

ICSE
LogiX 2.0 with BlueJ 

Computer Applications

A textbook for class X

Name

Class Section Roll No

Phone



Kips Learning Pvt. Ltd

C-12, Block C, Sector 3, Noida 201301, Uttar Pradesh

E-mail: info@kips.in **Website:** www.kips.in

Acknowledgements

All products and brand names used in this book are trademarks, registered trademarks or trade names of their respective owners.

© Kips Learning Pvt. Ltd: 2023

First published 2023

Fourth impression 2024

This book is sold subject to the condition that it shall not by way of trade or otherwise, be lent, resold, hired out, or otherwise circulated without the publisher's prior written consent in any form of binding or cover other than that in which it is published and without a similar condition including this condition being imposed on the subsequent purchaser and without limiting the rights under copyright reserved above, no part of this publication may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of both the copyright owner and the above-mentioned publisher of this book.

ISBN: 978-93-90676-93-4

e-book ISBN: 978-93-91164-03-4

Published by Kips Learning Private Limited

Head Office: C-12, Block C, Sector 3, Noida 201301, Uttar Pradesh

Registered Office: 45, Beniatola Lane, Kolkata 700 009, West Bengal, India

CIN: U22219WB2018PTC235056

Printed in India at Naba Mudran Pvt Ltd

All suggested use of the internet should be under adult supervision.

P R E F A C E



Scan the QR Code
for syllabus

“Learning to write programs stretches your mind, and helps you think better, creates a way of thinking about things that I think is helpful in all domains.”

—Bill Gates

Today, we live in a digital age, and most industries rely on data and software programs. Computer Science has touched our lives in all the spheres. Technology has made the world better, faster, and more connected. It is a part of our future, and thus, it is imperative that we equip young students, who are the future citizens of India, with skills around computing and communication devices. This course will help the students develop their critical thinking and problem-solving skills that go hand in hand.

KIPS, with its vast experience in the field of School Computer Education, has designed **Logix 2.0** series with a gradual learning approach towards the subject for a better learning experience of the students.

The focus of the books, **Logix 2.0** for classes IX and X, is to empower the students to learn and understand the concepts of object-oriented programming in Java using BlueJ. The contents of the books are in alignment with the latest ICSE syllabus for **Computer Applications (Code 86)**.

During the course, the students will learn to use both computer hardware and software.

The key features of these books are:

- **Exercises** at the end of the chapter with ample solved, unsolved, and application-based questions to reinforce the concepts learnt.
- **Programming Zone** with programs for hands-on practice and enhancing the programming skills of students.
- **Activity Zone** with additional questions for revision.
- **Board Questions** containing solved questions from previous years' exam papers.
- **Model Test Papers** both solved and unsolved for practice.
- **Viva Questions** and **Internal Assessment** to prepare students for the practical exam.
- **QR Codes** for videos explaining the important concepts in each chapter.

The author, **Mr Santosh Kumar**, is a senior Computer Science faculty in St. Xavier's Sr. Sec. School, Ranchi, since 2005. He is a Microsoft Certified System Engineer and has an expertise in designing and implementing various teaching strategies for making the learning of computer science easier.

We express our gratitude to **Mr Sanchayan K Ray**, Senior Subject Coordinator, Department of Computer Science, Don Bosco School, Kolkata, for his valuable review and inputs for improving the books.

We would also like to thank **Ms Poulami Das Akuli**, Computer Science Teacher in National Gems Higher Secondary School, Kolkata, for reviewing our books and giving suggestions for improving the content.

Suggestions for any improvement to the content are welcome so that they may be incorporated in the subsequent editions.

KIPS Content Development Team

Work Details

C O N T E N T S

1. Revision of Class IX	1
1.1 Procedure Oriented Programming (POP)	1
1.2 Object Oriented Programming (OOP)	2
1.3 Difference between Procedure and Object-Oriented Programming	3
1.4 Elements and Principles of OOP	3
1.5 Class as an Object Factory	7
1.6 Class as a User-defined Data Type	7
1.7 Message Passing between Objects	8
1.8 Introduction to Java	8
1.9 Introduction to BlueJ	11
1.10 Character Sets	18
1.11 Escape Sequence	18
1.12 Tokens	19
1.13 Data Types in Java	22
1.14 Variables	23
1.15 Constants	23
1.16 Type Conversion	24
1.17 Forms of Operators	25
1.18 Types of Operators	25
1.19 Precedence and Associativity of Operators	26
1.20 The println() and print() Methods	26
2. Revision of Class IX (Contd.)	39
2.1 Initialising and Assigning Data Values	39
2.2 Input using Parameters	40
2.3 Input using the Scanner Class	42
2.4 Types of Errors	44
2.5 Comments in Java	47
2.6 Packages in Java	48
2.7 Mathematical Functions in Java	50
2.8 Java Expressions	51
2.9 Decision Making Statements	58
2.10 if Statement	58
2.11 if-else Statement	59
2.12 if-else-if Statement	60
2.13 Nested if Statement	61
2.14 switch-case Statement	62
2.15 Difference between if-else and switch-case	63
2.16 Looping Statements	63
2.17 Classification of Loops	64
2.18 for Loop	65
2.19 while Loop	67
2.20 do-while Loop	68
2.21 Special Loops	69
2.22 Jump Statements	69
2.23 Interconversion of Loops	71
2.24 Comparison of Various Types of Loops	72
2.25 Nested Loops	73
3. Class as the Basis of all Computation	89
3.1 Objects and Classes	89
3.2 Objects Encapsulate State and Behaviour	90
3.3 Member Variables of a Class	91

3.4	Member Methods of a Class	93
3.5	Classes as Abstractions for Sets of Objects	93
3.6	Class as an Object Factory	94
3.7	Objects as Instances of a Class	95
3.8	Data Types	95
3.9	Variable Declarations	96
3.10	Some Real-life Examples of Classes and Objects	97
4.	User-defined Methods	111
4.1	Introduction	111
4.2	Defining a Method	112
4.3	Types of Parameters	115
4.4	Calling Methods	117
4.5	Using the return Statement in a Method	119
4.6	Programs Based on Methods	120
4.7	Pure and Impure Methods	122
4.8	Static and Non-static Methods	124
4.9	Method Overloading (Polymorphism)	126
5.	Constructors	144
5.1	Introduction	144
5.2	Characteristics of a Constructor	146
5.3	Difference between Constructors and Methods	146
5.4	Types of Constructors	146
5.5	Use of a Constructor	152
5.6	Constructor Overloading	153
5.7	Programs Based on Constructors	154
6.	Library Classes	172
6.1	Revisiting Data Types in Java	172
6.2	Wrapper Classes	173
6.3	Autoboxing and Unboxing	176
6.4	Methods of Wrapper Classes	177
7.	Encapsulation	188
7.1	Introduction	188
7.2	Access Specifiers	190
7.3	Visibility Rules for Access Specifiers	192
7.4	Scope of Variables	195
8.	Arrays	205
8.1	Introduction to Arrays	205
8.2	Array as a Composite Type	206
8.3	Types of Arrays	206
8.4	Declaring Single Dimensional Arrays	207
8.5	Initialising Single Dimensional Arrays	208
8.6	Declaring Double Dimensional Arrays	212
8.7	Initialising Double Dimensional Arrays	212
8.8	Finding the Length of an Array	215
8.9	Accessing the Elements of an Array	217
8.10	Searching Techniques	222
8.11	Sorting Techniques	227
9.	String Handling	249
9.1	Strings	249
9.2	Methods of String Class	250
9.3	Implementation of String Class Methods	254
9.4	String Array	259
	Practice Questions	285
	Internal Assessment	288
	Viva Questions	312

Chapter Highlights

Section A: Object Oriented Programming Concepts
Section B: Elementary Concepts of Objects and Classes
Section C: Values and Data Types
Section D: Operators in Java

Learning Outcomes

The student will be able to:

- know about the procedure-oriented and object-oriented programming.
- explain the principles of OOP.
- comprehend the use of classes and objects.
- understand the role of JVM and JDK.
- write and execute code in BlueJ.

Section A: Object Oriented Programming Concepts

Language is a very important part of any culture. It is the way in which people communicate with one another, build relationships, and create a sense of community. There are roughly 6,500 spoken languages in the world today, and each is unique in its own way. You all use a language to communicate with one another. Every language consists of the use of words in a structured way.

Similarly, in computing, a language that consists of a set of instructions that are clearly understood by the computer is known as the computer programming language. Each programming language has its own specific rules. A programming language tells computers what to do to perform certain tasks. The evolution of programming languages has made available so many high-level languages for programmers, such as C, C++, Java, Python, C#, PHP, Ruby, and so on. These languages enable programmers to write powerful application programs in an easy-to-understand manner.

Java is the most popular object-oriented programming language, developed by James Gosling at Sun Microsystems Inc. in 1991. You have already learnt the concepts of object-oriented programming in class IX. Let us quickly revise the different programming paradigms and later you will be introduced to the elementary concepts of OOP in detail. Furthermore, you will learn about the use of data types and operators in Java.



1.1 Procedure Oriented Programming (POP)

The procedure-oriented programming paradigm follows a process-centric approach. In this type of programming paradigm, a program is divided into functions when it becomes lengthy. Each function has a clearly defined purpose. In short, this type of paradigm puts more stress on a process rather than an object.

Many instructions are written to carry out tasks. Further, instructions are grouped into functions to perform those tasks. This paradigm uses a linear top-down approach and is mainly used for medium-sized applications.

The basic drawback of the procedural programming approach is that data is not secured because data is global and can be accessed by any function. Moreover, the code is not flexible enough to change. The application software needs to be upgraded as the requirements change. This results in more complexity in the program.

Features of POP

POP has the following features:

- ✦ It focuses on process rather than data.
- ✦ Several functions are written to solve a problem.
- ✦ Data moves openly around the system from function to function.
- ✦ Functions share global data.
- ✦ It uses a top-down approach.
- ✦ Some procedure-oriented programming languages are FORTRAN, ALGOL, COBOL, BASIC, Pascal, and C.

Limitations of POP

The following are certain limitations of POP:

- ✦ As data values are global to all the functions, any change in data values needs modification in all functions.
- ✦ It is not suitable for solving complex problems.
- ✦ It is often difficult to design because functions and data structures do not model the real world.
- ✦ It is difficult to create new data types. The ability to create a new data type of its own is called extensibility. Structured programming languages are not extensible.

1.2 Object Oriented Programming (OOP)

The Object Oriented Programming (OOP) paradigm is based on the concept of objects, i.e., it follows a problem-solving approach where all computations are carried out using objects. In this type of programming paradigm, more stress is given on objects rather than functions.

OOP does not allow data to flow freely. It is a modular approach that allows the data to be applied within a stipulated area. It also provides the reusability feature that allows you to reuse the code. Some examples of object-oriented programming languages are C++, Java, Smalltalk, Visual Basic, and Simula 67.

Features of OOP

OOP has the following features:

- ✦ It helps in breaking down the requirements into objects.
- ✦ Troubleshooting is simple and easy in these languages.
- ✦ More stress is given to objects rather than functions.

- ✦ It is more suitable for creating complex programs.
- ✦ New functionality can be easily developed by creating objects and functions.
- ✦ The feature of reusability supports programs or sections of programs to be reused.
- ✦ Multiple instances of a class can be generated to co-exist without any interference.
- ✦ The concept of data hiding can provide security to a program.
- ✦ Class can be extended through inheritance.

1.3 Difference between Procedure and Object-Oriented Programming

The object-oriented programming paradigm overcomes all the drawbacks of procedural programming. It is close to the real world, so it becomes easy to simulate the programming with your day-to-day life.

Procedure Oriented Programming	Object Oriented Programming
POP gives more stress on procedures.	OOP gives more stress on objects.
Data is shared among all the functions participating, thereby risking data safety and security.	Data and procedures both have equal importance. These are encapsulated to ensure data safety and security.
Examples: FORTRAN, ALGOL, COBOL, BASIC, Pascal, and C	Examples: C++, Java, Smalltalk, Visual Basic, and Simula 67

1.4 Elements and Principles of OOP

The following are different elements and principles of object-oriented programming:



Figure 1.1: Elements and Principles of OOP

Object

An object is a unique entity with some characteristics and behaviours. An object is a component in the program that knows how to perform certain actions and how to interact with other elements of the program. Objects are the basic units of object-oriented programming.

Example 1

A simple example of an object is a **mobile phone**. Every mobile phone has different attributes, such as name, brand, colour, model, weight, size, and camera resolution, which are referred to as its properties. The behaviour of an object can be correlated with activities that you do on the mobile phone, like dialling the number, starting the call, ending the call, taking a picture, etc.

Object



Attributes

name, brand, colour, model, weight, size, and camera resolution

Behaviour

dialling the number, starting the call, ending the call, taking a picture

Example 2

Consider another real-world object 'car'. Its properties are colour, model, brand, etc., and its behaviour is to drive, park, start, stop, accelerate, and so on.

In the real world, you find many objects around you, like books, trains, etc. All these objects have a state and behaviour. For example, somebody could be either 'happy', 'sad', or 'angry'. So, these are the behaviours, if you take these characteristics as 'mood'.

Class

A class is a group of objects that share common characteristics and behaviour. Objects are instances of a class. A class is a blueprint from which individual objects are created. You can think of a class as a concept, and the object is the implementation of that concept. You need to have a class before creating the objects.

Example 1

Consider Fruits as a class because fruits have some attributes (like fruit names: mangoes, apples, and oranges, colours: yellow, red, and orange) and behaviours (like taste and smell).

Class: Fruits



Example 2

Consider another class Dog, where different dogs can be defined as objects of that class.

Class: Dog



Object 1

Characteristics

breed, colour, age, gender, owner

Behaviour

eat, sleep, run, bark



Object 2

Characteristics

breed, colour, age, gender, owner

Behaviour

eat, sleep, run, bark

Encapsulation

The process of wrapping up data and associated functions together into a single unit is known as **encapsulation**. The basic functions of encapsulation are:

- ✦ Binding the data with the code that manipulates it
- ✦ Keeping the data and the code safe from external interference

Encapsulation improves the maintainability and flexibility of the program. It protects your class against any unauthorised access.

Example

Consider the example of a burger. A burger is made with a pair of buns, a cutlet, cucumber slices, lettuce, cheese, onion, and other ingredients. All of these ingredients can be arranged in a specific order to produce the final product, a tasty and nutritious burger. While having a burger, you are not bothered about the ingredients (data) and process (functions). The only thing you need to know is the final result, which is the 'burger'. The same concept is used for a class. A class combines data and functions into a single unit, which is referred to as encapsulation.



Abstraction

Abstraction is the act of representing the essential components and hiding the background details. It is a process where you show only "relevant" data and "hide" unnecessary details of an object from the user.

Example 1

Consider the example of a coffee machine. To get a cup of coffee, you press the dispensing buttons of the coffee machine. The dispensing buttons hide all the complexity of making coffee behind them.

Example 2

Consider another example of an ATM machine. To withdraw money from an ATM machine, you insert your debit card in to the provided slot and follow the on-screen instructions. After providing the correct PIN number and entering the desired amount, you collect the money from the machine. Here, the user interacts only with the essential features of an ATM machine: the card reading slot, the touch screen for input, and the money dispensing slot. The user does not need to know or understand the background processes the machine performs.

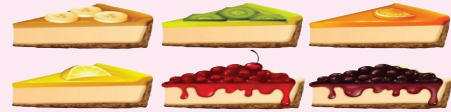


Polymorphism

Polymorphism is the concept of using the same function name multiple times in a program. It processes objects differently based on their data types. For example, a `draw()` method can draw a circle, rectangle or a triangle; an `accelerate()` method can accelerate a train, a car, or an aeroplane.

Example

Consider the example of pastries. Different pastries have different flavours, and their ingredients are also different. This is an example of polymorphism where the name would remain the same but the parameters are different.

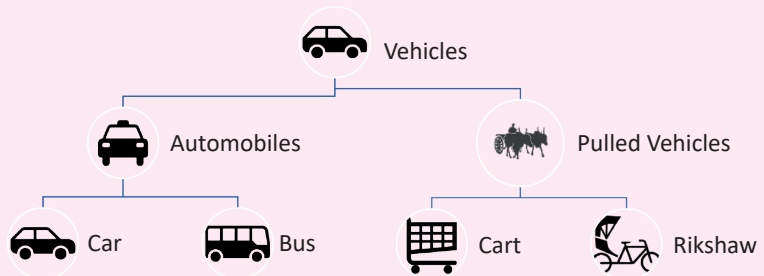


Inheritance

Inheritance enables an object of a class to acquire some properties from the objects of another class. It is the technique of building new classes from previously existing classes. A class that gets inherited is called the **parent/base/super** class. The class that inherits from the parent class is known as the **child/derived/sub** class.

Example

Consider the example of a class 'Car'. It inherits its properties from the class 'Automobiles', which inherits some of its properties from another class 'Vehicles'.



Section B: Elementary Concept of Objects and Classes

As you know, objects and classes are the basic units of object-oriented programming. Objects are referred to as entities and identified by distinct names, such as flower, chair, book, and so on. Classes are collections of objects that share common characteristics and behaviour. Object-oriented programming is a technique that enables the use of classes and objects to build programs more efficiently.

The following are the basic differences between class and objects:



Figure 1.2: Class and Objects

Class	Objects
A class is a blueprint from which objects are created.	An object is an instance of a class.
It has a logical existence.	It has a physical identity.
It is known as an object factory.	It is known as an instance of a class.
It is declared using the keyword <code>class</code> .	It is declared using the keyword <code>new</code> .

1.5 Class as an Object Factory

The term factory means a place where manufacturing takes place. Objects are created by using information available in the class. A class is a prototype of an object. Each object created by a specific class possesses the data and methods defined within the class. With one class, you can create several objects of a similar nature. Hence, a class is also referred to as an **object factory**.

When you create several objects from a class, each object gets unique properties of its own, and these objects exist in the dynamic memory of the computer. Thus, an object is called an **instance of a class**.

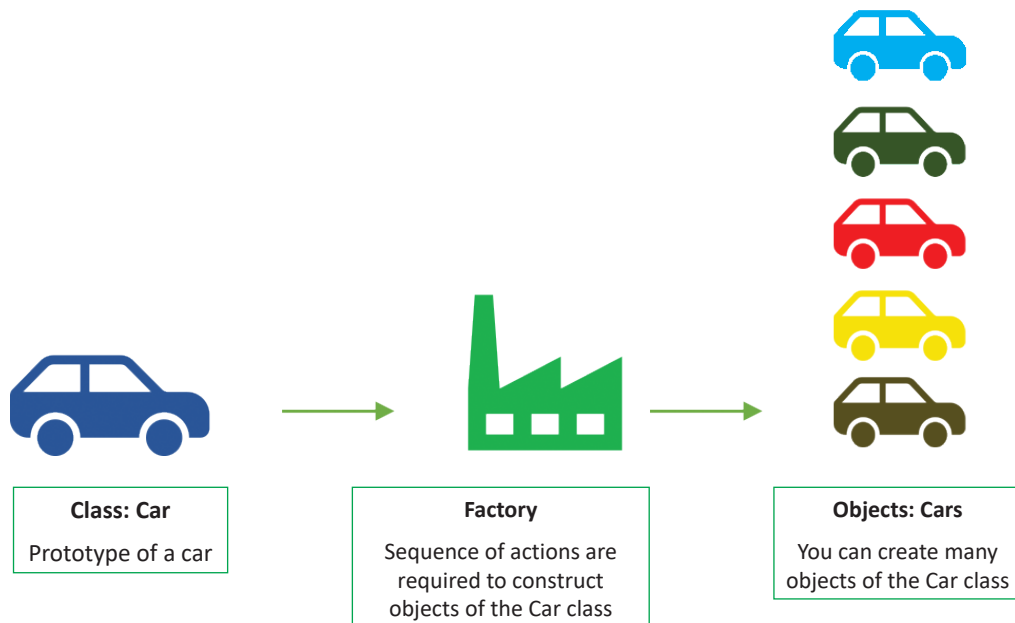


Figure 1.3: Class as an Object Factory

1.6 Class as a User-Defined Data Type

In the real world, more complicated objects are needed. OOP allows you to make real-world object models by using the concept of class. A class is a way to bind the data and associated functions together. It is a **user-defined** data type, which holds its own data members and member functions that can be accessed and used by creating an instance object of that class.

Example

```
class Car
{
    //data members
    String brand;
    int model;
    //methods
    speed();
}
```

Here, 'brand' and 'model' are variables, and 'string and int' are their data types, respectively.

The user-defined data type, 'Car' is used to create two objects: C1 and C2.

```
Car c1=new Car();
```

```
Car c2=new Car();
```

In the above example, different data types and objects are being used. Also, you can observe that a class consists of various fundamental data types. Since the class is created by the user, it is called a **user-defined data type**.



Quick Quiz

1. What are objects?

Objects are the unique entities with some characteristics and behaviours.

2. What is encapsulation?

The process of wrapping up data and associated functions together into a single unit is known as encapsulation.

1.7 Message Passing between Objects

In terms of computers, message passing is communication between processes. It is a form of communication used in object-oriented programming.

In Java, message passing means sending a message from one object to another. A **message** is a request to an object to perform an action (behaviour).

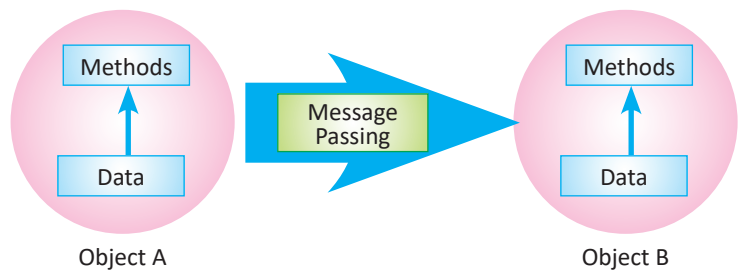


Figure 1.4: Message Passing

Consider the concept of producer and consumer. The consumer will only be able to consume what the producer produces. Similarly, objects communicate between each other by passing messages to perform some operations.

Example

Suppose, Alexa is an object and Music Player is a class. If you want Alexa to play a song, you simply send a message to the object, like:

```
Alexa.PlaySong();
```

The above statement is a request to the Alexa object to invoke its member method PlaySong() using the dot(.) operator.

1.8 Introduction to Java

Java is a full-featured, general-purpose, object-oriented programming language. Writing, compiling, and debugging a program is easy in Java. It helps to create modular programs and reusable code. The history of the Java programming language is usually associated with the World Wide Web (WWW). Its origin predates the web. Java's earlier name was Oak. Java applications are platform-independent, which means that Java applications can run on any platform.

Features of Java

The following are the different features of Java:

Platform-independent Language: Java is called a platform-independent language. The compiler converts the source code (.java file) to the byte code (.class file). This byte code can run on any platform, such as Windows, Linux, Mac OS, etc., which means a program that is compiled on Windows can run on Linux and vice-versa.

Object-Oriented Language: Java is an object-oriented programming language. It emphasises on the use of objects in programs. Each object represents an instance of a class.

Robust Language: Robust means reliable. Java programming language is developed in a way that puts a lot of emphasis on early checking for possible errors, that is why, the Java compiler can detect errors that are not easy to detect in other programming languages. The main features of Java that make it robust are garbage collection, exception handling, and memory allocation.

Distributed Language: Java programs can be distributed on more than one system that are connected using the internet connection.

Multithreading: Java supports multithreading. Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilisation of CPU.

Java Applications and Applets

The following are two types of Java programs:

Standalone Programs (Java Applications): Java is widely accepted as a development technology for standalone (desktop applications) development. Such programs or applications are destined to run on desktop applications, like word processors or spreadsheets.

Internet Applets (Java Applets): Internet Applets are small programs that need a web browser and are securely run on the machine by Java capable browsers. Applets are designed to be transferred to web browsers. This is the reason why an applet has a built-in graphic window.

Compilation in Java

Compilation is the process of converting a high-level programming language into a machine language. Language translators are used for compiling program codes. The following are the two types of language translators:

Compiler: A compiler is a program that translates the code written in a high-level language into machine code, in one go.

Interpreter: An interpreter is a program that translates the source code instructions into machine code, line-by-line.

In Java compilation, the source code is first converted to an intermediate code, i.e., the Java **byte code** which in turn is converted to the **machine code**. The following layout shows the steps involved from Java program to final execution:

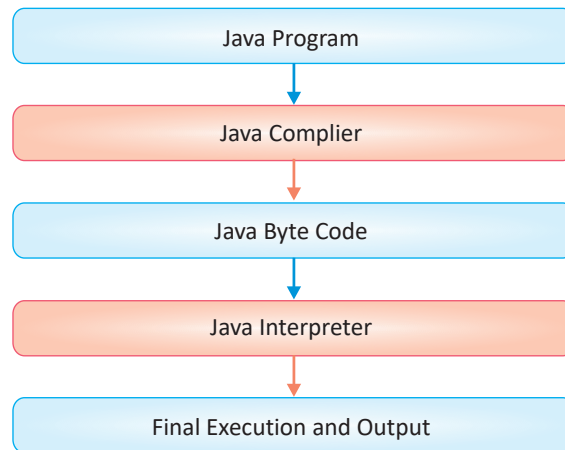


Figure 1.5: Step-by-Step Approach of a Java Program



Quick Quiz

1. **What is a compiler?**

A compiler is a program that translates the code written in a high-level language into machine code, in one go.

2. **What are the byte codes?**

Byte codes are the intermediate codes that are generated during compilation. It is processed by the Java Virtual Machine (JVM) to translate it into machine code.

Java Development Kit (JDK)

The Java Development Kit (JDK) is a product of Sun Microsystems used to develop Java software and is an extended subset of the Java **Software Development Kit (SDK)**. JDK consists of the API classes, a Java compiler, and the Java Virtual JDK. As the name implies, it is a complete Java development kit that includes **Java Runtime Environment (JRE)**, compilers, and various tools, like Javadoc, Java debugger, etc.

In order to create, compile, and run a Java program, you need to install JDK on your computer. It is used to compile Java applications and applets. It includes tools for developing Java applets and applications. JDK contains Java class libraries for different purposes.

These library classes contain some useful packages as listed below:

Package	Description
java.lang	It contains classes, like String, Character, Math, Integer, Thread, etc.
java.io	It is used to process the input and produce the output.
java.applets	It generates applets-specific environment.
java.util	It contains Scanner, Printer, Calendar, and various other utility classes.
java.awt	It supports abstract window toolkit.

Packages

A package is a collection of Java classes that are grouped on the basis of their functionality. The **import** keyword is used to load a package into a Java program.

Syntax:

```
java.package_name.*;
```

Example

```
import java.io.*;
```

A **Java package** is a group of similar types of classes, interfaces, and sub-packages. Packages in Java can be categorised into two forms, built-in package and user-defined package. There are many built-in packages, such as `java.lang`, `java.awt`, `java.url`, `java.io`, `java.util`, etc.

Java Bytecode

Java bytecode is an intermediate code that is obtained as a result of the compilation of a Java program. The Java bytecode gets processed by the **Java Virtual Machine (JVM)** instead of the processor.

Java Virtual Machine (JVM)

The **Java Virtual Machine (JVM)** is an interpreter of Java. It executes the bytecode produced by the compiler and converts it into machine code. Each operating system has a different JVM. However, the output produced by the JVM after the execution of bytecode is the same across all operating systems. This is why, Java is referred to as a platform-independent language.

1.9 Introduction to BlueJ

BlueJ is an **Integrated Development Environment (IDE)**, which includes an editor, a debugger, and a viewer. Let us learn how to create a class in BlueJ.

Creating a Class

To start with a new class, follow these steps:

- Click on the **New Class** button in the BlueJ editor. A dialog box **Create New Class** appears. Enter the class name as **trial**. Select the **Class Type** as **Class** and click on **OK** button.

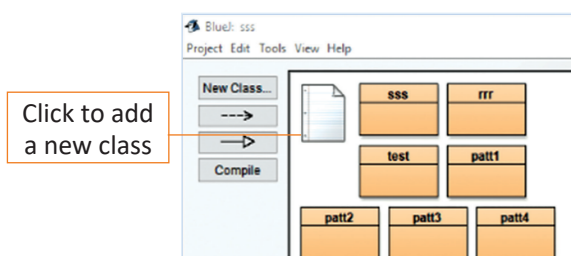


Figure 1.6: BlueJ Project Window

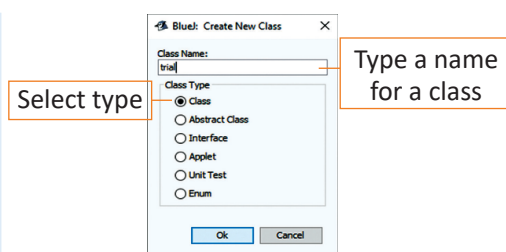


Figure 1.7: Creating a new Class

- In the **Project** window, you will get an icon representing the newly created class **trial**. The stripes on the icon indicate that the class has not been compiled yet.

Editing Class Code

In order to edit the class code, you need to open the class in the editor. To do so, follow these steps:

- In the Project window, double-click or right-click on the class icon and select the **Open Editor** option.

- ♦ The editor opens up. It contains some pre-written codes. Erase the code and type the following code to print a message.

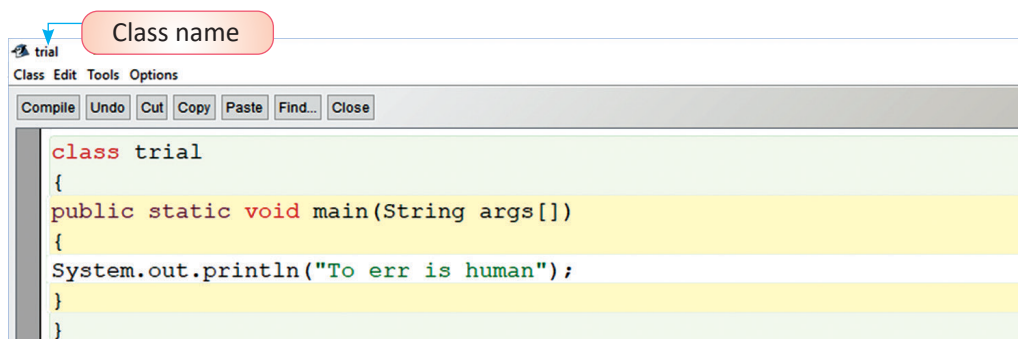


Figure 1.8: Editor Window

Explanation

Go through the following explanation to understand what each line of the code does.

Line 1: The keyword **class** denotes a class with the name trial.

Line 2: The first open curly bracket marks the beginning of the class code.

Line 3: This line consists of many keywords as listed below:

- ♦ **static**: It means main method can be called without an object.
- ♦ **public**: It is an access modifier, which means the compiler can execute this from anywhere.
- ♦ **void**: It is used to define the return type of the method. It defines what the method can return. Void means the method will not return any value.
- ♦ **main()**: It is the name of the method. This method name is searched by Java Virtual Machine (JVM) as a starting point for an application with a particular signature only.
- ♦ **String []args**: The JVM always looks for a specific method signature to start execution of an application, and that would be **public static void main (String args[])**. Here **args** is an argument of the type **String** array. String array argument can also be written as **String[] args**.

Line 4: The opening curly bracket indicates the beginning of the method code.

Line 5: The **System.out.println()** statement prints the text in double quotes (" ") as an output. The semicolon (;) indicates the end of a statement. In Java programming, you have to terminate every statement with a semicolon. The absence of this leads to a compile-time error.

Line 6: The closing curly bracket indicates the end of the method code.

Line 7: The last closing curly bracket marks the end of the class code.

Compiling Code

After writing the code, click on the **Compile** button located on the extreme left of the window or right-click on the class icon in the Project window and select **Compile**.

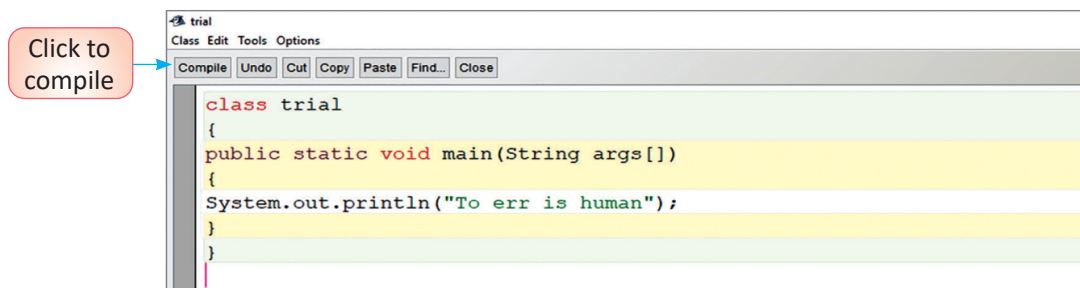


Figure 1.9: Compiling a Program

- ◆ If there is an error in the program, the cursor will move to the location of the error, and it will be marked with a red underline. The error message will be displayed in the text floating nearby.
- ◆ If there are no errors, you will get the message: Class compiled - no syntax errors and your changes are automatically saved. After a successful compilation, the Java compiler creates a machine language file (a bytecode file with the .class extension) in the project location folder.
- ◆ In this example, the default method **void main(String args[])** is used, so you can directly execute the code by right-clicking on the class icon **trial** and selecting the following method: **void main(String[] args)**

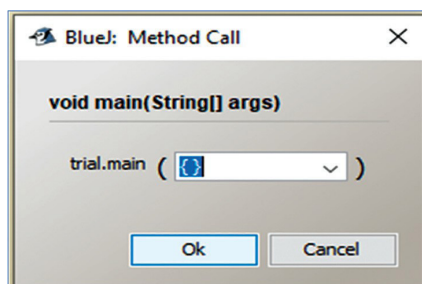


Figure 1.10: Calling Method

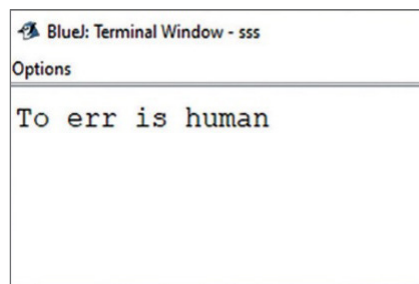


Figure 1.11: Output

- ◆ Otherwise, right-click on the class icon and select **newclass_name ()**. A dialog box appears. Enter the name of the object in the **Name of Instance** field and click on OK. An object icon will be created at the bottom of the Project window. Right-click on the object icon and select the method you want to execute.



Let's Revise

- ◆ A language that consists of a set of instructions that is clearly understood by the computer is known as the computer programming language.
- ◆ The procedure-oriented programming paradigm puts more stress on a process than an object.
- ◆ The object-oriented programming paradigm is based on the concept of objects.
- ◆ An object is a unique entity with some characteristics and behaviours.
- ◆ A class represents a set of objects that share common characteristics and behaviours. Objects are instances of a class.
- ◆ The process of wrapping up data and associated functions together into a single unit is known as encapsulation.
- ◆ The act of representing the essential components and hiding the background details is called abstraction.
- ◆ The ability of a method to behave in more than one form is called polymorphism.
- ◆ Inheritance is the mechanism by which an object acquires some or all the properties of another object.
- ◆ Java is a simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multi-threaded, and dynamic language.
- ◆ The Java Development Kit (JDK) is used to develop Java software and is an extended subset of the Java Software Development Kit (SDK).

- ❖ In Java, message passing means sending a message from one object to another. A message is a request to an object to perform an action (behaviour).
- ❖ The Java Virtual Machine (JVM) is an interpreter of Java. It executes the bytecode produced by the compiler.
- ❖ A package is a collection of classes and interfaces that provide a high-level layer of access protection and named space management.
- ❖ The import keyword is used to load a package into a Java program.
- ❖ A bytecode is an intermediate code that is obtained as a result of the compilation of a Java program.
- ❖ BlueJ is an Integrated Development Environment (IDE) for writing, editing, compiling, testing, executing, and debugging the Java programming language.

Key Terms

Program: A program is a collection of steps needed to perform a particular task.

POP: It is a programming language that puts more stress on functions rather than objects.

OOP: Object oriented programming puts more stress on objects rather than processes.

Source Code: It is a set of instructions written by a programmer.

Bytecode: It is an intermediate code that is obtained when the compiler compiles the Java source code.

JVM: It is the interpreter of Java that converts Java bytecode to machine code.

BlueJ: BlueJ is an IDE. It combines the features of an editor, a debugger, and a viewer.

Modelling: Modelling is the process of creating or giving a shape to any structure.

Entity: An entity refers to anything that has an identifiable body.



SECTION 1 (Solved Questions)

A. Fill in the blanks.

1. The wrapping of data and functions in an object or class is called _____.
2. A _____ is a group of similar objects.
3. JVM is an _____ of Java.
4. The basic function of JVM is to execute the _____ produced by the compiler.
5. An _____ is an instance of a class.
6. A _____ is a named collection of Java classes that are grouped on the basis of their functionality.

Ans.	1. Encapsulation	2. Class	3. Interpreter	4. Bytecode	5. Object	6. Package

B. Very short answer type questions.

1. Give examples of any two object-oriented languages.

Ans. C++ and Java

2. Name any two OOP's principles.

Ans. Encapsulation and Abstraction

3. Define JVM.

Ans. The Java Virtual Machine is an interpreter that converts the Java bytecode into machine code.

4. What are the two types of Java programs?

Ans. The two types of Java programs are:

a. Standalone Applications

b. Internet Applets

5. "Java uses both a compiler and an interpreter." Explain the statement.

Ans. The Java compiler converts the program to Java bytecode, which in turn is converted to machine code by the interpreter — the Java Virtual Machine.

6. Name the machine independent code generated by Java compiler.

Ans. Java bytecode

7. Define standalone application.

Ans. A standalone Java application refers to a Java program that can run independently on a computer.

8. What is an object?

Ans. An object is a unique entity that has some characteristics and behaviour.

9. What is a class?

Ans. A class is a group of objects that share some common characteristics and behaviour.

10. A class is termed as an "object factory." Define.

Ans. With one class, you can create several objects of a similar nature. Hence, a class is termed an "object factory".

C. Short answer type questions.

1. Describe the object-oriented programming.

Ans. The object-oriented programming language is a modular approach to programming in which equal stress is given to data as well as methods. It allows the data to be applied within the stipulated program area. It also provides reusability features to develop productivity logic. Examples of OOP based languages are Simula, C++, Java, Python, and Smalltalk.

2. Write any two features of OOP?

Ans. The features of OOP are as follows:

✦ It gives equal stress to data and function.

✦ Data can be easily modified without any change in the function.

3. What do you mean by object in Java?

Ans. It is a unique entity with some characteristics and behaviour. For example, a car is an object. It has many characteristics, like colour, make, model, and engine size.

4. What is class in Java?

Ans. A class is a group of objects that share common characteristics and behaviour. It is a user-defined data type that binds the data and associated functions together.

5. What is the need of a class in Java?

Ans. Classes in Java are needed to represent real-world entities, which have data type properties. Classes provide convenient methods for packing together a group of logical related data items and functions. It serves as a template for creating, or instantiating, specific objects within a program.

6. Define message passing.

Ans. In Java, message passing means sending a message from one object to another. A message is a request to an object to perform an action (behaviour).

7. What is data abstraction?

Ans. The act of representing only essential features without including the background details is called data abstraction. For example, to drive a car, only the accelerator, clutch, and brake controls need to be learnt rather than the workings of the engine and other details.

8. Explain polymorphism.

Ans. The ability of a method to behave in more than one form is called polymorphism. Function overloading is a way to implement it, in which more than one function has the same name but different argument list.

9. Define inheritance.

Ans. The ability of a class to derive properties from another class is called inheritance. The class that inherits is called the derived/sub/child class, and the class from which it is derived is called base/super/parent class.

10. What is a bytecode?

Ans. Java compiler generates machine independent bytecode instead of machine dependent code. Bytecode is a set of pseudo machine language instructions that are understood by the JVM (Java Virtual Machine) and are independent of the underlying hardware.

11. Describe the use of BlueJ.

Ans. BlueJ is an Integrated Development Environment (IDE) for writing, editing, compiling, testing, executing, and debugging the Java programming language.

D. Application-based questions.

1. Saniya's teacher has asked her about the role of JVM. Describe JVM's role to assist her.

Ans. JVM stands for Java Virtual Machine. It is a Java interpreter that executes the Java bytecode and converts it into machine code.

2. Why is an object called an instance of a class? Explain.

Ans. When you create several objects from a class, each object gets unique properties of its own, and these objects exist in the dynamic memory of the computer. Thus, an object is called an instance of a class.

SECTION 2 (Unsolved Questions)

A. Very short answer type questions.

1. Define the terms JDK and IDE.
2. What is source code in Java?
3. Define bytecode.
4. Why is Java referred to as platform-independent?
5. How can you create objects in Java?

- ### C. Application-based questions.

- ## SECTION 3 (Board Questions)

Object	Class
An object is an instance of a class.	A class is a blueprint from which objects are created.
It is a physical identity.	It is a logical identity.
It can be created in multiple instances.	It is declared only once.
It is created using the 'new' keyword.	It is created using the 'class' keyword.

To learn any programming language, you should be acquainted with its building blocks. Let us revise the fundamentals of Java in detail.

1.10 Character Sets

A character set is a set of all the letters of the English alphabet and some special characters that are valid in the Java language. The Java character set consists of the following:

Letters: A, B, C ... X, Y, Z (uppercase characters) and a, b, c... x, y, z (lowercase characters)

Digits: 0, 1, 2, 3, ...9

Special Characters or Delimiters: !, ~, #, @, ^, <>, -\$, *, (), etc.

Operators: +, -, *, %, >=, !=, etc.

The character set of Java acts as building blocks to form basic programming elements, such as constants, variables, and operators. Java also makes use of certain combinations of characters, such as `\n`, and `\t` to represent special conditions, like new line and new tab, respectively. There are two standard character sets, ASCII and UNICODE, that are commonly used in Java.

ASCII Code

ASCII is an acronym for the **American Standard Code for Information Interchange**. It is a code that uses numbers to represent characters. Each letter is assigned a number between 0 and 127.

ASCII is a 7-bit character set containing 128 characters. It contains the numbers from 0-9, the uppercase letters A to Z, and lowercase letters from a to z, and some special characters. The character sets used in modern computers, in HTML, and on the internet are all based on ASCII.

UNICODE

UNICODE is a 16-bit character encoding standard. It is an international standard of character encoding, which is capable of representing almost every character in well-known languages around the world. Before UNICODE, there were multiple standards for representing character encoding:

- ♦ ASCII for the United States
- ♦ ISO 8859-1 for Western European Language
- ♦ KOI-8 for Russian
- ♦ GB18030 and BIG-5 for Chinese

To support multinational application codes, some characters use a single byte and some characters use two bytes. Even the same code may represent a character differently in two different languages.

To overcome these shortcomings, the UNICODE system was developed where each character is represented by 2 bytes.

In UNICODE, a character holds 2 bytes, so Java also uses 2 bytes for characters.

- ♦ **Lowest value:** `\u0000`
- ♦ **Highest value:** `\uFFFF`

1.11 Escape Sequence

An escape sequence is a combination of characters that has a special meaning to the Java compiler. It is represented by a backslash (`\`) followed by one or more characters. In other words, an escape sequence is defined as a sequence of characters that does not represent itself when used inside a

character or string literal, but rather is translated into another character or a sequence of characters that may be difficult or impossible to represent directly in the program code. The following table shows some commonly used escape sequences:

Escape Sequences	Non-graphic Characters
\a	Audible bell
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\'	Single quote
\"	Double quotes
\\	Backslash

1.12 Tokens

A token is the smallest unit of a program that is easily understood by the compiler. It is the basic building block of a Java program. Tokens can be classified into the following:

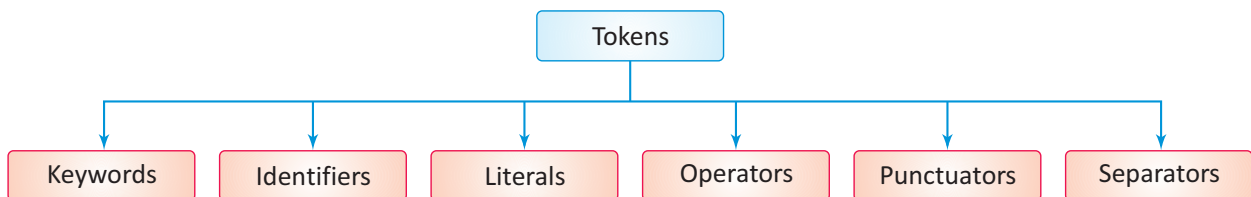


Figure 1.12: Tokens in Java

Keywords

Keywords are reserved words that convey a special meaning to the Java compiler. Keywords are a special part of a language's definition. They are predefined, i.e., the meaning of the keywords has already been described to the Java compiler, and it cannot be changed. Thus, keywords cannot be used as variable names because that would try to change the existing meaning of the keyword, which is not allowed.

Keywords are part of the syntax and they cannot be used as an identifier. The following are the different keywords that you can use in Java:

abstract	continue	for	new	switch
assert	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instance of	return	transient

catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const*	float	native	super	while

Figure 1.13: Keywords

NOTE

The keywords marked with an asterisk (*) are not in use. For example, 'const' and 'goto' are reserved, even though they are not currently used; true, false, and null might seem like keywords, but they are actually literals. You cannot use them as identifiers in your programs.

Identifiers

Identifiers are the names of variables, methods, classes, packages, and interfaces. You should keep in mind the following rules while writing an identifier:

- ✦ All identifiers must start with either a letter (a to z or A to Z) or a special character, such as underscore (_) or dollar sign(\$).
- ✦ They must not begin with a digit.
- ✦ After the first character, an identifier can have any combination of characters.
- ✦ A Java keyword cannot be used as an identifier.
- ✦ Identifiers in Java are case sensitive, 'book' and 'Book' are two different identifiers.
- ✦ They can be of any length.

The following are some **valid identifiers**:

Emp_Name	empid	EMPAGE	x	i
\$myvariable	_9pins	android	CS_BigB	_myvariable

The following are some **invalid identifiers**:

My Variable	It has a space.
9pins	It begins with a digit.
a+c	The plus sign is not an alphanumeric character.
testing1-2-3	The hyphen is not an alphanumeric character.
O'Shalu	Apostrophe is not an alphanumeric character.
SPS_&_Associates	Ampersand is not an alphanumeric character.



Quick Quiz

1. What is an escape sequence?

An escape sequence is a combination of characters that has a special meaning to the Java compiler.

2. What does ASCII stand for?

ASCII is an acronym for American Standard Code for Information Interchange.

Literals

A literal is sequence of characters used in a program to represent a constant value. They are made of digits, letters, and other characters. For example, 123, 16.243, "ABCD" etc. Java supports the following types of literals:

Integer Literals: Integer literals are values that represent whole numbers only. They can have positive or negative values. Examples: 13, 5, -27, -137, etc.

Real Literals: Real literals are also called floating point literals. They represent values with a decimal point. Examples: 3.14, -23.75, 24.023, etc.

Character Literals: Literals of this type represent exactly one character inside single quotes.

Examples: 'a' '8', '@', etc.

String Literals: Zero or more characters enclosed inside double quotes are called String literals.

Examples: "Shiva", "Bytes", "28/01/2022", etc.

Boolean Literals : The logical values "true" and "false" are known as the boolean literals.

Null Literal: This is a special kind of literal which is represented by the keyword 'null' or escape sequence '\0'.

Operators

Operators are special symbols that carry out operations on one or more operands. They can be used to perform a variety of arithmetic, relational, and logical operations. You will learn about operators in detail in Section D.

Punctuators

Punctuators are special punctuation marks used in Java programs as listed below:

Punctuator	Name	Use
.	Period (Dot)	It is used to separate the package name from the sub-package name and class name. It is also used to access member variables and methods.
;	Semicolon	It is used to terminate a statement in Java.

Separators

Separators help to define the structure of a program. The following are some characters that are generally used as separators in Java:

Separator	Name	Use
()	Parenthesis	It holds the list of parameters in method definition. Also, it is used in control statements and type casting.
{ }	Curly Brackets	It is used to define the block/scope of code, classes, and methods.
[]	Square Brackets	It is used in array declarations.
,	Comma	It is used to separate the consecutive parameters in the method definition. It is also used to separate the multiple variables under the same declaration.

1.13 Data Types in Java

In Java programs, you need to handle various types of data, so it becomes necessary to select an appropriate data type according to the data taken in a program. There are two types of data types: primitive and non-primitive.

Primitive Data Types

Primitive types are the built-in data types in Java. They are predefined by the language. These data types are independent of any other data type. There are eight primitive data types in Java as described below:

Data Type	Size	Range	Default Value	Description
byte	8 bits (1 bytes)	-128 to +127	0	It is the smallest integer data type that can store whole numbers from -128 to 127.
short	16 bits (2 bytes)	-32768 to +32768	0	It is probably the least used integer data type in Java. It is used to declare short integers.
int	32 bits (4bytes)	-2^{31} to $2^{31} - 1$	0	It is the most commonly used data type for declaring integer variables.
long	64 bits (8bytes)	-2^{63} to $2^{63} - 1$	0	It is used to store long integers in Java programs. You should use this data type when you need a range of values wider than those provided by int.
float	32 bits (4bytes)	$-3.4E + 38$ to $+3.4 + 38$	0.0f	It is a floating-point data type. It is useful when you require a fractional component, but do not require a large degree of precision.
double	64 bits (8bytes)	$-1.7E + 308$ to $1.7E+308$	0.0d	It is another floating-point data type that has an greater range of values. It is used when you need to maintain accuracy over many iterative calculations or manipulate large-valued numbers.
char	16 bits (2 Bytes)	0-65536	'\u0000'	It is used to hold a character.
boolean	Occupies 1 byte Uses 1 bit	true or false	false	It is used to represent logical values as either true or false.

Non-primitive Data Types

Non-primitive data types are also known as reference types. Non-primitive data types depend on other data types. Classes, Interfaces, and Arrays are examples of non-primitive data types.

1.14 Variables

A variable is a memory location that holds the data value of a particular data type. Each variable has a name by which it is identified in the program. The value stored in a variable can be changed during program execution. In Java, all the variables must be declared before use. Declaring a variable means getting a memory location allocated to store a value of the specified data type. This memory location can be accessed by the specified data type.

Syntax:

```
<data-type> <variable-name>;
```



Where the data type can be any **primitive data** type (like String, int, double, char, and so on) and the **variable name** is the name of the variable.

Example

```
int emp_age;
```

The given statement will declare an integer type variable `emp_age`. You can also assign a value to the variable at the time of its declaration. For example, the following statement would declare an integer type variable `emp_age` and assign an initial value 25 to it.

```
int emp_age = 25;
```

Rules for Naming Variables

You should keep in mind the following rules while declaring a variable:

- ✦ Any alphabetic character, any number, and two special characters, such as underscore (`_`) and the dollar sign (`$`), can be used in variable names.
- ✦ You cannot begin a variable name with a number. Also, variables cannot contain any whitespace.
- ✦ A declaration statement must end with a semicolon.
- ✦ It is a good idea to give your variables mnemonic names that are closely related to the values they hold.
- ✦ Variables must be declared before they are used. Any changes in variables cause actual changes in the memory location.

1.15 Constants

Constants are data values that never change their values during program execution. In Java, if you precede a variable declaration with the keyword **final**, the identifier becomes a constant. For example, the value of 'g' can be used in the program as a constant for various calculations. It can be declared as follows:

```
final float g = 9.8f;
```

In the above statement, the value of 'g' cannot change anywhere in the program.

1.16 Type Conversion

The process of converting one predefined type into another is called type conversion. There are two types of data type conversions: implicit and explicit.

Implicit Type Conversion

Implicit type conversion or **widening casting**, often known as **coercion**, is the default type conversion of an expression. When an expression has more than one data type, the compiler performs this type of conversion automatically. During this conversion, the Java compiler converts all data types to their higher types.

Example

```
byte a=10; int c = a + 10;
```

Here, a is converted to int automatically.

NOTE

Hierarchy indicates the increasing order of data types. If two data of different types are operated then result automatically gets converted to their higher type as shown below:

byte -> short -> char -> int -> long -> float -> double

Explicit Type Conversion

An explicit type conversion is performed manually through user intervention. It is also known as **narrowing casting** or **typecasting**. This type of conversion is required when you assign a larger data type to a smaller type.

Example

```
float f = (float) a/ float(b); if a, b are double.
```



Quick Quiz

1. **What do you understand by type conversion?**

The process of converting one data type to another is called type conversion.

2. **How many primitive data types are supported by Java?**

Java supports 8 primitive data types.

Let's Know More

When primitive data type is converted to its corresponding object of its class, it is called **boxing**. The opposite operation is known as unboxing, i.e. the process of converting an object into a corresponding primitive data type is known as **unboxing**.

Section D: Operators in Java

As you know, operators are special symbols that can be used to perform some operations on one or more operands. A combination of operators and operands is referred to as an **expression**.

The values on which operations are performed are referred to as **operands**. For example, in expression, $Z=X*Y$, Z, X, Y are the operands and the symbols * and = are the operators.

1.17 Forms of Operators

There are three forms of operators in Java: unary, binary, and ternary as discussed below:

Unary Operators	Unary operators require only a single operand. They perform various operations, such as incrementing/ decrementing a value by one, negating an expression, or inverting the value of a boolean. Unary operators include the unary plus (+), unary minus (-), increment (++), decrement (--), and logical not (!).
Binary Operators	Binary operators work on two operands. Operators and operands together form an expression. There are three types of binary operators: arithmetic, relational, and logical.
Ternary Operator	The ternary operator is also known as conditional operator. It works on three or more operands. This operator is used to decide for a value based on given conditions. It is a condensed form of the if-else statement that also returns a value.

1.18 Types of Operators

The following are the different types of operators in Java:

Operators	Description
Arithmetic Operators (+, -, *, /, %)	Arithmetic operators are used to perform basic mathematical operations like addition, subtraction, multiplication, and so on.
Relational Operators (>, <, ==, !=, >=, <=)	Relational operators are used to compare two values. They either return True or False according to the condition.
Logical Operators (&&, , !)	Logical operators are used to compare two or more than two boolean expressions. They return the result by yielding true or false depending upon the outcomes of different boolean expressions that you have included in your logical expression.
Assignment Operator (=)	The assignment operator is used to assign a value to a variable. For example, a = 2. Here, 2 is assigned to the variable 'a'. It assigns the value on its right to the variable on its left.
Increment Operator (++)	The increment operator is used to increase the value of a variable by 1, i.e., i = i + 1
Decrement Operator (--)	The decrement operator is used to decrease the value of a variable by 1, i.e., i = i - 1
new Operator	It is a special operator that is used to create new objects of a class. It dynamically allocates the memory required for the new object.
Dot Operator (.)	The dot operator is also known as the member operator. It is used to invoke the member of a class or package.

Know the Fact

The increment and decrement operators can be assigned either before or after the operand. When the increment/ decrement operator is used before the operand, such as ++x or --x, it is known as the **prefix**, and when it is used after the operand, such as x++ or x--, it is known as the **postfix**.

Read More

Counters and accumulators are special variables. A counter variable stores the number of times a particular situation occurs in a program, whereas an accumulator is used to store the partial results while some repeated tasks are performed.

1.19 Precedence and Associativity of Operators

Precedence is the order in which a program evaluates the operations in an expression. An operator with a higher precedence value is evaluated first then the operator having a lower precedence value. If two operators have the same precedence, then they are either evaluated from "Left to Right" or from "Right to Left". This is termed as **associativity**. It tells the direction of execution of operators. The following table shows the precedence and associativity of operators:

Precedence	Operators	Associativity
[], (), .	Array Index, Method Call, Member Access	Left to Right
++, --	Postfix Increment, Postfix Decrement	Right to Left
++, --, +, -, ~, !	Prefix Increment, Prefix Decrement, Unary	Right to Left
*, /, %	Multiplication, Division, Modulus	Left to Right
+, -,	Addition, subtraction String concatenation	Left to Right
<, >, <=, >=, instance of	Relational	Left to Right
==, !=	Equality	Left to Right
&&	Logical AND	Left to Right
	Logical OR	Left to Right
? :	Ternary	Right to Left
=, +=, -=, *=, /=, %=, &=, ^=, =, <<=, >>=, >>>=	Assignment	Right to Left

1.20 The println() and print() Methods

The `println()` method prints a string and moves the cursor to a new line, whereas the `print()` method prints a string, but does not move the cursor to a new line. Hence, subsequent printing instructions will print on the same line.

Consider the following programs to understand the use of concept you have learnt in the chapter.

Program 1

Write a program to display the use of typecasting.

```
1 public class Typecasting
2     public static void main(String args[])
3     {
4         // Declaring and initialising different variables
5         byte b = 42;
6         char c = 'a';
7         short s = 1024;
8         int i = 50000;
9         float f = 5.67f;
```

OUTPUT

Result = 626.7784146484375

```

10     double d = .1234;
11     double result = (f*b)+(i/c)-(d*s);           //automatic typecasting
12     System.out.println("Result = " + result); // use of println() method
13 }
14 }

```

NOTE

A line that begins with a double backslash ("//") in a program is referred to as a single-line comment. A comment statement is ignored by the compiler and interpreter. It is provided to give information about the code or logic.

Program 2

Write a program to display the result of arithmetic calculations, if a=40 and b=20.

```

1  public class Calculation    //defining class
2  {
3      public static void main(String[] args)
4          //main method of class
5      {
6          int a = 30, b = 40;
7          System.out.println("a + b = " + (a + b)); //use of + operator
8          System.out.println("a - b = " + (a - b)); //use of - operator
9          System.out.println("a * b = " + (a * b)); //use of * operator
10         System.out.println("a / b = " + (a / b)); //use of / operator
11         System.out.print("a % b = " + (a % b));    //use of % operator
12     }
13 }

```

OUTPUT

```

a + b = 60
a - b = 20
a * b = 800
a / b = 2
a % b = 0

```

Program 3

Write a program to display the result either as true or false using relational operators, if a=274, b=294, and c=3000.

```

1  public class Calculation    //defining class
2  {
3      public static void main(String[] args)
4          //main method of class
5      {
6          int a = 274, b = 294, c = 3000;    //declaring variables
7          System.out.println("a > b = " + (a > b));
8          //use of > operator
9          System.out.println("a < b = " + (a < b));
10         System.out.println("a <= b = " + (a <= b));
11         System.out.println("a >= b = " + (a >= b));
12         System.out.println("a == b = " + (a == b));
13         System.out.println("a != b = " + (a != b));
14     }
15 }

```

OUTPUT

```

a > b = false
a < b = true
a <= b = true
a >= b = false
a == b = false
a != b = true

```

```

//use of < operator
8   System.out.println("a <= c = " + (a <= c));
//use of <= operator
9   System.out.println("a >= c = " + (a >= c));
//use of >= operator
10  System.out.println("b == c = " + (b == c));
//use of == operator
11  System.out.println("a!=b =" + (a!=b));
//use of ! operator
12  }
13  }

```

Program 4

Write a program to display the use of all the logical operators, if a=360, b=240, and c=200.

```

1   public class Logical1
2   {
3       public static void main(String[] args)
4       {
5           int a = 360, b = 240, c=200;
6           //using logical AND, OR, and NOT
7           System.out.println("a and b both are even numbers:" + ((a%2==0)&& (b%2==0)));
8           System.out.println("a is greater than b or c: " + ((a>b)|| (a>c)));
9           System.out.println("a is not greater than b and c:" + (!((a>b)&&(a>c)));
10      }
11  }

```

OUTPUT

and b both are even numbers: true
a is greater than b or c: true
a is not greater than b and c: false

Program 5

Write a program to find the greater numbers using the ternary operator.

```

1   public class operators
2   {
3       public static void main(String[] args)
4       {
5           int a = 31, b = 19, result;
6           result = (a > b ? a : b); // ternary expression
7           System.out.println("Greater number is:" + result);
8       }
9   }

```

OUTPUT

Greater number is: 31

Program 6

Write a program to access display () method of the class Student to show the use of new and dot operators.

<pre>1 public class Student 2 { 3 void display() //member function 4 { 5 String name="Rohan"; // String variable 6 int age=30; 7 double salary = 20000; 8 System.out.println("Name:" + name); 9 System.out.println("Age:" + age); 10 System.out.println("Salary:" + salary); 11 } 12 public static void main(String args[]) 13 { 14 Student S1 = new Student();//using the new operator to create object class 15 S1.display(); //using the dot operator to access member function 16 } 17 }</pre>	OUTPUT Name: Rohan Age: 30 Salary: 20000.0

NOTE

String is not a primitive data type. The java.lang package provides the String class, therefore, it is an object type. You can create a string variable directly like any other variables.



Let's Revise

- ◆ An escape sequence is a combination of characters that has a special meaning to the Java compiler.
- ◆ A token is the smallest unit of a program that is easily understood by the compiler.
- ◆ Keywords are reserved words that convey a special meaning to the Java compiler.
- ◆ Identifiers are the names of variables, methods, classes, packages, and interfaces.
- ◆ A literal is a sequence of characters used in a program to represent a constant value.
- ◆ Punctuators are special punctuation marks used in Java programs.
- ◆ Separators help to define the structure of a program.
- ◆ There are eight primitive data types in Java: byte, short, int, long, char, float, double, and boolean.
- ◆ Classes, Interfaces, and Arrays are known as non-primitive data types.

- ❖ Variables are containers for storing data values.
- ❖ Implicit type conversion takes place automatically by the compiler when an expression has more than one data type and converts all data types to their higher types.
- ❖ An explicit type conversion is performed manually through user intervention. It is required when you assign a larger data type to a smaller type.
- ❖ Operators are special symbols that can be used to perform operations on one or more operands.
- ❖ Unary operators require only a single operand and binary operators work on two operands.
- ❖ Ternary operator is also known as conditional operator. It works on three or more operands.
- ❖ Arithmetic operators are used to perform basic mathematical operations on various data items.
- ❖ Relational operators are used to compare two values.
- ❖ Logical operators are used to compare two or more than two boolean expressions. They return the result by yielding true or false depending upon the outcomes of different boolean expressions.
- ❖ The assignment operator is used to assign some value to a variable.
- ❖ The new operator is used to create objects of a class. It dynamically allocates the memory required for the new object.
- ❖ The dot (.) operator is used to invoke the member of a class or package.
- ❖ Precedence is the order in which a program evaluates the operations in a formula or expression.

Key Terms

Data Type: It is the type of data used for storage in memory.

Keyword: These are the reserved words that convey some special meaning to the Java compiler.

Punctuator: Punctuators are special punctuation marks used in a Java program.

Separator: Separators are symbols used to distinguish sections of a Java program.

Operands: Operands are the data items on which operations are performed.

Operator: It is a symbol that works on some operands to generate results.

Precedence of operators: It is the order in which operators perform the operations.



SECTION 1 (Solved Questions)

A. Fill in the blanks.

1. The smallest unit used in a Java program is _____.
2. _____ is a word which conveys a special meaning to the Java program.
3. The keyword _____ used while declaring a variable makes it constant.

4. The character set of Java is called _____.
5. _____ data types are the built-in data types in Java.
6. _____ operators work with two operands.
7. The statement `i+=5` is equal to _____.
8. The _____ operator is used to find the remainder when one number is divided by another.
9. The _____ operator is also known as the conditional operator.
10. The output of the statement `(5>2 && 7<3)` is: _____.

Ans.	1. Token	2. Keyword	3. final	4. UNICODE	5. Primitive
	6. Binary	7. <code>i=i+5</code>	8. Modulus	9. Ternary	10. False

B. Very short answer type questions.

1. What is a token?

Ans. A token is the smallest unit used in a Java program.

2. What are the different types of tokens?

Ans. The different types of tokens are: keywords, literals, identifiers, operators, punctuators, and separators.

3. What are literals?

Ans. Literals are constants whose value does not change during the execution of the program.

4. What is a variable?

Ans. A variable is a memory location that holds the data value of a particular data type. Each variable has a name by which it is identified in the program.

5. Name any four primitive data types.

Ans. The four primitive data types are: byte, short, int, and long.

6. What is the size of the 'float' and 'double' data types?

Ans. float: 4 bytes

double: 8 bytes

7. What are operators?

Ans. Operators are symbols that perform some arithmetic, relational, or logical operations.

8. How are operators and operands interrelated?

Ans. An operator uses operands to perform any operation in order to generate a meaningful result. Thus, operators and operands are interrelated.

9. What is the difference between '=' and '=='?

Ans. The operator '=' is an assignment operator used to assign a value to a variable, whereas the operator '==' is a relational operator used to check whether two values are the same or not.

10. If a and b are integer variables and the value of a is 3, then what will be the values of a and b after executing the statement: `(a++)-(++a) = b`?

Ans. a = 5, b = -2

C. Short answer type questions.

1. Write any of the four rules for naming identifiers.

Ans. The following are the rules for naming identifiers:

- ◆ An identifier name must not begin with a digit.

- ◆ It should not be a keyword.
- ◆ It should not contain any special characters other than the underscore (_) and dollar (\$).
- ◆ It should not contain whitespace in between.

2. What is literal? List different types of literals.

Ans. A literal is sequence of characters used in a program to represent a constant value. The following are the different types of literals:

- ◆ Integer literals
- ◆ Floating literals
- ◆ Boolean literals
- ◆ Character literals
- ◆ String literals

3. What do you understand by declaring a variable?

Ans. Declaring a variable means getting a memory location allocated to store a value of the specified data type. This memory location can be accessed by the specified data type.

4. What are primitive data types?

Ans. Primitive data types are the built-in data types in Java. They are defined by the language and do not depend on other data types. Examples: byte, short, int, long, char, float, double, and boolean.

5. Why is a class known as the composite data type?

Ans. A class is defined by the use of various primitive data types, such as int, double, etc. That is why, it is known as a composite or non-primitive data type.

6. What is the difference between unary and binary operators?

Ans. Unary operators require only a single operand. They perform various operations, such as incrementing or decrementing a value by one, negating an expression, or inverting the value of a boolean. Binary operators work on two operands. There are three types of binary operators: arithmetic, relational, and logical.

7. Explain the ternary operator.

Ans. The ternary operator is also known as the conditional operator. It works on three or more operands. This operator is used to decide for a value based on given conditions.

8. What will be the values of m and n after executing the code?

```
int n = 20;
int m = n++%5;
```

Ans. m = 0, n = 21

9. What are logical operators? List different logical operators.

Ans. Logical operators are used to compare two or more than two boolean expressions. They return the result by yielding true or false depending upon the outcomes of different boolean expressions that you have included in your logical expression. The three logical operators used in Java are: AND(&&), OR(||), NOT(!).

10. What are counters and accumulators?

Ans. Counters and accumulators are special variables. A counter variable stores the number of times a particular situation occurs in a program, whereas an accumulator is used to store the partial results while some repeated tasks are performed.

D. Application-based questions.

1. Write statements to declare the variables for the following data types.

a. double	b. char
c. boolean	d. float

Ans. a. double d;
b. char c='S';
c. boolean b=true;
d. float f=22.0/7;

2. Evaluate the expression: $n+ = n++ + ++n$, if $n=10$.

Ans. $n+= n++ + ++n$
 $n=n + (n++ + ++n)$
 $n=10 + (10 + 12)$
 $n=32$

3. Evaluate the expression: $(x>=y) \parallel !(z==y) \&\&(z<x)$, if $x=20$, $y=7$, and $z=25$.

Ans. $(x>=y) \parallel (z==y) \&\&(z<x)$
 $\text{true} \parallel \text{true} \&\& \text{false}$
 $\text{true} \parallel \text{false}$
 true

4. Rewrite the following decision-making statement using the conditional operator.

```
if(x>0)
t=1;
else
t=0;
```

Ans. $t=(x>0)?1:0;$

SECTION 2 (Unsolved Questions)

A. Very short answer type questions.

1. Name any two primitive and two non-primitive data types.
2. What are constants?
3. What are punctuators?
4. Explain the separators.
5. What does ASCII stand for?
6. How do you declare a variable? Give syntax.
7. What is an expression?
8. State the use of the new operator.
9. Describe the precedence and associativity of operators.
10. What will be the output of the following, if $x = 5$ initially?
 - a. $5 * ++x$
 - b. $5 * x++$

B. Short answer type questions.

1. Why are the data types required?
2. Differentiate between variables and constants.
3. What are identifiers and tokens?
4. Explain the difference between relational and logical operators.
5. What are escape sequences? Give any two examples.
6. What is type conversion? How is an implicit conversion different from an explicit conversion?
7. Define keywords. Give any two examples.
8. Explain the following:
 - a. Postfix and prefix increments
 - b. Postfix and prefix decrements
9. What will be the output of the following program segment?

```
int a, b;  
a = 6;  
b = 8;  
b = ++a;  
System.out.println(+a);
```
10. Describe the use of the new operator.

C. Application-based questions.

1. Predict the return data type of the following code segments:

```
int p;  
double q;  
r = p+q;  
System.out.println(r);
```
2. Write programs for the following:
 - a. A program to input the length and breadth of a rectangle and find its area.
 - b. A program to input a two-digit number and find the sum of its digits.
 - c. A program to input three numbers and find their average.
3. Explain the methods `print()` and `println()`.

SECTION 3 (Board Questions)

1. State the difference between a Boolean literal and a Character literal. [2009]

Ans.	Boolean	Character
	It can contain either true or false.	It can contain any character.
	It uses one byte for storage.	It uses two bytes for storage.

2. What are keywords? Give an example. [2012] [2016]

Ans. Keywords are words that convey some special meaning to the Java program. Some of the keywords used in Java are: `int`, `if`, `public`, `static`, and `for`.

3. Give one example of each of primitive and non-primitive data types? [2012]

Ans. Primitive: `int`
Non-primitive: `class`

4. What is meant by precedence of operators?

[2013]

Ans. Precedence of operators in which operands are operated in an expression. For example, division has higher precedence over addition so it will be executed before. So, the expression $15+20/4$ will return 15 as a result.

5. Identify the literals listed below:

[2015]

- a. 0.5 b. 'A' c. false d. "a"

Ans. a. Real b. character c. boolean d. String

6. Evaluate the value of n, if value of p=5 and q=19.

[2015]

```
int n=(q-p)>(p-q)?(q-p):(p-q);
```

Ans. $n=(q-p)>(p-q)?(q-p):(p-q);$

$= (19-5)>(5-19)?(19-5):(5-19);$

$= 14>-14?14:-14$

$= 14$

7. Rewrite the following using ternary operator.

[2016]

```
if(x%2==0)
```

```
System.out.println("Even");
```

```
else
```

```
System.out.println("Odd");
```

Ans. `System.out.println((x%2==0)?"Even":"Odd");`

8. Write one difference between / and % operators?

[2017]

Ans. The division operator(/) divides the first operand by the second operand and returns the quotient whereas the modulus operator(%) returns the remainder when the first operand is divided by the second. For example, $5/3=1$ and $5\%3=2$.

9. State the number of bytes required by char and int data types?

[2017]

Ans. char: 2 bytes int: 4 bytes

10. `int res='A';` What is the value of res?

[2018]

Ans. 65

11. Write the output of the following:

[2018]

```
System.out.println("Incredible"+"\\n"+"world");
```

Ans. Incredible

World

12. What is the value of y after evaluating the expression given below when `int a=8`?

[2018]

```
a += ++a + a-- + --a;
```

Ans. `a += ++a + a-- + --a;`

```
a= a + (++a + a-- + --a);
```

```
= 8 + (9 + 9 + 7)
```

```
= 33
```

Case Study I

In Java, the ternary operator can be used to replace certain types of if...else statements. It is also possible to use one ternary operator inside another ternary operator. It is called the nested ternary operator in Java. You do not need to assign the contents of the ternary operator to a variable. For instance, you could write a ternary operator in a `System.out.println()` statement as:

```
System.out.println("Greater number is:" + ((a > b)? a : b));
```

This will let you see the result of your ternary operator in the Java console. Unlike an "if" statement, the ternary operator does not accept an "else" keyword. The ternary statement uses the colon (:) to represent an "else" condition.

1. Which is the correct ternary expression for the following code?

```
if(a>b)
```

```
max=a;
```

```
else
```

```
max=b;
```

a. <code>max=(a>b)?a:b;</code>	b. <code>max=(a>b):b?a;</code>
c. <code>max=(a>b)=a;</code>	d. <code>max=(a>b)=b;</code>
2. The symbol of ternary operator is _____ .
- | | |
|---------------------|---------------------|
| a. <code>? =</code> | b. <code>{ }</code> |
| c. <code> </code> | d. <code>?</code> |
3. The ternary statement uses the colon (:) to represent a/an _____ .
- | | |
|----------------------|-------------------|
| a. if condition | b. else condition |
| c. if-else condition | d. None of these |
4. Write the output of the following statement:
- ```
r = (5%2==0)?12:13
```
- |       |       |
|-------|-------|
| a. 2  | b. 12 |
| c. 13 | d. 5  |

Ans. 1. a      2. d      3. b      4. c

## Case Study II

A variable is a memory location that holds the data value of a particular data type. Each variable has a name by which it is identified in the program. The value stored in a variable can be changed during program execution. In Java, all variables must be declared before use. Declaring a variable means getting a memory location allocated to store a value of the specified data type. This memory location can be accessed by the specified data type. Java uses various kinds of data types. However, the data types are mainly categorised into two types: primitive data types and non-primitive data types.

Primitive data types are already hard coded into the compiler to be recognised when the program is executed. Examples are int, float, etc. Non-primitive data types are special types of data that are user-defined, i.e., the program contains their definition. Some examples are classes, interfaces, etc.

1. \_\_\_\_\_ is a named memory location used to store a value.
- |             |              |
|-------------|--------------|
| a. Variable | b. Delimiter |
| c. Static   | d. Address   |

2. \_\_\_\_\_ data type stores non-decimal numeric data.
- float
  - double
  - int
  - char
3. \_\_\_\_\_ data type could be used to test for success or failure of a code.
- char
  - boolean
  - long
  - check
4. \_\_\_\_\_ a variable means getting a memory location allocated to store a value of the specified data type.
- Declaring
  - Initialisation
  - Assigning
  - None of these
- Ans. 1. a      2. c      3. b      4. a

## Programming ONE

1. Write a program to compute the surface area of a sphere.

**Hint:** Surface area of sphere =  $4\pi r^2$

Ans. `public class Sphere`

```
{
 public static void main(String args[])
 {
 final double PI = 3.14159; //constant variable
 double radius = 17.8;
 double surfaceArea;
 surfaceArea = 4 *PI * radius * radius;
 System.out.println("Surface area of sphere is" + surfaceArea);
 }
}
```

2. Write a program to find and display the value of the given expressions.

$(a + 3)/6 - (2a + 5)/3$ ; taking the value of  $a = 5$

Ans. `public class exp1`

```
{
 public static void main(String args[])
 {
 int a = 5;
 double value = ((a + 3) / 6.0) - ((2 * a + 5) / 3.0);
 System.out.println("Result = " + value);
 }
}
```

3. Bhanu Shing is paid ₹500 per day he works and fined ₹10 for each day he does not. Write a program to compute and display his monthly income if he worked for 20 days and afterwards went on vacation for 10 days.

Ans. `public class` MonthlyIncome

```
{
 public static void main(String args[])
 {
 int salary = 30 * 500;
 int fine = 10 * 10;
 int netSalary = salary - fine;
 System.out.println("Monthly Income is = " + netSalary);
 }
}
```

4. Write programs for the following:

- To calculate the area and perimeter of the rectangle, if length=5 and breadth=7.
- To calculate the perimeter of a triangle having sides of length 2,3 and 5 units.
- To check if the two numbers 23 and 45 are equal or not.
- To check whether the both conditions 'a < 50' and 'a < b' are true or false, if values if a=55 and b=70.
- To swap the values of the two variables, if a=6 and b=8.



Link

Scan the given QR Code to the println( ) and print( ) methods.



## SDG Goal 15

### Life on Land

Protect, restore, and promote sustainable use of terrestrial ecosystems, sustainably manage forests, combat desertification, and halt and reverse land degradation and halt biodiversity loss





## Chapter Highlights

Section A: Input in Java

Section B: Mathematical Library Methods

Section C: Conditional Constructs in Java

Section D: Iterative Constructs in Java

## Learning Outcomes

The student will be able to:

- understand the various types of input methods used in Java.
- know about the Scanner class and its methods.
- identify common errors and use comments.
- know about different types of packages and their usage.
- use the conditional and iterative constructs.

You have revised the basic concepts of Java in the previous chapter, like data types and operators. In this chapter, you will revise the different input methods, methods of the Math class, and conditional and iterative constructs in Java.

## Section A: Input in Java

The most versatile aspect of a computer is that you can perform different operations on different types of data. The computer expects data to be provided by the user for performing the various operations on it.

Generally, Java is considered the default choice for various scientific applications due to its safe, portable, secure, and easily maintainable features. You must be aware that every application requires some data as an input. Java has used a variety of data input methods since its inception. Backward compatibility has also been taken into consideration during the development process.

I/O is the communication between an information processing system, such as a computer, and the outside world, possibly a human or another information processing system. Inputs are the signals or data received by the system and outputs are the signals or data sent from it.

## 2.1 Initialising and Assigning Data Values

The process of assigning a value to a variable for the first time in a program is called **initialisation**. Before the program execution begins, you must decide what type of data you want to use to initialise your variables. Also, ensure that the variables are declared before they are initialised. Look at the following statements:

```
int x; String n; char c;
x = 7; n = "CS"; c = 'A';
```

In the above statements, the variables x, n, and c have been declared first, and then they are assigned with some values. For example, x has been assigned an integer value of 7, n has been assigned the

### Let's Know More

The process of assigning a value to a variable for the first time in a program is called **initialisation**.

### Know the Fact

Modifiers are keywords that are added to the definitions to change their meanings. Modifiers in the Java language are divided into two categories: Access modifiers and Non-access modifiers. To use a modifier, you must include its keyword in the class, method, or variable definition.

### Read More

The Java Scanner class allows the user to take console input. It is part of the 'java.util' package. It is used to read primitive type values, such as byte, int, double, long, short, and float as an input. It is the easiest method of reading input in a Java program.

string value 'CS', and c has been assigned the character value A. You can also assign the values to the variables in the same line of declaration as given below:

```
int x = 31;
String n = "Raman";
boolean r = false;
```

### Program 1

Write a program to display the name and age of a student.

```
1. public class Student
2. {
3. public static void main(String args[])
4. {
5. String name;
6. int age;
7. name = "Khushi";
8. age = 15;
9. System.out.println("Name:"+name);
10. System.out.println("Age: "+age);
11. }
12. }
```

### OUTPUT

Name: Khushi  
Age: 15

In the above program, 'name' is a String variable and 'age' is an integer variable. 'Khushi' and 15 are the values that are assigned to the String and integer variables, respectively.



## 2.2 Input using Parameters

In Java, you can input your data using parameters. Parameters are specified after the method name (function name) and inside the parentheses. The values (inputs) necessary for a method (function) to perform the task are sent as parameters. You can add as many parameters as you want in a method, separated by a comma.

Parameters can be actual parameters or formal parameters. The key difference between actual parameters and formal parameters is that actual parameters are the values that are passed to the function when it is invoked, while formal parameters are the variables defined by the function that receive values when the function is called. When you take input through parameters, you need to enter the required values at runtime, i.e., during the program execution.

Consider the following example, in which the total( ) function is defined with two integer parameters. When you execute the program, you need to enter values for these two parameters to get the sum of their values.

```

class cl_parameter
{
 public static void total(int a, int b)
 {
 int c=a+b;
 System.out.println("Sum="+c);
 }
}

```

**Figure 2.1:** Total ( ) Function with Two Parameters

**Figure 2.2:** Entering Two Integer Values at Run Time

**Figure 2.3:** Sum of Entered Values as Output

Consider another example in which the display( ) function is defined with a String and an integer. When you execute the program, you need to enter values for these two parameters to display the information.

```

class cl_display
{
 public static void display(String s, int m)
 {
 System.out.println("Name="+s);
 System.out.println("Marks="+m);
 }
}

```

**Figure 2.4:** Total ( ) Function with Two Parameters

**Figure 2.5:** Entering a String and an Integer Value at Run Time

**Figure 2.6:** Output

#### NOTE

Variables whose values are to be used must be declared in a function or method while providing input through parameters.

## Program 2

Write a program that takes the parameter 'name' as a String.

```
1. public class DisplayName
2. {
3. static void find(String name)
4. {
5. System.out.println("Hi !!!" +name);
6. }
7. public static void main(String[] args)
8. {
9. find("Amar");
10. find("Akbar");
11. find("Anthony");
12. }
13. }
```

### OUTPUT

```
Hi !!!Amar
Hi !!!Akbar
Hi !!!Anthony
```

// Calling functions

When a parameter is passed to the method in the function call, it is called an **argument**. In the above example, 'name' is a parameter, while 'Amar', 'Akbar', and 'Anthony' are the arguments.

## 2.3 Input using the Scanner Class

The Scanner class is one of the predefined classes that is used to read the data dynamically from the keyboard, i.e., at run time. To use the Scanner class, you need a predefined package **java.util**, which can be called in the program using the following statement:

```
import java.util.Scanner;
```

### NOTE

To use the Scanner class you need jdk 1.5 or above with BlueJ 2.5 or above.



## Constructor of the Scanner Class

To read data from the keyboard, you need to create an instance (object) of the Scanner class, which is passed to the **System.in** (an input stream which is used to read data from the keyboard) in the constructor of the Scanner class. The **new** operator is used to create an instance of the Scanner class. For example, the following statement creates an object of the Scanner class by taking an object of the InputStream class.

```
Scanner sc = new Scanner(System.in);
```

## Advantages of Providing Input through the Scanner Class

The advantages of providing input through the Scanner class are listed below:

- ✦ It accepts data in the token form, so there is no need to mention the type of data.
- ✦ String manipulation is quite easy as the Scanner class provides various functions.

- ✦ It also can be used for powerful pattern matching.
- ✦ There is no need for exception handling.

## Working of the Scanner Class

The input received by the Scanner class is broken into tokens using a delimiter pattern. Let us discuss tokens and delimiters here.

### Tokens

A token is a series of characters that ends with a delimiter. Thus, if you read a line that has a series of numbers separated by blanks, the Scanner will take each number as a separate token. The tokens are considered to be of the string type and manipulated through various methods in the Scanner class.

### Example

Consider the string "ALL THE BEST". The tokens in this string are "ALL", "THE", and "BEST".

### Delimiter

A delimiter is a character used to separate two tokens (data items). It can be a whitespace (default delimiter), a tab character, a carriage return, or the end of file (EOF) character. It is used to finish the processing of one piece of data and move on to the next one. Most delimiters are characters that will not be used in the data, such as spaces or commas. It must also be consistent throughout the data.

## Methods of the Scanner Class

You can use the methods of the Scanner class to read tokens (data) through the keyboard. Java provides various next methods to accept the input from the user in various formats. For example, to read the input as a float value, you can invoke the `nextFloat()` method of the Scanner class. The following table lists various next methods along with their usage:

| Data Type         | Methods                   | Description                                            |
|-------------------|---------------------------|--------------------------------------------------------|
| byte              | <code>nextByte()</code>   | To scan the next token of the input as a byte          |
| short             | <code>nextShort()</code>  | To scan the next token of the input as a short integer |
| int               | <code>nextInt()</code>    | To scan the next token of the input as an integer      |
| long              | <code>nextLong()</code>   | To scan the next token of the input as a long integer  |
| float             | <code>nextFloat()</code>  | To scan the next token of the input as a float         |
| double            | <code>nextDouble()</code> | To scan the next token of the input as a double        |
| String (word)     | <code>next()</code>       | To scan the next token of the input as a single word   |
| String (sentence) | <code>nextLine()</code>   | To scan the next token of the input as a sentence      |

Consider the following program to understand the use of a `nextDouble` method.

### Program 3

Write a program to input the principal, rate, and time. Calculate the simple interest and final amount.

```
1 import java.util.*;
2 public class scan3
3 {
4 public static void main(String args[])
5 {
6 Scanner input = new Scanner(System.in);
7 double p,t,r,si,a;
8 System.out.println("Enter principal, rate and time");
9 p = input.nextDouble();
10 t = input.nextDouble();
11 r = input.nextDouble();
12 si = (p*r*t)/100;
13 a = p+si;
14 System.out.println("Simple Interest="+si);
15 System.out.println("Amount="+a);
16 }
17 }
```

#### OUTPUT

```
Enter principal, rate and time
200000
10
2
Simple Interest=40000.0
Amount=240000.0
```



### Quick Quiz

#### 1. What is initialisation in Java?

The process of assigning a value to a variable for the first time in a program is called initialisation.

#### 2. What is an argument?

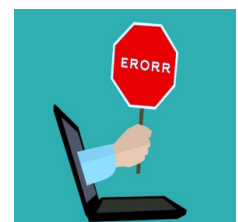
When a parameter is passed to the method in the function call, it is called an argument.

#### 3. Distinguish between the nextInt( ) and nextFloat( ) methods.

The nextInt( ) method scans the next token of input as an int, whereas the nextFloat( ) method scans the next token of input as a float.

## 2.4 Types of Errors

An error is an illegal operation performed by the user that results in the abnormal working of the program. Programming errors often remain undetected until the program is compiled or executed. Some of the errors inhibit the program from getting compiled or executed. Thus, they should be removed before execution.



There are three types of errors in Java, as discussed below:

## Runtime Errors

Runtime or execution errors occur during the execution of the program. Sometimes, these are exposed when the user enters invalid data or data that is not relevant. In other words, runtime errors occur when a program does not contain any syntax errors, which means it is syntactically correct, but has a logical issue that is only detected during the program execution. For example, if the user inputs data in string format when the computer is expecting an integer, there will be a runtime error.

To handle runtime errors, you can use the try-catch block. The try block holds a set of statements where an exception (error) can occur, whereas the catch block holds the statements that are used to handle the exception that occurs in the associated try block. A try block is always followed by a catch block.

### Example

```
// A Java program to demonstrate the runtime error
public class Run_Error
{
 public static void main(String args[])
 {
 int a = 10;
 int b = 0;
 int c= a/b;
 System.out.println (c);
 }
}
```

The above-written program will result in an error because the variable 'c' will not be able to obtain any value due to division by zero.

## Syntax Errors

Syntax errors occur due to the incorrect use of syntax in a program, such as a missing semicolon at the end of a statement or a missing bracket, class not found, etc. These errors are detected by the Java compiler and an error message is displayed on the screen while compiling. These errors are sometimes also referred to as **compile time errors**.

Syntax errors are easy to spot and rectify because the Java compiler finds them for you. The compiler will tell you which piece of code in the program got in trouble and what type of error is there. Usually, the compiler indicates the exact line where the error is, or sometimes the line just before it. However, if the problem is with incorrectly nested braces, the actual error may be at the beginning of the block. Syntax errors can be considered as grammatical errors that are caused due to mistakes in the programming language. For example, wrong variable names or method names, and incorrect use of punctuation in the code.

### Example

```
// A Java program to demonstrate the syntax error
public class Syn_error
{
 public static void main(String args[])
 {
 int a = 40,
 int b = 60;
 int sum = a + b;
 System.out.println("Sum of variables is " = sum);
 }
}
```

In the above code you can notice two errors: The **first error** is the absence of a semi colon (;) after the first line and the **second error** is the absence of (+) in the print statement. The correct code is given below:

```
public class Syn_error
{
 public static void main(String args[])
 {
 int a = 40;
 int b = 60;
 int sum = a + b;
 System.out.println("Sum of variables is " + sum);
 }
}
```

### Logical Errors

Logical errors occur when your program compiles and executes but does something incorrectly, such as producing an inaccurate result or having no output when it should. These errors are detected neither by the compiler nor by the interpreter JVM. The Java system has no idea what your program is supposed to do, so it provides no additional information to help you find the error.

Logical errors are also called **semantic errors**. These errors are caused by an incorrect idea or concept used by a programmer while coding. For example, if a programmer accidentally adds two variables when he meant to subtract them, the program will not provide any error and execute successfully, but with an incorrect result.

### Example

```
// A Java program that displays the logical error
public class Log_error
{
 public static void main(String args[])
 {
```



```

{
 int a = 12, b = 8;
 int c=a-b;
 System.out.println("sum="+ c);
}
}

```

The above-written code will result in wrong output because the use of the wrong symbol (instead of '+' symbol '-' has been used).

## 2.5 Comments in Java

A comment is an explanation of the source code of the program or the part of the program that is added to make the source code easy to understand. A comment statement is generally ignored during the compilation of the program. Sometimes, comments can also be used to mark a few lines for non-execution. This is used for understanding the logic of the program and also for testing the program.

In a program, comments play a part in making the program more human-readable by putting the details of the code involved. Proper use of comments makes maintenance easier and finds bugs easily. In Java, there are three types of comments as listed below:

### Single-line Comments

Single-line comments are used for explanations in a single line. By using single-line comments, you can make simple notes in the relevant areas of the source code of the program. These are denoted by two forward slashes (//).

#### Syntax:

```
//Comments here (Text in this line is considered as a comment.)
```

#### Example

```

//Java program to show single line comments
public class Scomment
{
 public static void main(String args[])
 {
 System.out.println("Learning Java"); // Printing a message
 }
}

```

In the above program, statements starting with two forward slashes (//) are called single-line comments.

### Multi-line Comments

It can be tedious to use single-line comments to describe a full method in code or a complex snippet because you have to give two forward slashes (//) at every line. To overcome this, multiline comments can be used. These comments are generally used for a long description of program code or to describe a block of code while debugging programs.

### Syntax:

```
/*Comment starts
continues
continues
Comment ends*/
```

### Example

```
//Java program to show multiline comments
public class Mcomment
{
 public static void main(String args[])
 {
 System.out.println("Java is a platform independent language.");
 /*The System.out.println() is a statement, which prints the argument
 passed to it.
 The println() method displays results on the monitor.
 Usually, a method is invoked by objectname.methodname().*/
 }
}
```

In the above program, statements written within `/*....*/` are called multiline comments.

## 2.6 Packages in Java

A package is a group of Java classes that are packed together based on their functionality. It contains multiple classes, each providing a special functionality that can be used in the program whenever required.

You may consider package as a mechanism to encapsulate a group of classes, sub-packages, and interfaces. Packages are used in Java programming for the following reasons:

- ◆ Packages are used to prevent naming conflicts. For example, there can be two classes with name 'staff' in two different packages, `teaching.staff` and `non_teaching.staff`.
- ◆ Packages make searching/locating and usage of classes, interfaces, enumerations, and annotations easier.
- ◆ They provide controlled access to data items.
- ◆ The packages can be considered as data encapsulation.

### Accessing Classes Inside a Package

You can access the classes of a particular package in two ways:

- ◆ To import only one class from a package  
`import java.util.vector;` ————— Specify the name of the class that you want to import
- ◆ To import all the classes from a package  
`import java.util.*;` ————— Use an asterisk symbol after the name of the package

The first statement is used to import the **vector** class from the **util** package, which is contained inside the Java package. The second statement imports all the classes from the util package.

## Types of Packages

There are two types of packages in Java: built-in packages and user-defined packages.

### Built-in Packages

These packages consist of various classes that are included in the Java API. Some of the commonly used built-in packages are:

| Package     | Description                                                                                                                                                                                               |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| java.lang   | It contains classes for basic programming, primitive data types (Wrapper class), String and Math functions.<br>[Note: This is the only package that is loaded by default when a Java program is created.] |
| java.util   | It holds the basic utility classes required by the programmer, such as Scanner and Printer.                                                                                                               |
| java.io     | It has classes for input and output of data.                                                                                                                                                              |
| java.awt    | It contains classes to implement or manage Windows (Graphical User) interface.                                                                                                                            |
| java.net    | It has classes for networks.                                                                                                                                                                              |
| java.applet | It contains classes for using applets.                                                                                                                                                                    |

All these standard Java classes are stored in a package called **Java**. To use a package in your program you need to call that package using the **import** keyword as given below:

#### Syntax:

```
import java.package_name.*;
```

#### Example:

```
import java.util.*;
```

### User-defined Packages

These are the packages that are defined by the user. A programmer can create their package by grouping all the classes designed for an application. These packages are called user-defined packages.

#### Rules for Naming a User-defined Package

- ✦ It must not be a keyword.
- ✦ Space and special characters are not allowed.
- ✦ The first character of the package name must be an alphabet.
- ✦ After the first alphabet, you can use digits, letters, or underscore.
- ✦ Every package name must be unique.
- ✦ The package name should be meaningful.

## Section B: Mathematical Library Methods

A method is a set of statements that perform a specific task in a class. It can be defined in a program by the user or built by the software developers. Let us learn about some commonly used mathematical methods in Java.

### 2.7 Mathematical Functions in Java

The **java.lang** package contains the classes that are essential to a Java program. It is loaded in a Java program, by default. This package has many important classes as its members. One of these members is the **Math** class. It contains methods for performing the basic mathematical operations, such as calculating the power, logarithm, square root, cube root, and trigonometric functions.

Let us now learn some basic methods from the package `java.lang.Math`.

| Math Function              | Syntax                     | Explanation                                                                                                                            | Example                                                                                                                                                 |
|----------------------------|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>Math.sqrt( )</code>  | <code>Math.sqrt(n)</code>  | This function returns the square root of a positive integer. It returns a double type value.                                           | <code>Math.sqrt(36) = 6.0</code>                                                                                                                        |
| <code>Math.cbrt( )</code>  | <code>Math.cbrt(n)</code>  | This function returns the cube root of a positive integer. It returns a double type value.                                             | <code>Math.cbrt(125) = 5.0</code>                                                                                                                       |
| <code>Math.pow( )</code>   | <code>Math.pow(a,b)</code> | This function returns the power of a variable raised to the other. It returns a double type value.                                     | <code>Math.pow(2, 5) = 32.0</code>                                                                                                                      |
| <code>Math.max( )</code>   | <code>Math.max(a,b)</code> | This function is used to find the bigger of the two numbers. Its return type will depend upon the type of values used in the function. | <code>Math.max(15,45)=45</code><br><code>Math.max(7.5,14.5)=14.5</code>                                                                                 |
| <code>Math.min( )</code>   | <code>Math.min(a,b)</code> | This function is used to find the smaller of two numbers. Its return type will depend upon the type of values used in the function.    | <code>Math.min(15,45)=15</code><br><code>Math.min(7.5,14.5)=7.5</code>                                                                                  |
| <code>Math.abs( )</code>   | <code>Math.abs(a)</code>   | This function returns the absolute value of a number given within the brackets. Its return type will depend on the data provided.      | <code>Math.abs(10)=10</code><br><code>Math.abs(-10)=10</code><br><code>Math.abs(15.10)=15.10</code>                                                     |
| <code>Math.round( )</code> | <code>Math.round(a)</code> | This function returns the rounded value of a number given within the brackets. Its return type is <code>int</code> .                   | <code>Math.round(10.4) = 10</code><br><code>Math.round(10.7) = 11</code><br><code>Math.round(15.10) = 15</code><br><code>Math.round(-10.5) = -10</code> |

| Math Function  | Syntax                                       | Explanation                                                                                                                                                                        | Example                                                                                               |
|----------------|----------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| Math.ceil( )   | Math.ceil(a)                                 | This function returns the next higher integer value of a number given within the brackets. Its return type is double.                                                              | Math.ceil(10.4)=11.0<br>Math.ceil(10.7)=11.0<br>Math.ceil(-15.10)=-15.0<br>Math.ceil(-10.5)=-10.0     |
| Math.floor( )  | Math.floor(a)                                | This function returns the previous lower integer value of a number given within the brackets. Its return type is double.                                                           | Math.floor(10.4)=10.0<br>Math.floor(10.7)=10.0<br>Math.floor(-15.10)=-16.0<br>Math.floor(-10.5)=-11.0 |
| Math.random( ) | r = (int) (min + (max-min) * Math.random()); | This function is used to return a pseudorandom double type number greater than or equal to 0.0 and less than 1.0. The default random numbers are always generated between 0 and 1. | // Statement to generate a random number from 1 to 6.<br>int r = 1 + (int) (Math.random()*6);         |

## 2.8 Java Expressions

An expression is a combination of operands and operators arranged in a logical manner. The operators of an expression indicate which operations are to be performed on the operands. The order of evaluation of operators in an expression is determined by the precedence and associativity of the operators.

To convert a mathematical expression to a Java expression, you need to follow the correct syntax based on the precedence of operators and their respective data types.

### Example

For the mathematical statement:  $s = ut + \frac{1}{2}at^2$ , the equivalent Java statement would be:  

```
double s = u*t + (1.0/2) * a* Math.pow(t,2);
```

Consider some more examples to understand how to write expressions in Java:

| Mathematical Expression            | Equivalent Java Expression                                   |
|------------------------------------|--------------------------------------------------------------|
| $\frac{(0.05 - 2yx^3)}{(x-y)}$     | double r = (0.05 - 2*y*Math.pow(x,3))/(x-y);                 |
| $\frac{-b + \sqrt{b^2 - 4ac}}{2a}$ | double r = (-b + Math.sqrt(Math.pow(b,2) - 4*a*c))/(2*a);    |
| $\frac{3}{8} \sqrt{b^4 - c^3}$     | double d = 3.0/8 * Math.sqrt(Math.pow(b,4) - Math.pow(c,3)); |



## Section Recap

- ◆ Inputs are the signals or data received by the system and outputs are the signals or data sent from it.
- ◆ When a parameter is passed to the method in the function call, it is called an argument.
- ◆ The Scanner class is one of the predefined classes that is used to read the data dynamically from the keyboard, i.e., at run time.
- ◆ A token is a series of characters that ends with a delimiter.
- ◆ A delimiter is a character used to separate data items. It can be a whitespace, a tab character, a carriage return, or the end of file (EOF) character.
- ◆ Runtime errors, logical errors, and syntax errors are three types of errors that commonly occur during the execution of a program.
- ◆ A comment is an explanation of the source code of the program or the part of the program that is added to make the source code easy to understand.
- ◆ The comments are of two types-single-line comments and multiline comments.
- ◆ The package contains multiple classes, each providing a special functionality that can be used in the program as and when required.
- ◆ The Math class present in the java.lang package contains methods for performing basic numeric operations, such as calculating the power, logarithm, square root, cube root, and trigonometric functions.
- ◆ An expression is a combination of operands and operators arranged in a logical manner.

### Key Terms

**Input:** It refers to providing some data to the computer.

**Initialisation:** Assigning some data for the first time to a variable is referred to as initialisation.

**Import:** It is a keyword used to call a package.

**Scanner:** It is a class present in the java.util package.

**Package:** It is a collection of classes that are grouped.

**java.lang:** All the basic functions of Java are stored in this package. It is loaded by default.

**Comments:** These can be included in a program to provide an explanation of the code.

**Package:** A package is a group of Java classes that are packed together based on their functionality.



### SECTION 1 (Solved Questions)

#### A. Fill in the blanks.

1. The \_\_\_\_\_ package is required to import the Java Scanner class.
2. The \_\_\_\_\_ keyword is used to include a package in a program.

3. \_\_\_\_\_ errors occur due to the incorrect use of syntax in a program.
4. Division of a number by zero will result into a \_\_\_\_\_ error.
5. A \_\_\_\_\_ is a character used to separate data items.
6. The \_\_\_\_\_ function is used to scan the next token of the input as a sentence.
7. An object of a Scanner class can be created by using the \_\_\_\_\_ keyword.
8. The return data type of Math.round() is \_\_\_\_\_.
9. The \_\_\_\_\_ method is used to find the square root.
10. The output of Math.floor(29.99) is \_\_\_\_\_.

|             |                 |           |           |                |              |
|-------------|-----------------|-----------|-----------|----------------|--------------|
| <b>Ans.</b> | 1. java.util    | 2. import | 3. Syntax | 4. Runtime     | 5. Delimiter |
|             | 6. nextLine ( ) | 7. new    | 8. int    | 9. Math.sqrt() | 10. 29.0     |

### B. Very short answer type questions.

1. Name the package that contains the Scanner class.

**Ans.** java.util

2. What are semantic errors?

**Ans.** Semantic errors are caused by an incorrect idea or concept used by a programmer while coding.

3. List any two functions of the Mathematical library of Java.

**Ans.** The two functions of the Mathematical library of Java are: Math.min( ) and Math.abs( )

4. What is an expression in Java?

**Ans.** An expression is a combination of operands and operators arranged in a logical manner. The operators of an expression indicate which operations are to be performed on the operands.

5. Given a package named 'oops.java'. How would you import a class named 'robust' in the package? Write a single-line statement.

**Ans.** import oops.java.robust;

6. Write down the syntax for the following functions:

- i. To find the square root of a number n
- ii. To find the rounded value of a number x

**Ans.** i. Math.sqrt(n);                      ii. Math.round(x);

7. What will be the return type of Math.ceil() function?

**Ans.** double

8. Write the equivalent Java expression for the following Mathematical expression:

$$(3 - \sqrt{x})^2 + y - 6\sqrt{5}$$

**Ans.** double d = Math.pow(3-Math.sqrt(x),2) + y + 6\*Math.sqrt(5);

### C. Short answer type questions.

1. What is the Scanner class?

**Ans.** The Scanner class is one of the predefined classes in Java that is used to read data dynamically from the keyboard, i.e., at run time. To use the Java Scanner class, you need to have a predefined package java.util.

2. What is the difference between next() and nextLine() methods?

**Ans.** The next() method scans the next token of the input as a single word whereas the nextLine() method scans the next token of input as a sentence.

3. List the different types of errors that can occur in a Java program.

Ans. The following are the different types of errors that can occur in a Java program:

- ◆ Syntax errors
- ◆ Runtime errors
- ◆ Logical errors

4. What are comments in Java?

Ans. A comment is an explanation of the source code of the program or the part of the program that is added to make the source code easy to understand.

5. Explain any two built-in Java packages briefly.

Ans. **java.lang:** It contains the classes required for basic programming, primitive data types, String and Math functions, etc.

**java.util:** It contains the basic utility classes required by the programmer, such as Scanner and Printer classes.

6. Mention any two advantages of providing input in a program using the Scanner class.

Ans. The following are the advantages of using the Scanner class to read input in Java:

- ◆ It accepts data in token form, so there is no need to mention the type of data.
- ◆ String manipulation is quite easy as the Scanner class provides various in-built functions.

7. Differentiate between the Math.ceil( ) and Math.floor( ) functions.

Ans. The difference between the Math.ceil( ) and Math.floor( ) functions is as follows:

| Math.ceil( )                                                                                                                        | Math.floor( )                                                                                                                   |
|-------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| This method returns the smallest double value that is greater than or equal to the argument and is equal to a mathematical integer. | This method returns the largest double value that is less than or equal to the argument and is equal to a mathematical integer. |
| <b>Example:</b><br>double a = Math.ceil(12.4);<br>Output: 12.0                                                                      | <b>Example:</b><br>double b = Math.floor(235.24);<br>Output: 236.0                                                              |

8. Explain the purpose of the Math.random( ) function. Give syntax to use this function with a range.

Ans. The random() function is used to return a pseudorandom double type number greater than or equal to 0.0 and less than 1.0. The default random numbers are always generated between 0 and 1. You can use the random() method to generate a random number between a given minimum and maximum value.

**Syntax:**

```
r = (int) (min + (max-min) * Math.random());
```

#### D. Application-based questions.

1. Write a program to take the name and age of a person as input and display them.

Ans.

```
import java.util.Scanner;
public class cl_Scan
{
 public static void main(String[] args)
 {
 Scanner s = new Scanner(System.in);
 System.out.println("Enter name, age, and salary");
 String name = s.nextLine();
```



```

 int age = s.nextInt();
 double salary = s.nextDouble();
 System.out.println("Name: " + name);
 System.out.println("Age: " + age);
 System.out.println("Salary: " + salary);
 }
}

```

2. Write a program to take the name and marks of a student in five subjects. Calculate the average marks and display them.

Ans. **import** java.util.Scanner;

```

public class cl_average
{
 public static void main(String[] args)
 {
 Scanner s = new Scanner(System.in);
 System.out.println("Enter name: ");
 String name = s.nextLine();
 System.out.println("Enter marks in English: ");
 int eng = s.nextInt();
 System.out.println("Enter marks in Math: ");
 int maths = s.nextInt();
 System.out.println("Enter marks in Science: ");
 int sci = s.nextInt();
 System.out.println("Enter marks in Social Studies: ");
 int sst = s.nextInt();
 System.out.println("Enter marks in Hindi: ");
 int hindi = s.nextInt();
 double avg=(eng + maths + sci + sst + hindi)/5.0;
 System.out.println("Hello " + name);
 System.out.println("The average marks are: " + avg);
 }
}

```

3. Use the Mathematical functions to calculate the hypotenuse of a right-angled triangle.

Ans. **class** Hypotenuse

```

{
 public static void main(String[] args)
 {
 double p=4, b=3, h;
 h=(Math.sqrt((Math.pow(p,2))+(Math.pow(b,2))));
 System.out.println("The length of the hypotenuse is: " + h);
 }
}

```

**A. Very short answer type questions.**

- What will be the output of the following code?  

```
float x = 7.87f;
System.out.println(Math.ceil(x));
System.out.println(Math.floor(x));
```
- Name the next methods of the Java Scanner class that are used to read or scan the input as:
  - An integer
  - A String
- Write the output of the following statements:
  - `System.out.println(-2%3);`
  - `System.out.println(Math.round(2.4));`
  - `System.out.println(Math.sqrt(225));`
  - `System.out.println(Math.pow(8,3));`

**B. Short answer type questions.**

- What are tokens and delimiters?
- Differentiate between actual and formal parameters.
- Name the following:
  - A package that is invoked by default
  - A keyword to use the classes defined in a package
- Correct the following statements:
  - `Scanner ss = new Scanner();`
  - `int a = intNext.ss();`
  - `String nm = ss.nextString();`
  - `Float p;`
- Identify the types of errors (syntax, runtime, or logical) in the following conditions:
  - Division by zero
  - Square root of a negative number
  - Using `(-)` in an arithmetic operation instead of `(+)`
  - Missing semicolon `(;)`
  - Using a variable that has not been previously defined
- Differentiate between syntax and runtime errors?
- Does an error-free program after compilation ensure that there would not be any runtime error? Justify your statement with the help of an example.

**C. Application-based questions.**

- Write a program to input the name of a student and his marks in three subjects. Print the total marks obtained along with his percentage.

- Write a program to input the principal, rate, and time using the Scanner class. Calculate the amount using simple interest.
- Write a program to input the length and breadth of a rectangle using the Scanner class. Calculate its area and perimeter.
- Write a program to input the distance travelled by a car in kilometres and the time taken in hours. Find the speed of the car. Now convert the speed into m/s.
- Write a program to input a two-digit number using the Scanner class. Find the number obtained by reversing its digits.

### SECTION 3 (Board Questions)

1. What is the use of the keyword import? [2010]

Ans. The import keyword is used to call a package or a class in a Java program.

2. Complete the code given below to create an object of the Scanner class. [2012]

Scanner sc=\_\_\_\_\_ Scanner(\_\_\_\_\_)

Ans. Scanner sc=new Scanner (System.in);

3. Give output of the following program segment: [2012]

```
double x = 2.9, y = 2.5;
System.out.println(Math.min(Math.floor(x), y));
System.out.println(Math.max(Math.ceil(x), y));
```

Ans. 2.0

3.0

4. Name the type of error (syntax, runtime, or logical error) in each case given below: [2012]

- Division by a variable that contains zero
- Multiplication operator used when the operation should be division
- Missing semicolon

Ans. a. Runtime error

b. Logical error

c. Syntax error

5. Name the method of the Scanner class that is used to: [2014]

- Input an integer data from the standard input device
- Input a string data from the standard input device

Ans. a. nextInt( )

b. next( ) or nextLine( )

6. What is meant by a package? Name any two Java Application Programming Interface packages. [2014]

Ans. A package is a collection of Java classes that are grouped on the basis of their functionality. Two Java application programming interface packages are: java.io and java.lang

7. What are the final values stored in variable 'x' and 'y' below? [2014]

```
double a = -6.35;
double b = 14.74;
double x = Math.abs(Math.ceil(a));
double y = Math.round(Math.max(a,b));
```

Ans. The final value stored in variable x is 6.0 and y is 15.0.

8. Which of the following are valid comments?

[2015]

- |                             |                             |
|-----------------------------|-----------------------------|
| a. <code>/*comment*/</code> | b. <code>/*comment</code>   |
| c. <code>//comment</code>   | d. <code>*/comment*/</code> |
- Ans. a. Valid comment  
b. Invalid comment  
c. Valid comment  
d. Invalid comment

9. Name the package which contains the Scanner class.

[2015]

Ans. `java.util`

10. Name the type of error (syntax, runtime, or logical error) in each case given below:

[2016]

- |                                   |                            |
|-----------------------------------|----------------------------|
| a. <code>Math.sqrt(36-45);</code> | b. <code>int a;b;c;</code> |
|-----------------------------------|----------------------------|

Ans. a. Runtime error  
b. Syntax error

## Section C: Conditional Constructs in Java

In your daily life, you come across many situations when you must choose from a variety of options. For example, you cross the road while driving if the traffic signal is green. Otherwise, you stop there. Similarly, while writing a program, sometimes you need to perform some set of actions based on some condition, so you need to include control statements in the program.

A programming language uses control statements to control the flow of execution of a program based on certain conditions. There are two types of control statements: **decision-making** statements and **looping** statements. Let us first concentrate on decision-making statements.

### 2.9 Decision Making Statements

Decision-making statements are also known as **selection statements**. A decision-making statement is included in a program when you want to execute a specific block of code based on the result of a certain condition. Depending on the result of the conditional block, it is decided whether the statement will be executed or not.

The various decision-making statements in Java are as listed below:

- ♦ `if`
- ♦ `if-else`
- ♦ `if-else-if`
- ♦ `Switch-case`

Let us discuss these statements one-by-one.

### 2.10 if Statement

The **if** statement is the simplest decision-making statement. It executes a statement or block of statements associated with it if a test expression (condition) evaluates to true. Otherwise, the flow of control comes out of the 'if' block and executes the next statement.

**Syntax:**

```
if(condition)
statement;
```

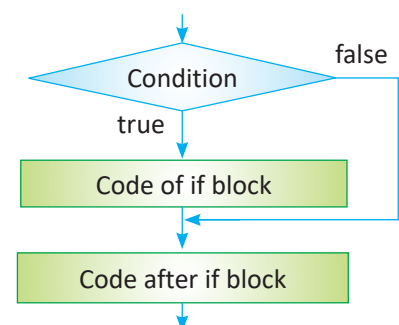


Figure 2.7: Flowchart of if Statement

### Example:

```
int a;
a=input.nextInt();
if(a>=75)
System.out.println("Distinction");
```

In the above example, suppose the value of variable 'a' is 20, which is less than 75, so the test expression evaluates to false and nothing will be printed because there is no statement after the 'if' block.

## 2.11 if-else Statement

The **if-else** statement is an extended version of the if statement. If the test expression evaluates to true, the statement or a block of statements associated with the 'if' block gets executed. If the test expression evaluates to false, the flow of control transfers to the 'else' block, and statements associated with it get executed.

### Syntax:

```
if(condition)
statement 1;
else
statement 2;
```

### Example:

```
int m;
m=input.nextInt();
if(m>18)
System.out.println("Eligible");
else
System.out.println("Not eligible");
```

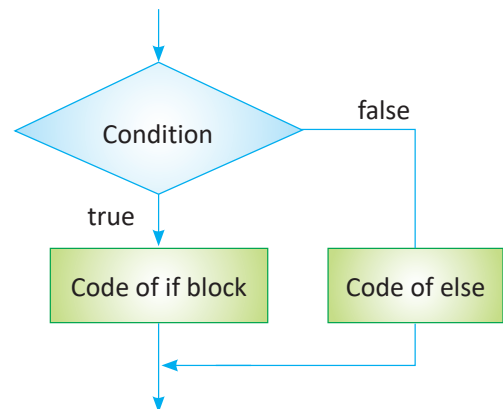


Figure 2.8: Flowchart of if-else Statement

### Program 4

Write a program to input two numbers and print the smaller of the two numbers.

```
1. import java.util.*;
2. public class Small_Number
3. {
4. public static void main(String args[])
5. {
6. Scanner input=new Scanner(System.in);
7. int x,y;
8. System.out.println("Enter two numbers:");
9. x=input.nextInt();
10. y=input.nextInt();
11. if(x<y)
```

#### OUTPUT

```
Enter two numbers:
20
30
Smaller number=20
```

```

12. System.out.println("Smaller number="+x);
13. else
14. System.out.println("Smaller number="+y);
15. }
16. }

```

## 2.12 if-else-if Statement

You can use the **if-else-if** statement in your program if you have multiple conditions to decide an action to be performed. This type of structure is known as the **else-if ladder**. The test expressions are evaluated from top to bottom. As soon as the test expression evaluates to true, the statement or a block of statements associated with it gets executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the 'else' block gets executed.

### Syntax:

```

if(condition 1)
statement 1;
else if (condition 2)
statement 2;
else if (condition 3)
statement 3;
.
.
else
statement;

```

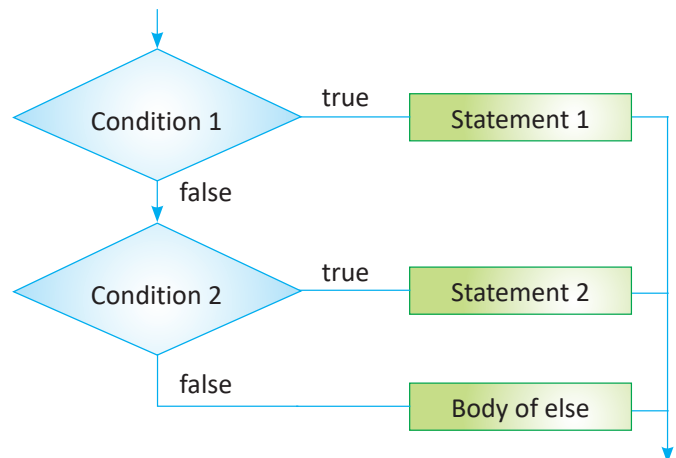


**Figure 2.9:** Flowchart of if-else-if Statement

### Program 5

Write a program to display the name of the day according to the number entered by the user.

```

1 import java.util.*;
2 public class Week_Day
3 {
4 public static void main(String args[])
5 {
6 Scanner input=new Scanner(System.in);
7 int m;
8 System.out.println("Enter a number from 1 to 7:");
9 m=input.nextInt();
10 if(m==1)
11 System.out.println("Monday");
12 else if(m==2)
13 System.out.println("Tuesday");
14 else if(m==3)
15 System.out.println("Wednesday");

```

### OUTPUT

```

Enter a number from 1 to 7:
6
Saturday

```

```

16 else if(m==4)
17 System.out.println("Thursday");
18 else if(m==5)
19 System.out.println("Friday");
20 else if(m==6)
21 System.out.println("Saturday");
22 else if(m==7)
23 System.out.println("Sunday");
24 else
25 System.out.println("Wrong input");
26 }
27 }

```

## 2.13 Nested if Statement

A nested **if** is a statement that has another **if** or **else** in its body. Here, the **inner if** block condition executes only when **outer if** block condition is true.

### Syntax:

```

if(Condition 1)
{
 if(condition 2)
 Statement 1;
 else
 Statement 2;
}
else
{
 if(condition 3)
 Statement 3;
 else
 Statement 4;
}

```

### Example

```

//Use of nested if
public class Test
{
 public static void main(String args[])
 {
 int x = 50;
 int y = 70;
 if(x == 50)
 {
 if(y == 70)
 {
 System.out.print("X=50 and Y=70");
 }
 }
 }
}

```

## Dangling else Problem

The nested if-else statement introduces a source of potential ambiguity referred to as the 'dangling-else' problem. This problem arises when in a **nested if** structure, the number of 'if' is more than the number of 'else' clauses.

### Syntax:

```

if(condition 1)
 if(condition 2)
 statement 1;
else
 statement 2;

```

The matching if statement of this else cannot be identified.

### Example

```

if(ch>='A')
 if(ch<='Z')
 ++upcase;
else
 ++other;

```

## 2.14 switch-case Statement

The **switch-case** statement is mostly used for creating menu-driven programs where the user has to select an option from given multiple statements. It is a multiway branch statement that provides an easy way to execute the different parts of the code based on the value of the expression. The expression used in switch-case must be an integer or a character. Also, in the case of characters, only a single character is allowed.

### Syntax:

```
switch(ch)
{
case 1:
statement;
break;
case 2:
statement;
break;
.
.
.
default:
statement;
}
```

### Let's Know More

Note a few important points while using the switch-case statements:

- ◆ An expression for the switch can be of any type: byte, short, int, char. In JDK7, the expression can also be of type String.
- ◆ Duplicate case values are not allowed.
- ◆ The default statement is optional.
- ◆ The break statement is used inside the switch to terminate a statement sequence.
- ◆ The break statement is optional. If omitted, the execution of the next case will continue.

The **break** statement takes the control out of the switch case. If the condition of the switch statement does not match with any of the cases, the **default** statement gets executed.

### Fall through Situation

The term 'fall through' refers to the way the switch statement executes its various case sections. Every statement that follows the selected case section will be executed unless a break statement is encountered. Consider the following program to understand the concept of fall through situation.

#### Program 6

Write a program to input a letter and check whether it is an uppercase vowel, lowercase vowel, or a consonant.

```
1. import java.util.*;
2. public class check_letter
3. {
4. public static void main(String args[])
5. {
6. Scanner input=new Scanner(System.in);
7. char ch;
8. System.out.println("Enter a character:");
9. ch=input.next().charAt(0);
10. switch(ch)
11. {
```

#### OUTPUT

```
Enter a character:
A
Uppercase vowel
```



```

12. case 'A': case 'E': case 'I': case 'O': case 'U':
13. System.out.println("Uppercase vowel");
14. break;
15. case 'a': case 'e': case 'i': case 'o': case 'u':
16. System.out.println("Lowercase vowel");
17. break;
18. default:
19. System.out.println("Consonant");
20. }
21. }
22. }

```

If you remove the 'break' from all the case blocks in the above-given program, a fall through situation occurs and you get the output as shown below on entering the case expression:

Enter a character: \_\_\_\_\_

A

Uppercase vowel

Lowercase vowel

Consonant \_\_\_\_\_

Output in a fall through situation

## 2.15 Difference between if-else and switch-case

The following table summarises the differences between 'if-else' and 'switch-case' statements:

| if-else                                                                                               | switch-case                                                                                                                    |
|-------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| The if statement is used to select between two alternatives.                                          | The switch statement is used to select among multiple alternatives.                                                            |
| The values used in the if statement are based on constraints.                                         | A switch can have values based on user choice.                                                                                 |
| Any data type, like int, float, long, double, or char can be used for comparison in the if statement. | The expression used in switch must be an integer or a character. Also, in case of character, only single character is allowed. |
| It can be used to check for a range of values.                                                        | It can be used only for a single value, not for a range of values.                                                             |

## Section D: Iterative Constructs in Java

By now you have revised decision-making statements. Now, let us recall about the iterative constructs. An iterative construct is required when you need to perform a task repeatedly until a certain condition is met in a program.

## 2.16 Looping Statements

Sometimes, it is required to repeatedly execute a set of instructions, which is possible through iterative statements. Iterative statements are also known as **looping statements** or **repetition statements**. These statements enable you to control the flow of execution by repetitively performing a set of statements as long as the condition remains true.

To carry out iterations, Java provides three looping statements: **for**, **while**, and **do-while**. The basic functionality of all these loops is similar. They differ only in the syntax and the way they check condition (before or after iteration).

## Elements of a Loop

Every loop consists of the following main elements:

### Initialisation

This part of the loop initialises the **loop control variable** or **counter** before beginning the loop. The loop control variable keeps a record of the number of times the loop is executed. The initialisation expression gets executed only once at the beginning of the loop.

**Example:**

```
int count = 1;
```

### Test Expression

This part of loop holds a conditional expression that is evaluated before entering the loop. The result of the test expression is either true or false. This result determines whether the loop will be executed or not. If the test condition evaluates to true, the body of the loop gets executed.

**Example:**

```
[test condition];
```

### Increment/Decrement

This part of the loop helps in progressing the loop. It can be an increment or decrement statement. The increment or decrement (update value of the loop control variable) takes place unless the test condition becomes false.

**Example:**

```
a++
b--
```

### Body of the Loop

Once the test condition is evaluated to true, the statements given in the body of the loop are executed.

### Terminating the Loop

When the test condition eventually becomes false, the loop terminates. A loop that never terminates is called an **infinite** loop.



## Quick Quiz

### 1. What is a block?

A block is a set of instructions enclosed in braces or curly brackets.

### 2. What is the use of the break statement?

The break statement is used in the switch construct to terminate the sequential execution of statements.

## 2.17 Classification of Loops

On the basis of the evaluation of the test condition, loops can be classified into two types:

### Entry-controlled Loops

A loop in which the test condition is checked before entering the body of the loop is called an entry-controlled loop. In Java, **for** loop and **while** loop are the entry-controlled loops.

## Exit-controlled Loops

A loop in which the test condition is evaluated after executing the body of the loop is called an exit-controlled loop. In this type of loop, the code must always be executed first, and then the test condition is checked. In Java, **do-while** loop is the exit-controlled loop.

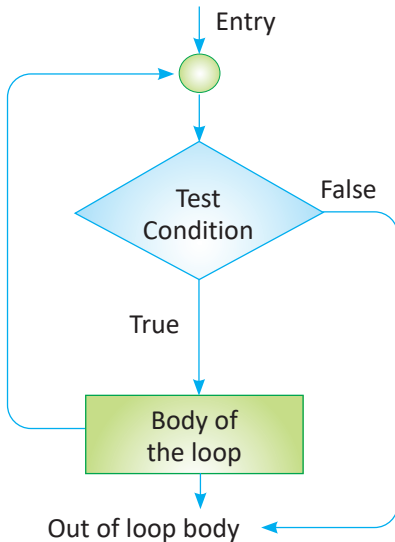


Figure 2.10: Entry-controlled Loop

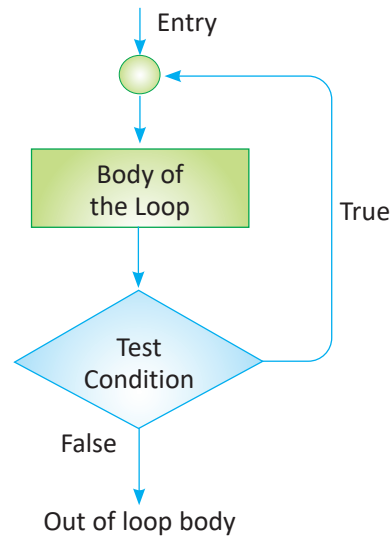


Figure 2.11: Exit-controlled Loop

## 2.18 for Loop

The **for** loop is the simplest loop in Java. It is an entry-controlled loop and performs a fixed number of iterations. The number of iterations depends on the test condition given inside the for loop.

### Syntax:

```
for (initialisation; test condition; increment/decrement)
{
 statements;
}
```

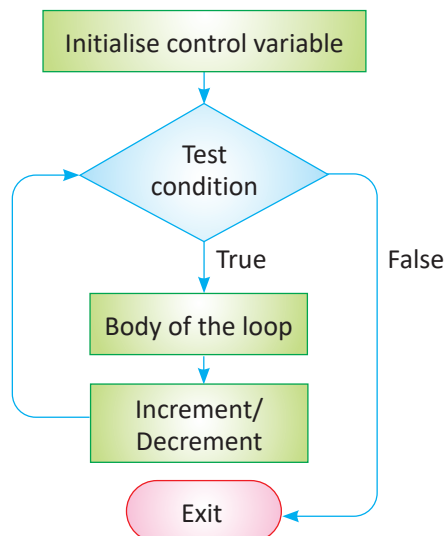


Figure 2.12: Flowchart of for Loop

## Example

```
class forLoop
{
 public static void main(String args[])
 {
 for (int i = 1; i <= 5; i++)
 System.out.print(i + " ");
 }
}
```

The following is the step-by-step explanation of the above-given example:

- First, the variable 'i' is initialised to 1.
- In the next step, the test condition, i.e.,  $i \leq 5$  is checked.
- As the value of 'i' is less than 5, the loop will execute and the value of 'i' will be printed.
- Next, the value of 'i' is incremented by 1 and it becomes 2. Again, the test condition will become true and the loop continues.
- The loop continues till the value of 'i' becomes greater than 5.
- As soon as the value of 'i' becomes 6, the test condition becomes false, and the control comes out of the loop.
- The loop terminates after the test condition becomes false and you will get the output as: 1 2 3 4 5.

### Let's Know More

A compound statement is a set of two or more statements that are enclosed between curly brackets. It is also called a block. A compound statement can be observed in an if construct or within a loop.

#### Example:

```
int i, a;
for(i=1; i<=5; i++)
{
 s=s+i;
 a=a+s;
}
```

## Program 7

Write a program to print the sum of all natural numbers up to 20.

```
1 public class for_example
2 {
3 public static void main(String args[])
4 {
5 int i,sum=0;
6 for(i=1;i<=20;i=i+1)
7 {
8 s=s+i;
9 }
10 System.out.println(s);
11 }
12 }
```

### OUTPUT

220

## Program 8

Write a program to print the first 10 multiples of 5.

```
1 public class for_example
2 {
3 public static void main(String args[])
4 {
5 int i;
6 for(i=5;i<=50;i=i+5)
7 {
8 System.out.print(i + " ");
9 }
10 }
11 }
```

### OUTPUT

5 10 15 20 25 30 35 40 45 50

## 2.19 while Loop

The **while** loop is another entry-controlled loop in which the test condition is checked at the beginning. In general, a while loop can be used in a program where the number of iterations is not fixed.

### Syntax:

```
initialisation
while (condition)
{
 statement;
 step value;
}
```

### Example

```
// A program to print the odd numbers from
1 to 10.
class cl_odd
{
 public static void main(String args[])
 {
 int i=1;
 while(i<=10)
 {
 System.out.print(i + " ");
 i=i+2;
 }
 }
}
```

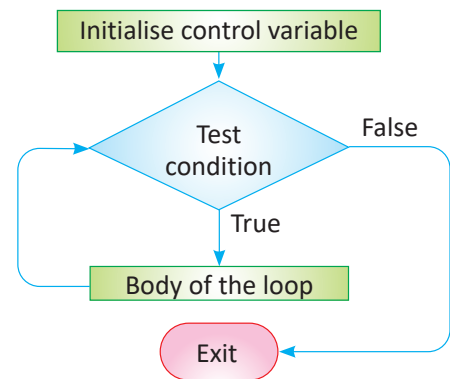


Figure 2.13: Flowchart of while Loop



### Quick Quiz

1. Which constructs are used for unknown number of iterations?  
while loop and do while loop
2. Give examples of entry-controlled loops.  
for loop and while loop

The following is the step-by-step explanation of the above-given example:

- First, the variable 'i' is initialised to 1. Then, the test condition is checked, i.e.,  $i \leq 10$ .
- Here, the test condition evaluates to true, so the loop body is executed and the value of 'i', i.e., 1 is printed.
- Then, the value of 'i' is incremented by 2. Again, the test condition is checked and the same process continues till the test condition becomes false.
- As soon as the value of 'i' becomes 11, the test condition becomes false and the loop terminates. So, on executing the loop, you will get the output as: 1 3 5 7 9.

## 2.20 do-while Loop

The **do-while** loop is used in a program when the number of iterations is not fixed. In this loop, the control enters the loop without checking the defined condition, executes the body of loop, and after that checks the condition for the further continuation of the tasks. Thus, this type of loop executes the statements given inside the loop body **at least once**. If the test condition is not satisfied, then control exits from the loop and the loop terminates. That is why the do-while loop is also known as the exit-controlled loop.

### Syntax:

```
initialisation
do
{
 statement;
 step value;
}
while (condition);
```

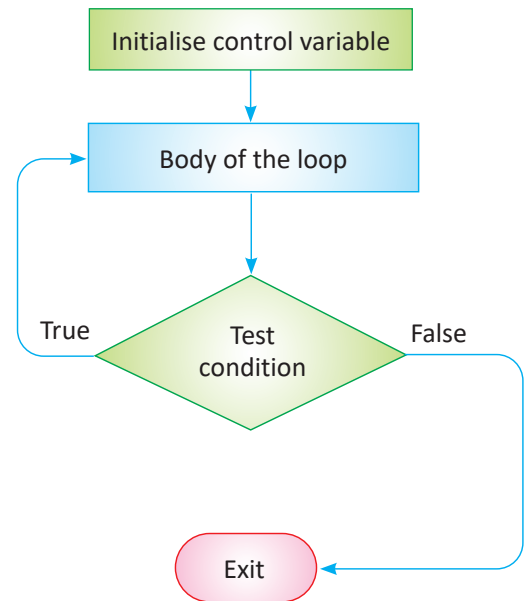


Figure 2.14: Flowchart of do-while Loop

### Example

```
// A program to print the even numbers from 1 to 10 using do-while loop.
class cl_natural
{
 public static void main(String args[])
 {
 int i=0;
 do
 {
 System.out.print(i + " ");
 i=i+2;
 }
 while(i<=10);
 }
}
```

The following is the step-by-step explanation of the above-given example:

- ✦ First, the loop control variable *i* is initialised to 0. Then, the control enters the loop body.
- ✦ The statements inside the loop body are executed. So, the value of '*i*' is printed, which is 0. After that test expression of 'while' is checked.
- ✦ The test condition given inside while is checked.
- ✦ The test condition evaluates to true as  $0 < 10$ , so the control again enters the loop body and prints the next value of '*i*', which is now  $i+2$ , i.e., 2.
- ✦ This process continues till the test condition becomes false. On executing the above loop, you will get the output as: 0 2 4 6 8 10.

## 2.21 Special Loops

Apart from the three main loops, Java also supports two types of special loops, i.e., **infinite loop** and **null loop**.

### Infinite Loop

An infinite loop is a sequence of instructions in a computer program that repeats endlessly. This can happen due to any of the following reasons:

- ✦ Loop has no terminating condition.
- ✦ Loop has a condition that can never be met.
- ✦ Loop has a condition that causes the loop to start over.

An infinite loop is useful for those applications that accept user input and generate output continuously until the user exits the application manually.

### Null Loop

A **null** loop is a loop that does not contain any statements within itself. It can be used as a **delay counter** in which the loop executes for a predefined number of times. It does not have a loop body.

#### Example

```
int m=1;
while(m>0)
{
 System.out.println(m);
 m++;
}
```

The loop in the above example will continue forever because there is no terminating condition.

#### Example

```
for (x=0; x<500; x++)
{
}
Or
for (x=0; x<500; x++);
```

## 2.22 Jump Statements

Jump statements are used in programming languages to control the flow of a program's execution under a particular condition. These are used to force the execution flow to advance and branch based on the changes in a program's state. Two commonly used jump statements are **break** and **continue**.

### break Statement

The **break** statement takes the program control out of a loop. It is mainly used in situations where you want the program to exit out of the loop when a particular situation arises.

### Example

```
//A program to show the use of break statement in a loop
public class cl_break
{
 public static void main(String args[])
 {
 int i=1;
 while(i<=10)
 {
 System.out.print(i+" ");
 i++;
 if(i==5)
 break;
 }
 }
}
```

In the above given example, when the expression `i == 5` evaluates to true, the `break` statement will be encountered. The `break` statement breaks the flow of execution and the control comes out of the loop. So, you will get the output as: 1, 2, 3, 4.

### continue Statement

Sometimes, it is useful to force an early iteration of a loop. That is, you might want to continue running the loop but stop processing the further statements for a particular iteration. The statements will not be executed for that particular instance, but for the other iterations, the statements will be executed. Hence, the `continue` statement skips the rest of the current iteration of the loop, jumps back to the beginning of the loop, and continues with the next iteration.



### Quick Quiz

**1. Name two jump statements used in Java.**

The two jump statements used in Java are `break` and `continue`.

**2. Which statement is used to terminate the loop?**

The `break` statement is used to terminate the loop.

### NOTE

The `continue` statement does not terminate the loop, it just skips some part of the loop.

### Example

```
//A program to show the use of continue statement to print odd numbers
public class odd_num
{
 public static void main(String args[])
 {
 for (int i=0; i<10; i++)
```



```

 {
 if(i%2 ==0)
 continue;
 System.out.print(i + " ");
 }
 }
}

```

In the above example, the continue statement skips the iteration of the loop when the test expression  $i\%2==0$  evaluates to true. Hence, only the odd numbers are printed as: 1 3 5 7 9.

## 2.23 Interconversion of Loops

All the three types of loops, for loop, while loop, and do-while loop, can be changed into one another with just a few changes to the program's statements. You can write a program using while or do-while loop that you have written using for loop or vice-versa. You can select any loop while writing a program as per your convenience. The interconversion of loops can be done in the following ways:

### Converting a for Loop to a while Loop

The 'for' loop is used when the number of iterations is fixed. On the other hand, the 'while' loop is used when the number of iterations is not fixed. However, you can convert a 'for' loop to a 'while' loop by following the given steps:

- ✦ Initialisation of the loop control variable is done before the loop starts.
- ✦ The test condition is given with the while keyword within the parenthesis.
- ✦ The statements of the loop body are given within the curly brackets.
- ✦ At last, the increment/decrement statement for the loop control variable is given.

### Example

|                                                                                                                                                                                                                            |                                                                                                                                                                                                                                                        |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> //Using a for loop to print a series class for_demo {     public static void main(String args[])     {         for (int i=1; i&lt;=5; i++)         {             System.out.print(i + " ");         }     } } </pre> | <pre> //Converting a for loop into a while loop class while_demo {     public static void main(String args[])     {         int i=1;         while(i&lt;=5)         {             System.out.print(i + " ");             i++;         }     } } </pre> |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Observe the above-given 'for' and 'while' loops. In the 'for' loop, initialisation, testing condition, and increment are given in a single statement of for loop. On the other hand, in the 'while' loop,

initialisation is done before the loop starts, then the test condition is written within the parenthesis of while, and the increment value is given in the loop's body. Although the syntax of both the loops is different, but a 'for' loop can easily be converted into a 'while' loop as shown above. On executing these loops, you will get the same output, which is 1 2 3 4 5.

## Converting a while Loop to a do-while Loop

You can also convert a 'while' loop to a 'do-while' loop by following the given steps:

- ✦ Initialisation of the loop control variable is done before the loop starts.
- ✦ Then, the statements of the loop body are written after the do keyword within the curly brackets.
- ✦ The increment/decrement statements are given in the loop body only.
- ✦ After the closing bracket of do, the test condition is given with the while keyword within the parentheses.

### Example

```
//Using a while loop to print a series
class while_demo
{
 public static void main (String args[])
 {
 int m=1;
 while(m<=5)
 {
 System.out.print(m);
 m++;
 }
 }
}
```

```
//Converting a while loop into a do-while loop
class dowhile_demo
{
 public static void main(String args[])
 {
 int m=1;
 do
 {
 System.out.print(m);
 m++;
 } while(m<=5);
 }
}
```

Observe the syntax of both the loops is different, but a 'while' loop can easily be converted into a 'do-while' loop as shown above. On executing these loops, you will get the same output, which is 1 2 3 4 5.

## 2.24 Comparison of Various Types of Loops

The following table summarises the differences between for, while, and do-while loops:

| for Loop                                                                                     | while Loop                                                                                                                      | do-while Loop                                                                                                                                                       |
|----------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| The for loop is a control flow statement that iterates a part of the program multiple times. | The while loop is a control flow statement that executes a part of the program repeatedly based on the given boolean condition. | The do-while loop is a control flow statement that executes a part of the program at least once and the further execution depends upon the given boolean condition. |
| If the number of iterations is fixed, it is recommended to use the for loop.                 | If the number of iterations is not fixed, it is recommended to use the while loop.                                              | If the number of iterations is not fixed and you must have to execute the loop at least once, it is recommended to use the do-while loop.                           |



|                                                                                         |                                                                       |                                                                          |
|-----------------------------------------------------------------------------------------|-----------------------------------------------------------------------|--------------------------------------------------------------------------|
| <b>Syntax:</b><br>for(init;condition;incr/<br>decr)<br>{<br>// code to be executed<br>} | <b>Syntax:</b><br>while(condition)<br>{<br>//code to be executed<br>} | <b>Syntax:</b><br>do<br>{<br>//code to be executed<br>}while(condition); |
|-----------------------------------------------------------------------------------------|-----------------------------------------------------------------------|--------------------------------------------------------------------------|

## 2.25 Nested Loops

A **nested** loop means a loop within a loop. In some cases, it is required to construct such loops. It can be applied when the internal loop repeats a given number of times for each repetition of the external loop. Nested loops may be constructed using 'while', 'do-while', or 'for' loops. Let us now see the working of nested 'for' loop.

### Syntax:

```
for(<initialisation>;<test condition>;<increment/decrement>) //Starting of loop 1
{
 for(<initialisation>;<test condition>;<increment/decrement>) //Starting of loop 2
 {
 Statements block
 } //End of loop 2
} //End of loop 1
```

Observe that the **inner loop** closes prior to the **outer loop**. You can also have multiple nested loops in the same way.

### Program 9

Write a program to show the use of nested for loop.

```
1. class day_week
2. {
3. public static void main(String[] args)
4. {
5. int weeks = 2;
6. int days = 7;
7. // outer loop prints weeks
8. for(int i = 1; i <= weeks; ++i)
9. {
10. System.out.println("Week: " + i);
11. // inner loop prints days
12. for(int j = 1; j <= days; ++j)
13. {
14. System.out.println(" Day: " + j);
15. }
16. }
17. }
18. }
```

### OUTPUT

|         |         |
|---------|---------|
| Week: 1 | Week: 2 |
| Day: 1  | Day: 1  |
| Day: 2  | Day: 2  |
| Day: 3  | Day: 3  |
| Day: 4  | Day: 4  |
| Day: 5  | Day: 5  |
| Day: 6  | Day: 6  |
| Day: 7  | Day: 7  |

## SECTION 1 (Solved Questions)

### A. Fill in the blanks.

- The \_\_\_\_\_ statement executes a block of statements associated with it when a test-expression evaluates to true. Otherwise, the flow of control comes out of it.
- The absence of \_\_\_\_\_ in the switch-case statement causes a fall through situation.
- When the if statement is written within another if, it is called a \_\_\_\_\_.
- The expression used in switch-case must be an \_\_\_\_\_ or a \_\_\_\_\_.
- The \_\_\_\_\_ statement is optional in switch-case.
- \_\_\_\_\_ are used to perform a task repeatedly a number of times.
- The \_\_\_\_\_ statement is used to skip the current iteration of the loop.
- The do-while loop is also called an \_\_\_\_\_ loop.
- The \_\_\_\_\_ loop first checks the condition and then executes the body of the loop.
- The \_\_\_\_\_ loop is used for known number of iterations.
- An \_\_\_\_\_ loop is an endless loop.
- In the \_\_\_\_\_ construct, a block of statements gets executed as soon as the test expression evaluates to true, and the rest of the ladder is bypassed.

|             |             |                    |              |                       |              |                |
|-------------|-------------|--------------------|--------------|-----------------------|--------------|----------------|
| <b>Ans.</b> | 1. if       | 2. break           | 3. nested if | 4. Integer, Character | 5. default   | 6. Loops       |
|             | 7. continue | 8. exit-controlled | 9. while     | 10. for               | 11. infinite | 12. if-else-if |

### B. Very short answer type questions.

1. What are the decision-making statements in Java?

**Ans.** A decision-making statement is used to take certain decisions based upon a condition and produce the needed output.

2. List the different selection statements.

**Ans.** In Java, you can use the different selection statements, such as if, if-else, if-else-if ladder, and select-case.

3. What is the use of the 'break' statement in a switch-case?

**Ans.** The break statement is used in the switch-case construct to terminate the sequential execution of statements and transfer the control outside the switch block.

4. What do you understand by an entry-controlled loop?

**Ans.** A loop in which the condition is checked before entering the loop is called an entry-controlled loop.

5. Name the two jump statements used in Java.

**Ans.** Two jump statements used in Java are break and continue.

6. What are the parameters used in 'for' loop?

**Ans.** The for loop needs an initial value, test condition, and an increment or decrement value, i.e., update expression.

7. What is an infinite loop?

**Ans.** An infinite loop is an endless loop. It repeats indefinitely as there is no condition that turns false to terminate the execution of the loop.

8. Why is a null loop known as a delay loop?

Ans. A null loop does not repeat the execution of any statement, but it keeps the control engaged until all iterations have been completed. That is why, a null loop is called a delay loop.

9. Write a 'for' loop statement that initialises a loop variable x as 2, increments it by 2 in every iteration and continues the loop till x is 20.

Ans. `for(x=2;x<=20;x=x+2)`

10. Write a while statement to repeat the loop until character variable 'ch' is equal to character 'N'.

Ans. `while (ch== 'N')`

### C. Short answer type questions.

1. What is the use of the 'else' statement in the 'if-else' construct? Write the syntax of 'if-else' statement.

Ans. The 'else' statement is used with the 'if' statement to execute a block of code when the condition is false.

**Syntax:**

```
if(condition)
{
 statement 1;
 statement n;
}
else
{
 statement 1;
 statement 2;
}
```

2. What is a compound statement?

Ans. A compound statement is a set of two or more statements that are enclosed between curly brackets. It is also called a block. A compound statement can be observed in decision-making structures and loops.

3. Differentiate between the 'for' loop and the 'while' loop.

| Ans. | for Loop                                                 | while Loop                                                     |
|------|----------------------------------------------------------|----------------------------------------------------------------|
|      | It is used for a fixed number of iterations.             | It is used for an unknown number of iterations.                |
|      | The control variable is initialised along with the loop. | The counter variable is initialised before beginning the loop. |

4. Differentiate between the 'while' loop and the 'do-while' loop.

| Ans. | while Loop                                                                            | do-while Loop                                                                                       |
|------|---------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------|
|      | It is an entry-controlled loop.                                                       | It is an exit-controlled loop.                                                                      |
|      | It checks the condition first and then the execution begins.                          | It checks the condition after executing the statement in the loop.                                  |
|      | In a while loop, the body of the loop is not executed if the test condition is false. | In a do-while loop, the body of the loop is executed at least once, even if the condition is false. |

5. Define nested loops. When are they used?

Ans. A nested loop means a loop within a loop. It can be used when the internal loop repeats a given number of times for each repetition of the external loop.

#### D. Application-based questions.

1. Find the output of the following looping constructs:

```
for(i=1;i<=5;i++)
{
 System.out.print(2*i + " ");
}
```

Ans. Output is: 2 4 6 8 10

2. Write the output of the following:

```
for(x=5;x>1;x--)
System.out.print(x);
System.out.println();
System.out.print(i);
```

Ans. Output is: 5432

1

3. Write an appropriate 'for' loop for the following:

- A loop is to be repeated 30 times.
- A loop is to be executed 100 times. But the loop will be terminated when the value of the variable  $x=75$ .

Ans. a.

```
for(i=1;i<=30;i++)
{
 Statements;
}
```

b.

```
for(x=1;x<=100;x=x+1)
{
 if(x==75)
 break;
}
```

4. Find the output of the following looping constructs:

a. 

```
for(i=1; i<=5; i++)
{
 System.out.println(2*i);
}
```

b. 

```
int i = 1;
int d = 5;
do
{
 d = d * 2;
 System.out.println(d);
 i++;
}
while (i <= 5);
```

Ans. a. Output:

2  
4  
6  
8  
10

b. Output:

10  
20  
40  
80  
160

## SECTION 2 (Unsolved Questions)

### A. Very short answer type questions.

1. Name any two iterative constructs.
2. Write one main difference between 'if' and 'switch-case'.
3. What is the purpose of the 'default' keyword in a switch-case statement?
4. Is it necessary to use 'break' statement in a switch-case statement?
5. Define 'do-while' loop.
6. Define 'if-else-if' ladder.
7. What is the significance of the control variable in a 'for' loop?
8. What is meant by an infinite loop?

### B. Short answer type questions.

1. Why is the 'continue' statement also called a 'jump' statement?
2. In what situations will the 'if-else' statement be more beneficial than a switch-case statement?
3. Give an example of an infinite loop.
4. Which loop would you prefer if you want the statement to be executed in any case?
5. Explain the function of each of the following with an example.
  - a. break
  - b. continue

### C. Application-based questions.

1. How many times will the following loop be executed?

```
int x=1000;
while(true)
{
 if(x<500)
 break;
 x=x-100;
}
```
2. Rewrite the following program code by using 'do-while' loop.

```
int a;
for(a=1;a<=50;a++)
 System.out.println("OOPS");
```
3. Rewrite the following program code by using the 'for' loop.

```
int a=10;
```

```
while(a>0)
{
 System.out.println("POP");
 a--;
}
```

4. Rewrite the following statements by using the 'while' loop.

```
for(int n=1,int s=0;n<=10;n=n+1)
s+=n;
```

5. Rewrite the following program code by using the 'do-while' loops.

```
int p,q=1;
for(p=1;p<=5;p++)
q*=p;
System.out.println(q);
```

6. Rewrite the following program code after correcting errors, if any.

```
int p;q;
for(p=1;p<=5;p++);
{
 q=p*P*P
System.out.println(q);
}
```

7. Write a Java program that asks the user to provide a single character from the alphabet. Print the message 'vowel' or 'consonant' depending on the user input. If the user input is not a letter (between a and z or A and Z), or is a string of length > 1, print an error message.
8. Write a program to find the sum of all odd numbers less than 100.
9. Write a program to input the number of students in a class. For every student, enter their height and find the average height of the students of the class.
10. Write a menu-driven program to perform the following:
- Display the factors of the number
  - Print the first 5 multiples of the number
  - Find the sum of all numbers starting from 1 till that number

### SECTION 3 (Board Questions)

1. The following is a segment of a program:

[2009]

```
x=1;y=1;
if(n>0)
{
 x=x+1;
 y=y+1;
}
```

What will be the value of 'x' and 'y', if 'n' assumes a value of:

- 1
- 0



Ans. a. when  $n=1$   
 $x=x+1=1+1=2$   
 $y=y+1=1+1=2$   
 $x=2, y=2$

b. when  $n=0$   
 $x=1, y=1$

2. Predict the output of the following code snippet, when:

[2009]

```
a. opn='b' b. opn='x' c. opn='a'
switch(opn)
{
case 'a':
System.out.println("Platform Independent");
break;
case 'b':
System.out.println("Object Oriented");
break;
case 'c':
System.out.println("Robust and Secure");
break;
default:
System.out.println("Wrong Input");
}
```

Ans. a. Object Oriented  
b. Wrong Input  
c. Platform Independent

3. Rewrite the following program segment using 'if-else-if' statement instead of ternary operator.

```
String grade=(marks>=90)?"A":(marks>=80)?"B":"C";
```

[2014]

```
Ans. if(marks>=90)
 grade="A";
 else if(marks>=80)
 grade="B";
 else
 grade="C";
```

4. Give the output of the following program segment and also mention the number of times the loop is executed. **[2010]**

```
int a,b;
for(a=6,b=4;a<=24;a=a+6)
{
 if(a%b==0)
 break;
}
System.out.println(a);
```

Ans.

| Iteration | a<=24 | a  | b | a%b       | a=a+6 |
|-----------|-------|----|---|-----------|-------|
| 1         | True  | 6  | 4 | !=0       | 12    |
| 2         | True  | 12 | 4 | =0, break |       |

5. Name the two jump statements and write their uses.

[2010]

Ans. **Break:** A break statement terminates the current loop and takes the program control out of the loop.

**Continue:** A continue statement halts the program execution for a particular iteration. It continues with the next iteration.

6. State the difference between an entry-controlled loop and an exit controlled loop.

[2011]

Ans.

| Entry Controlled Loop                                       | Exit Controlled Loop                                                       |
|-------------------------------------------------------------|----------------------------------------------------------------------------|
| Condition is checked before entering into the loop          | Condition is checked after executing the body of the loop.                 |
| If the condition is not true the statement is not executed. | Even if the condition is not true the statement is executed at least once. |

7. Write the output of the following code:

[2015]

```
char ch;
int x=97;
do
{
ch=(char)x;
System.out.println(ch+ " ");
if(x%10==0)
break;
++x;
}
while(x<=100);
```

Ans. Output is:

a  
b  
c  
d

8. Give the output of the following program segment and also mention how many times the loop will execute.

[2018]

```
int i;
for(i=5;i>10;i++)
System.out.println(i);
System.out.println(i*4);
```

Ans. The output will be 20 and the loop will not be executed at all.

9. What is the final value of 'ctr' after the iteration process given below executes?

[2013]

```
int ctr=0;
for(int i=1;i<=5;i++)
```

```
for(int j=1;j<=5;j+=2)
++ctr;
```

**Ans.** The final value of ctr will be 15.

**10. Study the given method and answer the given questions:**

**[2014]**

```
public void sampleMethod()
{
 for(int i=0;i<3;i++)
 {
 for(int j=0;j<2;j++)
 {
 int number=(int)(Math.random()*10);
 System.out.println(number);
 }
 }
}
```

- a. How many times does the loop execute?
- b. What is the range of possible values stored in the variable number?

**Ans.** a. The outer loop executes three times, and for every iteration of the outer loop, the inner loop executes twice. So, the total number of iterations is 6.

b. The range of possible values stored in the variable 'number' will be 0-9 (both inclusive).

## SECTION 4 (Case Study Based Questions)

### Case Study I

Loops are a fundamental concept in programming. In order to understand what loops are, you have to look at some real-life examples of loops. Consider your daily routine: you get up, brush your teeth, put on clothes, go to work, come home, eat, and then sleep. Next day, you do the same things in the same order and go to sleep again. This cycle keeps on repeating. This concept of repetitive actions performed time and again is called a loop.

Loops make it very easy to group all the code that is needed to be repetitively processed and throw it under scope. The loop does the remaining job. Looping in Java is defined as performing some lines of code in an ordered fashion until a condition is false. The condition is important because you do not want the loop to run forever. As soon as this condition is false, the loop stops.

1. Which of the following is not a loop statement in Java?
  - a. while
  - b. do-while
  - c. for
  - d. if
2. The \_\_\_\_\_ statement is used to quit a loop suddenly.
  - a. break
  - b. quit
  - c. leave
  - d. continue

3. A \_\_\_\_\_ statement is used to perform any task for repeated number of times.
- conditional
  - arithmetic
  - looping
  - selection
4. The \_\_\_\_\_ loop is an exit-controlled loop.
- for
  - while
  - do-while
  - break

Ans. 1. d                      2. a                      3. c                      4. c

### Case Study II

Decision-making means the action of taking decisions and choosing an action plan accordingly. Consider a real-life example, you are driving a car on a road to a concert, and you see your fuel light flashing. You immediately decide that you will stop at the next petrol pump. But if the light had not blinked at all, you would have continued driving. That is where you make a decision and plan your action.

Decision-making in Java executes a particular segment of code based on the result of a Boolean condition. It is important because conditions define the flow of programs and the output of a particular program. The decision-making principles in Java chiefly consist of if-else statements, continue, break, and switch statements. It decides the flow of the program control during the execution of the program.

- Which of the following is not a decision-making statement in Java?
  - while
  - if-else
  - nested if
  - if
- The \_\_\_\_\_ statement is used to terminate a switch case.
  - leave
  - quit
  - break
  - continue
- The if statement is a:
  - One way selection statement
  - Two way selection statement
  - Multi way selection statement
  - Logical statement
- The \_\_\_\_\_ statement is optional in switch-case.
  - switch
  - case
  - default
  - None of these

Ans. 1. a                      2. c                      3. a                      4. c

1. Write a program to input the marks obtained by a student. If the marks obtained by the student is 75 or above display “Qualified for admission”, otherwise no output should be displayed.

Ans. 

```
import java.util.*;
public class qual
{
 public static void main(String args[])
 {
 Scanner input=new Scanner(System.in);
 int m;
 System.out.println("Enter the marks:");
 m=input.nextInt();
 if(m>=75)
 System.out.println("Qualified for admission.");
 }
}
```

2. Write a program to input the marks obtained by a student. If the marks obtained by the student is 75 or above display “Qualified for admission”, otherwise display “Try next time”.

Ans. 

```
import java.util.*;
public class admission_result
{
 public static void main(String args[])
 {
 Scanner input=new Scanner(System.in);
 int m;
 System.out.println("Enter the marks:");
 m=input.nextInt();
 if(m>=75)
 System.out.println("Qualified for admission.");
 else
 System.out.println("Try next time.");
 }
}
```

3. Write a program to input the marks obtained by a student. Display the grades obtained by the student according to the following criteria:

| 80 and above | Distinction     |
|--------------|-----------------|
| 60-79        | First Division  |
| 45-59        | Second Division |
| 40-44        | Pass            |
| Below 40     | Fail            |

Ans. 

```
import java.util.*;
public class admission_grade
{
```

```

public static void main(String args[])
{
 Scanner input=new Scanner(System.in);
 int m;
 System.out.println("Enter the marks:");
 m=input.nextInt();
 if(m>=80)
 System.out.println("Distinction");
 else if(m>=60 && m<80)
 System.out.println("First Division");
 else if(m>=45 && m<60)
 System.out.println("Second Division");
 else if(m>=40 && m<45)
 System.out.println("Pass");
 else
 System.out.println("Fail");
}
}

```

4. Write a program using switch-case to input the first letter of the direction and print the name of the direction.

Ans. `import java.util.*;`

```

public class direction
{
 public static void main(String args[])
 {
 char ch;
 Scanner input = new Scanner(System.in);
 System.out.print("Enter your choice: ");
 ch = input.next().charAt(0);
 switch (ch)
 {
 case 'E': case 'e':
 System.out.println("East");
 break;
 case 'W': case 'w':
 System.out.println("West");
 break;
 case 'N': case 'n':
 System.out.println("North");
 break;
 case 'S': case 's':
 System.out.println("South");
 break;
 default:
 System.out.println("Invalid choice");
 }
 }
}

```

5. Write a program using switch-case to develop a menu and based upon it perform either of the conversions.

a. Rupees to paise

b. Paise to rupees

Ans. `import java.util.*;`

`public class conversion`

`{`

`public static void main(String args[])`

`{`

`Scanner input = new Scanner(System.in);`

`int ch;`

`double r1,r2,p1,p2;`

`System.out.println("Enter 1 for Rupee to Paise conversion: ");`

`System.out.println("Enter 2 for Paise to Rupee conversion: ");`

`System.out.print("Enter your choice: ");`

`ch = input.nextInt();`

`switch (ch)`

`{`

`case 1:`

`System.out.print("Enter rupee amount: ");`

`r1 = input.nextDouble();`

`p1 = r1 * 100;`

`System.out.println(r1 + " Rupees => " + p1 + " Paise");`

`break;`

`case 2:`

`System.out.print("Enter Paise amount: ");`

`p2 = input.nextDouble();`

`r2 = p2/100;`

`System.out.println(p2 + " Paise => " + r2 + " rupees");`

`break;`

`default:`

`System.out.println("Incorrect Choice");`

`}`

`}`

`}`

6. Write a program using switch case to find the volume of a cube, a sphere, and a cuboid. For an incorrect choice, an appropriate error message should be displayed.

Volume of a cube =  $s * s * s$

Volume of a sphere =  $(4/3) * \pi * r * r * r$  ( $\pi = (22/7)$ )

Volume of a cuboid =  $l * b * h$

Ans. `import java.util.*;`

`public class geometry`

`{`

`public static void main(String args[])`

`{`

`Scanner input = new Scanner(System.in);`

```

int ch;
double cs,cv,r,sa,l,b,h,vol;
System.out.println("1. Volume of Cube");
System.out.println("2. Volume of Sphere");
System.out.println("3. Volume of Cuboid");
System.out.print("Enter your choice: ");
ch = input.nextInt();
switch(ch) {
 case 1:
 System.out.print("Enter side of cube: ");
 cs = input.nextDouble();
 cv = Math.pow(cs, 3);
 System.out.println("Volume of cube = " + cv);
 break;
 case 2:
 System.out.print("Enter radius of sphere: ");
 r = input.nextDouble();
 sa = (4 / 3.0) * (22 / 7.0) * Math.pow(r, 3);
 System.out.println("Volume of sphere = " + sa);
 break;
 case 3:
 System.out.print("Enter length of cuboid: ");
 l = input.nextDouble();
 System.out.print("Enter breadth of cuboid: ");
 b = input.nextDouble();
 System.out.print("Enter height of cuboid: ");
 h = input.nextDouble();
 vol = l * b * h;
 System.out.println("Volume of cuboid = " + vol);
 break;
 default:
 System.out.println("Invalid Choice");
}
}
}

```

7. Write a program to input the number of students in a class. For every student input the marks and find the highest marks obtained in the class.

Ans. 

```
import java.util.*;
public class marks
{
 public static void main(String args[]) {
 Scanner input = new Scanner(System.in);
 int n,i,m,max=0;
 System.out.print("Enter number of students: ");
 n = input.nextInt();
 for(i=1;i<=n;i++)
```



```

 {
 System.out.print("Enter marks: ");
 m = input.nextInt();
 if (m>max)
 max=m;
 }
 System.out.println("Highest Marks = " + max);
}
}

```

8. Write a program to input a number. Check and display whether it is a Niven number or not. (A number is said to be Niven if it is divisible by the sum of its digits).

Ans. `import java.util.*;`  
`public class niven`  
`{`  
 `public static void main(String args[])`  
 `{`  
 `Scanner input = new Scanner(System.in);`  
 `System.out.print("Enter number: ");`  
 `int n = input.nextInt();`  
 `int t = n;`  
 `int s = 0;`  
 `while (n != 0)`  
 `{`  
 `int d = n % 10;`  
 `n /= 10;`  
 `s += d;`  
 `}`  
 `if (t != 0 && t % s == 0)`  
 `System.out.println(t + " is a Niven number");`  
 `else`  
 `System.out.println(t + " is not a Niven number");`  
 `}`  
 `}`  
`}`

9. Write a program that inputs the number of runs made by a cricket player on each ball. Entering runs as greater than 6 should display the message that the player is out. Finally, it should display the number of runs made and balls played by the player.

Ans. `import java.util.*;`  
`public class cricket`  
`{`  
 `public static void main(String args[])`  
 `{`  
 `Scanner input = new Scanner(System.in);`  
 `int r = 0, b = 0, t = 0;`  
 `do {`  
 `System.out.print("Enter runs: ");`

```

 r = input.nextInt();
 if (r > 6)
 System.out.println("Player is out");
 else
 t=t+r;
 b++;
 }
 while(r<=6);
 System.out.println("Total Runs made: " + t);
 System.out.println("Total Balls faced: " + b);
}
}

```

10. Write a program to input a number and check and print whether it is a Pronic number or not. [A pronic number is the product of two consecutive integers.]

Examples:  $42 = 6 * 7$  and  $72 = 8 * 9$

Ans. `import java.util.*;`  
`public class pronic`  
`{`  
     `public static void main(String args[])`  
     `{`  
         `Scanner input = new Scanner(System.in);`  
         `int n,i,x=0;`  
         `System.out.println("Enter a number: ");`  
         `n = input.nextInt();`  
         `for (i = 1; i <= n+1; i++)`  
         `{`  
             `if (i * (i + 1) == n)`  
             `{`  
                 `x=1;`  
                 `break;`  
             `}`  
         `}`  
         `if (x==1)`  
             `System.out.println(n + " is a pronic number");`  
         `else`  
             `System.out.println(n + " is not a pronic number");`  
     `}`  
`}`



**Link**

Scan the given QR Code to learn the use of jumping statements.



## Chapter Highlights

- 3.1 Objects and Classes
- 3.2 Objects Encapsulate State and Behaviour
- 3.3 Member Variables of a Class
- 3.4 Member Methods of a Class
- 3.5 Classes as Abstractions for Sets of Objects
- 3.6 Class as an Object Factory
- 3.7 Objects as Instances of a Class
- 3.8 Data Types
- 3.9 Variable Declarations
- 3.10 Some Real-life Examples of Classes and Objects

## Learning Outcomes

The student will be able to:

- understand the concept of classes and objects.
- know about member variables and member methods.
- create an instance of a class.
- distinguish between primitive and non-primitive data types.
- define and declare variables for both types.

Objects correspond to things found in the real world. For example, the school bus that you take to go to school, the book you study, and the pen you use to write are all different objects. These objects possess different attributes and show different behaviours in various situations. Let us learn about objects and classes in detail.

### 3.1 Objects and Classes

In object-oriented programming, data and functions operating on that data are stored in a single unit, which is referred to as an **object**. In general, an object may refer to a book, a person, a store, or any item that closely resembles the object in the real world. An object encapsulates both data and functions operating on that particular data.

For example, you can consider a doctor, patient, and nurse as the objects of a hospital management system.

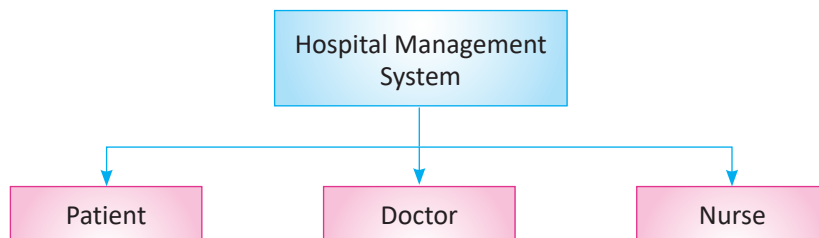


**Figure 3.1:** Objects of Hospital Management System

A **class** represents a set of objects that share common characteristics and behaviour. It is a way to bind the data describing an entity and its associated functions together. It can be considered as a collection of objects that contains a set of data items (states) and related functions (behaviour) to operate on the given data items. For example, a teacher, a clerk, or a gatekeeper are the objects of the Staff class, because all of these have similar features, such as name, address, age, blood group, and other personal details.

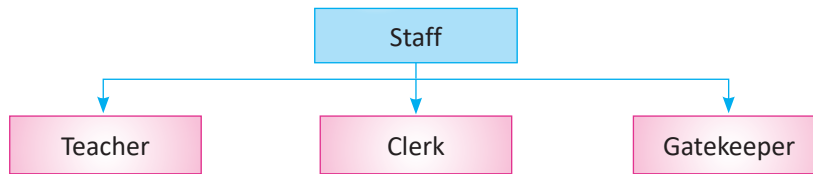


**Figure 3.2:** Objects of Staff Class

In object-oriented programming languages, the data items remain confined within an area of their application so that they remain hidden to the outside world and allow only some use of the essential features. This concept of data hiding is an important concept in OOP, which can be implemented by using classes.

In Java, an object is a physical and even a logical entity, whereas a class is only a logical entity. It is a blueprint from which objects are created. So, an object is the instance of a class. Some real-life examples of objects are as follows:

- ◆ Vehicles running on the road
- ◆ Patients in a hospital
- ◆ Students in a school
- ◆ Furniture in your house
- ◆ Software installed in your computer
- ◆ Grocery items in a grocery shop

As an object is considered as a physical entity in a program, each instance of a class may have certain real values instead of variables. The following are the characteristics of an object:

### Identity

An object has an identity that differentiates one object from the other, i.e., each object is a distinct individual. For example, a teacher teaches a subject, a doctor has a specialisation, and a vehicle has a model.

### State

An object has a state. For example, a teacher works in a school, a doctor goes to a clinic, and a vehicle is manufactured by a company.

### Behaviour

The behaviour of an object specifies its properties. For example, a teacher teaches in the class, a doctor examines the patients, and a vehicle is used for travelling.

## 3.2 Objects Encapsulate State and Behaviour

An object stores its state in member variables and exposes its behaviour through the member methods. The member methods operate on member variables and serve as the mechanism to interact with the various objects. Only the member methods that are wrapped inside the class can access the data and change its state. Hence, the state and behaviour are said to be encapsulated by the object, hiding the internal state and performing all the interaction through the methods of the object.

Encapsulation prevents free access to data within an object. The data can be accessed only through the interface provided by the associated functions. As encapsulation hides the data from other functions. This feature of object oriented programming is called data hiding.

## Data Hiding

The concept of data hiding states that the information obtained should be hidden from the rest of the system. It means that modules should be specified and constructed in such a way that information stored within one module is inaccessible to other modules that do not need it. For example, `object_name1` should not access the data of `object_name2`.

Data hiding has the following advantages:

- ✦ Any changes made to one object have no effect on the other.
- ✦ It reduces the interaction between various modules and makes the system more reliable.

## Encapsulation

Encapsulation refers to the wrapping up of data and functions together into a single unit called class. In other words, encapsulation provides a protective shield that prevents the data from being accessed by the code outside this shield. Technically, in encapsulation, the variables or data of a class are hidden from any other class and can be accessed only through the member functions of the class in which they are declared.

Data in a class is hidden from other classes using the concept of data hiding, which is achieved by making a class's members or methods private. The class is exposed to the end-user without providing any details behind its implementation using the abstraction concept. So, you can say that encapsulation is a combination of two concepts: data hiding and abstraction.

Encapsulation can be achieved by declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables.

### Advantages of Encapsulation

The following are the advantages of encapsulation:

**Data hiding:** The user will have no idea about the inner implementation of the class. It will not be visible to the user how the class is storing values in the variables.

**Reliability:** You can make the variables of the class read-only or write-only depending on your requirement.

**Reusability:** Encapsulation also improves the reusability and is easy to change with new requirements.

**Easy troubleshooting:** Encapsulated code is easy to troubleshoot.

## 3.3 Member Variables of a Class

All the variables declared inside a class are called **member variables** or **data members** of a class. They are all present in the body of the class instead of a method.

**Syntax:**

```
class <class name>
{
 Member variables; //Variables to be used
}
```

## Classification of Member Variables

The data members of a class can be of the following two types:

### Static Variables (Class Variables)

Static variables are also known as class variables. These variables are declared just like instance variables. The difference is that static variables (static data members) are declared using the 'static' keyword within a class outside of any method, constructor, or block. Unlike instance variables, you can only have one copy of a static variable per class, irrespective of how many objects you create. Static variables are created at the start of program execution and destroyed automatically when execution ends.

#### Syntax:

```
static data_type var1, var2, var3 ... varn;
```

### Instance Variables

Instance variables (non-static data members of a class) are declared in a class but outside any method, constructor, or block. As instance variables are declared in a class, they are created when an object of the class is created and destroyed when the object is destroyed. Every instance variable must be accessed with the respective object name.

#### Syntax:

```
data_type var1, var2, var3 ... varn;
```

The following example shows the declaration of instance and static variables in a class.

#### Example

```
public class Student
{
 String name; //Instance variable
 int roll_no; //Instance variable
 static float s_class //Class variable because of keyword static
}
```

## Difference between Instance Variables and Static Variables

| Instance Variables                                                                                                                                     | Static Variables                                                                                                                  |
|--------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| Each object will have its own copy of the instance variable.                                                                                           | You can only have one copy of a static variable per class, irrespective of how many objects you create.                           |
| Changes made to an instance variable using one object will not be reflected in other objects as each object has its own copy of the instance variable. | In case of static variables, changes will be reflected in other objects as static variables are common to all objects of a class. |
| You can access the instance variables through object references.                                                                                       | Static variables can be accessed directly using class name.                                                                       |

### 3.4 Member Methods of a Class

Like member variables, a class also holds **member methods**, which are also known as **member functions** that deal with instance variables. The functions declared inside the class are known as member functions, like `display_information()`, `Calculation()`, and so on. Member variables and member functions together define the properties and behaviour of the objects in a class.

Here, only a brief introduction to the methods is given. The detailed explanation of methods is given in the next chapter.

A method declaration must have the method name and its return type, indicating the type of data that will be returned when the method is called.

#### Syntax:

```
return_type method_name()
{
 Method body
}
```

#### Example

```
int sum(int a, int b) //A method
{
 int c=a+b;
 return c;
}
```



### 3.5 Classes as Abstractions for Sets of Objects

**Abstraction** refers to representing only the essential features of an object without revealing its inner details. It is related to the implementation of data hiding. While analysing a program, the programmer focuses on one module at a time, without knowing the internal details of the other modules. In short, a concept that defines the overall external behaviour of the system without knowing the internal details of a system is called an abstraction.

Data abstraction can also be considered as a property by virtue of which only the essential details are displayed to the user. The non-essential details are not displayed to the user. For example, a motorbike is viewed as a complete unit rather than its individual components.

Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of a car or applying brakes will stop the car, but he does not know how on pressing the accelerator, the speed is actually increasing. This means he does not know about the inner mechanism of the car or the implementation of the accelerator, brakes, etc., in the car. This situation is also referred to as abstraction.

#### Encapsulation vs Abstraction

The following points describe the comparison between encapsulation and data abstraction:

- ◆ Encapsulation is data hiding, while abstraction refers to representing essential components only.
- ◆ Encapsulation groups together data and methods that act upon the data, whereas abstraction deals with exposing the interface to the user and hiding the internal details.

#### Advantages of Abstraction

The following are some advantages of using the concept of abstraction:

- ◆ It reduces the complexity of a program.
- ◆ It avoids the repetition of data.

- ◆ It increases the reliability of the program.
- ◆ It helps to increase the security of an application or program.

### 3.6 Class as an Object Factory

Classes in Java are needed to represent real-world entities. They provide a convenient method for packing together a group of logically related data items and functions that work on them. In Java, the data items are called **fields**, and the functions are called **methods**.

Declaring a class means creating a **user-defined data type**. Objects of a class can be created within a program as many times as you want. Each object created by a specific class possesses the data and methods defined within the class. A class produces the objects of similar types. Hence, it is termed as an **object factory**. A class contains the details of the object's internal data and when the object is brought into existence.

#### Syntax:

The syntax to create a class is as follows:

```
<Access Modifier> class <class name>
```

```
{
 ↑ ↑ ↑
 Data members;

 Member methods()

}
```

Attributes required to declare a class

Consider the following example to understand how to declare a class.

#### Example

```
public class Student
{
 //Data members
 String name;
 int class;
 int age;
 //Member methods
 InputData();
 PrintData();
}
```

#### Syntax:

The syntax for creating an object of the class is as follows:

```
<Class Name> <Object Name> = new<Constructor Name>();
```



Notice that, a constructor is used to initialise an object of a class. It has the same name as that of the class. You will study constructors in the further chapters.

Now, the objects for the class 'Student' can be created using the following statements:

```
Student name1=new Student();
```

```
Student name2=new Student();
```

Here, name1 and name2 are the names of the objects of class Student. In this way, you can create as many objects of a class as you wish.

### 3.7 Objects as Instances of a Class

An object is the basic unit of the object-oriented programming, which represents a particular instance of a class. There can be more than one instances of every object. Every instance of the object has its own relevant data. In Java, a class is used to create objects for defining various data types and methods.

Once a class has been defined, you can create many objects belonging to that class. An object belonging to a particular class is known as an **instance** of that class. Therefore, you can say that an orange is an instance of the **fruit** class. The terms object and instance are often interchangeable.

Consider the objects created below for the Student class.

```
Student Rahul=new Student();
```

```
Student Neha=new Student();
```

Here, Rahul and Neha, are the two objects of the class 'Student' can also be considered as its instances. The process of creating objects of a class is called **instantiation**. You can have as many number of instances within a class as you wish. Each object has a different set of values for its data members, however, the various methods of a class behave identically for all the objects.

The definition statement of an object has the following three parts:

**Declaration:** It tells the computer that the data type of the declared variable is a class.

**Instantiation:** It uses the new keyword to allocate memory space for an object.

**Initialisation:** The new keyword calls the constructor, which initialises the attributes of the new object.

### 3.8 Data Types

As you know, data types refer to the different types of values that can be stored in a variable. Each member variable needs a data type depending on the type of data used to declare it. Data types are divided into two categories: **primitive** and **non-primitive (composite)**.

#### Primitive Data Types

Primitive data types are Java's fundamental data types that are built-in and are defined by Java language specification. These data types will always has some value. They cannot be null. There are eight primitive data types in Java: byte, short, int, long, char, float, double, and boolean.

### Example

```
int marks = 75; //Integer
float temperature = 98.4; //Floating point number
char grade = 'D'; //Character
boolean result = true; //Boolean
String name = "Harshit"; //String
```

## Composite or Non-Primitive Data Types

Non-primitive data types are created by the programmers as they are not predefined like primitive data types. These data types are also called **reference data types** because they refer to the memory location where data is stored. Classes, Objects, Arrays, Strings, and Interfaces are examples of non-primitive or composite data types as they are created using primitive data types. Composite data types can be null also. The following are different examples of non-primitive data type variables:

### Example

```
Student ob=new Student(); //Object of class Student
int marks[]=new marks[]; //Declaring an array, a non-primitive data type
```

## Difference between Primitive and Non-Primitive Data Types

| Primitive Data Types                                                                 | Non-Primitive Data Types                                                                         |
|--------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------|
| Primitive data types are Java's fundamental data types.                              | Non-primitive or composite data types are created by the programmers using primitive data types. |
| Primitive data types are built-in data types defined by Java language specification. | Composite data types are defined by the programmer.                                              |
| A primitive data type always has a value.                                            | Non-primitive data type variables can be null also.                                              |
| Examples: byte, short, int, long, float, double, char, Boolean                       | Examples: Classes, Objects, Arrays, Strings, and Interfaces                                      |

## 3.9 Variable Declarations

As you know, a variable is a memory location that holds the data value of a particular data type. Each variable has a name by which it is identified in the program. The value stored in a variable can be changed during program execution. In Java, all variables must be declared before use. Declaring a variable means getting a memory location allocated to store a value of the specified data type. This memory location can be accessed by the specified data type. In Java, you can declare different types of variables as listed below:

**String:** It stores text, such as "Hello Java". String values are enclosed in double quotes.

**int:** It stores integers (whole numbers), without decimals, such as 1234 or –2345. This type of variable includes negative values also.

**float:** It stores floating point numbers, with decimals, such as 32.57 or –45.99.

**char:** It stores single characters, such as 'a' or 'A'; char values are enclosed in single quotes.

**boolean:** It stores values with two states: true or false.

## Declaring a Variable

You need to specify the data type and name for a variable while declaring it.

### Syntax:

```
<data-type> <variable-name>;
```

### Example:

```
String name;
```

In this way, a name can only be given to a memory location.

## Initialising Variables

To initialise a variable, you need the following three components:

**datatype:** It is the type of data that can be stored in the variable.

**variable\_name:** It specifies the name given to the variable.

**value:** It specifies the initial value stored in the variable.

The following example shows initialisation of a variable:

### Syntax:

```
Data_type variable_name = value;
```

### Example

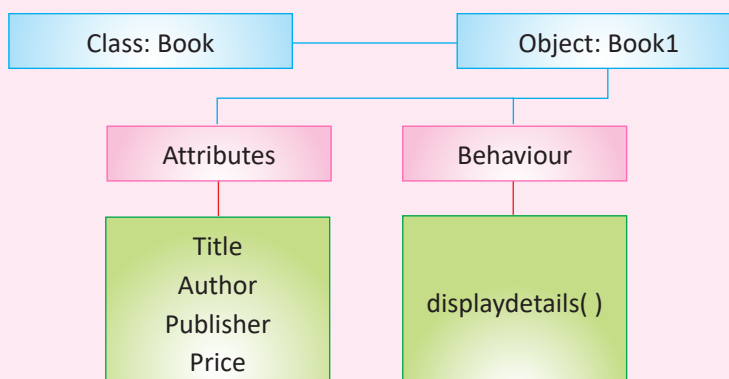
```
String name = "Harshit";
```

## 3.10 Some Real-life Examples of Classes and Objects

The following section describes some real-life examples of classes and objects. Also, the programming implementation of the same has also been discussed here.

### Example 1

Consider the example of a class Book and its object Book1 having the attributes and behaviour as shown below:



## Program 1

Design a class Book in Java and define its data members and member methods. Create an object of the class and use the data members and member methods.

```
1 public class Book
2 {
3 String title; //Data member
4 String author; //Data member
5 String publisher; //Data member
6 double price; //Data member
7 void displaydetails () //Member method
8 {
9 System.out.println("The title of the book is: " +title);
10 System.out.println("The name of the author of the book is: " +author);
11 System.out.println("The publisher of the book is: " +publisher);
12 System.out.println("The price of the book is: " +price);
13 }
14 public static void main(String[] args)
15 {
16 Book Book1 = new Book(); //Creating object or instance of a class
17 Book1.title = "Computer Applications";
18 Book1.author = "Santosh Kumar";
19 Book1.publisher = "Kips Learning";
20 Book1.price = 550;
21 Book1.displaydetails();
22 }
```

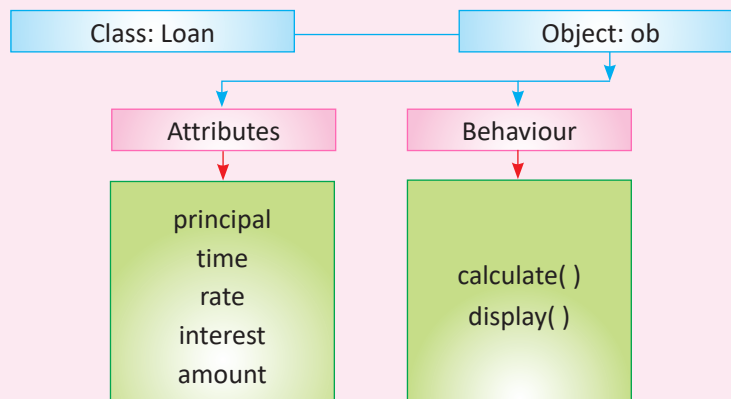
### OUTPUT

The title of the book is: Computer Applications  
The name of the author of the book is: Santosh Kumar  
The publisher of the book is: Kips Learning  
The price of the book is: 550

Using data members and methods of a class

## Example 2

Consider the example of a class Loan and its object ob having the attributes and behaviour as shown below:



## Program 2

Bank charges interest for the vehicle loan as given below:

| Number of years                | Rate of interest |
|--------------------------------|------------------|
| Up to 5 years                  | 15%              |
| More than 5 and up to 10 years | 12%              |
| Above 10 years                 | 10%              |

Write a program to create the class 'Loan' with the specifications given below:

| Data Members     | Purpose                                  |
|------------------|------------------------------------------|
| int time         | Time period for which loan is sanctioned |
| double principal | Amount sanctioned                        |
| double rate      | Rate of interest                         |
| double interest  | To store the interest                    |
| double amt       | Amount to pay after given time           |

| Member Methods   | Purpose                                                                                                |
|------------------|--------------------------------------------------------------------------------------------------------|
| void getdata()   | To accept principal and time                                                                           |
| void calculate() | To find interest and amount<br>[interest = (principal*rate*time)/100<br>amount = principal + interest] |
| void display()   | To display interest and amount                                                                         |

```
1. import java.util.*;
2. public class Loan
3. {
4. int time;
5. double principal, rate, interest, amt;
6. public void getdata()
7. {
8. Scanner in = new Scanner(System.in);
9. System.out.print("Enter principal: ");
10. principal = in.nextInt();
11. System.out.print("Enter time: ");
12. time = in.nextInt();
13. }
14. public void calculate()
15. {
```

### OUTPUT

```
Enter principal: 500000
Enter time: 3
Interest = 225000.0
Amount payable = 725000.0
```

```

16. if (time <= 5)
17. rate = 15.0;
18. else if (time <= 10)
19. rate = 12.0;
20. else
21. rate = 10.0;
22. interest = (principal * rate * time) / 100.0;
23. amt = principal + interest;
24. }
25. public void display()
26. {
27. System.out.println("Interest = " + interest);
28. System.out.println("Amount payable = " + amt);
29. }
30. public static void main(String args[])
31. {
32. Loan ob = new Loan();
33. ob.getdata();
34. ob.calculate();
35. ob.display();
36. }
37. }

```



## Let's Revise

- ◆ An object is an identifiable entity that has a set of attributes, behaviour, and state.
- ◆ A class is a group of objects that share common characteristics and behaviours.
- ◆ The process of creating objects of a class is called instantiation. You can have as many number of instances within a class as you wish.
- ◆ The concept of data hiding states that the information obtained should be hidden from the rest of the system.
- ◆ Abstraction refers to representing only the essential features of an object without revealing its inner details.
- ◆ Data types refer to the different types of values that can be stored in a variable.
- ◆ Primitive data types are built-in data types in Java. They are defined by Java language specification.
- ◆ The non-primitive data types are created by the programmers as they are not predefined like primitive data types.
- ◆ A variable is a memory location that holds the data value of a particular data type.

## Key Terms

**Class:** A class is a logical construct whereas an object has a physical reality.

**Object Factory:** A class is called as an object factory because you can create as many objects as you want using a class.

**Encapsulation:** It is the wrapping up of data and functions into a single unit called class.

**Abstraction:** It is the act of representing essential components and hiding the background details.

**Data Members:** Variables declared inside a class are known as data members.

**Member Functions:** The functions declared inside the class are known as member functions.

**Instance Variables:** These are declared in a class but outside of any method, constructor, or block. These are created when an object of the class is created and destroyed when the object is destroyed.

**Static Variables:** These are class variables declared with the static modifier (keyword) of which a single copy exists, irrespective of how many objects you create.



## SECTION 1 (Solved Questions)

### A. Fill in the blanks.

1. An object is referred as an \_\_\_\_\_ of a class.
2. A class is a \_\_\_\_\_ data type.
3. The \_\_\_\_\_ keyword is used to create an object of a class.
4. \_\_\_\_\_ is the process of wrapping up of data and functions together into a single unit.
5. \_\_\_\_\_ refers to the concept of representing essential components and hiding background details.
6. The data types 'int' and 'float' are examples of \_\_\_\_\_ data types.
7. A \_\_\_\_\_ is used to initialise an object of a class.
8. The \_\_\_\_\_ variables are created at the start of program execution and destroyed automatically when the execution ends.

|      |                |                  |                |                  |
|------|----------------|------------------|----------------|------------------|
| Ans. | 1. Instance    | 2. Non-primitive | 3. new         | 4. Encapsulation |
|      | 5. Abstraction | 6. Primitive     | 7. Constructor | 8. static        |

### B. Very short answer type questions.

1. Write any two non-numeric primitive data types.

Ans. char and boolean

2. Why is a class known as composite data type?

Ans. A composite data type is the one which is composed using various primitive data types. A class can contain various primitive data types as its members, so it can be considered as a composite data type.

3. Write the syntax for creating an instance of a class.

Ans. Syntax to create an instance of a class is:

```
<Class Name> <Object Name>= new <Constructor Name>();
```

4. Mention any two attributes required for class declaration.

Ans. Two attributes required for class declaration are: access specifier and class name.

5. What are instance variables?

Ans. Instance variables are non-static variables that are declared in a class, but outside of any method, constructor, or block.

### C. Short answer type questions.

1. What is a class? Give a real-life example to explain the concept of class.

Ans. A class is a blueprint for objects that share common attributes and behaviours. For example, a lion, a giraffe, and a cow can be kept in the 'Animal' class as all three can share some common attributes (four legs, two eyes, etc.) and behaviours (eating, sleeping, etc.).

2. What are the member methods in a class?

Ans. The functions declared inside the class are known as member functions or member methods, like `display_information()`, `Calculation()`, and so on. Member variables and member methods together define the properties and behaviour of the objects in a class.

3. Why is an object called an instance of a class?

Ans. An object is called an instance of a class because every object has its own instance of the member variables and methods.

4. Differentiate between instance (non-static) variables and class (static) variables.

| Ans. | Instance Variables                                              | Class Variables                                                |
|------|-----------------------------------------------------------------|----------------------------------------------------------------|
|      | They are declared without using the 'static' keyword.           | They are declared using the 'static' keyword.                  |
|      | Each object of a class gets its own copy of instance variables. | All objects of a class share the same copy of class variables. |
|      | They can be accessed only through an object of the class.       | They can be accessed using the class name or object.           |

5. Differentiate between built-in data types and user-defined data types.

| Ans. | Built-in Data Types                                                                        | User-defined Data Types                                                        |
|------|--------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------|
|      | Built-in data types are fundamental data types defined by the Java language specification. | User-defined data types are created by the user.                               |
|      | The size of built-in data types is fixed.                                                  | The size of the user-defined data types depend upon their constituent members. |
|      | Built-in data types are available in all parts of a Java program.                          | Availability of user-defined data types depends upon their scope.              |
|      | Built-in data types are independent components.                                            | User-defined data types are composed of built-in data types.                   |

### D. Application based questions.

1. Write a program to accept two numbers as instance variables of a class. Use the following functions for the given purposes:

**Class name:** Calculate

**Member methods:**

`void inputdata( )`: To input both the values

`void calculate( )`: To find sum and difference



void outputdata( ): To print sum and difference of both the numbers

Use a main( ) method to call all the above-mentioned functions.

Ans. 

```
import java.util.*;
public class Calculate
{
 int a,b,sum,diff;
 public void inputdata() {
 Scanner in = new Scanner(System.in);
 System.out.print("Enter first number: ");
 a = in.nextInt();
 System.out.print("Enter second number: ");
 b = in.nextInt();
 }
 public void calculate()
 {
 sum = a + b;
 diff = a - b;
 }
 public void outputdata()
 {
 System.out.println("Sum = " + sum);
 System.out.println("Difference = " + diff);
 }
 public static void main(String args[])
 {
 Calculate ob= new Calculate();
 ob.inputdata();
 ob.calculate();
 ob.outputdata();
 }
}
```

2. Write a program with the following specifications:

**Class name:** Triplet

**Data members:** int a, b, c

**Member methods:**

void getdata( ): To accept three numbers

void findprint( ): To check and display whether the numbers are Pythagorean triplets or not

Ans. 

```
import java.util.*;
public class Triplet
{
 private int a;
 private int b;
 private int c;
```

```

void getdata()
{
 Scanner in = new Scanner(System.in);
 System.out.print("Enter a: ");
 a = in.nextInt();
 System.out.print("Enter b: ");
 b = in.nextInt();
 System.out.print("Enter c: ");
 c = in.nextInt();
}
void findprint()
{
 if ((Math.pow(a, 2) + Math.pow(b, 2)) == Math.pow(c, 2)
 || (Math.pow(b, 2) + Math.pow(c, 2)) == Math.pow(a, 2)
 || (Math.pow(a, 2) + Math.pow(c, 2)) == Math.pow(b, 2))
 System.out.print("Numbers are Pythagorean Triplets");
 else
 System.out.print("Numbers are not Pythagorean Triplets");
}
public static void main(String args[])
{
 Triplet ob = new Triplet();
 ob.getdata();
 ob.findprint();
}
}

```

### 3. Define a class discount having the following description:

#### Data members

int cost: To store the price of an article

String name: To store the customer's name

double dc: To store the discount

double amt: To store the amount to be paid

#### Member functions

void input( ): To store the cost of the article and name of the customer

void cal( ): To calculate the discount and amount to be paid

void display( ): To displays the name of the customer, cost, discount, and amount to be paid

Write a program to compute the discount according to the following conditions and display the output as per the given format.

| List Price              | Rate of discount      |
|-------------------------|-----------------------|
| Up to ₹5,000            | No discount           |
| From ₹5,001 to ₹10,000  | 10% on the list price |
| From ₹10,001 to ₹15,000 | 15% on the list price |
| Above ₹15,000           | 20% on the list price |

**Output:**

| Name of the customer | Discount | Amount to be paid |
|----------------------|----------|-------------------|
| .....                | .....    | .....             |
| .....                | .....    | .....             |

```
Ans. import java.util.*;
public class Discount
{
 int cost;
 String name;
 double dc,amt;
 public void input()
 {
 Scanner in = new Scanner(System.in);
 System.out.print("Enter customer name: ");
 name = in.nextLine();
 System.out.print("Enter article cost: ");
 cost = in.nextInt();
 }
 public void cal()
 {
 if (cost <= 5000)
 dc = 0;
 else if (cost <= 10000)
 dc = cost * 0.1;
 else if (cost <= 15000)
 dc = cost * 0.15;
 else
 dc = cost * 0.2;
 amt = cost - dc;
 }
 public void display()
 {
 System.out.println("Name of the customer\tDiscount\tAmount to be paid");
 System.out.println(name + "\t" + dc + "\t" + amt);
 }
 public static void main(String args[])
 {
 Discount ob = new Discount();
 ob.input();
 ob.cal();
 ob.display();
 }
}
```

**A. Very short answer type questions.**

1. What are the data types?
2. Name the different primitive data types.
3. Define instance variables.
4. Why is the class called an object factory?
5. List the characteristics of an object.
6. Define class variables. Give an example.

**B. Short answer type questions.**

1. Which of the following declarations are not allowed and why?
  - a. `class abc{...}`
  - b. `public class NumberOfDaysWorked{...}`
  - c. `private int x;`
  - d. `private class abc{...}`
  - e. `default key getKey(...)`
2. Differentiate between built-in data types and user-defined data types.
3. How is the concept of abstraction useful in Java?
4. What is the need for a class in Java?
5. What are data members?

**C. Application based questions.**

1. Define a class 'Electricity\_Bill' with the following specifications:

**Data members**

String n: The name of the customer

int units: The number of units consumed

double bill: The amount to be paid

**Member methods**

void accept( ): To store the name of the customer and number of units consumed

void calculate( ): To calculate the bill as per the tariff table given below

void print( ): To prints the details in the format given below

**Output:**

Name of the customer: .....

Number of units consumed: .....

Bill amount: .....

**Tariff table**

| Number of units | Rate per unit |
|-----------------|---------------|
| First 100 units | ₹2.00         |
| Next 200 units  | ₹3.00         |
| Above 300 units | ₹5.00         |

A surcharge of 2.5% charged if the number of units consumed is above 300 units.

Write a main( ) method to create an object of the class and call all the member methods.

2. Design a class 'Railway Ticket' with following description:

**Data members**

String name: To store the name of the customer

String coach: To store the type of coach customer wants to travel in

long mob\_no: To store customer's mobile number

int amt: To store basic amount of ticket

int total\_amt: To store the amount to be paid after updating the original amount

**Member methods**

void accept( ): To take input for name, coach, mobile number, and amount

void update( ): To update the amount as per the coach selected (extra amount to be added in the amount as per the table below)

void display( ): To display all details of a customer such as name, coach, total amount, and mobile number

| Type of Coaches | Amount |
|-----------------|--------|
| First_AC        | ₹700   |
| Second_AC       | ₹500   |
| Third_AC        | ₹250   |
| Sleeper         | None   |

Write a main( ) method to create an object of the class and call all the member methods.

### SECTION 3 (Board Questions)

1. Why is an object called an instance of a class? [2010]

**Ans.** A class can create objects of itself with different characteristics and common behaviour. So, you can say that an object represents a specific state of the class.

2. What is the difference between an object and a class? [2011]

**Ans.** An object is an unique entity which has certain characteristics and behaviours whereas a class is a group of objects.

An object is an instance of a class whereas a class is an object factory.

3. What does a class encapsulate? [2013]

**Ans.** A class encapsulates data and behaviour.

4. Find out the composite data types from the list given below: [2014]

```
static int x;
arr[i]=10;
obj.display();
boolean b;
private char chr;
String str;
```

**Ans.** The composite data types are:

```
arr[i]=10;
obj.display();
String str;
```

## SECTION 4 (Case Study Based Questions)

Java is an object oriented programming language. In order to implement this concept, Java has features called as classes and objects in Java. These are fundamental concepts in the field of Object Oriented Programming and having a good grasp of these concepts is essential for further concepts of OOP. A class is the blueprint of all the objects that are derived from it. It represents the set of methods and properties that are common to all the objects which are referenced from it. It has the methods and variables that are common to all objects.

1. A class is declared by the keyword \_\_\_\_\_.  
a. class                      b. new                      c. constructor                      d. import
2. A class binds together data and \_\_\_\_\_.  
a. Objects                      b. Instances                      c. Functions                      d. Classes
3. A \_\_\_\_\_ is also called an object factory.  
a. Class                      b. Method                      c. Function                      d. Instance
4. The \_\_\_\_\_ are real-life entities in a class.  
a. Instance                      b. Objects                      c. Characteristics                      d. Modules

**Ans.** 1. a      2. c      3. a      4. b

## Programming ZONE

1. Write a program to calculate the factorial of a given number using classes and objects.

**Ans.** `class` Calculation

```
{
 void fact(int n)
 {
 int fact=1;
 for(int i=1;i<=n;i++)
 {
 fact=fact*i;
 }
 System.out.println("factorial is "+fact);
 }
 public static void main(String args[])
 {
 new Calculation().fact(5);//calling method with anonymous object
 }
}
```

2. Write a program to demonstrate the working of a banking-system where you deposit and withdraw amount from your account. Create an Account class which has deposit( ) and withdraw( ) methods.

**Ans.** `class` Account

```
{
```

```

int acc_no;
String name;
float amount;
//Method to initialize object
void insert(int a, String n, float amt)
{
 acc_no=a;
 name=n;
 amount=amt;
}
//deposit method
void deposit(float amt)
{
 amount=amount+amt;
 System.out.println(amt+" deposited");
}
//withdraw method
void withdraw(float amt)
{
 if(amount<amt)
 {
 System.out.println("Insufcient Balance");
 }
 else
 {
 amount=amount-amt;
 System.out.println(amt+" withdrawn");
 }
}
//method to check the balance of the account
void checkBalance()
{
 System.out.println("Balance is: "+amount);
}
//method to display the values of an object
void display()
{

```

```

 System.out.println(acc_no+" "+name+" "+amount);
 }
 //Creating a test class to deposit and withdraw amount
 class TestAccount
 {
 public static void main(String[] args)
 {
 Account a1=new Account();
 a1.insert(832345,"Ankit",1000);
 a1.display();
 a1.checkBalance();
 a1.deposit(40000);
 a1.checkBalance();
 a1.withdraw(15000);
 a1.checkBalance();
 }
 }

```

3. Write a program to create an Employee class. Create a method insert to add the details of the employee and display method to display the details.
4. Write a program to create a class Shape. Design the corresponding methods to define different shapes with different sides and calculate their area respectively.



**Link**

Scan the given QR Code to learn more about classes and objects.





## Chapter Highlights

- 4.1 Introduction
- 4.2 Defining a Method
- 4.3 Types of Parameters
- 4.4 Calling Methods
- 4.5 Using the return Statement in a Method
- 4.6 Programs Based on Methods
- 4.7 Pure and Impure Methods
- 4.8 Static and Non-static Methods
- 4.9 Method Overloading (Polymorphism)

## Learning Outcomes

### The student will be able to:

- understand the concept of methods.
- define methods.
- describe the types of parameters used in a method.
- know how to call a method.
- use the return statement in methods.
- differentiate between pure and impure methods.
- differentiate between static and non-static methods.
- understand method overloading.

Methods are the main building blocks of a Java program. Just like your body has multiple organs that work in collaboration with each other and have their own specific tasks. Similarly, each method in Java has its own specific task, and they work in combination with other components of the program.

## 4.1 Introduction

Lengthy programs are more prone to errors, and finding and correcting those errors is time-consuming. To overcome this, programs are broken into a number of small logical components, each of which serves a specific task. The process of splitting the longer and complex programs into a number of smaller units is called **modularisation**. Programming with such an approach is called modular programming, and these smaller units are referred to as methods or functions.

## Methods

A **method** is a set of instructions to carry out a particular task. It is also known as a function, procedure, module, subroutine, or sub-program. Methods can be reused in programs as many times as you want. It hides the computational complexity of the program. Using methods, you can structure your programs in a modular way. In a Java program, at least one method is needed, that is the `main( )` method.

## Need of using Methods

Methods are an important component of every Java program. A class has no capability without methods, because the objects created by such a class cannot perform any operations. Therefore, you must include methods that are necessary for processing the data contained in the class.

The various reasons for which methods are used in Java programs are listed below:

- ✦ Methods help in improving the readability of code.
- ✦ The code reusability feature of methods allows you to reuse the same chunks of code multiple times without retyping them.
- ✦ Methods reduce the complexity of the program by dividing a more complex task into individual segments.
- ✦ They make the program easy to troubleshoot.

## Types of Methods

There are two types of methods that you can use in your Java programs, as listed below:

**System-defined methods:** These are built-in methods or library methods that are predefined and available for use. For example, `Math.sqrt()` and `Math.round()`.

**User-defined methods:** These are defined by users as per their requirements. They can be modified whenever required.

This chapter deals with the discussion of user-defined methods. You will learn how to create your own custom methods and use them in your programs.



## 4.2 Defining a Method

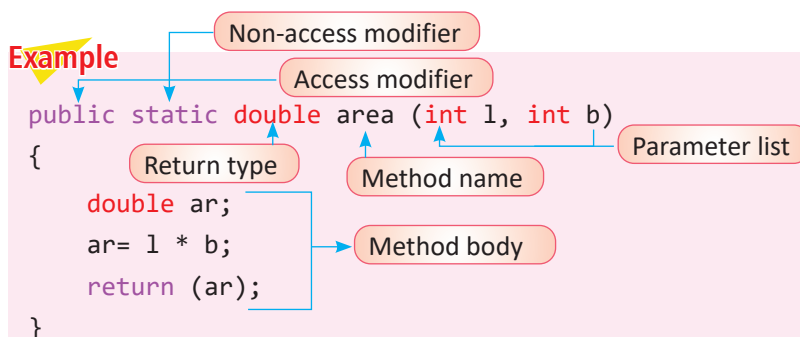
In a class definition, the member methods are defined after the data members. The following syntax is used to define a method:

**Syntax:**

```
<Access modifier><Non-access modifier><return_type><Method_name> (parameter_list)
{
 variable declaration;
 statement1;
 statement2;

 return(value_computed);
}
```

The example below shows how to define a method using the various components specified in the above-given syntax.



Let us know about these components of a method in detail.

## Access Modifiers

An access modifier or specifier specifies the scope of a method. It could be **public**, **private**, **protected**, or **default**.

**Public:** This type of method can be used from anywhere, i.e., it can be accessed within the class, outside the class, within the package, and outside the package.

**Protected:** This type of method has a scope within the class or package. You can access a protected method outside of the class or package through the child (sub-class) class. If you do not make the child class, it cannot be accessed from outside the package.

**Private:** This type of method is confined to the class only. It cannot be accessed from the other classes, inherited classes, or objects of classes.

**Default:** When no access specifier is specified, then it is assumed to be 'default'. The scope of this type of method is within the package.

The following table summarises the scope of the access modifiers:

| Access Specifier | Inside Class | Inside Package | Outside package subclass | Outside package |
|------------------|--------------|----------------|--------------------------|-----------------|
| Private          | Yes          | No             | No                       | No              |
| Default          | Yes          | Yes            | No                       | No              |
| Protected        | Yes          | Yes            | Yes                      | No              |
| Public           | Yes          | Yes            | Yes                      | Yes             |

## Non-Access Modifiers

Like access modifiers or specifiers, Java supports many non-access modifiers, such as **static**, **final**, **abstract**, **synchronised**, **volatile**, **transient**, and **native**. They do not change the accessibility of methods, but they provide special properties to them. That is why, they are commonly known as non-access specifiers.

For example, if a method is declared **final**, it cannot be overridden. If a method is declared **static**, you do not need to create an object to access it. The static methods can be accessed by using the class name. In other words, the static methods belong to the class rather than to the objects of that class. So, they are often referred to as **class methods**.

Normally, you use the static modifier in your programs. The use of the final modifier will be discussed in later chapters.

## Return Type

It specifies what type of data (int, float, double, etc.) is to be returned back on a method call. The return type of a method can be any of the primitive data types. For example, if a method has an int return type then it returns an integer value.

**Example:**

```
int sum()
```

If no return type is specified, then the return type is taken as void. The **void** data type specifies an empty set of values. It is used as the return type for methods that do not return a value.

## Example

```
void area_circle(double radius)
{
 double area, pi=3.14;
 area = pi * radius * radius;
 System.out.println("Area of a circle: " + area);
}
```

The return type of this method is void so it does not return any value.

## Function Name

It specifies the unique name given to the method. It must correspond to the method's functionality. For example, if you are developing a method for calculating the average of marks, then the method name can be Calculation( ), Calc( ), or CalcAverage( ). A method is invoked by its name.

## Parameter List

A parameter list refers to a list of comma-separated variables enclosed within the parentheses of a method. It contains the data type and variable name, that the method requires in order to perform computations on values passed during the method call.

## Body of a Method

The method-body within curly brackets contains a block of statements that are executed to perform a task on a method call. Basically, a method contains all the actions to be performed.

## Method Prototype and Signature

The first line of the method definition is called a **method prototype** or **method header**. It essentially describes a function's interface to the compiler by providing details. For example, it tells the number of arguments passed to the function, the data types of each of the passed arguments, and the order in which the arguments are passed to the function. This information is used by the compiler to ensure that the function calls have the correct number of arguments or parameters. It also ensures that each argument or parameter has the appropriate data type.

**Method signature** basically refers to the number of parameters and their types. It is the part of the prototype.



## Quick Quiz

1. Write the prototype of the function 'challenge', which receives two characters 'ch1' and 'ch2' as parameters and an integer 'n', and returns 'true' or 'false'.

boolean challenge(char ch1, char ch2, int n)

2. Write the prototype of the function 'test' which receives two double values and a char as parameters, and returns a double value.

double test (double a, double b, char c)

## 4.3 Types of Parameters

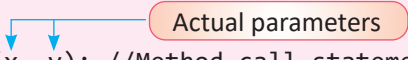
While defining a method, you declare a list of variables within a pair of parentheses followed by the function name. These are called parameters. In Java, there are two types of parameters: actual and formal. Let us learn about these in detail.

### Actual Parameters

The variables used in the method reference or method call statements are called **actual parameters** or **arguments**. They are written within the parentheses, followed by the name of the method. When these type of parameters are passed to a method, they take the place of the variables used in the method definition.

#### Example

```
public class pass1
{
 static int product(int a, int b) //Defining a method
 {
 return(a*b);
 }
 public static void main(String args[])
 {
 int x = 5;
 int y = 7;
 int result = product(x, y); //Method call statement
 System.out.println(" The product is: " + result);
 }
}
```



#### OUTPUT

The product is: 35

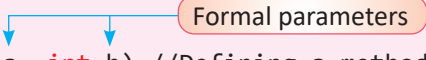
In the above example, the variable 'x' and 'y' are the actual parameters as they are passed to the method during a method call.

### Formal Parameters

The variables used in a method definition are referred to as **formal parameters**. Whenever you define formal parameters, the compiler gets information about variables that are being used in the method.

#### Example

```
public class pass2
{
 static int product(int a, int b) //Defining a method
 {
 return(a * b);
 }
}
```



#### OUTPUT

The product is: 20

```

public static void main(String args[])
{
 int x = 4;
 int y = 5;
 int result = product(x, y); //A method call statement
 System.out.println("The product is: " + result);
}
}

```

Actual parameters

In the above example, the variables 'a' and 'b' are the formal parameters as they are given in a method definition. These two parameters will receive the values from a method call statement.

Note a few important points related to the arguments and parameters:

- Both parameters and arguments are the variables used in a program or method.
- The parameter list must be separated by commas.

### Differences between Formal and Actual Parameters

| Formal Parameter                                                                    | Actual Parameter                                       |
|-------------------------------------------------------------------------------------|--------------------------------------------------------|
| It contains the list of variables along with their data types, separated by commas. | It contains the actual variables used in the function. |
| It is used in the method definition.                                                | It is used in the method call statement.               |
| Example: void sum (int a, int b)                                                    | Example: sum (3, 5)                                    |

## 4.4 Calling Methods

The method does not get executed until you invoke or call it. The method can be invoked or called by a **method call** statement. A method call statement holds the method name followed by the parameter list in the round brackets.

### Example

The following are examples of method call statements:

```

Display();
Product(x, y);
Sum (2,5);

```

The methods can be invoked or called in two ways:

- Pass by value (Call by value)
- Pass by reference (Call by reference)

Let us discuss these two ways of invoking methods in detail.

### Pass by Value

In the **pass by value** method, the parameter values are copied to the other variables and then passed. That is why, it is named 'pass by value' because the actual value of the parameters does not change. It can alternatively be defined as the process of copying actual parameters into formal parameters. Any modification to the formal parameters has no impact on the actual parameters. This is also known as **call by value**.

Consider the following example to understand the concept of call by value.

### Example 1

```
public class cval
{
 static void swap(int x, int y) //Defining a method
 {
 int t;
 t=x; x=y;
 y=t;
 System.out.println("Inside method: x= "+x +" & y= "+y);
 }

 public static void main(String args[])
 {
 int a=5,b=7;
 System.out.println("Before swapping: a= "+a +" & b= "+b);

 swap(a, b); //A method call statement

 System.out.println("After swapping: a= "+a +" & b= "+b);
 }
}
```

**Formal parameters**

**OUTPUT**

Before swapping: a = 5 & b = 7  
Inside method: x = 7 & y = 5  
After swapping: a = 5 & b = 7

Values of a and b are copied to x and y

**Actual parameters**

In the above example, the values of the actual parameters 'a' and 'b' before the method call are 5 and 7. When the swap( ) method is invoked, these values get copied to the formal parameters 'x' and 'y', and change to 7 and 5, respectively on executing the instructions written inside the method. However, the values of the actual parameters 'a' and 'b' remain the same even after the method call, i.e., 5 and 7, as shown in the output.

Consider another example to get a better understanding of the concept of call by value.

### Example 2

```
public class cvalue
{
 static void change(int a) //Defining a method
 {
 a = a + 50;
 System.out.println ("On calling method, a is: " + a);
 }

 public static void main(String args[])
 {
 int a = 30;
 System.out.println ("Before calling method, a is: " + a);
 change(a); //A method call statement
 System.out.println("After calling method, a is: " + a);
 }
}
```

**OUTPUT**

Before calling method, a is: 30  
On calling method, a is: 80  
After calling method, a is: 30

In the above example, the value of the actual parameter 'a' before the method call is 30. When the change( ) method is invoked, this value gets copied to the formal parameter 'a' and changes to 80 on executing the instructions written inside the method. However, the value of the actual parameter 'a' remains the same even after the method call, i.e., 30, as shown in the output.

Thus, the key advantage of 'call by value' method is that you cannot change the variables used to call the methods because any change that occurs inside the method is on the method's copy of the argument values. The original copy of the argument values remains intact.

## Pass by Reference

In pass by reference, the called method receives the reference (address of actual parameters) of the passed parameters, and through this reference, it accesses the original data. Any changes that take place are reflected in the original data also. You can pass the non-primitive data types, such as **objects, arrays, classes, strings, interfaces, and enum** as references in this method.

Notice a few points while passing the objects as references to the methods:

- ✦ If you make changes to the called method, the original value is altered in the case of pass by reference.
- ✦ You must pass the class object in place of the actual parameters to pass the references because as the formal parameters of a class object type have the same references to each other. As a result, any modifications will be reflected in both formal and actual objects.

Consider the following example to understand the concept of call by reference.

### Example 1

```
public class Cref
{
 static int a;
 static void update(Cref cf) //Defining a method
 {
 Cref.a +=10;
 }
 public static void main(String args[])
 {
 Cref Ob=new Cref();
 Ob.a=30;
 System.out.println("Before function calling: "+Ob.a);
 Ob.update(Ob); //A method call statement
 System.out.println("After function calling: "+Ob.a);
 }
}
```

#### OUTPUT

Before function calling: 30  
After function calling: 40

Formal parameters

Object reference is passed

Actual parameter

In the above example, the address of the actual parameter 'Ob' gets passed to the formal parameter 'cf' when the Cref( ) method gets invoked. This method now works with the original data, and the change in the value of the formal parameter is reflected in the value of the actual parameter, as shown in the output.



Consider another example to get a better understanding of the concept of call by reference.

### Example 2

```
public class pass3
{
 int num=20;
 static void update(pass3 ob) //Defining a method
 {
 ob.num=ob.num+20;
 }
 public static void main(String args[])
 {
 pass3 ob=new pass3();
 System.out.println("Before change: "+ob.num);
 ob.update(ob); //A method call statement
 System.out.println("After change: "+ob.num);
 }
}
```

#### OUTPUT

Before change: 20  
After change : 40

In the above example, the address of the actual parameter 'ob' gets passed to the formal parameter 'ob' when the update( ) method gets invoked. This method now works with the original data, and the change in the value of the formal parameter is reflected in the value of the actual parameter, as shown in the output.

Thus, changes are reflected back to their original values in call by reference method. This is useful when the values of the original variables need to be changed using a method.

## 4.5 Using the return Statement in a Method

The return statement is used to return a value from a method or a constructor. It causes the program control to transfer back to the caller of the method. The return statement immediately terminates the method in which it is executed.

### Syntax:

```
return(expression);
```

You can use the return statement only inside the method body. The return value's data type must be the same as the type given in the method header.

The following example demonstrates the use of return statement in a method:

### Program 1

Write a program to find the square of a given number using the return statement in the method.

```
1 public class number
2 {
3 int sq(int a)
4 {
```

```

5 int s;
6 s = a*a;
7 return (s);
8 }
9 public static void main(String args[])
10 {
11 number n = new number();
12 int x;
13 x=n.sq(5);
14 System.out.println("Square of 5 is: " +x);
15 }
16 }

```

## 4.6 Programs Based on Methods

Consider the following programs to familiarise yourself with the use of methods. These programs demonstrate how to write a method definition, pass arguments in a method call, and return a value to method.

### Program 2

Write a program to input the marks of a student. Pass it to a method, which returns the grades as per the details given below:

| Marks | Grade |
|-------|-------|
| 90    | A     |
| 75-89 | B     |
| 60-74 | C     |
| 40-59 | D     |
| <40   | E     |

```

1 import java.util.*;
2 public class grade
3 {
4 char findgrade(int m)
5 {
6 char g;
7 if(m>=90)
8 g='A';
9 else if(m>=75 && m<90)
10 g='B';
11 else if(m>=60 && m<75)
12 g='C';
13 else if(m>=40 && m<60)
14 g='D';
15 else

```

```

16 g='E';
17 return(g);
18 }
19 public static void main(String args[])
20 {
21 grade ob=new grade();
22 int m1;
22 Scanner input=new Scanner(System.in);
23 System.out.println("Enter marks:");
24 m1=input.nextInt();
25 char x;
26 x=ob.findgrade(m1);
27 System.out.println("Grade="+x);
28 }
29 }

```

### Program 3

Write a program to input two numbers and return their sum.

```

1 class number
2 {
3 public int sum(int a,int b)
4 {
5 int c;
6 c=a+b;
7 return(c);
8 }
9 public static void main(String args[])
10 {
11 number ob=new number();
12 int x;
13 x=ob.sum(450, 500);
14 System.out.println("Sum="+x);
15 }
16 }

```

#### OUTPUT

Sum = 950

### Program 4

Write a program to input the principal amount, rate of interest, and the time period. Pass these values to a method to calculate the simple interest and then return the final amount.

```

1 import java.util.*;
2 public class money
3 {

```

| 4<br>5<br>6<br>7<br>8<br>9<br>10<br>11<br>12<br>13<br>14<br>15<br>16<br>17<br>18<br>19<br>20<br>21<br>22<br>23<br>24 | <pre> double simple(int p,int t,int r) {     double si,a;     si=(p*t*r)/100;     a=p+si;     return(a); }  public static void main(String args[]) {     money ob=new money();     int p1,t1,r1;     Scanner input=new Scanner(System.in);     System.out.println("Enter the principal, time, and rate of interest:");     p1=input.nextInt();     t1=input.nextInt();     r1=input.nextInt();     double x;     x=ob.simple(p1,t1,r1);     System.out.println("Amount="+x); } </pre> | <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr style="background-color: #8e44ad; color: white;"> <th style="padding: 5px;">OUTPUT</th> </tr> </thead> <tbody> <tr> <td style="padding: 5px;">Enter the principal, time, and rate of interest: 10000</td> </tr> <tr> <td style="padding: 5px;">5</td> </tr> <tr> <td style="padding: 5px;">12</td> </tr> <tr> <td style="padding: 5px;">Amount=16000</td> </tr> </tbody> </table> | OUTPUT | Enter the principal, time, and rate of interest: 10000 | 5 | 12 | Amount=16000 |
|----------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|--------------------------------------------------------|---|----|--------------|
| OUTPUT                                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |        |                                                        |   |    |              |
| Enter the principal, time, and rate of interest: 10000                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |        |                                                        |   |    |              |
| 5                                                                                                                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |        |                                                        |   |    |              |
| 12                                                                                                                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |        |                                                        |   |    |              |
| Amount=16000                                                                                                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |        |                                                        |   |    |              |

## 4.7 Pure and Impure Methods

During program execution in Java, the methods may or may not change the state of an object. Based on this, the methods could be classified as pure and impure methods.

### Pure Methods

These methods return a value to their caller methods but do not modify the state of the objects.

A method is considered pure if:

- ✦ It does not change the original state of an object.
- ✦ It always gives the same result when the same arguments are passed.

For example, the in-built method `sin(0)` will always return 0 as a result. So it is a pure method. On the other hand, the `random()` method is an impure method, as it will give different outputs for the same function call.

Consider the following example to understand the concept of pure method:

#### Example

```

public class pure
{
 static int sum(int n1, int n2)
 {
 return n1 + n2;
 }
}

```

| OUTPUT  |
|---------|
| Sum = 5 |

```

public static void main(String args[])
{
 sum(2,3);
 System.out.println("Sum = "+sum(2,3));
}
}

```

In the above-given example, the resultant value of the sum( ) method depends only on the values of 'n1' and 'n2'. If you call the method sum(2,3), the output will always be 5. That is why, this function is referred to as a pure function.

## Impure Methods

Impure methods are those that can cause a change of state in the object. The values of the object's instance variables get modified or changed depending on the current state of the object on which the method operates. Such methods can cause some unwanted side effects if not carefully controlled by the access-modifier.

Impure method may or may not return a value but it definitely changes the state of the object arguments it has received.

A method is considered impure if:

- ✦ It changes the original state of an object.
- ✦ It does not return the same value every time it is called, even when the same arguments are given.

Consider the following example to understand the concept of impure method:

### Example

```

double trans(double bal, double wd)
{
 if(wd>bal)
 System.out.println("Sorry insufficient balance");
 else
 {
 bal=bal-wd;
 System.out.println("Withdrawal successful");
 }
 return(bal);
}

```

In this example, trans(double, double) is an impure method, because it causes a change in the object's state by changing the value of 'bal' variable.

## Difference between Pure and Impure Methods

The following table summarises the differences between pure and impure methods:

| Pure Method                                                               | Impure Method                                                            |
|---------------------------------------------------------------------------|--------------------------------------------------------------------------|
| This method never changes the state of the program.                       | This method may change the state of the program.                         |
| This method always returns the same result when given the same arguments. | This method may return a different result when given the same arguments. |
| It does not have any side effects.                                        | It may have side effects.                                                |

### 4.8 Static and Non-Static Methods

Based on the memory allocation to the variables, the functions or methods can be classified into static and non-static. Let us discuss them one-by-one.

#### Static Methods

A **static** method is a method that belongs to a class but does not belong to any of its instances. It can be called without the use of an instance or object of that class. A static method is declared by using the keyword **static** in its prototype. The main advantage of a static method is that it can be called without creating an object of the class in which it is declared. It can access the static data members of the class and also change their values. The best example of a static method is the **main( )** method.

##### Let's Know More

The `main( )` method is the first method that is called in a Java program. It is invoked in the system, by default. The keyword `static` makes the JVM call the `main( )` method without the need of creating an object.

Consider the following example understand how to create the static method:

#### Example

```
public class Display
{
 public static void main(String args[])
 {
 show();
 }
 static void show()
 {
 System.out.println("Hello Java!");
 }
}
```

##### OUTPUT

Hello Java!

In the above example, you can observe that the method `show( )` is called without creating any object of the class `Display`.

Now, note some important points regarding static methods:

- ✦ In static methods, memory allocation happens only once, because the `static` keyword fixes a particular memory location for that method in RAM. So, every time the method is called in a program, that particular memory is used. As a result, less memory is allocated.
- ✦ In the static method, the method uses compile time or early binding. For this reason, you can access static method without creating an instance of the class.

## Non-static Methods

Any method defined inside a class is known as a **non-static** method. It is also called an **instance** method. Before calling or invoking the instance method, it is necessary to create an object of its class. Every method in Java declared without the keyword 'static', by default, is a non-static method. Non-static methods cannot access any static methods or any static variables without creating an instance or object of the class.

Consider the following example to understand how to create the non-static method:

### Example

```
public class disp
{
 public static void main(String args[])
 {
 disp ob = new disp();
 System.out.println("The sum is: "+ob.add(23, 35));
 }
 int s;
 public int add(int a, int b)
 {
 s = a+b;
 return s;
 }
}
```

#### OUTPUT

The sum is: 58

In the above example, the method add( ) is a non-static method as it is called using the object 'ob' of the class 'disp'.

Now, note some important points regarding non-static methods:

- ✦ In non-static methods, the memory is not fixed in the RAM. So, the memory is allocated every time the method is called. So, memory is used more in non-static methods than the static methods.
- ✦ The non-static methods use runtime or dynamic binding. That is why, you cannot access a non-static method without creating an instance of the class.

## Difference between Static and Non-static Methods

| Static Methods                                                                       | Non-static Methods                                                                                                                                                 |
|--------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| A static method can access only static members and cannot access non-static members. | A non-static method can access both static as well as non-static members.                                                                                          |
| A static method uses compile time binding or early binding.                          | A non-static method uses run time binding or dynamic binding.                                                                                                      |
| A static method occupies less space, and memory allocation happens only once.        | A non-static method may occupy more space. Memory allocation happens when the method is invoked and it is deallocated after the execution of the method completes. |
| A static method is declared using the static keyword.                                | A normal method is not required to have any special keyword.                                                                                                       |

## 4.9 Method Overloading (Polymorphism)

It is the concept of using the same method name multiple times with different sets of parameters. Method overloading is the process of defining two or more methods in a class with the same name but distinct signatures. It is the process by which Java implements polymorphism.

### Example

```
void area(int l, int b)
void area(int s)
```

In the above example, two methods are given the same name and same return type, but different parameters with different data types.

If you call the `area()` method to calculate the area of a rectangle with two `int` type parameters, then the methods **`area(int l, int b)`** will be invoked. On the other hand, if you call the `area()` method to calculate the area of a square with one `int` type parameter, then the method **`area(int s)`** will be invoked.

Hence, the method `area()` is said to be overloaded. Consider the following program to understand the functionality of method overloading.

### Program 5

Write a program using function overloading to find the area of a rectangle, square, and a circle.

```
1 public class overload
2 {
3 void area(int l,int b)
4 {
5 int a=l*b;
6 System.out.println("Area of rectangle="+a);
7 }
8 void area(int s)
9 {
10 int a1=s*s;
11 System.out.println("Area of square="+a1);
12 }
13 void area(float r)
14 {
15 float a2=3.14f*r*r;
16 System.out.println("Area of circle="+a2);
17 }
18 public static void main(String args[])
19 {
20 overload ob=new overload();
21 ob.area(3,5);
```

### OUTPUT

```
Area of rectangle = 15
Area of square = 36
Area of circle = 98.4704
```



```

22 ob.area(6) ;
23 ob.area(5.6f);
24 }
25 }

```

## Need for Method Overloading

In Program 5, you have observed that there is only one method, `area()` with different parameters. If method overloading would not have been used here, then three different methods would have to be used as follows:

```

void areaRectangle(int l, int b)
void areaSquare(int s)
void areaCircle(float r)

```

Hence, it is reliable to use many functions with the same name for similar operations.

## Advantages of Method Overloading

There are many advantages of using method overloading in a program. Some of them are listed below:

- ✦ It not only implements polymorphism but also reduces the number of comparisons in a program.
- ✦ It increases the readability of the program as the code looks clean.
- ✦ It makes the programs run faster.
- ✦ It gives programmers more freedom because they can use the same name for different methods.

## Determining Method Overloading

When a method is declared many times in a class, Java uses the techniques given below as a guide to determine overloading:

- ✦ If a method's signature matches that of a prior method, that method is considered a redeclaration of the first method.
- ✦ The second declaration is marked as a syntax error if the signatures of the two methods match but the return types differ.
- ✦ If the type of parameters, the number of parameters, or both are different in the signatures of two methods, the two methods are deemed overloaded.

### Program 6

Write a program to perform the following functions to demonstrate the concept of function overloading.

**void check\_num():** It accepts two integers and a character as input and checks whether both numbers are even if the character is 'E', otherwise it checks whether the sum is prime or not.

**void check\_num():** It accepts two characters and an integer and finds the sum of the ASCII codes of the letters if the integer is even, otherwise it finds the word formed by combining both letters.

**void check\_num():** It accepts three integers and a character. If the character is 'A' then it finds the average of the integers, otherwise it finds the biggest of the three numbers using the ternary operator.

```

1 public class overload
2 {
3 void check_num(int a, int b, char c)
4 {
5 int s,i,f=0;
6 if (c=='E')
7 {
8 if (a%2==0 && b%2==0)
9 System.out.println("Both are even.");
10 else
11 System.out.println("Both are not even.");
12 }
13 else
14 {
15 s=a+b;
16 for(i=1;i<=s;i++)
17 {
18 if(s%i==0)
19 f++;
20 }
21 if(f==2)
22 System.out.println("Sum is prime.");
23 else
24 System.out.println("Sum is not prime.");
25 }
26 }
27 void check_num(char a,char b,int x)
28 {
29 if(x%2==0)
30 System.out.println((int)a+(int)b);
31 else
32 System.out.println(a+b);
33 }
34 void check_num(int a, int b, int c, char x)
35 {
36 double d;
37 if(x=='a')
38 {
39 d=(a+b+c)/3.0;
40 System.out.println("Average="+d);
41 }

```

## OUTPUT

Sum is not prime.  
142  
biggest number=9

```

42 else
43 {
44 int max=(a > b) ? (a > c ? a : c) : (b > c ? b : c);
45 System.out.println("biggest number="+max);
46 }
47 }
48 public static void main(String args[])
49 {
50 overload ob=new overload();
51 ob.check_num(5,9, 'c');
52 ob.check_num('A', 'M',5);
53 ob.check_num(5,9,3, 'c');
54 }
55 }

```

### Program 7

Design a class to overload a function polygon( ) to do the following tasks:

**void polygon(int n,char ch):** To draw a filled square of side 'n' using a character.

**void polygon(int x,int y):** To draw a filled rectangle of length 'x' and breadth 'y', using the symbol '@'.

**void polygon( ):** To draw a right angled triangle as shown.

\*  
\*\*  
\*\*\*  
\*\*\*\*  
\*\*\*\*\*

```

1 public class overload
2 {
3 int i,j;
4 void polygon(int n,char ch)
5 {
6 for(i=1;i<=n;i++)
7 {
8 for(j=1;j<=n;j++)
9 {
10 System.out.print(ch);
11 }
12 System.out.println();
13 }
14 }
15 void polygon(int x,int y)
16 {
17 for(i=1;i<=x;i++)
18 {

```

### OUTPUT

```

AAAA
AAAA
AAAA
AAAA
@@@@@
@@@@@
@@@@@
*
**


```

```

19 for(j=1;j<=y;j++)
20 {
21 System.out.print('@');
22 }
23 System.out.println();
24 }
25 }
26 void polygon()
27 {
28 for(i=1;i<=5;i++)
29 {
30 for(j=1;j<=i;j++)
31 {
32 System.out.print('*');
33 }
34 System.out.println();
35 }
36 }
37 public static void main(String args[])
38 {
39 overload ob=new overload();
40 ob.polygon(4,'A');
41 ob.polygon(3,5) ;
42 ob.polygon();
43 }
44 }

```



## Let's Revise

- ◆ The process of splitting the lengthier and complex programs into a number of smaller units is called modularisation.
- ◆ A method is a set of instructions to carry out a particular task.
- ◆ A list of variables in a method is referred to as parameters.
- ◆ The variables used in the method reference or method call are called actual parameters or arguments.
- ◆ The variables used in a method definition are referred to as formal parameters.
- ◆ Pass by value is the process of passing a copy of actual arguments to the formal parameters. Any changes made in the formal parameters do not reflect on the actual parameters.
- ◆ Pass by reference is the process of passing the reference (address) of actual arguments to the formal parameters, and any change made in the formal parameters will be reflected in the actual parameters.

- ◆ A method that returns a value to its caller method but does not modify the state of an object is called a pure method.
- ◆ A method that changes the state of an objects is called an impure method.
- ◆ A method that belongs to a class but does not belong to any of its instances is called a static method. It is declared with the static keyword.
- ◆ Any method defined inside a class is known as a non-static method. It is also called the 'instance method'. Before calling the non-static method, it is necessary to create an object of its class.
- ◆ Method overloading is the concept of using the same method name at multiple times with different sets of parameters.

## Key Terms

**Method Call or Caller Method:** A method does not get executed until you invoke or call it. The method can be invoked or called by a method call statement.

**Method Prototype:** It is the first line of the method, which contains the method name, return type, and the parameter list.

**Method Signature:** It refers to the number of parameters and their types. It is the part of the method prototype.

**Access Specifier:** It defines the accessibility of the method. It could be public, private, or protected.

**Return Type:** It specifies the type of data that a method returns. The return type of a method is any of the primitive data types.

**Pass by Value:** It means passing a copy of actual arguments to the formal parameters.

**Pass by Reference:** It means passing the reference (address) of actual arguments to the formal parameters.



## SECTION 1 (Solved Questions)

### A. Fill in the blanks.

1. A method can return only \_\_\_\_\_ value to its caller method.
2. A method returns a \_\_\_\_\_ type of data.
3. A method with a \_\_\_\_\_ return type does not return any value.
4. Parameters that appear in the method definition are called the \_\_\_\_\_ parameters.
5. \_\_\_\_\_ occurs when the same method name is used several times in a program with distinct sets of parameters.
6. The method that changes the state of an object is called \_\_\_\_\_.
7. The process of passing a copy of the actual arguments to the formal parameters is called \_\_\_\_\_.
8. A method that has a \_\_\_\_\_ keyword can be used from anywhere, like within the class, outside the class, within the package, and outside the package.

**Ans.**

1. One

2. Primitive

3. void

4. Formal

5. Method overloading

6. Impure method

7. Pass by value

8. public

## B. Very short answer type questions.

1. What is a method?

Ans. A method is a set of instructions to carry out a particular task.

2. Write the prototype of a method PARAM which returns Boolean type value.

Ans. `boolean PARAM()`

3. Write the prototype of a method root that accepts three integers and returns one floating type data.

Ans. `float root(int x, int y, int z)`

4. What do you understand by the return type of a method?

Ans. The return type specifies the type of data that will be returned when the method is called. It should be any of the primitive data types.

5. What are impure methods?

Ans. A method that changes the state of an object is called an impure method.

6. What do you understand by the scope of a method?

Ans. The scope of a method means how the different parts of the program can access that method.

7. What is the use of the return statement?

Ans. The return statement is used to return a value from a method. It is given inside the method body. The return value's data type must be the same as the type given in the method prototype.

8. What do you mean by 'invoking a method' or 'calling a method'?

Ans. Invoking a method or calling a method means using a method in a program to execute the instructions written inside it.

9. Differentiate between public and private modifiers.

Ans. A public modifier makes the class member methods accessible both within and outside their class, whereas a private modifier makes the class member methods accessible only within the class in which they are declared.

10. Describe the class methods.

Ans. The methods that are declared with the static keyword are called class methods.

## C. Short answer type questions.

1. What are the two ways of invoking a function?

Ans. The following are the two ways of invoking a function:

- ✦ Pass by value
- ✦ Pass by reference

2. When do you declare a member method static?

Ans. When a method needs to be called without associating it with an object, it is declared static method.

3. Why is the main method defined as a static method?

Ans. The main() method is the first method that is called in a Java program. The keyword static makes the JVM call the main method without the need of creating an object.

4. Mention the reason for using pass by reference.

**Ans:** In the 'pass by reference' method, changes are reflected back to their original values. This is useful when the values of the original variables need to be changed using a method.

**D. Application based questions.**

1. Write the output of the following function, if 12 is passed to 'n'

```
void test (int n)
{
 for (int x=1;x<=n;x++)
 {
 if (n%x==0)
 System.out.println(x);
 }
}
```

**Ans.** 1  
2  
3  
4  
6  
12

2. Write the output of the following function, if the values passed to the function are 4 and 17.

```
void test2 (int a, int b)
{
 while(a!=b)
 {
 if(a>b)
 a=a-b;
 else
 b=b-a;
 }
 System.out.println(a);
}
```

**Ans.** 1

3. Find errors in the following program segment and rewrite it after correcting all the errors.

```
Double trouble (int a, int b)
{
 a*b=c;
 b*c=d;
 return c,d
}
```

**Ans.** The corrected code is:

```
double trouble (int a, int b)
{
 int c=a*b;
 int d=b*c;
 return d;
}
```

4. Write the output of the following program.

```
class op
{
 boolean check()
 {
 return(75>65);
 }
 public static void main()
 {
 op x=new op();
 boolean t=x.check();
 System.out.println(t);
 }
}
```

Ans. true

## SECTION 2 (Unsolved Questions)

### A. Very short answer type questions.

1. Write the prototype of a function and check whether it returns true or false when given a character ch and an integer n.
2. Describe all the access specifiers that you use with methods.
3. Distinguish between static and non-static methods.
4. Give an example of method overloading.
5. Distinguish between actual and formal parameters.

### B. Short answer type questions.

1. Differentiate between pure and impure methods. Give a suitable example.
2. Describe the static methods.
3. What are non-static methods? Give an example.
4. What is method overloading? Give an example.
5. What is a method prototype? Explain. How is it different from the method signature?

### C. Application based questions.

1. Find errors in the following code segments and rewrite them after correcting all the errors.

```
a. static sum(void)
{
 int x; y;
 x=2;
 y=+x;
 system.out.println(x+" "+y);
}

b. public static void main(String args[])
{
```



```

 int x,y;
 y=sum(x);
 system.out.println(y);
 }
c. class 12design
{
 void sum1
 {
 int a;b;c;
 5=c;
 b=3(4-2);
 a=b+c
 system.out.println('sum = ' + a;
 }
}

```

2. Consider the following code segment and answer the following questions.

```

class Myclass
{
 short roll_no;
 long ssn;
 String stu_name;
}

```

- Define a method called accept( ) to assign values to the variables, roll\_no, ssn, and stu\_name as 1, 8765122, and Vivek, respectively.
- Define a method display( ) that displays roll\_no, ssn, and stu\_name.
- Execute all the created methods under the main( ) method.

3. Write the output of the following code snippets:

```

a. class OUT1
{
 boolean var;
 void output()
 {
 var=true;
 }
 public static void main(String args[])
 {
 OUT1 ob=new OUT1();
 System.out.println(ob.var);
 }
}

```

```

b. class OUT2
{
 int func(int x, int y)

```

```

{
 if (x%y == 0)
 return ++x;
 else
 return y--;
}

public static void main(String args[])
{
 int p=20, q=23;
 q=func (p,q);
 System.out.println(p+" "+q);
 p=func (q,p+3);
 System.out.println(p+" "+q);
}
}

```

c. class OUT3

```

{
 static public int fun(int x, int y)
 {
 y -= x;
 return y;
 }

 public static void main(String args[])
 {
 int a = 3, b = 7;
 b = fun(a, b);
 a = fun(b, a);
 System.out.println(a + " " + b);
 }
}

```

4. Write a function int Max(int,int) to accept two integers and return the largest integer.
5. Write a menu-driven program that uses method overloading to perform the following tasks:
  - a. void show(int N): To print all the odd numbers from 1 to N.
  - b. void show(int M, int N): To print all the even numbers from M to N.
  - c. void show(int M, int N, int P): To print all the numbers from M to N that are completely divisible by P.
6. Write a program that uses method overloading to do the following tasks:
  - a. void display (int n): To print n number of "\*"
  - b. void display(int m, int n): To print a rectangle filled up with "\*"
  - c. void display(char ch): To print the given character in 4 x 5 matrix form.

### SECTION 3 (Board Questions)

1. Write two advantages of using methods in a program.

[2010]

Ans. ♦ Methods help to manage the complexity of the program by dividing a bigger or complex task into smaller tasks.

♦ Methods help with code reusability.

2. Differentiate between static and non-static data members.

[2010]

| Static Data Members                                                | Non-Static Data Members                                                |
|--------------------------------------------------------------------|------------------------------------------------------------------------|
| They are declared using the 'static' keyword.                      | They are not declared with the 'static' keyword.                       |
| All objects of a class share the same copy of static data members. | Each object of the class gets its own copy of non-static data members. |
| They can be accessed using the class name or object.               | They can be accessed only through an object of the class.              |

3. What is the role of keyword 'void' in declaring methods?

[2011]

Ans. The keyword 'void' signifies that the method doesn't return a value to the calling method.

4. Give the prototype of a function 'search' that receives a sentence 'sent' and word 'w', and returns 1 or 0.

[2011]

Ans. `int search(String sent, String w)`

5. Differentiate between call by value and call by reference.

[2012]

| Call by Value                                                                  | Call by Reference                                                          |
|--------------------------------------------------------------------------------|----------------------------------------------------------------------------|
| Values of actual parameters are copied to formal parameters.                   | Reference of actual parameters is passed to formal parameters.             |
| Changes made to formal parameters are not reflected back to actual parameters. | Changes made to formal parameters are reflected back to actual parameters. |

6. Name the Java keyword that:

[2013]

- indicates that a method has no return type.
- makes the method as a class method.

Ans. a. void

b. static

7. The access specifier that gives the most accessibility is \_\_\_\_\_ and the least accessibility is \_\_\_\_\_.

[2015]

Ans. The access specifier that gives the most accessibility is public and the least accessibility is private.

8. Write a function prototype of the following:

[2016]

A function 'PosChar' which takes a string argument and a character argument and returns an integer value.

Ans. `int PosChar(String s, char ch)`

9. Name any two types of access specifiers.

[2016]

Ans. public and private

10. What are the two ways of invoking functions?

[2017]

Ans. The two ways of invoking functions are call by value and call by reference.

## Case Study I

Methods are the lines of code that perform a specific function in a program. They can either return a value or not return anything. The methods that do not return anything are of type void. The main advantage of methods in a program is code reusability. Applications around the world are built to solve problems. However, while building an application, there is absolutely no need to program each and every part of it. This is where methods come into play. If you need to implement a particular function in your application that is already programmed by someone else, you can directly implement that method in your application without worrying about that function at all. Methods simplify programs by making them easy to debug. It also enhances code readability and reusability.

- \_\_\_\_\_ are part of a program used to perform a particular task.
  - Methods
  - Class
  - Abstraction
  - static
- A method \_\_\_\_\_ return a value to its caller method.
  - must
  - always
  - may or may not
  - never
- You can execute a method through a/an \_\_\_\_\_.
  - Object
  - Method prototype
  - Method signature
  - Method call
- \_\_\_\_\_ parameters are passed in a method call.
  - Formal
  - Actual
  - Instance
  - Static

|      |      |      |      |      |
|------|------|------|------|------|
| Ans. | 1. a | 2. c | 3. d | 4. b |
|------|------|------|------|------|

## Case Study II

Overloaded methods are those methods that have the same name but different parameters. These methods make the program readable. This enables the programmer to name several methods with the same names, which leads to less confusion. However, if each method needs to have a different definition, only the arguments get changed, not the method name itself. You can easily define another method with the same name but with different arguments.

- A same method with many definitions are referred to as \_\_\_\_\_.
  - Method prototype
  - Method call
  - Method overloading
  - Main method
- If more than one functions exist, then they are identified by its \_\_\_\_\_.
  - Method name
  - Method signature
  - Data types
  - Variable names
- The concept of OOP which manages multiple functions with same name is called \_\_\_\_\_.
  - Abstraction
  - Multitasking
  - Encapsulation
  - Polymorphism
- You can use \_\_\_\_\_ overloaded methods in a Java program.
  - One
  - Two
  - Three
  - Any number of

|      |      |      |      |      |
|------|------|------|------|------|
| Ans. | 1. c | 2. b | 3. d | 4. d |
|------|------|------|------|------|

1. Write a main method to create an object of the class 'Library' and call its member methods.

Instance variables/data members:

int accNum: To store the accession number of the book.

String title: To store the title of the book.

String author: To store the name of the author.

Member methods:

void input(): To input and store the accession number, title, and author.

void compute(): To accept the number of days late, calculate, and display the fine charged at the rate of ₹2 per day.

void display(): To display the details in the following format:

| Accession Number | Title | Author |
|------------------|-------|--------|
| .....            | ..... | .....  |

```
Ans. import java.util.*;
class Library
{
 int accNum;
 String title, author;
 void input() {
 Scanner in = new Scanner(System.in);
 System.out.print("Enter book title: ");
 title = in.nextLine();
 System.out.print("Enter author: ");
 author = in.nextLine();
 System.out.print("Enter accession number: ");
 accNum = in.nextInt();
 }
 void compute() {
 Scanner in = new Scanner(System.in);
 System.out.print("Enter number of days late: ");
 int days = in.nextInt();
 int fine = days * 2;
 System.out.println("Fine = Rs." + fine);
 }
 void display() {
 System.out.println("Accession Number\tTitle\tAuthor");
 System.out.println(accNum + "\t\t" + title + "\t" + author);
 }
 public static void main(String args[]) {
 Library ob = new Library();
 ob.input();
 }
}
```

```

 ob.display();
 ob.compute();
 }
}

```

2. Design a class to overload a function area( ) as follows:

- a. double area (double a, double b, double c) with three double arguments, returns the area of a scalene triangle using the formula:

$$\text{area} = \sqrt{s(s-a)(s-b)(s-c)}$$

$$\text{where } s = (a+b+c) / 2$$

- b. double area (int a, int b, int height) with three integer arguments, returns the area of a trapezium using the formula:

$$\text{area} = (1/2)\text{height}(a + b)$$

- c. double area (double diagonal1, double diagonal2) with two double arguments, returns the area of a rhombus using the formula:

$$\text{area} = 1/2(\text{diagonal1} \times \text{diagonal2})$$

Ans. **class** over\_area

```

{
 double area(double a, double b, double c) {
 double s = (a + b + c) / 2;
 double x = s * (s-a) * (s-b) * (s-c);
 double result = Math.sqrt(x);
 return result;
 }
 double area (int a, int b, int height) {
 double result = (1.0 / 2.0) * height * (a + b);
 return result;
 }
 double area (double diagonal1, double diagonal2) {
 double result = 1.0 / 2.0 * diagonal1 * diagonal2;
 return result;
 }
}

```

3. Define a class 'ElectricBill' with the following specifications:

Instance variables / data members:

String n: To store the name of the customer.

int units: To store the number of units consumed.

double bill: To store the amount to be paid.

Member methods:

void accept( ): To accept the name of the customer and number of units consumed.

void calculate( ): To calculate the bill as per the following tariff:

| Number of units | Rate per unit |
|-----------------|---------------|
| First 100 units | ₹2.00         |
| Next 200 units  | ₹3.00         |
| Above 300 units | ₹5.00         |

Note: A surcharge of 2.5% charged if the number of units consumed is above 300 units.

**void print( ):** To print the details as follows:

Name of the customer: .....

Number of units consumed: .....

Bill amount: .....

Write a main method to create an object of the class and call the above member methods.

```
Ans. import java.util.*;
class ElectricBill
{
 String n;
 int units;
 double bill;
 public void accept() {
 Scanner in = new Scanner(System.in);
 System.out.print("Enter customer name: ");
 n = in.nextLine();
 System.out.print("Enter units consumed: ");
 units = in.nextInt();
 }
 public void calculate() {
 if (units <= 100)
 bill = units * 2;
 else if (units <= 300)
 bill = 200 + (units - 100) * 3;
 else {
 double amt = 200 + 600 + (units - 300) * 5;
 double surcharge = (amt * 2.5) / 100.0;
 bill = amt + surcharge;
 }
 }
 public void print() {
 System.out.println("Name of the customer\t\t: " + n);
 System.out.println("Number of units consumed\t: " + units);
 System.out.println("Bill amount\t\t\t: " + bill);
 }
 public static void main(String args[]) {
 ElectricBill ob = new ElectricBill();
 }
}
```

```

 ob.accept();
 ob.calculate();
 ob.print();
 }
}

```

4. Design a class to overload a function check( ) as follows:

- a. void check (String str , char ch ): To find and print the frequency of a character in a string.

Example:

Input:

str = "success"

ch = 's'

Output:

number of s present in str = 3

- b. void check(String s1): To display only vowels from string s1, after converting it to lowercase.

Example:

Input:

s1="computer"

Output : o u e

Ans. class over\_check

```

{
 void check (String str , char ch)
 {
 int count = 0;
 int len = str.length();
 for (int i = 0; i < len; i++)
 {
 char c = str.charAt(i);
 if (ch == c) {
 count++;
 }
 }
 System.out.println("Frequency of " + ch + " = " + count);
 }
 void check(String s1) {
 String s2 = s1.toLowerCase();
 int len = s2.length();
 System.out.println("Vowels:");
 for (int i = 0; i < len; i++)
 {
 char ch = s2.charAt(i);
 if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u')
 System.out.print(ch + " ");
 }
 }
}

```



5. Design a class to overload a function volume( ) as follows:

- a. double volume (double R) with radius (R) as an argument, returns the volume of sphere using the formula.

$$V = \frac{4}{3} \times \frac{22}{7} \times R^3$$

- b. double volume (double H, double R) with height(H) and radius(R) as the arguments, returns the volume of a cylinder using the formula.

$$V = \frac{22}{7} \times R^2 \times H$$

- c. double volume (double L, double B, double H) with length(L), breadth(B) and Height(H) as the arguments, returns the volume of a cuboid using the formula.

$$V = L \times B \times H$$

Ans. class overload

```
{
 double volume(double r)
 {
 return (4 / 3.0) * (22 / 7.0) * r * r * r;
 }
 double volume(double h, double r)
 {
 return (22 / 7.0) * r * r * h;
 }
 double volume(double l, double b, double h)
 {
 return l * b * h;
 }
 public static void main(String args[])
 {
 overload ob = new overload();
 System.out.println("Sphere Volume = " + ob.volume(7));
 System.out.println("Cylinder Volume = " + ob.volume(5, 4.5));
 System.out.println("Cuboid Volume = " + ob.volume(4.5, 7, 3));
 }
}
```



Link

Scan the given QR Code to learn how to call methods.



## Chapter Highlights

- 5.1 Introduction
- 5.2 Characteristics of a Constructor
- 5.3 Difference between Constructors and Methods
- 5.4 Types of Constructors
- 5.5 Use of a Constructor
- 5.6 Constructor Overloading
- 5.7 Programs Based on Constructors

## Learning Outcomes

## The student will be able to:

- know about constructors, its types, and features.
- differentiate between a constructor and a method.
- know how to overload a constructor.
- create programs based on constructors.

In the previous chapter, you learnt about methods that describe the behaviour of an object. In this chapter, you will learn about a special method of every class called a constructor. It is a method of a class that is invoked automatically whenever an object is created. Let us study more about it.

## 5.1 Introduction

A **constructor** is a special method that has the same name as the class name. It has no return type, not even void. It is used to initialise the objects of a class. It is invoked automatically when an object of a class is created. Like other member methods of a class, you can define a constructor as shown below:

## Example

```
public class MyClass
{
 //This is the constructor
 MyClass()
 {
 }
 ...
}
```

Class name

A member method with the same name as the class is called constructor

Constructor is an integral part of the class as it is used to create an instance of the class. It should be defined in the public section of a class, so that its objects can be created in any function.

## Invoking a Constructor

A constructor is automatically invoked when an object is created. The following syntax is used to invoke a constructor.

## Syntax:

```
<class name><object> = new <constructor_name(>>;
```

## Example

```
Employee Rajesh = new Employee();
```

You can also initialise an object using the constructor with default values as shown here:

```
Employee Ramesh = new Employee(123, "Ramesh", 5000);
```

Consider the following program to understand how to define and invoke a constructor.

### Program 1

A program that demonstrates the use of constructor.

```
1 public class number
2 {
3 int a,b;
4 number() //Constructor definition
5 {
6 a=0;
7 b=0;
8 }
9 void assign() //Method definition
10 {
11 a=2;
12 b=3;
13 }
14 void sum() //Method definition
15 {
16 int c=a+b;
17 System.out.println("Sum="+c);
18 }
19 public static void main(String args[])
20 {
21 number ob=new number(); //Invoking constructor
22 ob.assign();
23 ob.sum();
24 }
25 }
```

### OUTPUT

Sum=5

In the above example, whenever you create an object of the class 'number', the number() constructor is called automatically. The constructor assigns a default values of 0 to the member variables 'a' and 'b'.

## 5.2 Characteristics of a Constructor

The constructors have the following characteristics:

- ◆ A constructor must have the same name as the class.
- ◆ It does not have any return type, not even void because the implicit return type of a constructor is the class type itself. If you write the return type for the constructor, it will be treated as an ordinary method.
- ◆ A constructor can be defined with or without any parameters. It can have any number of parameters.
- ◆ A constructor is called automatically whenever an object of the class is created. It does not need to be called individually.
- ◆ A constructor should not be static because it will be invoked each time an object of a class is created.
- ◆ A constructor should not be of a private type, otherwise you will not be able to create an object of a class.
- ◆ Just like methods, constructors can also be overloaded.
- ◆ Constructors cannot be inherited at all.

### Know the Fact

A **destructor** is a special member method that gets invoked automatically when the object's scope ends. The memory allocated to objects by the constructors gets deallocated when a destructor gets invoked automatically. Destructors are not used in Java. Allocation and release of memory are implicitly handled by the garbage collector in Java.

## 5.3 Difference between Constructors and Methods

A constructor of a class is much like a class method structure-wise, but its functionality is totally different. The following table summarises the differences between a constructor and a method:

| Constructor                                                   | Method                                               |
|---------------------------------------------------------------|------------------------------------------------------|
| Name of the constructor should be the same as the class name. | You can use any name for a method while defining it. |
| It has no return type, not even void.                         | It can have any valid return type.                   |
| It is automatically called when an object is created.         | It executes only when it is called.                  |
| It creates an instance of the class.                          | It groups Java statements.                           |
| Constructors are always declared in the public section.       | Methods may be private, public, or protected.        |

## 5.4 Types of Constructors

There are following types of constructors:

- ◆ Default and non-parameterised constructors
- ◆ Parameterised constructors

### Default Constructors

A constructor that has no parameters is referred to as a **default** and **non-parameterised** constructor. In program 1, **number( )** is a default constructor. If you do not explicitly declare a constructor for a class, the Java compiler provides a default constructor for that class. The default constructor initialises the variables of a class with their default values, such as 0.0 for 'float and double', false for 'boolean', and 0 for 'byte, short, int, long' and null for 'String'. Look at the example to understand the working of a default constructor.

## Example

```
public class default_const
{
 default_const()
 {
 System.out.println("Default is created");
 }
 public static void main(String args[])
 {
 default_const ob = new default_const();
 }
}
```

In the above-given example, the default constructor is: **default\_const()**. It will be called the object of the class '**default\_const**' is created. So, you will get the output as: 'Default is created'.

Consider the following program to become familiar with the use of default constructor.

### Let's Know More

The **new** keyword is used to create an object of a class. This is mostly followed by the constructor to initialise the new object.

## Program 2

The basic salaries of the employees of 'ABC' organisation is undergoing a revision. Write a program to define a class called 'Upgrade' with the following specifications. Use a default constructor to initialise the data members and invoke the methods in the main( ) method.

### Data members:

String name: To store the name of the employee

int bas: To store the basic salary

int expn: To consider the length of service as an experience

double inc: To store the increment

double nbas: To store the new basic salary (basic + increment)

### Member methods:

Upgrade( ): A constructor to initialise all data members

void accept( ): To input name, basic, and experience

void display( ): To print all the details of an employee

void increment( ): To calculate the increment based on the experience as per the table given:

| Experience                         | Increment             |
|------------------------------------|-----------------------|
| Up to 3 years                      | ₹1,000 + 10% of basic |
| 3 years or more and up to 5 years  | ₹3,000 + 12% of basic |
| 5 years or more and up to 10 years | ₹5,000 + 15% of basic |
| 10 years or more                   | ₹8,000 + 20% of basic |

```

1 import java.util.*;
2 public class Upgrade
3 {
4 String name;
5 int bas, expn;
6 double inc, nbas;
7 Upgrade() // Default constructor
8 {
9 name = "";
10 bas = 0;
11 expn = 0;
12 inc = 0.0;
13 nbas = 0.0;
14 }
15 void accept() // Member method
16 {
17 Scanner input = new Scanner(System.in);
18 System.out.print("Enter name: ");
19 name = input.nextLine();
20 System.out.print("Enter basic: ");
21 bas = input.nextInt();
22 System.out.print("Enter experience: ");
23 expn = input.nextInt();
24 }
25 public void increment() // Member method
26 {
27 if (expn <= 3)
28 inc = 1000 + (bas * 0.1);
29 else if (expn <= 5)
30 inc = 3000 + (bas * 0.12);
31 else if (expn <= 10)
32 inc = 5000 + (bas * 0.15);
33 else
34 inc = 8000 + (bas * 0.2);
35
36 nbas = bas + inc;
37 }
38 void display() // Member method
39 {
40 System.out.println("Name: " + name);
41 System.out.println("Basic: " + bas);

```

## OUTPUT

```

Enter name: Atul
Enter basic: 20000
Enter experience: 22
Increment: 12000.0
New Basic: 32000.0

```

```

42 System.out.println("Experience: " + expn);
43 System.out.println("Increment: " + inc);
44 System.out.println("New Basic: " + nbas);
45 }
46 public static void main(String args[])
47 {
48 Upgrade ob = new Upgrade();/* Invoking the default constructor while
 creating instance of a class*/
49 ob.accept();
50 ob.increment();
51 ob.display();
52 }
53 }

```

## Parameterised Constructors

A constructor with the parameters is known as a parameterised constructor. It is used when you need to set distinct values for the various data elements of different objects. However, you can provide the same values as well. The values of the parameters are passed at the time of the creation of an object. Consider the following example to understand about the parameterised constructors:

### Example

```

public class param_const
{
 int m;
 String nm;
 param_const(int m1, String n1) // Parameters passed to the constructor
 {
 m=m1;
 nm=n1;
 System.out.println(m+" "+nm);
 }
 public static void main(String args[])
 {
 param_const ob = new param_const(99, "ABCD"); /*Values passed at the time
 of object creation*/
 }
}

```

The output of the above-given example is: 99 ABCD. This is because you have passed two parameters to the constructor while invoking it. These two values are assigned to the variables 'm' and 'nm', respectively. In the same way, you can pass distinct values for the various data elements of different objects. To understand this, consider Program 3.

### Program 3

Write a program that displays the roll number and name of two students using a parameterised constructor.

```
1 public class p_const
2 {
3 int roll;
4 String name;
5 p_const(int i,String n)
6 {
7 roll = i;
8 name = n;
9 }
10 void display()
11 {
12 System.out.println(roll+" "+name);
13 }
14 public static void main(String args[])
15 {
16 p_const ob1 = new p_const(123,"Amit");
17 p_const ob2 = new p_const(234,"Ajay");
18 ob1.display();
19 ob2.display();
20 }
21 }
```

#### OUTPUT

```
123 Amit
234 Ajay
```

Passing distinct values  
for the data elements of  
two different objects

Calling methods

In the above example, you have created the constructor 'p\_const' that has two parameters. So, you will pass two parameters whenever you create the object of the class to invoke the parameterised constructor.

You can have any number of parameters in the constructor. The following program demonstrates how you can pass more than two parameters in a constructor.

### Program 4

Write a program to define a class 'Bill' that calculates the telephone bill of a consumer with the following description:

#### Data members:

int bno: Bill number

String name: Name of consumer

int call: Number of calls made in a month

double amt: Bill amount to be paid by the customer

#### Member methods:

Bill(): Constructor to initialise data members with the default initial value

Bill(...): Parameterised constructor to accept billNo, name, and number of calls made

Display( ): To display the details

Calculate( ): To calculate the monthly telephone bill for a consumer as per the table given below:



| Units consumed                                         | Rate         |
|--------------------------------------------------------|--------------|
| First 100 calls                                        | ₹0.60 / call |
| Next 100 calls                                         | ₹0.80 / call |
| Next 100 calls                                         | ₹1.20 / call |
| Above 300 calls                                        | ₹1.50 / call |
| Fixed monthly rental applicable to all consumers: ₹125 |              |

```

1 import java.util.*;
2 public class Bill
3 {
4 int bno, call;
5 String name;
6 double amt;
7 Bill(int b, String n, int c) /*parameterised constructor to accept billno,
 name, and number of calls made*/
8 {
9 bno = b;
10 name = n;
11 call = c;
12 }
13 void calculate()
14 {
15 double charge;
16 if (call <= 100)
17 charge = call * 0.6;
18 else if (call <= 200)
19 charge = 60 + ((call - 100) * 0.8);
20 else if (call <= 300)
21 charge = 60 + 80 + ((call - 200) * 1.2);
22 else
23 charge = 60 + 80 + 120 + ((call - 300) * 1.5);
24 amt = charge + 125;
25 }
26 void display()
27 {
28 System.out.println("Bill No: " + bno);
29 System.out.println("Name: " + name);
30 System.out.println("Calls: " + call);
31 System.out.println("Amount Payable: " + amt);
32 }
33 public static void main(String args[])
34 {
35 Scanner input = new Scanner(System.in);
36 System.out.print("Enter Name: ");
37 String cName = input.nextLine();

```

#### OUTPUT

```

Enter Name: Ram
Enter Bill Number: 2345
Enter Calls: 250
Bill No: 2345
Name: Ram
Calls: 250
Amount Payable: 325.0

```

```

38 System.out.print("Enter Bill Number: ");
39 int bNum = input.nextInt();
40 System.out.print("Enter Calls: ");
41 int nCalls = input.nextInt();
42 Bill ob = new Bill(bNum, cName, nCalls); // Invoking constructor
43 ob.calculate(); //Calling method
44 ob.display(); //Calling method
45 }
46 }

```

In the above-given program, you are passing three parameters while creating an instance of the class to invoke the parameterised constructor. The parameterised constructor then assigns values to the data members.

## 5.5 Use of a Constructor

When you write a program in Java, it becomes tedious to initialise all the variables in a class each time an object is created. You may declare and call a special user-defined method to initialise variables. However, it is generally inconvenient to do so every time you write a new program. In Java, the initialisation process is performed automatically through the use of a constructor. If you do not explicitly initialise the value of data members through constructors, they may contain any ambiguous value, which could result in an incorrect output of the program. So, constructors are required in programs to initialise the variables with meaningful values, as shown the following example:

### Example

```

public class IPL
{
 String batsman;
 int age;
 IPL() //Constructor
 {
 batsman = "Dhoni";
 age = 40;
 }
 public static void main(String args[])
 {
 IPL p1 = new IPL();
 IPL p2 = new IPL();

 System.out.println("Name of batsman 1 =" +p1.batsman);
 System.out.println("Age of batsman 1 =" +p1.age);
 System.out.println("Name of batsman 2 =" +p2.batsman);
 System.out.println("Age of batsman 2 =" +p2.age);
 }
}

```

#### OUTPUT

```

Name of batsman 1 = Dhoni
Age of batsman 1 = 40
Name of batsman 2 = Dhoni
Age of batsman 2 = 40

```

Instances of a class that invokes the constructor

Second object will also have the same values

## 5.6 Constructor Overloading

Just like methods, constructor of a class can also be overloaded. Constructor overloading is the process of defining multiple constructors for the same class. Each constructor should have a unique signature or set of input arguments. In other words, constructor overloading is a technique that allows a single class to have many constructors, each of which has different sets of parameters. Each overloaded constructor in the class is used to perform a specific task.

In general, constructor overloading can be depicted by the following example:

### Example



```
public class overload
{
 overload()
 {
 ...
 }
 overload(String s)
 {
 ...
 }
 overload(int a, int b)
 {
 ...
 }
}
```

### Know the Fact

The `this` keyword is used to refer to the current class instance variable. If there is ambiguity between the instance variables and parameters, the `'this'` keyword resolves this problem. The member methods of all the objects have access to the keyword `'this'`, which points to the object itself.

### NOTE

Each overloaded constructor must have a different signature. This is because the Java compiler uses the parameter lists, parameter types, and number of input parameters to identify overloaded constructors. If any two constructors of a class have the same signature, it will lead to uncertainty. The Java compiler raises an error in this case.

Consider the following program to understand the concept of constructor overloading:

### Program 5

Write a program to display roll number, name, and age using constructor overloading.

```
1 public class overload
2 {
3 int roll, age;
4 String name;
5 overload(int i, String n) //Constructor with two parameters
6 {
7 roll = i;
```

### OUTPUT

```
121 Ajay 0
242 Vijay 15
```

```

8 name = n;
9 }
10 overload(int i,String n,int a) //Constructor with three parameters
11 {
12 roll = i;
13 name = n;
14 age = a;
15 }
16 void display()
17 {
18 System.out.println(roll+" "+name+" "+age);
19 }
20 public static void main(String args[])
21 {
22 overload ob1 = new overload(121,"Ajay");
23 overload ob2 = new overload(242,"Vijay",15);
24 ob1.display();
25 ob2.display();
26 }
27 }

```

In the above program, the class 'overload' has two different versions of the overload( ) constructor. The first constructor has two parameters, whereas the second constructor has three parameters. On invoking the first constructor, an integer value and a String are assigned to the variables 'roll\_number' and 'name', respectively. When you invoke the second constructors, two integers and a String are assigned to the variables 'roll\_number', 'name', and 'age', respectively.

## 5.7 Programs Based on Constructors

Consider the following programs to get familiar with the use of constructors:

### Program 6

Write a program to define the class 'Rectangle' with the following descriptions. Use the default constructor to initialise the data members and invoke the constructor and methods in the main( ) method.

#### Data Members/Instance variables:

length: To store length of the rectangle

breadth: To store breadth of the rectangle

area: To store area of the rectangle

#### Member Functions:

Rectangle( ): A default constructor to initialise data members

void input( ): To input length and breadth

void findArea( ): To find the area of the rectangle

void display( ): To print length, breadth, and area of the rectangle

```

1 import java.util.*;
2 public class Rectangle
3 {
4 int l, b, a;
5 Rectangle()
6 {
7 l = 0;
8 b = 0;
9 a = 0;
10 }
11 void input()
12 {
13 Scanner input = new Scanner(System.in);
14 System.out.println("Enter length and breadth");
15 l = input.nextInt();
16 b = input.nextInt();
17 }
18 void findarea()
19 {
20 a = l * b;
21 }
22 void display()
23 {
24 System.out.println("Length="+l);
25 System.out.println("Breadth="+b);
26 System.out.println("Area="+a);
27 }
28 public static void main(String args[])
29 {
30 Rectangle Ob=new Rectangle();
31 Ob.input();
32 Ob.findarea();
33 Ob.display();
34 }
35 }

```

#### OUTPUT

```

Length = 30
Breadth = 40
Area = 120

```

#### Program 7

Create a class 'cuboid' with data members length, breadth, and height. Use a parameterised constructor to initialise the object and two functions: surfacearea( ) and volume( ) to calculate the surface area and volume of the cuboid.

```

1 import java.io.*;
2 public class cuboid

```

```

3 {
4 int l,b,h;
5 cuboid(int x,int y,int z) //Parameterised constructor
6 {
7 l = x;
8 b = y;
9 h = z;
10 }
11 void findsurfacearea()
12 {
13 int a = 2 * (l * b + b*h + h*l);
14 System.out.println("Surface area="+a);
15 }
16 void volume()
17 {
18 int v = l * b *h;
19 System.out.println("Volume="+v);
20 }
21 public static void main(String args[])
22 {
23 cuboid ob = new cuboid(5,7,9);
24 ob.findsurfacearea();
25 ob.volume();
26 }
27 }

```

#### OUTPUT

Surface area=286  
Volume=315

### Program 8

Create a class 'compound' to calculate the compound interest using formula:  $CI = P(1+r/100)^t$ , where 'P' is the Principal amount, 'r' is the rate of interest, and 't' is the time period in years.

#### Data members:

pamt, rate (double data type to store principal amount and rate of interest), time (integer to store time period)

#### Member methods:

compound( ): A onstructor to assign default values to all the data members

void input( ): To input the principal, rate, and time from the user

double findInterest( ): To find and return the final amount using the given formula

void printData( ): To print the principal, rate time, and amount

Write the main( ) function to input required data for calculating and printing the compound interest.

```

1 import java.io.*;
2 import java.util.*;
3 public class compound
4 {
5 int time;
6 double pamt,rate,amt;
7 compound()
8 {
9 time = 0;
10 pamt = 0.0;
11 rate = 0.0;
12 amt = 0.0;
13 }
14 void input()
15 {
16 Scanner input=new Scanner(System.in);
17 System.out.println("Enter principal, time, and rate");
18 pamt=input.nextDouble();
19 time=input.nextInt();
20 rate=input.nextDouble();
21 }
22 double findinterest()
23 {
24 amt = pamt * Math.pow((1+rate/100),time);
25 return(amt);
26 }
27 void display()
28 {
29 System.out.println("Principal="+pamt);
30 System.out.println("Time="+time);
31 System.out.println("Rate="+rate);
32 double x=findinterest();
33 System.out.println("Amount="+x);
34 }
35 public static void main(String args[])
36 {
37 compound ob = new compound();
38 ob.input();
39 ob.display();
40 }
41 }

```

## OUTPUT

```

Enter principal, time, and rate:
10000
10
5
Principal=10000.0
Time=10
Rate=5.0
Amount=16288.94626777442

```



## Let's Revise

- ◆ A constructor is used to initialise the objects of a class. It is invoked automatically when an object of a class is created.
- ◆ A constructor must have the same name as the class.
- ◆ There are two types of constructors: default constructor and parametrised constructor.
- ◆ A constructor without parameters is called default constructor or non-parametrised constructor.
- ◆ A constructor that has parameters is called parametrised constructor.
- ◆ You can have any number of parameters in a constructor.
- ◆ Constructor overloading is a technique that allows a single class to have many constructors, each of which has a different sets of parameters.



### SECTION 1 (Solved Questions)

#### A. Fill in the blanks.

1. A constructor has the same name as the \_\_\_\_\_ name.
2. A constructor is used to initialise the instance \_\_\_\_\_ of a class.
3. A constructor can be invoked by the \_\_\_\_\_ operator.
4. A constructor has no \_\_\_\_\_ type.
5. When a class has many constructors of same name with different sets of parameters, it is referred to as constructor \_\_\_\_\_.
6. The constructor generated by the compiler is known as the \_\_\_\_\_.

**Ans.**

- |           |                |                        |
|-----------|----------------|------------------------|
| 1. Class  | 2. Variables   | 3. new                 |
| 4. return | 5. Overloading | 6. Default constructor |

#### B. Very short answer type questions.

1. What is a constructor?

**Ans.** A constructor is a special method that has the same name as that of the class. It does not have any return type.

2. Which unit of a class gets called when the object of the class is created?

**Ans.** A constructor gets called when the object of the class is created.

3. Name the two types of constructors.

**Ans.** Default constructor and parametrised constructor are the two types of constructors.

4. Define a parametrised constructor for the class 'rectangle' that assigns the length and breadth and pass it to the variables 'len' and 'br'.



Ans. `rectangle(int len, int br)`  
{  
length = len;  
breadth = br;  
}

5. Create a class with one integer variable. Initialise the variable using:

- Default constructor
- Parametrised constructor

Ans. `class name`  
{  
int a;  
name() // Default constructor  
{  
a = 0;  
}  
name(int t) // Parameterised constructor  
{  
a = t;  
}  
}

6. How many constructors can a class have?

Ans. A class can have any number of constructors but with different sets of parameters.

7. Why do the constructors not have any return type?

Ans. Constructors do not have any return type, not even void because the implicit return type of a constructor is the class type itself.

### C. Short answer type questions.

1. Write any two features of constructors.

Ans. A constructor must have the same name as the class.

It does not return any value, even void.

2. What is constructor overloading?

Ans. The process of defining more than one constructor for a class is called constructor overloading. The set of the parameters in every constructor must be different.

3. Why do the constructors need to be overloaded?

Ans. Constructor overloading allows different initial values for different objects. Therefore, the constructors need to be overload.

4. "Member method and constructor can be defined private". Write whether the given statement is true or false. Give reason.

Ans. A member method can be private but a constructor cannot. If you declare a private constructor, the object creation for the class may not be possible. Hence, the given statement is false.

## D. Application based questions.

1. Consider the following class 'club' and answer the following questions:

```
class club
{
 int x, y;
 int total;
 club(int a, int b)
 {
 x=a;
 y=b;
 total=x+y;
 }
 public static void main(String args[])
 {
 club ob = new club();
 System.out.println("Sum="+ob.total);
 }
}
```

a. Write all the data members and the constructor of the class 'club'.

b. In the above code, how many times club( ) will be executed?

**Ans.** a. Data members of class club is: x, y, and total, and constructor is: club( ).

b. The constructor club( ) will be invoked only once because only one object 'ob' is created.

2. Distinguish between the given pair of statements:

a. Date() { }

b. Date() {d=1,m=7}

**Ans.** If the class contains the constructor Date( ), the constructor gets invoked on the creation of an object but does nothing. On the other hand, the second constructor when invoked, would initialise data members 'd' as 1 and 'm' as 7.

3. Differentiate between the following statements:

a. abc p = new abc();

b. abc p = new abc(5, 7, 9);

**Ans.** The first statement is calling a non-parameterised constructor to create and initialise an object 'p' of class 'abc'. The second statement is calling a parameterised constructor that accepts three arguments to create and initialise an object 'p' of class 'abc'.

## SECTION 2 (Unsolved Questions)

### A. Very short answer type questions.

1. What happens if you provide a return type to a constructor?
2. Why do you need a constructor?
3. Describe the basic structure of a constructor.
4. What is the use of the keyword 'new'?
5. Define constructor overloading.

## B. Short answer type questions.

1. Write any four characteristics of a constructor.
2. Why do you need a constructor as a class member?
3. What is the use of the keyword 'this'?
4. Distinguish between the parameterised and non-parameterised constructor.

## C. Application based questions.

1. Create a class with the following descriptions. Also, define the main( ) function to create an object to invoke all the methods listed below:

**Class name:** Rectangle

**Data members/instance variables:**

length: To store length of the rectangle

breadth: To store breadth of the rectangle

area: To store area of the rectangle

**Member methods:**

Rectangle( ): A default constructor to initialise data members

void input( ): To input length and breadth

void findArea( ): To find area of the rectangle

void display( ): To print length, breadth, and area of the rectangle

2. Write a program to design a calculator according to the details given below.

**Class name:** Calculator

**Data members/instance variables:**

Result: To store result value

**Member methods:**

Calculator( ): A default constructor to initialise data members

void enterFirst(double x): Initialise result with x

void showResult( ): To print the result on screen

void add(int x): To add the argument to the result

void sub(int x): To subtract the argument from the result

void multiply(int x): To multiply the argument to the result

void divide(int x): To divide the argument by the result

void clear( ): To overwrite the result with 0

3. Create a class with the following descriptions Also, define the main( ) function to create an object to invoke all the methods listed below.

**Class name:** TravelPlan

**Data members/instance variables:**

PlanCode: To store code for travelling

Place: To store name of the place to be visited

Number\_of\_travellers: To input number of travellers

Number\_of\_buses: To input number of buses needed

### Member methods:

TravelPlan( ): A default constructor to initialise plan code as 1001, place as "AGRA", number\_of\_travellers is: 4, and number of buses as 1

void display( ): To show all the content of all the instance variables

void newPlan( ): To input plan code, place, and number\_of\_travellers

void Bus\_requirement( ): To find out number of buses required according to the following:

| Number of travellers | Number of buses required |
|----------------------|--------------------------|
| Less than 20         | 1                        |
| 20 to 40             | 2                        |
| 40 & above           | 3                        |

## SECTION 3 (Board Questions)

### 1. State two features of a constructor.

[2010]

Ans. a. A constructor has the same name as that of the class.

b. A constructor has no return type, not even void.

### 2. Explain the concept of constructor overloading with an example.

[2011]

Ans. Constructor overloading is a technique through which a class can have more than one constructor with different parameter lists. The different constructors of the class are differentiated by the compiler using the number of parameters in the list and their types. For example, the Rectangle class below has two constructors:

```
class Rectangle
{
 int length;
 int breadth;
 //Constructor 1
 public Rectangle()
 {
 length = 0;
 breadth = 0;
 }
 //Constructor 2
 public Rectangle(int l, int b)
 {
 length = l;
 breadth = b;
 }
 public static void main(String[] args) {
 //Constructor 1 called
 Rectangle ob1 = new Rectangle();
 //Constructor 2 called
 Rectangle ob2 = new Rectangle(10, 15);
 }
}
```

3. Create a class with one integer instance variable. Initialise the variable using:

[2012]

- Default constructor
- Parameterised constructor

Ans. class Number

```
{
 int a;
 public Number()
 {
 a = 0;
 }
 public Number(int x)
 {
 a = x;
 }
}
```

4. Give a difference between a constructor and a method.

[2013]

Ans. A constructor has the same name as class name whereas a method should have a different name than the class name.

5. What is a constructor? When is it invoked?

[2014]

Ans. A constructor is a member method that is written with the same name as the class name and is used to initialise the data members or instance variables. A constructor does not have a return type. It is invoked at the time of creating any object of the class.

6. Name the two types of constructors.

Ans. Parameterised constructors and non-parameterised (default) constructors.

7. What is a parameterised constructor?

[2016]

Ans. A parameterised constructor receives parameters at the time of creating an object and initialises the object with the received values.

8. Differentiate between a constructor and method.

[2017]

Ans.

| Constructor                                                               | Method                                                                                                                        |
|---------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| A constructor is a block of code that initialises a newly created object. | A function is a group of statements that can be called at any point in the program using its name to perform a specific task. |
| A constructor has the same name as the class name.                        | A method should have a different name than the class name.                                                                    |
| A constructor has no return type, not even void.                          | A method requires a valid return type.                                                                                        |

## SECTION 4 (Case Study Based Question)

A constructor in Java is a special method that is used to initialise objects. It is called when an object of a class is created. It can be used to set initial values for object attributes. Note that the constructor name must match the class name, and it cannot have a return type (not even void).

There are two types of constructors: default and parameters. Default constructors do not contain any parameters, whereas parameterised constructors contain parameters that are passed when an object of the class is invoked.

1. A method having the same name as the class name is known as \_\_\_\_\_.
  - a. Constructor
  - b. Template
  - c. Abstract method
  - d. Static method
2. A constructor has the following as the return type.
  - a. int
  - b. ob
  - c. void
  - d. no return type
3. A constructor that has parameters is known as \_\_\_\_\_.
  - a. Default constructor
  - b. Parameterised constructor
  - c. Copy constructor
  - d. Temporary constructor
4. A constructor is called when an \_\_\_\_\_ of the class is created.
  - a. Operation
  - b. Object
  - c. Value
  - d. Execution

Ans. 1. a                      2. d                      3. b                      4. b

## Programming ONE

1. Define a class Student as given below. Also, write a main( ) method to create an object of the class to invoke all the member methods.

Data members/instance variables:

name, age, m1, m2, m3 (marks in three subjects), maximum, average

Member methods:

A default constructor to initialise the data members

To accept the details of a student

To compute the average and the maximum marks

To display the name, age, marks in three subjects, maximum, marks of these subjects, and their average.

```
Ans. import java.util.*;
class Student
{
 String name;
 int age;
 int m1;
 int m2;
 int m3;
 int maximum;
 double average;
 Student()
 {
 name = "";
 age = 0;
 m1 = 0;
 m2 = 0;
```

```

 m3 = 0;
 maximum = 0;
 average = 0.0;
 }
 void accept()
 {
 Scanner input = new Scanner(System.in);
 System.out.print("Enter name: ");
 name = input.nextLine();
 System.out.print("Enter age: ");
 age = input.nextInt();
 System.out.print("Enter Subject 1 Marks: ");
 m1 = input.nextInt();
 System.out.print("Enter Subject 2 Marks: ");
 m2 = input.nextInt();
 System.out.print("Enter Subject 3 Marks: ");
 m3 = input.nextInt();
 }
 void compute()
 {
 if (m1 > m2 && m1 > m3)
 maximum = m1;
 else if (m2 > m1 && m2 > m3)
 maximum = m2;
 else
 maximum = m3;
 average = (m1 + m2 + m3) / 3.0;
 }
 void display()
 {
 System.out.println("Name: " + name);
 System.out.println("Age: " + age);
 System.out.println("Subject 1 Marks: " + m1);
 System.out.println("Subject 2 Marks: " + m2);
 System.out.println("Subject 3 Marks: " + m3);
 System.out.println("Maximum Marks: " + maximum);
 System.out.println("Average Marks: " + average);
 }
 public static void main(String args[]) {
 Student ob = new Student();
 ob.accept();
 ob.compute();
 ob.display();
 }
}

```

2. Define a class named 'FruitJuice' with the following description. Also, write a main( ) method to create an object of the class to invoke all the member methods.

**Data members:**

int product\_code: To store the product code number

String flavour: To store the flavour of the juice (e.g., orange, apple, etc.)

String pack\_type: To stores the type of packaging (e.g., tetra-pack, pet bottle, etc.)

int pack\_size: To store package size (e.g., 200 mL, 400 mL, etc.)

int product\_price: To store the price of the product

**Member methods:**

FruitJuice( ): A constructor to initialise integer data members to 0 and string data members to " "

void input( ): To input and store the product code, flavour, pack type, pack size, and product price

void discount( ): To reduce the product price by 10

void display( ): To display the product code, flavour, pack type, pack size, and product price

```
Ans. import java.util.*;
class FruitJuice
{
 int product_code;
 String flavour;
 String pack_type;
 int pack_size;
 int product_price;
 FruitJuice()
 {
 product_code = 0;
 flavour = "";
 pack_type = "";
 pack_size = 0;
 product_price = 0;
 }
 void input()
 {
 Scanner input= new Scanner(System.in);
 System.out.print("Enter Flavour: ");
 flavour = input.nextLine();
 System.out.print("Enter Pack Type: ");
 pack_type = input.nextLine();
 System.out.print("Enter Product Code: ");
 product_code = input.nextInt();
 System.out.print("Enter Pack Size: ");
 pack_size = input.nextInt();
 System.out.print("Enter Product Price: ");
 product_price = input.nextInt();
 }
 void discount()
 {

```



```

 product_price -= 10;
 }
 void display()
 {
 System.out.println("Product Code: " + product_code);
 System.out.println("Flavour: " + flavour);
 System.out.println("Pack Type: " + pack_type);
 System.out.println("Pack Size: " + pack_size);
 System.out.println("Product Price: " + product_price);
 }
 public static void main(String args[])
 {
 FruitJuice ob = new FruitJuice();
 ob.input();
 ob.discount();
 ob.display();
 }
}

```

3. Define a class named movieMagic with the following description. Also, write a main( ) method to create an object of the class to invoke all the member methods.

**Data members:**

int year: To store the year of release of a movie

String title: To store the title of the movie

float rating: To store the popularity rating of the movie (minimum rating=0.0 and maximum rating=5.0)

**Member methods:**

movieMagic( ): Default constructor to initialise numeric data members to 0 and String data member to " "

void accept( ): To input and store year, title, and rating

void display( ): To display the title of the movie and a message based on the rating as given:

| Rating     | Message to be displayed |
|------------|-------------------------|
| 0.0 to 2.0 | Flop                    |
| 2.1 to 3.4 | Semi-Hit                |
| 3.5 to 4.4 | Hit                     |
| 4.5 to 5.0 | Super-Hit               |

```

Ans. import java.util.*;
class movieMagic
{
 private int year;
 private String title;
 private float rating;
 movieMagic()
 {
 year = 0;
 title = "";
 rating = 0.0f;
 }
 void accept()
 {

```

```

Scanner input = new Scanner(System.in);
System.out.print("Enter Title of Movie: ");
title = input.nextLine();
System.out.print("Enter Year of Movie: ");
year = input.nextInt();
System.out.print("Enter Rating of Movie: ");
rating = input.nextFloat();
}
void display()
{
 String message = "Invalid Rating";
 if (rating <= 2.0f)
 message = "Flop";
 else if (rating <= 3.4f)
 message = "Semi-Hit";
 else if (rating <= 4.4f)
 message = "Hit";
 else if (rating <= 5.0f)
 message = "Super-Hit";
 System.out.println(title);
 System.out.println(message);
}
public static void main(String args[])
{
 movieMagic ob = new movieMagic();
 ob.accept();
 ob.display();
}
}

```

4. Define a class named **BookFair** with the following description. Also, write a **main( )** method to create an object of the class to invoke all the member methods.

**Data members/instance variables:**

String bname: To store the name of the book

double price: To store the price of the book

**Member methods:**

BookFair( ): Default constructor to initialise data members

void display( ): To display the name and price of the book after discount

void Input( ): To input and store the name and the price of the book

void calculate( ): To calculate the price after discount. Discount is calculated based on the following criteria:

| Price                                           | Discount     |
|-------------------------------------------------|--------------|
| Less than or equal to ₹1000                     | 2% of price  |
| More than ₹1000 and less than or equal to ₹3000 | 10% of price |
| More than ₹3000                                 | 15% of price |

**Ans.** `import java.util.*;`  
`class BookFair`

```

{
 String bname;
 double price;

 BookFair()
 {
 bname = "";
 price = 0.0;
 }
 void input()
 {
 Scanner input = new Scanner(System.in);
 System.out.println("Enter name of the book: ");
 bname = input.nextLine();
 System.out.println("Enter price of the book: ");
 price = input.nextDouble();
 }
 void calculate()
 {
 double disc;
 if (price <= 1000)
 disc = price * 0.02;
 else if (price <= 3000)
 disc = price * 0.1;
 else
 disc = price * 0.15;
 price -= disc;
 }
 void display()
 {
 System.out.println("Book Name: " + bname);
 System.out.println("Price after discount: " + price);
 }
 public static void main(String args[])
 {
 BookFair ob = new BookFair();
 ob.input();
 ob.calculate();
 ob.display();
 }
}

```

5. Design a class name 'ShowRoom' with the following description. Also, write a main( ) method to create an object of the class to invoke all the member methods.

**Data members/instance variables:**

String name: To store the name of the customer

long mobno: To store the mobile number of the customer

double cost: To store the cost of the items purchased

double dis: To store the discount amount

double amount: To store the amount to be paid after discount

#### Member methods:

ShowRoom(): default constructor to initialise data members

void input(): To input customer name, mobile number, and cost

void display(): To display customer name, mobile number, amount to be paid after discount

void calculate(): To calculate discount on the cost of purchased items, based on following criteria:

| Cost                                              | Discount (in percentage) |
|---------------------------------------------------|--------------------------|
| Less than or equal to ₹10000                      | 5%                       |
| More than ₹10000 and less than or equal to ₹20000 | 10%                      |
| More than ₹20000 and less than or equal to ₹35000 | 15%                      |
| More than ₹35000                                  | 20%                      |

```
Ans. import java.util.*;
class ShowRoom
{
 String name;
 long mobno;
 double cost;
 double dis;
 double amount;
 ShowRoom()
 {
 name = "";
 mobno = 0;
 cost = 0.0;
 dis = 0.0;
 amount = 0.0;
 }
 void input()
 {
 Scanner input = new Scanner(System.in);
 System.out.print("Enter customer name: ");
 name = input.nextLine();
 System.out.print("Enter customer mobile no: ");
 mobno = input.nextLong();
 System.out.print("Enter cost: ");
 cost = input.nextDouble();
 }
 void calculate()
 {
 int disPercent = 0;
 if (cost <= 10000)
 disPercent = 5;
 else if (cost <= 20000)
 disPercent = 10;
```

```

else if (cost <= 35000)
 disPercent = 15;
else
 disPercent = 20;

dis = cost * disPercent / 100.0;
amount = cost - dis;
}
void display()
{
 System.out.println("Customer Name: " + name);
 System.out.println("Mobile Number: " + mobno);
 System.out.println("Amount after discount: " + amount);
}
public static void main(String args[])
{
 ShowRoom ob = new ShowRoom();
 ob.input();
 ob.calculate();
 ob.display();
}
}

```



**Link**

Scan the given QR Code to learn about constructors.



## Game for Entrepreneur Skills

**This game develops teamwork, negotiation, and strategic thinking abilities. The players must determine which puzzle pieces are missing from their set and how to get them.**

### Rules to play:

- Select a variety of jigsaw puzzles for teams to begin.
- There must be a mix of all jigsaw puzzle pieces that are distributed among the teams in random order.
- The goal is to finish the puzzle before the other teams do.
- The players must come up with innovative ways to persuade the other team members to acquire the pieces they need. They can exchange a team member; go for merger, or barter.



## Chapter Highlights

- 6.1 Revisiting Data Types in Java
- 6.2 Wrapper Classes
- 6.3 Autoboxing and Unboxing
- 6.4 Methods of Wrapper Classes

## Learning Outcomes

The student will be able to:

- revise the concept of data types in Java.
- understand the concept of wrapper classes.
- describe autoboxing and unboxing.
- use methods of wrapper classes.

Java provides a library of built-in classes with predefined methods that you can include in your Java programs to perform several tasks. These library classes belong to their respective packages. A Java package is a collection of similar types of built-in classes (library classes) and sub-packages that offer the majority of the functionality. The most frequently used classes are the wrapper classes present in the 'java.lang' package. In this chapter, you will learn about wrapper classes in detail. Before learning about the concept of wrapper classes, let us first revise the various categories of data types.

## 6.1 Revisiting Data Types in Java

A data type specifies what type of data a variable will store. You have already learnt that Java has two main categories of data types: primitive and non-primitive (composite).

### Primitive Data Types

These are the data types that are predefined in Java. There are eight types of primitive data types, namely **int**, **float**, **char**, **Boolean**, **byte**, **short**, **long**, and **double**.

### Non-primitive Data Types

These data types are user-defined data types, which means their definition is given in the program by the user. These data types have instances like objects. As a result, they are called reference variables. They can be created using primitive data types, which is why they are also known as composite data types. The most popular types are classes, arrays, strings, and interfaces. A class is considered a composite data type because it is defined with many primitive data types, as shown in the following example:

#### Example

```
import java.util.*;
public class student
{
 int rollno, marks1, marks2, marks3;
 public float average()
```

```

{
 float avg1;
 avg1=(marks1+marks2+marks3)/3;
 return avg1;
}
public static void main(String args[])
{
 float a;
 student s = new student();
 s.rollno=101;
 s.marks1=55;
 s.marks2=65;
 s.marks3=75;
 a=s.average();
 System.out.println("The average is: "+a);
}
}

```

In the above example, the 'student' class has four data members (rollno, marks1, marks2, marks 3) of integer type, and a member method that returns a float value. Hence, you can say that a class is a composite data type.

### Difference between Primitive and Composite Data Types

The following table summarises the differences between primitive and composite data types:

| Primitive Data Types                           | Composite Data Types                                                                                            |
|------------------------------------------------|-----------------------------------------------------------------------------------------------------------------|
| These are predefined or inbuilt data types.    | These are made up of primitive data types.                                                                      |
| These data types are provided by Java.         | They are created by the user.                                                                                   |
| The size of the primitive data types is fixed. | The size of user-defined data types depends on the number and type of primitive data types used to create them. |
| Examples: int, char, float, double, etc.       | Examples: class, String, array, etc.                                                                            |



## 6.2 Wrapper Classes

Java uses primitive data types, such as int, float, double, and so on, to hold the values. These values can be passed to methods through a 'call by value' method. But sometimes, you may need to pass such data (for example, numeric values) to a method through an object, i.e., as a reference. To solve this problem, Java provides wrapper classes that encapsulate primitive data type values within objects.

A wrapper class wraps the value of a primitive data type in an object. It contains various methods to convert primitive values into objects and vice versa. The following table shows different wrapper classes provided by Java with their primitive data types. Notice that the name of a wrapper class begins with an uppercase letter while a primitive data type begins with a lowercase letter. For example, 'float' is a data type, whereas 'Float' is a wrapper class.

| Primitive Data Types | Wrapper Classes |
|----------------------|-----------------|
| boolean              | Boolean         |
| char                 | Character       |
| byte                 | Byte            |
| short                | Short           |
| int                  | Integer         |
| long                 | Long            |
| float                | Float           |
| double               | Double          |

### Let's Know More

The main difference between wrapper classes and primitive data types is that wrapper classes are used to convert primitive types into objects and objects back to primitive types, while primitive data types are predefined data types provided by the Java programming language.

## Need of Wrapper Classes

The wrapper classes are important in a program for the following reasons:

- ◆ Wrapper classes convert primitive data types into objects. Objects are needed to pass primitive values (arguments) as references to a method, whenever required.
- ◆ Wrapper classes make it possible to use primitive values as objects that can be used with any type of class and its methods.
- ◆ Wrapper classes provide various methods that help in converting various data types to the required data types. For example, a numeric value could be converted to a string, or a string could be converted to a numeric value.

## Using Wrapper Classes

Each wrapper class comes with its own constructor that you can use to create objects of that wrapper class. The **new** operator is used when you explicitly create objects of the wrapper class explicitly.

Notice the following example, where 'ob' is an object of **Integer** type (wrapper class) whereas 'a' is an integer variable, i.e., primitive data type.

### Example

```
int a = 5; //Primitive data type
Integer ob = new Integer(a); //Object of Integer wrapper class
```

### Program 1

Write a program that initialises wrapper class variables using the new operator.

```
1 import java.io.*;
2 public class Wrap1
3 {
4 public static void main (String args[])
5 {
6 Integer n = new Integer(20); // Object of Integer wrapper class
7 System.out.println("Value of n is: " + n);
8 }
9 }
```

### OUTPUT

Value of n is: 20



In the above example, the primitive data type's value is passed to the constructor of the Integer wrapper class to create a wrapper object of integer type. In the same way, you can create objects of other wrapper classes.

The different constructors for creating objects of wrapper classes and methods to obtain respective values are as listed below:

| Constructor                        | Method                  |
|------------------------------------|-------------------------|
| Character c1 = new Character('D'); | char charValue();       |
| Boolean b1 = new Boolean(true);    | boolean booleanValue(); |
| Byte b1 = new Byte(11);            | byte byteValue();       |
| Short s1 = new Short(26);          | short shortValue();     |
| Integer i1 = new Integer(234);     | int intValue();         |
| Long l1 = new Long(5432187)        | long longValue();       |
| Float f1 = new Float(35.6f)        | float floatValue();     |
| Double d1 = new Double(7923.43)    | double doubleValue();   |

Consider the following program to understand how to create and use objects of a wrapper class:

#### Program 2

Write a program to print primitive values of wrapper class objects.

```

1 import java.lang.*;
2 public class wrapper
3 {
4 public static void main(String[] args)
5 {
6 Character x = new Character('9'); //Creating a Character object
7 char ch = x.charValue();
8 Boolean b = new Boolean(true); //Creating a Boolean object
9 boolean value = b.booleanValue();
10 Integer a = new Integer(34); //Creating an Integer object
11 int i = a.intValue();
12 System.out.println("Primitive char value is:" + ch);
13 System.out.println("Primitive boolean value is:" + b);
14 System.out.println("Primitive Integer value is:" + i);
15 }
16 }
```

#### OUTPUT

```

Primitive char value is: 9
Primitive boolean value is: true
Primitive Integer value is: 34
```



### Quick Quiz

#### 1. Describe primitive data types.

These are the data types that are predefined in Java. There are eight types of primitive data types, namely **int**, **float**, **char**, **Boolean**, **byte**, **short**, **long**, and **double**.

#### 2. Why is a class called composite data type?

A class is called a composite data type because it is defined with many primitive data types.

## 6.3 Autoboxing and Unboxing

Autoboxing and unboxing are two important features of Java. Let us discuss them in detail.

### Autoboxing

Automatic conversion of primitive data types to object of their corresponding wrapper classes is known as **autoboxing**, and this is taken care of by the compiler. For example, conversion of **int to Integer**, **long to Long**, **double to Double**, etc.

Consider the following statement. Here, the character value 'A' is said to be 'autoboxed' into its corresponding wrapper class Character.

#### Example:

Character charVar='A'; //Boxing into a Character object

The Java compiler applies autoboxing:

- ◆ When a primitive value is passed as a parameter to a method that expects a wrapper class object.
- ◆ When a primitive value is assigned to a variable of the corresponding wrapper class.

The following program shows the conversion of a primitive value into an object:



#### Program 3

Write a program that demonstrates autoboxing.

```
1 public class box
2 {
3 public static void main (String args[])
4 {
5 int a = 35; //Primitive data type
6 Integer j = a; //Converting or boxing int to an Integer object
7 System.out.println("Value of integer variable is:"+ a);
8 System.out.println("Value of object is:"+ j);
9 }
10 }
```

#### OUTPUT

Value of integer variable is: 35  
Value of object is: 35

In the above example, when 'a' is assigned with an integer value, the compiler boxes this value to 'j' because 'j' is an integer object. So, you will get the output as shown.

### Unboxing

It is just the reverse process of autoboxing, it is also taken care of by the compiler. Converting an object of a wrapper class to its corresponding primitive type is known as unboxing. For example, **Integer to int**, **Long to long**, **Double to double**, and so on.

Consider the following statements. Here, object 'charVar' is said to be unboxed into the primitive type 'char'.

### Example

```
Character charVar = 'A';
char c = charVar; //Unboxing into a char primitive data type
```

The Java compiler applies unboxing:

- ◆ When a wrapper class object is passed as a parameter to a method that expects a primitive value.
- ◆ When a wrapper class object is assigned to a primitive variable.

The following program shows the conversion of an object into a primitive value.

#### Program 4

Write a program that demonstrates unboxing.

```
1 public class unbox
2 {
3 public static void main (String args[])
4 {
5 Integer Num = 20; // boxing int to an Integer object
6 int Number = Num*2; //unboxing the Integer to an int
7 System.out.println("Value of integer variable is:" + Num);
8 System.out.println("Value of object is:" + Number);
9 }
10 }
```

#### OUTPUT

```
Value of object is: 20
Value of variable is: 40
```

In the above example, 'Num' is an object of Integer wrapper class. So, the compiler unboxes the value of this object 'N', as it is to be multiplied into an integer variable. Hence, you will get the output as shown.

## 6.4 Methods of Wrapper Classes

As you know, wrapper classes are available in the 'java.lang' package. They are imported by default, whenever a Java program is loaded. The basic requirement of using wrapper classes is to convert primitive data values into its objects and vice-versa. There are various methods that are used to do various operations on the wrapper class objects, as discussed below:

### Parsing Methods

The wrapper classes Integer, Long, Float, and Double provide methods to parse a String value, as a numeric value as listed below:

| Methods                    | Description                                                                                                                                                                                        |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>parseInt(String s)</b>  | This method belongs to the <b>Integer</b> wrapper class. It converts a valid string representation of a numeric value into a primitive integer value.<br><b>Syntax:</b> int parseInt(String s)     |
| <b>parseLong(String s)</b> | This method belongs to the <b>Long</b> wrapper class. It converts a valid string representation of a numeric value into a primitive long integer value.<br><b>Syntax:</b> long parseLong(String s) |

|                              |                                                                                                                                                                                                          |
|------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>parseFloat(String s)</b>  | This method belongs to the <b>Float</b> wrapper class. It converts a valid string representation of a numeric value into a primitive floating-point number.<br><b>Syntax:</b> float parseFloat(String s) |
| <b>parseDouble(String s)</b> | This method belongs to the <b>Double</b> wrapper class. It converts a valid string representation of a numeric value into a primitive double value.<br><b>Syntax:</b> double parseDouble(String s)       |

## NOTE

Make sure the characters in the input string of the **parse( )** method are digits while using this function. If the input string does not contain a numeric value or if the first character of the string is not a digit, then it returns NaN, i.e., not a number. To indicate a positive or negative value, a plus ("+") sign or a minus ("-") sign is provided at the beginning, respectively.

## Program 5

Write a program that demonstrates the use of parsing methods.

```

1 public class parse
2 {
3 public static void main (String args[])
4 {
5 int x = Integer.parseInt("9");
6 float f = Float.parseFloat("123.45");
7 long l = Long.parseLong("9988776655");
8 double d = Double.parseDouble("3.14E10");
9 System.out.println(x);
10 System.out.println(f);
11 System.out.println(l);
12 System.out.println(d);
13 }
14 }
```

## OUTPUT

```

9
123.45
9988776655
3.14E10
```

## Character Methods

The Character wrapper class provides methods for manipulating characters as listed below:

| Methods                          | Description                                                                                                                                                                     |
|----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>boolean isDigit(char ch)</b>  | It checks whether the character value is a digit or not and returns true if the digit is in between 0-9, otherwise it returns false.<br><b>Syntax:</b> boolean isDigit(char ch) |
| <b>boolean isLetter(char ch)</b> | It checks whether the character value is a letter or not and returns true if the value is a letter, otherwise it returns false.<br><b>Syntax:</b> boolean isLetter(char ch)     |

|                                         |                                                                                                                                                                                                          |
|-----------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>boolean isLetterOrDigit(char ch)</b> | It checks whether the character value ch is a letter or digit, or not and returns true if the value is a letter or digit, otherwise it returns false.<br><b>Syntax:</b> boolean isLetterOrDigit(char ch) |
| <b>boolean isLowerCase(char ch)</b>     | It checks whether the character value is in lowercase or not, and returns true if the value is in lowercase, otherwise it returns false.<br><b>Syntax:</b> boolean isLowerCase(char ch)                  |
| <b>boolean isUpperCase(char ch)</b>     | It checks whether the character value is in uppercase or not and returns true if the value is in uppercase, otherwise it returns false.<br><b>Syntax:</b> boolean isUpperCase(char ch)                   |
| <b>boolean isWhitespace (char ch)</b>   | It checks whether the character value is a whitespace or not and returns true if the value is a whitespace, otherwise it returns false.<br><b>Syntax:</b> boolean isWhitespace (char ch)                 |
| <b>char toLowerCase(char ch)</b>        | It converts the character to lowercase.<br><b>Syntax:</b> char toLowerCase(char ch)                                                                                                                      |
| <b>char toUpperCase(char ch)</b>        | It converts the character to uppercase.<br><b>Syntax:</b> char toUpperCase(char ch)                                                                                                                      |



## Let's Revise

- ◆ There are eight types of primitive data types: int, float, char, boolean, byte, short, long, and double.
- ◆ A class is considered a composite data type because it is defined with many primitive data types.
- ◆ Java provides wrapper classes to convert primitive values into objects and vice versa.
- ◆ Wrapper classes are available in the 'java.lang' package.
- ◆ Boolean, Character, Byte, Short, Integer, Long, Float, and Double are known as wrapper classes.
- ◆ The name of a wrapper class begins with an uppercase letter, while the primitive data type begins with a lowercase letter.
- ◆ Each wrapper class comes with its constructor that you can use to create objects of that wrapper class.
- ◆ Automatic conversion of primitive data types to objects of their corresponding wrapper classes is known as autoboxing, and this is taken care of by the compiler.
- ◆ Automatic conversion of a wrapper class object to its corresponding primitive type is known as unboxing, and this is also taken care of by the compiler.
- ◆ The wrapper classes Integer, Long, Float, and Double provides parseInt( ), parseLong( ), parseFloat, parseDouble to parse string values as numeric values, respectively.
- ◆ The Character wrapper class provides methods for manipulating characters, like isDigit(char ch), isUpperCase(char ch), etc.

## Key Terms

**Package:** It is a collection of similar types of built-in classes.

**Wrapper Classes:** These are part of the 'java.lang' package.

**Boxing:** It is the process of encapsulating a value within an object.



## SECTION 1 (Solved Questions)

### A. Fill in the blanks.

1. The first letter of the wrapper class is \_\_\_\_\_.
2. Wrapper classes are available in the \_\_\_\_\_ package.
3. The \_\_\_\_\_ wrapper class contains the data type 'int.'
4. The \_\_\_\_\_ method is used to convert String data to float type.
5. \_\_\_\_\_ is the process of changing a primitive type into a wrapper object.
6. \_\_\_\_\_ is a wrapper class of char data type.

**Ans.**

1. Uppercase

2. java.lang

3. Integer

4. parseFloat(String s)

5. Autoboxing

6. Character

### B. Very short answer type questions.

1. Name any two wrapper classes.

**Ans.** Integer and Float

2. Name the wrapper class of 'char' type.

**Ans.** The wrapper class of 'char' is Character.

3. Write the purpose of the following functions:

- a. Float.parseFloat( )
- b. char toLowerCase(char ch)
- c. boolean isWhitespace (char ch)

**Ans.**

- a. It is used to convert string data into float data.
- b. It returns true if character value is in lowercase, otherwise it returns false.
- c. It returns true if value is a whitespace, otherwise it returns false.

4. Write the return type of the following library functions.

- a. isLetterOrDigit(char)
- b. toUpperCase(char)

**Ans.**

- a. boolean
- b. char

5. What is the name for the process of converting a wrapper class object to a primitive type?

Ans. Unboxing

### C. Short answer type questions.

1. What are wrapper classes?

Ans. Wrapper classes are used to convert the primitive values into objects and vice versa. They are present in 'java.lang' package. The different wrapper classes provided by Java are Boolean, Byte, Integer, Float, Character, Short, Long, and Double.

2. Write a statement to convert a number stored in a string variable 'n' to double data type.

Ans. The statement used to convert a string variable to double is:

```
double d=Double.parseDouble(n);
```

3. What is the difference between Boolean and boolean.

Ans. Boolean is a wrapper class of the boolean data type whereas boolean is a primitive data type.

4. What do you mean by autoboxing?

Ans. When Java automatically converts a primitive type into a corresponding wrapper class object, like int to Integer, then it's called autoboxing.

5. Why do you need the wrapper classes?

Ans. The wrapper classes are important in a program for the following reasons:

Wrapper classes make it possible to use primitive values as objects that can be used with any type of class and its methods.

Wrapper classes provide various methods that help in converting types. For example, a numeric value could be converted to a numeric value.

### D. Application based questions.

1. Write the output of the following code segment:

```
Integer n = new Integer(235);
Float f = n.floatValue();
System.out.println(f);
```

Ans. 235.0

2. State the value of 'p' after the following is executed:

```
char x='9';
p=Character.isLetter(x);
```

Ans. false

3. Give the output of the following code:

```
String s="5";
int p=Integer.parseInt(s);
p=p+55;
System.out.println(p);
```

Ans. 60

4. Give the output of the following code:

```
String a="20", b="22";
int p=Integer.parseInt(a);
int q=Integer.valueOf(b);
System.out.println(p+" "+q);
```

Ans. 20 22

5. Give the output of the following code:

```
char ch='A'
System.out.println(Character.isLetter(ch));
```

Ans. true

## SECTION 2 (Unsolved Questions)

### A. Very short answer type questions.

1. List the names of any four wrapper classes.
2. How do you create an object of the Integer class? Give example.
3. Give an example to show the concept of autoboxing and unboxing.
4. Define non-primitive data types.
5. Describe the use of the following methods:
  - a. `parseInt(String s)`
  - b. `parseLong(String s)`
6. Write the difference between the following two methods:
  - a. `boolean isLetterOrDigit(char ch)`
  - b. `boolean isDigit(char ch)`

### B. Short answer type questions.

1. Distinguish between primitive data types and wrapper classes.
2. Describe the need for wrapper classes.
3. Explain autoboxing and unboxing.
4. Write the use of any two methods of the Integer class.
5. Describe any two methods of Character class.

### C. Application based questions.

1. Write the output of the following code segments:
  - a. 

```
char c = 'A';
int n = (int) c + 32;
System.out.println((char)n);
```
  - b. 

```
String s = "7";
int t = Integer.parseInt(s);
t = t + 1000;
System.out.println(t);
```
2. Give the output of the following code segments:
  - a. 

```
char c = 'B';
int i = 4;
```



```

System.out.println(c+i);
System.out.println((int)c+i);
b. char ch = 'y';
 char chr = Character.toUpperCase(ch);
 int p = (int) chr;
 System.out.println(chr + "\t" + p);
c. int n = 97;
 char ch = Character.toUpperCase((char)n);
 System.out.println(ch + " Great Victory");

```

### SECTION 3 (Board Questions)

1. State the values of 'n' and 'ch' based on the statement: `char c = 'A';` [2012]

```

int n = c + 1;
char ch = (char)n;

```

**Ans.** Value of 'n' is 66 and 'ch' is B.

2. Name the wrapper classes of char type and boolean type. [2015]

**Ans.** Wrapper class of char type is Character and boolean type is Boolean.

3. State the data type and value of 'res' after the following is executed: [2019]

```

char ch = '9';
res Character.isDigit(ch);

```

**Ans.** Data type of 'res' is boolean as return type of method `Character.isDigit()` is boolean. Its value is true as '9' is a digit.

4. Give the output of the following code: [2017]

```

String A = "26", B = "100";
String D = A + B + "200";
int x = Integer.parseInt(A);
int y = Integer.parseInt(B);
int d = x + y;
System.out.println("result 1="+D);
System.out.println("result 2="+d);

```

**Ans.** result 1 = 26100200  
result 2 = 126

### SECTION 4 (Case Study Based Question)

The programming language Java uses primitive data types, such as int, float, double, etc., to hold the data. The values of the primitive data types can be passed to methods by value only. But sometimes, you may need to pass such data to a method through an object as a reference. A primitive data type can be encapsulated within an object using a wrapper class. The wrapper classes wrap the value of a primitive type in an object. Wrapper classes are present in the `java.lang` package. The wrapper class contains methods to convert primitive data types into objects and vice versa. The different wrapper

classes provided by Java are Boolean, Byte, Integer, Float, Character, Short, Long, and Double. The name of a wrapper class begins with an uppercase letter, while the primitive data type begins with a lowercase letter.

1. \_\_\_\_\_ are used to convert various primitive data types values to the required data types.
  - a. Super classes
  - b. Subclasses
  - c. Abstract classes
  - d. Wrapper classes
2. A \_\_\_\_\_ data type can be encapsulated within an object using wrapper class.
  - a. Primitive
  - b. Non-primitive
  - c. void
  - d. Abstract
3. The name of a wrapper class begins with a \_\_\_\_\_.
  - a. Uppercase letter
  - b. Lowercase letter
  - c. Digit
  - d. Special character
4. Wrapper class are present in \_\_\_\_\_ package.
  - a. java.io
  - b. java.lang
  - c. java.wrapper
  - d. java.util

Ans. 1. d                      2. a                      3. a                      4. b

## Programming ONE

1. Write a program to input two letters . Find the difference between their ASCII codes and display the message accordingly.

- ✦ If  $d=0$ , both letters are same.
- ✦ If  $d<0$ , first letter is smaller.
- ✦ If  $d>0$ , second letter is smaller.

```
Ans. import java.util.*;
public class letter_comp
{
 public static void main(String args[])
 {
 char c1,c2;
 int d;
 Scanner input = new Scanner(System.in);
 System.out.print("Enter first character: ");
 c1 = input.next().charAt(0);
 System.out.print("Enter second character: ");
 c2 = input.next().charAt(0);
 d = (int)c1 - (int)c2;
 if (d > 0)
 System.out.println("Second letter is smaller");
```

```

 else if (d < 0)
 System.out.println("First letter is smaller");
 else
 System.out.println("Both letters are same");
 }
}

```

2. Write a program to input five numbers. Find the sum and product of the numbers. Join the sum and product to form a single number. Display the concatenated number.  
[Example: Let sum = 20 and product = 720, then the number after joining sum and product will be 20720]

Ans. `import java.util.*;`  
`public class num_sum_pro`  
`{`  
 `public static void main(String args[])`  
 `{`  
 `int i,n;`  
 `String sm;`  
 `long r;`  
 `Scanner input = new Scanner(System.in);`  
 `System.out.println("Enter 5 integers:");`  
 `long s = 0, p = 1;`  
 `for (i = 1; i <= 5; i++)`  
 `{`  
 `n = input.nextInt();`  
 `s += n;`  
 `p *= n;`  
 `}`  
 `sm = Long.toString(s) + Long.toString(p);`  
 `r = Long.parseLong(sm);`  
 `System.out.println("Final concatenated number = " + r);`  
 `}`  
`}`

3. Write a program to input a letter. Find its ASCII code. Reverse the digits of the ASCII code and display the new character. [Example, if the input value is 'O', you will get its ASCII code as 79. On reversing the ASCII code, the value of the new character will be displayed. i.e., 97 = 'a'.]

Ans. `import java.util.*;`  
`public class rev_letter`  
`{`  
 `public static void main(String args[])`  
 `{`

```

char ch;
int a,r,d;
Scanner input = new Scanner(System.in);
System.out.print("Enter a letter: ");
ch = input.next().charAt(0);
a = (int)ch;
System.out.println("ASCII Code = " + a);
r = 0;
while (a > 0)
{
 d = a % 10;
 r = r * 10 + d;
 a = a/10;
}
System.out.println("Reversed ASCII Code = " + r);
System.out.println("New character = " + (char)r);
}
}

```

4. Write a program to input ten letters. Convert each letter into uppercase. Find and display the number of vowels and consonants present in the set.

Ans. `import java.util.*;`

```

public class letter_v_con
{
 public static void main(String args[])
 {
 Scanner input = new Scanner(System.in);
 System.out.println("Enter any 10 letters:");
 int i, v = 0, c = 0;
 char ch;
 for (i = 1; i <= 10; i++)
 {
 ch = input.next().charAt(0);
 ch = Character.toUpperCase(ch);
 if (ch == 'A' || ch == 'E' || ch == 'I' || ch == 'O' || ch == 'U')
 v++;

```

```

 else if (ch >= 'A' && ch <= 'Z')
 c++;
 }

 System.out.println("Number of Vowels = " + v);
 System.out.println("Number of Consonants = " + c);
}
}

```



**Link**

Scan the given QR Code to learn the use of methods of the wrapper classes.



## Emotional Strengthening

**Tips to self-regulate using neuropsychology:**

**If you are feeling-**

**Stressed:** Do breathing exercises.

**Anxious:** Take a brisk walk.

**Angry:** Look outside the window.

**Insecure:** Make a list of your strengths and read them.

**Low motivation:** Focus on one spot on your screen for one minute.

## Chapter Highlights

- 7.1 Introduction
- 7.2 Access Specifiers
- 7.3 Visibility Rules for Access Specifiers
- 7.4 Scope of Variables

## Learning Outcomes

### The student will be able to:

- define encapsulation.
- know the use of access specifiers.
- understand the scope and visibility of access specifiers.
- understand the scope of variables.

Suppose, you are logging in to your bank account through net banking. For this, you enter your customer ID and password. The information you entered is used to validate whether you are a valid user or not. To authenticate your identity, several internal processes take place. If the entered information is incorrect, you will not get access to your account. The bank follows this practice to keep your account safe from being misused. Thus, security can be achieved by hiding some of the details from others.

In Java programming, the concept of data hiding is used to hide internal object details, which is achieved through encapsulation. Encapsulation is one of the most significant characteristics of a class. Let us learn about encapsulation in detail.

## 7.1 Introduction

Encapsulation is one of the basic fundamental OOP concepts. It is a mechanism for wrapping up the data and functions together into a single unit called class. If a class is properly designed, the programmer who uses the class does not need to know all the implementation details of the class. He or she can use that class by using the description defined by the programmer, just like the Math class that you import into Java programs to use mathematical methods.

In encapsulation, data members (variables) of a class are hidden from other classes. They can be accessed only through the member methods of their own class.

To achieve encapsulation in Java:

- ✦ declare the variables of a class as **private**.
- ✦ provide the **public** member methods to set and get the values of variables.

Consider the following program that demonstrates how to achieve encapsulation in Java:

### Example 1

```
public class Student
{
 private String n;
 private String c;
 private int r;
```

```

public void input_roll(int r1) //Public member method
{
 r=r1;
}
public void input_name(String n1) //Public member method
{
 n=n1;
}
public void input_class(String c1) //Public member method
{
 c=c1;
}
public int display_roll() //Public member method
{
 return r;
}
public String display_name() //Public member method
{
 return n;
}
public String display_class() //Public member method
{
 return c;
}
}

```

In the above program, the class 'Student' is encapsulated as the variables are declared as private and all the member methods are public. These member methods provide access to the instance variables of the class. Normally, these methods are referred to as **setters** and **getters** since they are used to set and get the values of variables. Therefore, any class that needs to access the variables should access them through these getters and setters methods.

Now, consider another program that shows how you can access instance variables of a class (that you have defined in the 'Student' class, given in the previous example) in other classes.

## Example 2

```

public class Student_Details
{
 public static void main(String args[])
 {
 Student ob = new Student();
 ob.input_name("Preeti");
 }
}

```

### OUTPUT

Name: Preeti, Class: 10A

```

 ob.input_class("10 A");
 ob.input_roll(15);
 System.out.print("Name: ," + ob.display_name() +" Class: "
 + ob.display_class());
 }
}

```

When you compile both the classes 'Student' and 'Student\_Details' in BlueJ, an arrow appears in between them, indicating that the class 'Student Details' instantiates an object of the class 'Student'. When you execute the Student\_Details class, you will get the output as shown in Example 2.

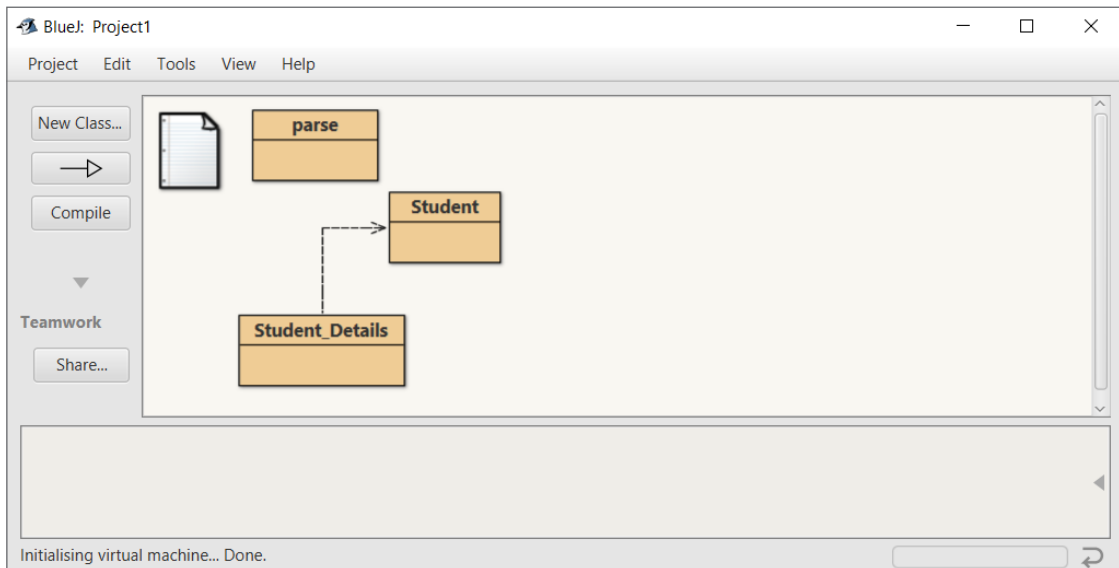


**Figure 7.1: BlueJ Project Window**

## Advantages of Encapsulation

Some advantages of encapsulation are listed below:

- ✦ The encapsulated code is more flexible and easier to change with new requirements.
- ✦ Encapsulation allows programmers to hide desired data of a class by making it private. Users have no idea about the internal implementation of the class. They only know how to update the data member's value by calling its getter or setter methods, but what these methods do is completely hidden from them.
- ✦ It keeps the data safe from external inheritance. Thus, encapsulation makes the program more secure.
- ✦ It provides easy maintenance of the application.

## 7.2 Access Specifiers

When you declare a new class in Java, you indicate the type of access permitted to its variables and methods. An access specifier or modifier restricts the access of the class, constructor, method, and data members to another class. There are four types of access specifiers used in a Java program: **public**, **private**, **protected**, and **default**.



## Public Access Modifier

The public access specifier does not put any restrictions on the access of the classes, methods, data members, or constructors. They can be accessed by other methods within the same class or any other class. The public access modifier is specified using the 'public' keyword at the beginning of the class or method declaration, as shown below:

### Example

```
public class cuboid //public member method
{
 public float volume(float l, float b, float h) //public member method
 {
 return (l*b*h);
 }
}
```

In the above example, the volume( ) method of the class 'cuboid' has a public access specifier, so it can be accessed in the same or any other class.

## Private Access Modifier

The private access modifier puts restrictions on the access of a class, method, or data member. If the access specifier of a class, method, or data member is declared as private, it can be accessed only within the class where it is defined. Any other class in the same package will not be able to access these members. The private access modifier is specified using the 'private' keyword at the beginning of the class or method declaration, as shown below:

### Example

```
public class cuboid //public member method
{
 //private member method
 private float volume(float l, float b, float h)
 {
 return (l*b*h);
 }
}
```

#### Let's Know More

A class cannot be declared as 'private' or 'protected'. If you do so, it would be completely useless because no one would be able to access it.

In the above example, the volume( ) method of the class 'cuboid' has a private access specifier, so it can be accessed in the same class only.

## Protected Access Modifier

The protected access modifier also puts restrictions on the access of a class, method, or data member. If the access specifier of a class, method, or data member is declared as protected, it can be accessed only by the classes of the same package. However, they are also accessible by the subclass present in any other package. If you do not make the subclass, it cannot be accessed from outside the package.

The protected access modifier is specified using the 'protected' keyword at the beginning of the class or method declaration, as shown below:

#### Example

```
public class cuboid
{
 protected float volume(float l, float b, float h) //Protected member method
 {
 return (l*b*h);
 }
}
```

In the above example, the volume( ) method of the class 'cuboid' has a protected access specifier, so it can be accessed within the package and outside the package through sub class.

### Default Access Modifier

When no access modifier is specified for a class, method, or data member, it has the default access modifier, by default. Such a type of class, method, or data member can be accessed only within the package.

#### Example

```
public class cuboid
{
 float volume(float l, float b, float h) //Method with default access modifier
 {
 return (l*b*h);
 }
}
```

In the above example, the volume( ) method of the class 'cuboid' has no access specifier (referred to as the default access specifier), so it is accessed within the same package.

## 7.3 Visibility Rules for Access Specifiers

The scope or visibility of an access specifier defines the scope of a class, method, data member, or constructor. It is summarised as given below:

- ✦ The **public** access specifier is the least restrictive access specifier. It provides the widest range of accessibility or visibility in comparison of all the other access specifiers. Its scope is visible to everyone in the project.
- ✦ The **private** access specifier has the scope and visibility only within the class where it is defined.
- ✦ The **protected** access specifier has the scope and visibility within the same package and all subclasses of other packages.
- ✦ The **default** access specifier has the scope and visibility only within the same package.

The scope and visibility of access specifiers has been explained through the following programs.

### Program 1

Write a program in Java that has two classes, one public and one default, to demonstrate the scope of the public access specifier.

```
1 public class SP1
2 {
3 public void display //Member method with public specifier
4 {
5 System.out.println("Hello");
6 }
7 }
8 public class SP2
9 {
10 public static void main(String args[])
11 {
12 SP1 ob = new SP1();
13 ob.display(); //Calling member method of a class to another
14 }
15 }
```

#### OUTPUT

Hello

The above example has two classes: 'SP1' and 'SP2'. The access specifier of 'SP1' is 'public' and 'SP2' is the 'default'. The main method is defined within the 'SP2' class, where an object of the class 'SP1' is created. As the access specifier of the display() method is public, it is accessible from any class. In the above example, it is accessed through the method call statement ob.display().

### Program 2

Write a program to demonstrate the scope of the private access specifier by creating two classes.

```
1 public class Area_of_circle1
2 {
3 private double r = 3.5;
4 private double circle(double r1) //Member method with private specifier
5 {
6 return(3.14*r*r);
7 }
8 }
9 public class Area_of_circle2
10 {
11 public static void main(String args[])
12 {
13 Area_of_circle1 ob = new Area_of_circle1();
14 System.out.println(ob.circle(4.2)); //Wrong access of the 'circle' method
15 }
16 }
```

The above example has two classes: 'Area\_of\_circle1' and 'Area\_of\_circle2'. The main method is defined within the 'Area\_of\_circle 2' class where an object of class 'Area\_of\_circle 1' is created.

As the access specifiers of the data member 'r' and method 'circle(double r1)' of the class 'Area\_of\_circle1' are 'private', the program will give the following two errors on compilation:

- ◆ r has private access in Area\_of\_circle1
- ◆ circle(double) has private access in Area\_of\_circle1

Thus, variables or methods that are specified as private can be accessed within the same class only.

### Program 3

Write a program that demonstrates the scope of the protected access specifier by creating two packages.

#### Package 1

```
1 package P1;
2 public class Sample_Class1
3 {
4 protected void design() //Protected member method
5 {
6 System.out.println("Troubleshooting");
7 }
8 }
```

#### Package 2

```
1 package P2;
2 import P1.*;
3 class Sample_Class2 extends Sample_Class1 /*Creating a subclass by including
4 extends keyword in another package*/
5 {
6 public static void main(String args[])
7 {
8 Sample_Class2 ob=new Sample_Class2(); // Creating object of Sample_Class2
9 ob.design(); /*Calling a protected method of Sample_Class1 using object
10 of subclass*/
11 }
12 }
```

The above example has two packages: 'P1' and 'P2'. In the 'P1' package, 'Sample\_Class1' is specified as 'public' to make it accessible in the 'P2' package. However, the design( ) method of 'Sample\_Class1' is specified as 'protected', so it can only be accessed by creating an object of the subclass of another package. Here, the subclass is 'Sample\_Class2'. So, on executing the 'Sample\_Class2' package, you will get the output as shown.

#### NOTE

To create a package in BlueJ, click on **Edit > New Package**. Enter the name of the package. Inside packages, you can define and execute classes.

### Program 4

Write a program to demonstrate the use of the default access specifier by creating two packages containing classes with the default access specifier. Try to access the class from one package to the class of another package.

## Package 1

```
1 package P1;
2 class Sample_Class1
3 {
4 void spdesign()
5 {
6 System.out.println("Troubleshooting");
7 }
8 }
```

## Package 2

```
1 package P2;
2 import P1.*;
3 class Sample_Class2
4 {
5 public static void main(String args[])
6 {
7 Sample_Class2 ob=new Sample_Class2();
8 ob.spdesign();
9 }
10 }
```

The above example also has two packages: 'P1' and 'P2'. When you execute 'Sample Class2' of the 'P2' package, you will get an error message indicating "Cannot find symbol" because a class or method with the 'default' access specifier can only be accessed within the same package.

The following table shows the scope of all the access specifiers in Java:

| Access Modifier | Within Class | Within Package | Outside Package by Subclass Only | Outside Package |
|-----------------|--------------|----------------|----------------------------------|-----------------|
| Private         | Y            | N              | N                                | N               |
| Default         | Y            | Y              | N                                | N               |
| Protected       | Y            | Y              | Y                                | N               |
| Public          | Y            | Y              | Y                                | Y               |

## 7.4 Scope of Variables

In Java programming, a variable can be declared and defined inside a class, method, or block. The 'scope of a variable' means how a specific variable is accessed within the program or across the classes. In other words, 'scope of a variable' refers to the visibility or accessibility of a variable. The variables used in a class can be any of the following types:

### Local Variables

Local variables are declared inside methods, blocks, or constructors. A block is a set of statements enclosed within curly brackets. The scope of local variables is limited to the methods, blocks, or constructors. So, local variables are accessible only to the methods, blocks, or constructors in which they are declared. They are not accessible to the rest of the class.

## Argument Variables

Argument variables are the variables that are passed during a function call. The number of arguments in a function call must be the same as the number of parameters in the function definition. Their scope is limited to the method where they are being used. Therefore, argument variables are visible only to the methods in which they are used. These variables are not accessible to the rest of the class.

## Instance Variables (Non Static)

Instance variables are the member variables that are declared in a class but outside of constructors, methods, or blocks. They are used to store the individual states of an object. These variables are called instance variables because their values are unique to each instance of the class. The scope of instance variables is within the class. They can be accessed from all the methods, constructors, and from the beginning of its program block to the end of the program block in the class.

## Class Variables (Static)

Static variables are declared in a class with the 'static' modifier. The static variable will be available as a single copy to all the instances of the class. The scope of static variables is also within the class. All the methods, constructors, and blocks within a class can access static variables by using the class name.

### Program 5

Write a program that demonstrate the scope of different types of member variables in a class.

```
1 public class number
2 {
3 static double x = 2.5; ← Class variable
4 private int n; ← Instance variable
5 public void setNum(int a)
6 {
7 n = a;
8 }
9 public double calc()
10 {
11 double r = x * x + n * n;
12 return (r);
13 }
14 public static void main(String args[])
15 {
16 int t = 7;
17 double s; ← Local variables
18 number ob = new number();
```

### OUTPUT

Value of x = 2.5

Value of n = 4

Result = 22.25

```

19 ob.setNum(t); ← Argument variable
20 s = ob.calc();
21 System.out.println("Value of x =" + number.x);
22 System.out.println("Value of n =" + ob.n);
23 System.out.println("Result =" + s);
24 }
25 }

```



## Let's Revise

- ◆ The process of wrapping up the data and functions together into a single unit is called encapsulation.
- ◆ An access specifier restricts the access of the class, constructor, method, and data members to another class.
- ◆ There are four types of access specifiers used in a Java program: public, private, protected, and default.
- ◆ The public access specifier is the least restrictive access specifier. It provides the widest range of accessibility or visibility in comparison of all the other access specifiers. Its scope is visible to everyone in the project.
- ◆ The private access specifier has the scope or visibility only within the class where it is defined.
- ◆ The protected access specifier has the scope or visibility within the same package and all subclasses of other packages.
- ◆ The default access specifier has the scope and visibility only within the same package.
- ◆ The 'scope of a variable' refers to the visibility or accessibility of a variable.
- ◆ Local variables are declared inside a method, block, or constructor. The scope of local variables is limited to the methods, blocks, or constructors in which they are declared.
- ◆ Argument variables are the variables that are passed during a function call. The scope of argument variables is limited to the method where they are being used.
- ◆ Instance variables are the member variables that are declared in a class but outside of constructors, methods, or blocks. The scope of instance variables is within the class.
- ◆ Static variables are declared in a class with the 'static' modifier. The scope of static variables is also within the class.

## Key Terms

**Encapsulation:** It is the wrapping up of data and functions together into a single unit called class.

**Access Specifier:** It specifies the accessibility of a method or variable.

**Scope of Variables:** It refers to the visibility or accessibility of a variable.

## SECTION 1 (Solved Questions)

### A. Fill in the blanks.

1. To achieve encapsulation in a Java program, you declare the variables of a class as \_\_\_\_\_.
2. The \_\_\_\_\_ of a variable defines the part or section of the program code where the variable is accessible.
3. The \_\_\_\_\_ access specifier has the scope and visibility within the same package, and all the subclasses of other packages.
4. The private access specifier has the scope and visibility only within the \_\_\_\_\_.
5. The \_\_\_\_\_ access specifier is the least restrictive access specifier.
6. Variables that are declared within body of a method are termed as \_\_\_\_\_.

|             |            |           |                    |
|-------------|------------|-----------|--------------------|
| <b>Ans.</b> | 1. private | 2. Scope  | 3. protected       |
|             | 4. class   | 5. public | 6. local variables |

### B. Very short answer type questions.

1. What is encapsulation?

**Ans.** Encapsulation is the process of wrapping up of data and functions together into a single unit.

2. Name any two access specifiers.

**Ans.** public and private

3. Can a 'private' member be accessed by the following?

- a. A member of the same class
- b. A member of the other class
- c. A member method of other class

**Ans.** a. Yes                      b. No                      c. No

4. Which access specifier makes the elements of a class visible to all?

**Ans.** public

5. Name the variable that is bounded to the object itself.

**Ans.** Instance variable

### C. Short answer type questions.

1. How are private members of a class different from public members?

**Ans.** Private members can be accessed or visible only within the class where they are defined, whereas public members can be accessed or visible to all, i.e., they can be accessed within the same class, outside the class, even outside the package.

2. Write any two advantages of encapsulation.

**Ans.** ♦ Encapsulation allows programmers to hide desired data of a class by making it private.  
 ♦ It keeps the data safe from external inheritance. Thus, encapsulation makes the program more secure.



### 3. Describe the different types of access specifiers.

**Ans.** The following are the different types of access specifiers:

**public:** The public access specifier indicates that method and data members can be accessed anywhere. It is the least restrictive access specifier and provides the accessibility or visibility to everyone in the project.

**private:** The private access specifier has the scope and visibility only within the class where it is defined.

**protected:** The protected access specifier has the scope and visibility within the same package and all subclasses of other packages.

**default:** The default access specifier has the scope and visibility only within the same package.

### 4. Explain the instance variables.

**Ans.** Instance variables are the member variables that are declared in a class but outside of constructors, methods, or blocks. They are used to store the individual states of objects.

### 5. Describe the class variables.

**Ans.** The variables that are declared in a class with the 'static' modifier are called static or class variables.

### 6. Define the scope of local variables.

**Ans.** The scope of local variables is limited to the methods, blocks, or constructors in which they are declared.

## SECTION 2 (Unsolved Questions)

### A. Very short answer type questions.

1. What is encapsulation?
2. What are the various access specifiers in Java?
3. Differentiate between private and protected access specifiers.
4. Describe the scope of the argument variables.
5. Describe the scope of the default access specifier.

### B. Short answer type questions.

1. What do you mean by the scope of variables in Java?
2. Why do you need to use static variables in Java?
3. Why can't you declare a class as private?
4. Differentiate between private and protected visibility modifiers.
5. What is the difference between local variables and instance variables?

## SECTION 3 (Board Questions)

### 1. Differentiate between static and non-static data members.

[2010]

**Ans.**

| Static Data Members                                                | Non-Static Data Members                                                |
|--------------------------------------------------------------------|------------------------------------------------------------------------|
| They are declared using the 'static' keyword.                      | They are declared without using the 'static' keyword.                  |
| All objects of a class share the same copy of static data members. | Each object of the class gets its own copy of non-static data members. |
| They can be accessed using the class name or object.               | They can be accessed only through an object of the class.              |

### 2. Name the keyword that declares class variables.

[2011]

**Ans.** static

3. What does a class encapsulate?

[2013]

Ans. A class encapsulates data and behaviour.

4. Consider the following class:

[2014]

```
public class myClass {
 public static int x=3, y=4;
 public int a=2, b=3;}

```

- Name the variables for which each object of the class will have its own distinct copy.
- Name the variables that are common to all objects of the class.

Ans. a. a, b

b. x, y

5. The access specifier that gives the most accessibility is \_\_\_\_\_ and the least accessibility is \_\_\_\_\_.

[2015]

Ans. The access specifier that gives the most accessibility is 'public' and the least accessibility is 'private'.

6. Name any two types of access specifiers.

[2016]

Ans. protected and private

#### SECTION 4 (Case Study Based Questions)

Encapsulation means wrapping up of data and functions together into a single unit. It is one of the most significant characteristics of a class. To achieve encapsulation in Java, variables are declared as private. The private variables can be accessed through public member methods. Normally, these methods are referred to as "setters" and "getters" since they are used to set and get the values of private variables. The property that controls the accessibility of member variables and member methods in a class is known as the access specifier. There are four types of access specifiers: public, private, protected, and default.

- Protected members of a class are visible within the \_\_\_\_\_.
  - Package
  - Class
  - Block
  - Both a and b
- If member variables are private, what can you do to access them from the class object?
  - Create private member methods to access those member variables.
  - Create protected member methods to access those member variables.
  - Create public member methods to access those member variables.
  - Private data members cannot directly accessed from outside the class.
- Which access specifier is used when no access specifier is used with a member of class (java)?
  - private
  - default
  - protected
  - public
- If variables are listed within a method block, they are known as \_\_\_\_\_.
  - Instance variables
  - Class variables
  - Argument variables
  - Local variables

Ans. 1. d

2. c

3. d

4. d

1. Write a program to print factors of an entered number. Create the class 'factors' with the following specifications:

**Data member:**

private int n: To define a private instance variable to store an integer value

**Member functions:**

void input( ): To take an integer number from user

void find\_fact( ): To print all the factors of the number

Ans. 

```
import java.util.*;

public class factors
{
 private int n; //Private instance variable
 public void input() //Public member method
 {
 Scanner input = new Scanner(System.in);
 System.out.println("Enter a number: ");
 n = input.nextInt();
 }
 public void find_fact() //Public member method
 {
 for (int i = 1; i <= n; i++)
 {
 if(n%i==0)
 System.out.println(i);
 }
 }
 public static void main(String args[])
 {
 factors ob = new factors(); //Creating object of a class
 ob.input(); //Invoking getter method to access private variable
 ob.find_fact(); // Invoking setter method to display value of a private variable
 }
}
```

2. Write a program to check and display whether the entered number is prime or not. Create the class 'check\_prime' with the following specifications:

**Data members:**

int n: To define an instance variable to store an integer value

#### Member methods:

void input( ): To take an integer number from user

void check( ): To check and display whether the number is prime or not

```
Ans. import java.util.*;
public class check_prime
{
 int n;
 public void input()
 {
 Scanner input = new Scanner(System.in);
 System.out.print("Enter the number: ");
 n = input.nextInt();
 }
 public void check()
 {
 int i, f=0;
 for (i = 1; i <=n; i++)
 {
 if (n % i == 0)
 f++;
 }
 if (f==2)
 System.out.println("Prime Number");
 else
 System.out.println("Not a Prime Number");
 }
 public static void main(String args[])
 {
 check_prime ob = new check_prime();
 ob.input();
 ob.check();
 }
}
```

3. Write a program to check and display whether the number is automorphic or not. Create the class 'automorphic' with the following specifications:

#### Data members:

int num, temp, square, digit: To define global variables to store integer values for different variables

#### Member methods:

void input( ): To take an integer number from user

void check( ): To check and display whether the number is automorphic or not

```

Ans. import java.util.*;
class automorphic
{
 int num, temp, square, digit=0;
 void input()
 {
 Scanner input=new Scanner(System.in);
 System.out.print("Enter a number:");
 num=input.nextInt();
 }
 void check()
 {
 temp=num;
 while(num!=0)
 {
 digit=digit+1;
 num=num/10;
 }
 square=temp*temp;
 int power=(int)Math.pow(10,digit);
 if(square%power==temp)
 System.out.println("Automorphic Number");
 else
 System.out.println("Not Automorphic Number");
 }
 public static void main(String args[])
 {
 automorphic ob = new automorphic();
 ob.input();
 ob.check();
 }
}

```

- Write a program to reverse the entered number. Also calculate and print the average of both the numbers (original number and reverse number). Create the class 'Reverse\_Average' with the following specifications:

**Data members:**

int num, temp, temp1, reverse: To define global variables to store integer values for different variables

double average: To define a local variable to store double value

### Member methods:

void input( ): To take an integer number from user

void display( ): To calculate the average of the reverse number and to display

Ans. `import java.util.*;`

`class Reverse_Average`

```
{
 int num, temp, temp1, reverse=0;
 double average;
 void input()
 {
 Scanner input=new Scanner(System.in);
 System.out.print("Enter a number: ");
 num=input.nextInt();
 }
 void display()
 {
 temp=num;
 while(num!=0)
 {
 temp1=num%10;
 reverse=reverse*10+temp1;
 num=num/10;
 }
 average=(temp + reverse)/2.0;
 System.out.println("Reverse number is: " +reverse);
 System.out.println("Average of original and its reverse number is: " +average);
 }
 public static void main(String args[])
 {
 Reverse_Average ob = new Reverse_Average();
 ob.input();
 ob.display();
 }
}
```



**Link**

Scan the given QR Code to learn the use of access specifiers and scope of variables.



### Chapter Highlights

- 8.1 Introduction to Arrays
- 8.2 Array as a Composite Type
- 8.3 Types of Arrays
- 8.4 Declaring Single Dimensional Arrays
- 8.5 Initialising Single Dimensional Arrays
- 8.6 Declaring Double Dimensional Arrays
- 8.7 Initialising Double Dimensional Arrays
- 8.8 Finding the Length of an Array
- 8.9 Accessing the Elements of an Array
- 8.10 Searching Techniques
- 8.11 Sorting Techniques

### Learning Outcomes

The student will be able to:

- know about arrays and their uses.
- create one-dimensional and two-dimensional arrays.
- determine the length of an array.
- access the elements of an array.
- use searching and sorting techniques on arrays.

You are familiar with the use of primitive variables in Java programming. Also, you have observed that a single value can be stored in a single variable. So, if you need to work with numerous variables of the same data type in a program, you need to declare all of them.

During the course of a program, remembering the names of all these variables is a difficult task. To overcome this, Java provides a linear data structure that allows you to store many values of the same data type under the same name. It is known as an 'array'. Let us learn about it in detail.

## 8.1 Introduction to Arrays

An **array** is a group of similar type of data items (homogeneous) that share a common name. It is a **non-primitive** or **reference** data type. It is useful when you have multiple variables of the same data type. Suppose you have the roll numbers of 50 students in a class, then you can use only an array of integer type instead of using 50 variables.

Furthermore, the elements of an array are stored in a **contiguous memory location**. The variables in the array are ordered, and each has an **index** or **subscript**. The index starts from 0 and goes up to n-1, where 'n' is the length of the array.

Look at the following figure to understand the structure of an array. Here, the array length is 9, the first index is 0, and the last index is 8. Indices are used for accessing values of an array. Each value in the array is called an **element**.

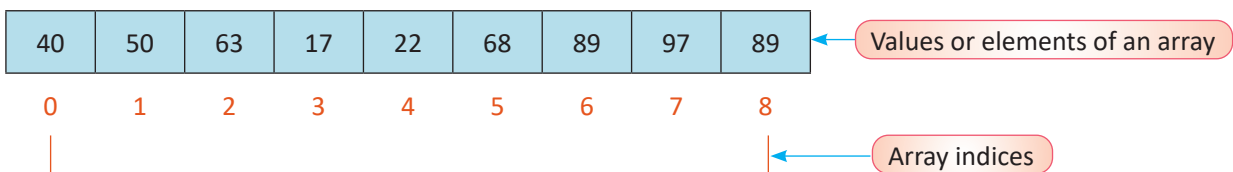


Figure 8.1: An Array

### Let's Know More

An array can hold primitive values of the same type that are referenced by a common name. You specify the type of values that the array stores at the time of the declaration.

### Know the Fact

The size of an array cannot be changed once you define it. However, if you try to assign a value to an array element that is larger than its size, a run time exception will be generated.

### Know the Fact

The size of an array cannot be negative. If you give size of an array as negative, you do not get any errors while compiling. But, you will get 'Negative Array Size Exception' at run time.

### Read More

If the array uses more than one index, it is called a multi-dimensional array. In a two-dimensional array, the first index stands for a row, and the second index stands for a column.

### Fact File

The **subscript** is the index of the element in the array, whereas the **subscripted variable** is the name of the array when it is used with a subscript to access a single element in the array.

## Advantages of Using Arrays

The following are some of the advantages of using an array:

**Same type of data items:** Arrays represent multiple values of the same type using a single name. This avoids the confusion of using multiple variables.

**Easy access of elements:** In an array, elements can be accessed easily by using their index numbers.

**Contiguous memory:** An array allocates memory to all of its elements in contiguous memory locations. Hence, there is no chance of extra memory being allocated in the case of arrays. This reduces the overhead time.

**Code readability:** An array allows you to store multiple values of the same type by defining a single variable. This helps in the improvement of the readability of the code. If no array is used in this case, then you need to store multiple values in multiple variables.

## 8.2 Array as a Composite Type

You have used the primitive data types, like int, float, double, etc. As array can be created using these primitive data types, therefore it can be considered as a **composite data type**.

In Java, the arrays can be declared by using the square brackets [ ] after the name of the array. For example, the array **Number** will be declared in Java as:

```
int Number[] = new int[9];
```

Alternatively, you can declare it as:

```
int Number[];
Number= new int[100];
```

All its elements will be accessed using the indices from 0 to 8 as Number[0], Number[1], Number[2], and so on.

As the array is a collection of data elements of the same type under single declaration, so it is called a composite data type.

## 8.3 Types of Arrays

There are two types of arrays in Java as discussed below:

### Single Dimensional Arrays

A single-dimensional array is also known as **one-dimensional** array. It contains the elements of the same data type in a contiguous manner.



It represents one row of elements that can be accessed by their indices or subscripts. Consider the following single dimensional array:

```
int n[] = {10, 20, 30, 40};
```

Where, all the elements are stored in contiguous memory locations in the order of their indices. The starting index is num[0], the last index is num[3]. These indices are holding different values as shown:

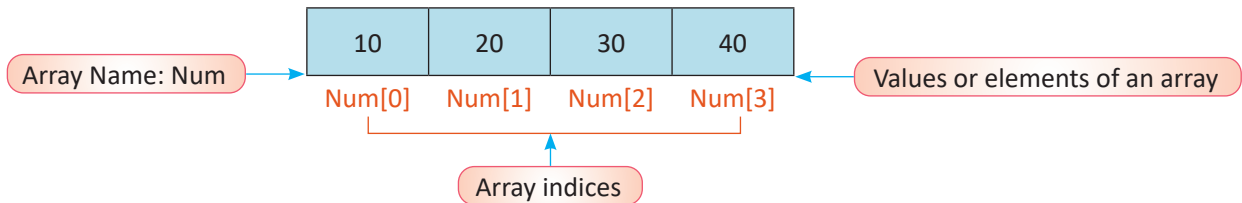


**Figure 8.2:** Single Dimensional Array

## Double Dimensional Arrays

A double-dimensional array is also known as a **two-dimensional** or **2D array**. It represents data in a tabular form in which data is stored into rows and columns. In simple words, you can say that a two dimensional is an array of multiple one-dimensional arrays where data can be accessed using the row index and column index, like [0][0], [0][1]. and so on. Consider the following 2D array to understand its structure in a better way:

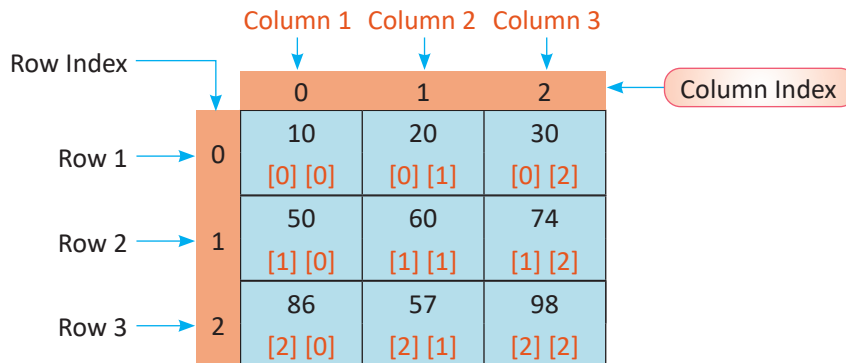


**Figure 8.3:** Two Dimensional Array

Here, the array **Number [3][3]** has three rows and three columns. The value in the first square bracket represents a row and the value in the second square bracket represents a column. So, the index starts from [0][0] in the upper-left corner to [2][2] in the lower-right corner.

## 8.4 Declaring Single Dimensional Arrays

Before using any variable in a program, you need to declare it. Similarly, you declare an array before using it.

### Syntax:

The syntax to declare an array is:

```
<data_type> <array name>[] = new <data_type>[size];
```

Or

```
<data_type>[] <array name>= new <data_type>[size];
```



### Example:

To declare an array **marks** to store the marks of 50 students, you may use the statement:

```
int marks[] = new int[50];
```

Or

```
int[] marks = new int[50];
```

You can also write the above statements in two steps:

```
int marks[];
```

```
marks = new marks[50];
```

Based on their data types, arrays can also be declared as follows:

| Data Type | Array                        |
|-----------|------------------------------|
| Byte      | byte ar[]=new byte[5];       |
| Short     | short ar[]=new short[5];     |
| Long      | long ar[]=new long[5];       |
| Float     | float ar[]=new float[5];     |
| Double    | double ar[]=new double[5];   |
| Boolean   | boolean ar[]=new boolean[5]; |
| Character | char ar[]=new char[5];       |
| String    | String ar[]=new String [5];  |

## 8.5 Initialising Single Dimensional Arrays

Initialising an array means assigning values to the array using the same variable name. There are multiple methods to assign values to the arrays as listed below:

- ◆ Using direct assignment
- ◆ Using array initialisation
- ◆ Using Scanner class
- ◆ Using method argument in BlueJ

Let us discuss these methods in detail.

### Using Direct Assignment

In this method, you directly use the index of the elements to assign the values. For example, if you want to assign a value to the fifth element of the **marks** array, then you can use the following statement:

```
marks[4] = 50;
```

The 5th element has an index n-1, i.e., 5-1 = 4.

Consider the following program to see how the direct assignment of values in an array is done.

### Program 1

Write a program to declare an array marks[ ] to store and display the marks of five students using the direct assignment method.

```
1. public class Stu_marks
2. {
3. public static void main(String args[])
4. {
5. int marks[] = new int[5]; //Declaring an array
6. marks[0] = 90;
7. marks[1] = 95;
8. marks[2] = 75;
9. marks[3] = 80;
10. marks[4] = 57;
11. System.out.println("marks[0] " + marks[0]);
12. System.out.println("marks[1] " + marks[1]);
13. System.out.println("marks[2] " + marks[2]);
14. System.out.println("marks[3] " + marks[3]);
15. System.out.println("marks[4] " + marks[4]);
16. }
17. }
```

#### OUTPUT

```
marks[0] 90
marks[0] 95
marks[0] 75
marks[0] 80
marks[0] 57
```

Direct assignment of values in an array

### Using Array Initialisation

You can assign values to the array elements at the time of array initialisation. This method defines and initialises values for the array variable. In this method, elements are passed to an array initialisation statement, which defines the size of the array. That means, you do not need to define the size of the array beforehand.

Consider the following program that shows how to assign values to the array at the time of array initialisation.

### Program 2

Write a program to declare an array marks[ ] and display the marks of the students alongwith their roll numbers. Assign values to the array while initialising it.

```
1. public class Stu_marks
2. {
3. public static void main(String args[])
4. {
5. int marks[] = {90, 95, 75, 80, 57};
```

Assigning values  
at the time of array  
initialisation

```

6. System.out.println("Roll No.\t Marks");
7. System.out.println("-----\t -----");
8. for (int i = 0; i < 5; i++)
9. {
10. System.out.println(i + "\t\t" + marks[i]);
11. }
12. }
13. }

```

| OUTPUT   |       |
|----------|-------|
| Roll No. | Marks |
| -----    | ----- |
| 1        | 90    |
| 2        | 95    |
| 3        | 75    |
| 4        | 80    |
| 5        | 57    |

## Using Scanner Class

In this method, the values are entered by the user at the execution time and then they are stored in the array at their respective positions. The following program demonstrate how to assign values to an array at the runtime.

### Program 3

Write a program to declare an array price[] to store the prices of six items. The array accepts the price of the items at runtime.

```

1. public class items
2. {
3. public static void main(String args[])
4. {
5. Scanner input = new Scanner(System.in);
6. float price[] = new float[5];
7. for (int i = 0; i < 5; i++)
8. {
9. System.out.println("Enter price for item" + (i+1));
10. price[i]=input.nextFloat();
11. }
12. System.out.println("Item \t Price");
13. System.out.println("-----\t -----");
14. for (int i = 0; i < 5; i++)
15. {
16. System.out.println((i+1) + "\t" + price[i]);
17. }
18. }
19. }

```

| OUTPUT                 |        |
|------------------------|--------|
| Enter price for item 1 |        |
| 200.50                 |        |
| Enter price for item 2 |        |
| 300.50                 |        |
| Enter price for item 3 |        |
| 150.75                 |        |
| Enter price for item 4 |        |
| 450.75                 |        |
| Enter price for item 5 |        |
| 325.25                 |        |
| Item                   | Price  |
| ----                   | ----   |
| 1                      | 200.5  |
| 2                      | 300.5  |
| 3                      | 150.75 |
| 4                      | 450.75 |
| 5                      | 325.25 |

## Using Method Argument in BlueJ

In this method, the input values are provided by the user at runtime in the BlueJ environment and values are saved in an array at the respective indices. The array is passed as an argument to the main() function. To understand this method, follow the given steps:

- ♦ Write the following code in the BlueJ editor.

## Program 4

A program to declare an array of marks[] that accepts the arguments when the method is invoked.

```
1. import java.util.*;
2. public class InputMethod
3. {
4. public static void main(int marks[])
5. {
6. System.out.println("Marks in 5 subjects are: ");
7. for (int i = 0; i < 5; i++)
8. {
9. System.out.println(marks[i]);
10. }
11. }
12. }
```

Passing an array  
as an argument to  
the method

- ◆ After writing the code, compile it by clicking on the **Compile** button in the editor window.
- ◆ Now, switch to the main window and right-click on the compiled class **InputMethod**.
- ◆ Select the method: `void main(int[] marks)` from the context menu as shown in Figure 8.4.
- ◆ The **Method Call** window opens. Enter five integer values for marks enclosed within the curly brackets as shown in Figure 8.5 and click on **OK**.

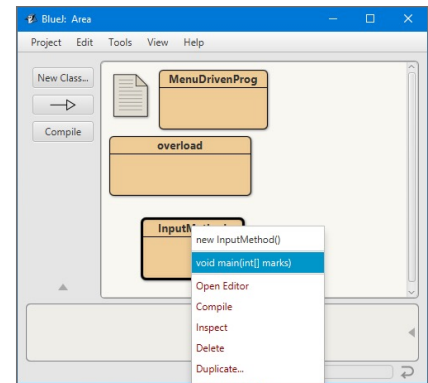


Figure 8.4: Invoking Method

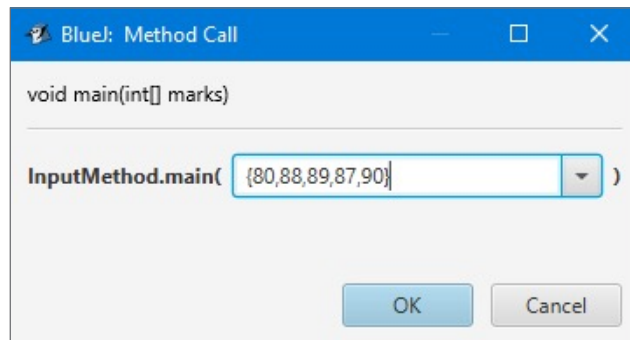


Figure 8.5: Method Call

- ◆ You will get the output as shown in the output window.

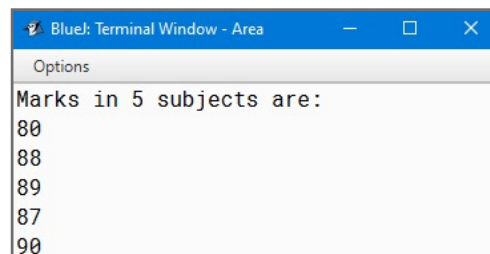


Figure 8.6: Output

## 8.6 Declaring Double Dimensional Arrays

As you know, the double-dimensional arrays have two dimensions, i.e., a row number and a column number. So, while declaring a double-dimensional array, both the dimensions are required.

### Syntax:

The syntax to declare a double-dimensional array is:

```
<data_type> <array_name>[][] = new <data_type>[row_size][column_size];
```

Or

```
<data_type>[][] <array_name> = new <data_type>[row_size][column_size];
```

In the syntax, the data type refers to the data that will be stored in the array. The first index, row\_size, indicates to the number of rows in the array, while the second index, column\_size, indicates the number of columns in the array.

For example, to declare a two dimensional array **matrix** to store the integer values, you may use the following statement:

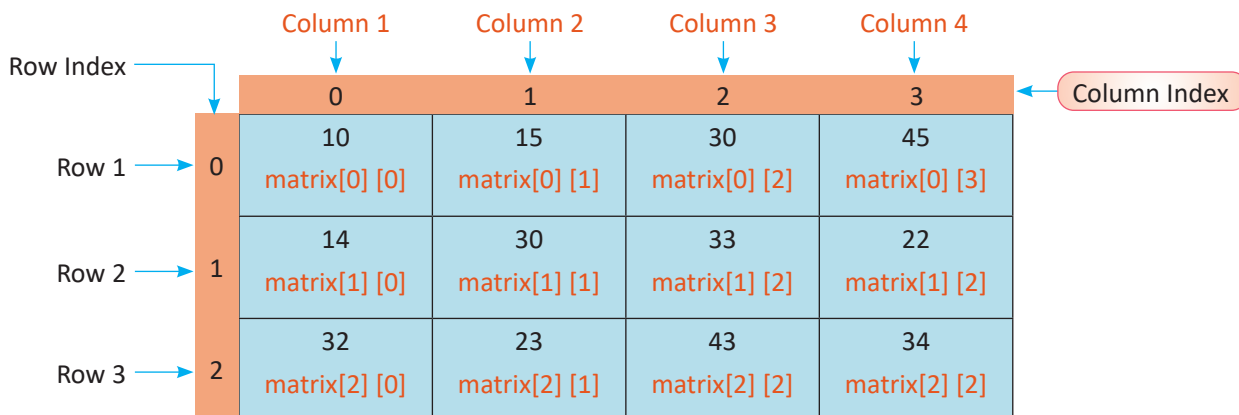
```
int matrix[] = new int[3][4];
```

Or

```
int[] matrix = new int[3][4];
```

The matrix can have the values as mentioned here:

```
int matrix[3][4] = {{10, 15, 30, 45}, {14, 30, 33, 22}, {32, 23, 43, 34}};
```



The diagram illustrates a 3x4 matrix. The columns are labeled 'Column 1' through 'Column 4' at the top, with arrows pointing to the column indices 0, 1, 2, and 3 respectively. The rows are labeled 'Row 1' through 'Row 3' on the left, with arrows pointing to the row indices 0, 1, and 2 respectively. Each cell in the matrix contains a numerical value and its corresponding array access notation. A 'Column Index' label with an arrow points to the column indices at the top right.

|           |   | Column 1           | Column 2           | Column 3           | Column 4           |              |
|-----------|---|--------------------|--------------------|--------------------|--------------------|--------------|
| Row Index |   | 0                  | 1                  | 2                  | 3                  | Column Index |
| Row 1     | 0 | 10<br>matrix[0][0] | 15<br>matrix[0][1] | 30<br>matrix[0][2] | 45<br>matrix[0][3] |              |
| Row 2     | 1 | 14<br>matrix[1][0] | 30<br>matrix[1][1] | 33<br>matrix[1][2] | 22<br>matrix[1][3] |              |
| Row 3     | 2 | 32<br>matrix[2][0] | 23<br>matrix[2][1] | 43<br>matrix[2][2] | 34<br>matrix[2][3] |              |

Figure 8.7: Double Dimensional Array: Matrix [3][4]

## 8.7 Initialising Double Dimensional Arrays

Like single dimensional arrays, there are multiple methods to assign values to the two-dimensional arrays as listed below:

- ◆ Using direct assignment
- ◆ Using array initialisation
- ◆ Using Scanner class
- ◆ Using method argument in BlueJ

Let us discuss these methods in detail.



## Using Direct Assignment

In this method, the values are directly assigned to the arrays using indices. Consider the following program to understand this.

### Program 5

Write a program to declare a 2D array and assign values to it using indices.

```
1. public class Array_2D
2. {
3. public static void main(String args[])
4. {
5. int matrix[][] = new int[3][4]; ← Array initialisation
6. matrix[0][3] = 180; ← Assigning values in an array using indices
7. matrix[1][3] = 200;
8. matrix[2][3] = 150;
9. System.out.println("matrix [0][3] = " + matrix [0][3]);
10. System.out.println("matrix [1][4] = " + matrix [1][3]);
11. System.out.println("matrix [2][3] = " + matrix [2][3]);
12. }
13. }
```

#### OUTPUT

```
matrix [0][3] = 180
matrix [1][3] = 200
matrix [2][3] = 150
```

## Using Array Initialisation

While declaring a two-dimensional array, you can provide the values to the array in the initialisation statement as shown in the following program. So, you do not need to specify the dimensions in the declaration statement.

### Program 6

Write a program to declare a 2D array. Assign values to it at the time of declaration.

```
1. public class Matrix1
2. {
3. public static void main(String args[])
4. {
5. int items[][] = {{11, 22, 33, 44}, ← Assigning values
6. {10, 20, 30, 40}, ← to an array while
7. {11, 21, 31, 41}}; ← declaring it
8. for (int i = 0; i < 3; i++)
9. {
10. for (int j = 0; j < 4; j++)
11. {
12. System.out.print(items[i][j] + " ");
13. }
14. System.out.println();
15. }
16. }
17. }
```

#### OUTPUT

```
11 22 33 44
10 20 30 40
11 21 31 41
```

## Using Scanner Class

You can also use the Scanner class to provide the values to a two-dimensional array. The values are entered at runtime by the user as shown in the Program 7.

### Program 7

Write a program to declare a 2D array that accepts values at run time.

```
1. import java.util.*;
2. public class arr_scan
3. {
4. public static void main(String args[])
5. {
6. Scanner arr_input = new Scanner(System.in);
7. System.out.print("Enter number of rows:");
8. int r = arr_input.nextInt();
9. System.out.print("Enter number of columns:");
10. int c = arr_input.nextInt();
11. int arrscanner[][] = new int[r][c];
12. System.out.println("Enter " + (r * c) + " numbers:");
13. for(int i = 0; i < r; i++)
14. {
15. for (int j = 0; j < c; j++)
16. {
17. arrscanner[i][j] = arr_input.nextInt();
18. //Reading input values of a matrix
19. }
20. }
21. System.out.println("Array values are:");
22. for(int i = 0; i < r; i++)
23. {
24. for(int j = 0; j < c; j++)
25. {
26. System.out.print(arrscanner[i][j] + " ");
27. //Displaying entered values of a matrix
28. }
29. System.out.println();
30. }
```

Scanner class object  
to read input

Declaring and creating array objects

### OUTPUT

```
Enter number of rows: 3
Enter number of columns: 3
Enter 9 numbers:
23
43
56
75
46
67
58
34
48
Array values are:
23 43 56
75 46 67
58 34 48
```



## Using Method Argument in BlueJ

You can also initialise a two-dimensional array while writing a program in BlueJ. Using this method, the values are entered at the runtime. Follow the given steps to pass an array as an argument to the main ( ) function:

- ♦ Write the following program in the BlueJ editor.

### Program 8

A program to declare an array of price [ ] that accepts the arguments when the method is invoked.

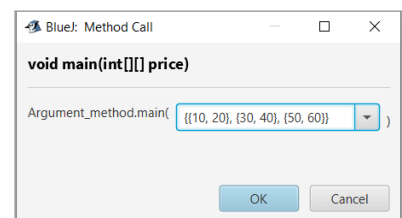
```
1. public class Argument_method
2. {
3. public static void main(int price[][])
4. {
5. System.out.println("2-D array is:");
6. for(int i = 0; i < 3; i++)
7. {
8. for (int j = 0; j < 2; j++)
9. {
10. System.out.print(price[i][j] + " ");
11. }
12. System.out.println();
13. }
14. }
15. }
```

### OUTPUT

2-D array is:

```
10 20
30 40
50 60
```

- ♦ Compile the program and from the main window, right-click on the compiled class. Select the method **void main(int[][] price)** from the shortcut menu.
- ♦ The Method call window opens up. Enter the values in the input box as: `{{10, 20}, {30, 40}, {50, 60}}`, Click on OK. You will get the output as shown in the output box.



## 8.8 Finding the Length of an Array

The length of an array determines the maximum number of elements it can hold. Although Java does not have any built-in methods for calculating array length, but it provides an attribute 'length' to find the length of an array. To use the length attribute, you need to provide the length keyword, which is followed by the dot (.) operator and the array name.

**Example:**

```
int money[]=new int[8];
```

```
int length_of_array=money.length; ← To find total length of an array
```

In the above example, money is an integer array with the length of 8, i.e., it can hold 8 elements. But, the logical size of the array is n-1, because the index starts from 0.

Consider the following programs to get familiar with the use of the length property of the array.

#### Program 9

Write a program to declare an integer array. Also, calculate its length.

```
1. public class Arr_len
2. {
3. public static void main(String args[])
4. {
5. int n[] = new int[10];
6. int l=n.length; // Calculating length of an array
7. System.out.println("The length of the array is: "+ l);
8. }
9. }
```

#### OUTPUT

The length of the array is: 10

Note that the length property determines the maximum number of elements that the array can contain or the capacity of the array. It does not count the actual number of elements that are inserted into the array. That is, it returns the total size of the array.

For arrays whose elements are initialised at the time of its creation, length, and size are the same.

#### Program 10

Write a program to declare an array that holds the names of the students. Also, calculate its length.

```
1. public class Arr_len
2. {
3. public static void main(String args[])
4. {
5. String nm = {"Preeti", "Khushi", "Shalini", "Smiley"};
6. int l=nm.length;
7. System.out.println("The size of the array is: " + l);
8. }
9. }
```

#### OUTPUT

The size of the array is: 4

In the above example, the length of the array is 4, as four elements are initialised for it.

## Array Index Out of Bounds Exception

When trying to access the array's elements, keep the array's size in mind. The Java Virtual Machine (JVM) throws an **ArrayIndexOutOfBoundsException** exception if you try to access an element whose index is larger than or equal to the array's size or maximum value of index. Also, if you declare an array with a negative length value, the JVM throws the same exception.

Look at the following example to understand this issue in a better way:

## Example

```
//A program that demonstrates the case of 'ArrayIndexOutOfBoundsException' exception in an array.
public class arr_exec
{
 public static void main(String args[])
 {
 int arr[]={5,6,7,8};
 for(int i=0; i<=arr.length; i++)
 {
 System.out.println(arr[i]);
 }
 }
}
```

Array elements equal to the length of the array are tried to be accessed.

Observe that the above program will produce an 'ArrayIndexOutOfBoundsException' exception: **Index 4 out of bounds for length 4**. This is because the array elements are accessed using the index equal to the maximum size of the array.

## 8.9 Accessing the Elements of an Array

You can access the elements of one-dimensional or two-dimensional arrays using their indices. The index of the array starts with 0 and ends with n-1, where 'n' is the size of the array. You need to use the for loop to access all the elements of an array.

### Accessing the Elements of Single Dimensional Arrays

Consider the following programs to learn how to access elements from single dimensional arrays:

#### Program 11

Write a program to input five numbers in an array and find their sum.

```
1. import java.util.*;
2. public class Arr_Sum
3. {
4. public static void main(String args[])
5. {
6. Scanner input=new Scanner(System.in);
7. int i,s=0;
8. int n[]=new int[5];
9. System.out.println("Enter five numbers for an array:");
10. for(i=0;i<5;i++)
11. {
12. n[i]=input.nextInt();
13. s=s+n[i];
14. }
15. System.out.print("Five numbers in the array are:");
```

#### OUTPUT

```
Enter five numbers
for an array:
5
4
3
2
1
Five numbers in
the array are: 5 4
3 2 1
The sum of the
entered numbers
is: 15
```

```

16. for(i=0;i<5;i++)
17. {
18. System.out.print(" "+ n[i]);
19. }
20. System.out.println(" ");
21. System.out.println("The sum of the entered numbers is:"+s);
22. }
23. }

```

### Program 12

Write a program to input 10 numbers in an array and find the biggest element of the array.

```

1. import java.util.*;
2. public class Big_Num
3. {
4. public static void main(String args[])
5. {
6. Scanner input=new scanner(System.in);
7. int i,max=0;
8. int n[]=new int[10];
9. System.out.println("Enter 10 numbers: ");
10. for(i=0;i<10;i++)
11. {
12. n[i]=input.nextInt();
13. if(n[i]>max)
14. max=n[i];
15. }
16. System.out.println("The biggest number is: " +max);
17. }
18. }

```

#### OUTPUT

Enter 10 numbers:

15

20

10

40

45

98

90

80

78

90

The biggest number is: 98

### Program 13

Write a program to accept the number of students in a class. Create a single-dimensional array to store the marks of all the students and find the average marks.

```

1. import java.util.*;
2. public class avg_marks
3. {
4. public static void main(String args[])
5. {
6. Scanner input=new Scanner(system.in);
7. int n,i,s=0;
8. double avg;
9. System.out.println("Enter the number of students:");

```

```

10. n=input.nextInt();
11. int m[]=new int[n];
12. for(i=0;i<n;i++)
13. {
14. System.out.println("Enter the marks:");
15. m[i]= input.nextInt();
16. s=s+m[i];
17. }
18. avg=s/n;
19. System.out.println("Average marks is: "+av);
20. }
21. }

```

#### OUTPUT

```

Enter the number of
students:
5
Enter the marks:
45
Enter the marks:
47
Enter the marks:
43
Enter the marks:
49
Enter the marks:
48
Average marks is: 46.0

```

### Program 14

Write a program to input 10 numbers in a single dimensional array. Find the largest, second largest, and the smallest element of the array.

```

1. import java.util.*;
2. public class numbers
3. {
4. public static void main(String args[])
5. {
6. Scanner input=new Scanner(System.in);
7. int i,max,min,smax;
8. int n[]=new int[10];
9. System.out.println("Enter 10 numbers");
10. for(i=0;i<10;i++)
11. {
12. n[i]= input.nextInt();
13. }
14. max=smax=min=n[0];
15. for(i=0;i<10;i++)
16. {
17. if(n[i]>max)
18. {
19. smax=max;
20. max=n[i];
21. }
22. if(n[i]>smax && n[i]<max)
23. {
24. smax=n[i];
25. }
26. if(n[i]<min)
27. {

```

#### OUTPUT

```

Enter 10 numbers
23
3
25
54
36
76
33
9
66
58
Largest number is: 76
Second largest number is: 66
Smallest number is: 3

```

```

28. min=n[i];
29. }
30. }
31. System.out.println("Largest number is: "+max);
32. System.out.println("Second largest number is: "+smax);
33. System.out.println("Smallest number is: "+min);
34. }
35. }

```

## Accessing the Elements of Two Dimensional Arrays

The elements of a two-dimensional array can also be accessed in the same manner as that of a single dimensional array, except that you need to follow nested for loops. One is for rows and another is for columns. Consider the following programs to see how you can access the elements of two-dimensional arrays.

### Program 15

Write a program to input numbers in a 4 x 5 array and find the largest element in each row.

```

1. import java.util.*;
2. public class largest
3. {
4. public static void main(String args[])
5. {
6. Scanner input=new Scanner(System.in);
7. int a[][]=new int[2][3];
8. int i, j, max;
9. System.out.println("Enter the elements in the array:");
10. for(i=0;i<2;i++)
11. {
12. for(j=0;j<3;j++)
13. {
14. a[i][j]=input.nextInt();
15. }
16. }
17. for(i=0;i<2;i++)
18. {
19. max=0;
20. for(j=0;j<3;j++)
21. {
22. if(a[i][j]>max)

```

### OUTPUT

```

Enter the elements in the array:
12
23
34
43
21
31
Largest element of row 1 is: 34
Largest element of row 2 is: 43

```

```

23. max=a[i][j];
24. }
25. System.out.println("Largest element of row "+(i+1)+" is: "+max);
26. }
27. }
28. }

```

### Program 16

Write a program to input numbers in a 4 x 4 array and display the sum of its left and right diagonals.

```

1. import java.util.*;
2. public class sum_diag
3. {
4. public static void main(String args[])
5. {
6. Scanner input=new Scanner(System.in);
7. int i,j,sl=0,sr=0;
8. int n[][]=new int[4][4];
9. for(i=0;i<4;i++)
10. {
11. for(j=0;j<4;j++)
12. {
13. System.out.println("Enter a number");
14. n[i][j]=input.nextInt();
15. }
16. }
17. for(i=0;i<4;i++)
18. {
19. sr=sr+n[i][i];
20. sl=sl+n[i][3-i];
21. }
22. System.out.println("sum of right diagonal is: "+sr);
23. System.out.println("sum of left diagonal is: "+sl);
24. }
25. }

```

### OUTPUT

Enter the elements:

10

20

30

40

50

60

70

80

11

22

33

44

55

66

77

88

Sum of right diagonal is: 191

Sum of left diagonal is: 187

### Program 17

Write a program to input numbers in a 4 x 4 array. Check whether the matrix is symmetric or not.

```

1. import java.util.*;
2. public class mat_sym
3. {

```

```

4. public static void main(String args[])
5. {
6. Scanner input=new Scanner(System.in);
7. int i,j,k=0;
8. int n[][]=new int[4][4];
9. System.out.println("Enter the elements:");
10. for(i=0;i<4;i++)
11. {
12. for(j=0;j<4;j++)
13. {
14. n[i][j]=input.nextInt();
15. }
16. }
17. for(i=0;i<4;i++)
18. {
19. for(j=0;j<4;j++)
20. {
21. if(n[i][j] != n[j][i])
22. {
23. k=1;
24. break;
25. }
26. }
27. }
28. if(k==0)
29. System.out.print("Symmetric Matrix");
30. else
31. System.out.print("Not a Symmetric Matrix");
32. }
33. }

```

## OUTPUT

```

Enter the elements:
10
22
33
11
44
55
20
30
66
77
80
90
40
44
33
99
Not a Symmetric Matrix

```

## 8.10 Searching Techniques



Searching is one of the basic operations that can be performed on arrays. When a programmer works on a large amount of data stored in arrays, it is necessary to determine whether an array contains a value that matches a certain value or not. The process of finding a particular element in the array is known as searching. There are two types of searching techniques: linear search and binary search. Let us discuss about these in detail.

### Linear Search

The linear search technique is also known as **sequential search**. It compares each element of the array with the search element. In linear search, the search starts from the first element of the array (0<sup>th</sup> index of the array) and continues sequentially across the array, checking each element with the



search element until a match is found or the array's last element is reached. The algorithm exits once the search value is found and returns the position (index) of the target value in the array. When the search process is over, a relevant message is displayed to convey the message whether the search process has been successful or unsuccessful.

### Algorithm for Linear Search

**Step 1:** Input the elements in the array `ar[ ]` having `n` elements

**Step 2:** Input the element to be searched, `ns`

**Step 3:** `ctr=0, p=0`

**Step 4:** While(`ctr<n`), repeat steps 5 and 6

**Step 5:** If (`ar[ctr] == ns`) then `k=1`

**Step 6:** `ctr=ctr+1`

**Step 7:** If `k=1`, then print "Search successful " else print "Search unsuccessful"

**Step 8:** End

| Variables Description |                            |
|-----------------------|----------------------------|
| <b>ar [ ]:</b>        | name of the array          |
| <b>n:</b>             | length of array            |
| <b>ns:</b>            | key element to be searched |
| <b>k:</b>             | flag variable              |
| <b>ctr:</b>           | counter variable           |

The above algorithm shows the process of linear search. Linear search examines each value until the target is found or the end of the list is reached.

Consider the following array `ar[ ]` containing 10 elements in which element 35 is being searched for. So, the loop begins from the 0<sup>th</sup> position and compares each element of the array with 35 as shown:

|    |    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|----|
| 12 | 23 | 19 | 24 | 27 | 18 | 35 | 37 | 41 | 26 |
| 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  |

Figure 8.8: An array

|             |             |             |             |             |             |             |             |             |             |
|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| 12          | 23          | 19          | 24          | 27          | 18          | 35          | 37          | 41          | 26          |
| AR[0] == 35 | AR[1] == 35 | AR[2] == 35 | AR[3] == 35 | AR[4] == 35 | AR[5] == 35 | AR[6] == 35 | AR[7] == 35 | AR[8] == 35 | AR[8] == 35 |
| False       | False       | False       | False       | False       | False       | True        |             |             |             |

Figure 8.9: Linear Search

As you can see, a match has been found, i.e., the search is successful. Therefore, the control will not proceed further. If the searched item is not found, you will get the message, "Search Unsuccessful". The linear search method is good for small and unsorted arrays.

### Program 18

Write a program to input 10 numbers in a single-dimensional array. Check whether the element is present in the array or not using the linear search technique.

```
1. import java.util.*;
2. class sda_lin_search
```

```

3. {
4. public static void main(String args[])
5. {
6. Scanner input=new Scanner(System.in);
7. int i,ns,x=0;
8. int n[]=new int [10];
9. System.out.println("Enter the elements in the array: ");
10. for(i=0;i<10;i++)
11. {
12. n[i]= input.nextInt();
13. }
14. System.out.println("Enter a number to search:");
15. ns=input.nextInt();
16. for(i=0;i<10;i++)
17. {
18. if(n[i]==ns) /*Comparing an array element
19. to the searched number*/
20. {
21. x=1;
22. break;
23. }
24. }
25. if(x==1)
26. System.out.println("Number is available");
27. else
28. System.out.println("Number is not available");
29. }

```

#### OUTPUT

```

Enter the elements in the
array:
35
23
46
59
65
76
82
13
54
65
Enter a number to search:
76
Number is available

```

## Binary Search

Binary search is the search technique that works on **sorted arrays**. Hence, in order to search an element in an array using binary search technique, you must ensure that the array is sorted.

Binary search follows divide and conquer approach in which, the list is divided into two halves and the item is compared with the middle element of the list. If the match is found then, the location of middle element is returned otherwise, the search is carried on in either of the halves depending upon the result produced through the match. If the element is found then the search ends, otherwise it results in an unsuccessful search.

The binary search is more efficient than the linear search because it reduces number of iterations.

## Algorithm for Binary Search

**Step 1:** Input the elements in the array `ar[ ]` having `n` elements in sorted order

**Step 2:** Input the element to be searched, `ns`

**Step 3:** `lb=0, ub=n-1, m=0, k=0`

**Step 4:** while(`lb<=ub`) and `k=0` repeat steps 5 to 7

**Step 5:** `m=(lb+ub)/2`

**Step 6:** if(`ns==ar[m]`) then `k=1`

**Step 7:** if(`ns<ar[m]`) then `ub=m-1`, else `lb=m+1`

**Step 8:** if `k=1` then print "search succesful" else print "search unsuccessful"

**Step 9:** End

The above algorithm shows the process of binary search. Each comparison in a binary search eliminates half of the remaining values under consideration.

Let us understand this technique with the help of an example. Assume that you have the following sorted array: `Price[ ]`, and you want to search for number 38 in this array.

|           |    |    |    |    |    |    |    |    |    |    |    |
|-----------|----|----|----|----|----|----|----|----|----|----|----|
|           | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 |
| Search 38 | 02 | 05 | 08 | 12 | 16 | 23 | 38 | 56 | 72 | 91 | 97 |

**Figure 8.10:** Sorted Array, `Price[ ]`

For this, you need to divide the sorted array into two halves by using the middle point. The middle point can be calculated as **(Lowest point + Highest point)/2**, where:

**Lowest point:** The first index of the array

**Highest point:** The last index of the array

**Middle point:** The middle point of the array

So, the middle point is:  $(0+10)/2=5$ .

After getting the middle element, compare it with the searching element (38) to check whether the middle element value (23) is equal to or greater than the searching element or not.

- ◆ If the middle element is equal to the searching element, you can declare "search successful" and end the search process.
- ◆ If the middle element value is less than the searching element, you will find the answer in the second half. In this case, you ignore the first half of the array and only search through the second half.
- ◆ If the middle element value is greater than the searching element, you will find the answer in the first half. In this case, you ignore the second half of the array and repeat the searching process only in the first half.

Here, the middle element value (23) is less than the searching element (38) and so you will find the answer in the second half. In this case, you ignore the first half of the array and repeat the searching process only in the second half as shown:

### Variables Description

**ar [ ]:** name of the array

**n:** length of array

**ns:** key element to be searched

**lb:** lower bound of array

**ub:** upper bound of array

**k:** flag variable

**m:** middle element of array

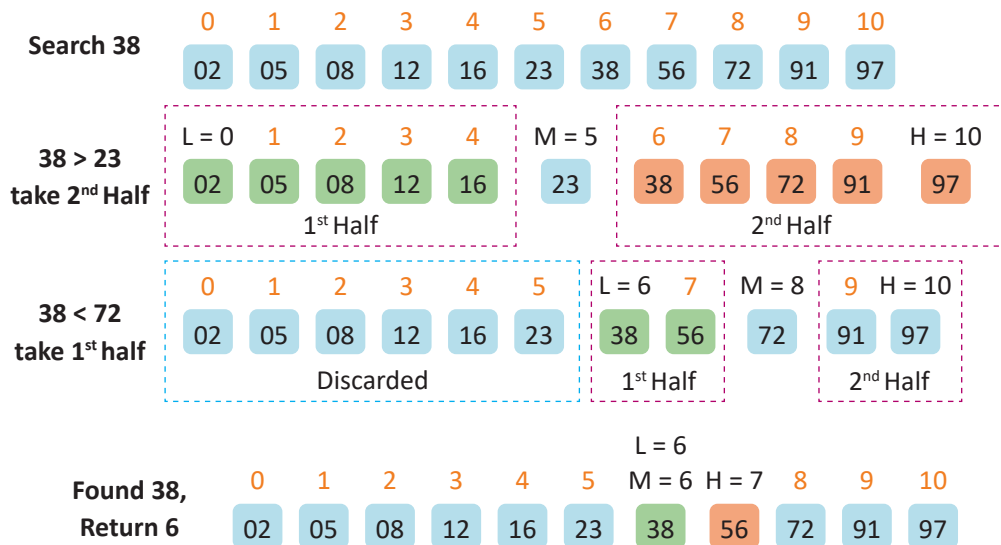


Figure 8.11: Binary Search

### Program 19

Write a program to input 10 numbers in a single-dimensional array. Check whether the element is present in the array or not using the binary search technique.

```

1. import java.util.*;
2. class bin_search1
3. {
4. public static void main(String args[])
5. {
6. Scanner input=new Scanner(System.in);
7. int i,lb=0,ub=9,m,ns,k=0;
8. int n[]=new int[10];
9. System.out.println("Enter the array elements in ascending order: ");
10. for(i=0;i<10;i++)
11. {
12. n[i]=input.nextInt();
13. }
14. System.out.println("Enter a number to search: ");
15. ns= input.nextInt();
16. while(lb<=ub)
17. {
18. m=(lb+ub)/2;
19. if(ns<n[m])
20. ub=m-1;
21. if(ns>n[m])
22. lb=m+1;
23. if(ns==n[m])
24. {
25. k=1;
26. break;

```

### OUTPUT

```

Enter the elements in the array:
23
25
32
38
44
56
69
74
85
99
Enter a number to search:
44
Number is available

```

```

27. }
28. }
29. if(k==1)
30. System.out.println("Number is available");
31. else
32. System.out.println("Number is not available");
33. }
34. }

```

## Difference between Linear Search and Binary Search

| Linear Search                                                                                        | Binary Search                                                                                                         |
|------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| A linear search can be performed on a sorted array as well as unsorted array.                        | A binary search can only be performed on a sorted array.                                                              |
| In linear search, the search operation begins from the first element and goes till the last element. | In binary search, the array is divided into two halves and the element to be searched is compared with the mid value. |
| Linear search is also known as sequential search.                                                    | Binary search is also known as non-sequential search.                                                                 |
| It is relatively slower, and takes $n-1$ iterations to find the $n^{\text{th}}$ element.             | It is faster than linear search as it reduces the number of iterations.                                               |

## 8.11 Sorting Techniques

Sorting is a method of arranging the elements in ascending or descending order. If the elements are numbers, then they are arranged numerically in ascending or descending order. If the elements are characters or strings, then they are arranged alphabetically in ascending or descending order.

There are many techniques for sorting available in Java:

◆ Selection Sort    ◆ Bubble Sort    ◆ Merge Sort    ◆ Insertion Sort

In this chapter, you will learn about the selection sort and bubble sort.



### Selection Sort

Selection sort is a technique in which successive elements are selected in an order and placed into their proper sorted positions.

In order to arrange the elements in ascending order, this algorithm searches for the smallest element in the array and swaps it with the element in the first position. Then, it searches for the second smallest element and swaps it with the element in the second position. This process continues until  $n-1$  smallest elements are found and moved to their respective places. The  $n^{\text{th}}$  element is the last, and it is already in place. It is called selection sort because it repeatedly selects the next-smallest element and swaps it into the right place.

### Algorithm for Selection Sort

**Step 1:** Input the elements in the array `ar[ ]` having  $n$  elements

**Step 2:** for  $i=0$  to  $n-1$  repeat steps 3 to 8

**Step 3:** min=i

**Step 4:** for j=i+1 to n repeat 5

**Step 5:** if(ar[j]<ar[min]) then min=j

**Step 6:** t=ar[min]

**Step 7:** ar[min]=ar[i]

**Step 8:** ar[i]=t

**Step 9:** End

Let us consider the unsorted array as below. You can see that in every step the element is swapped at the correct sorted position where it should be present.

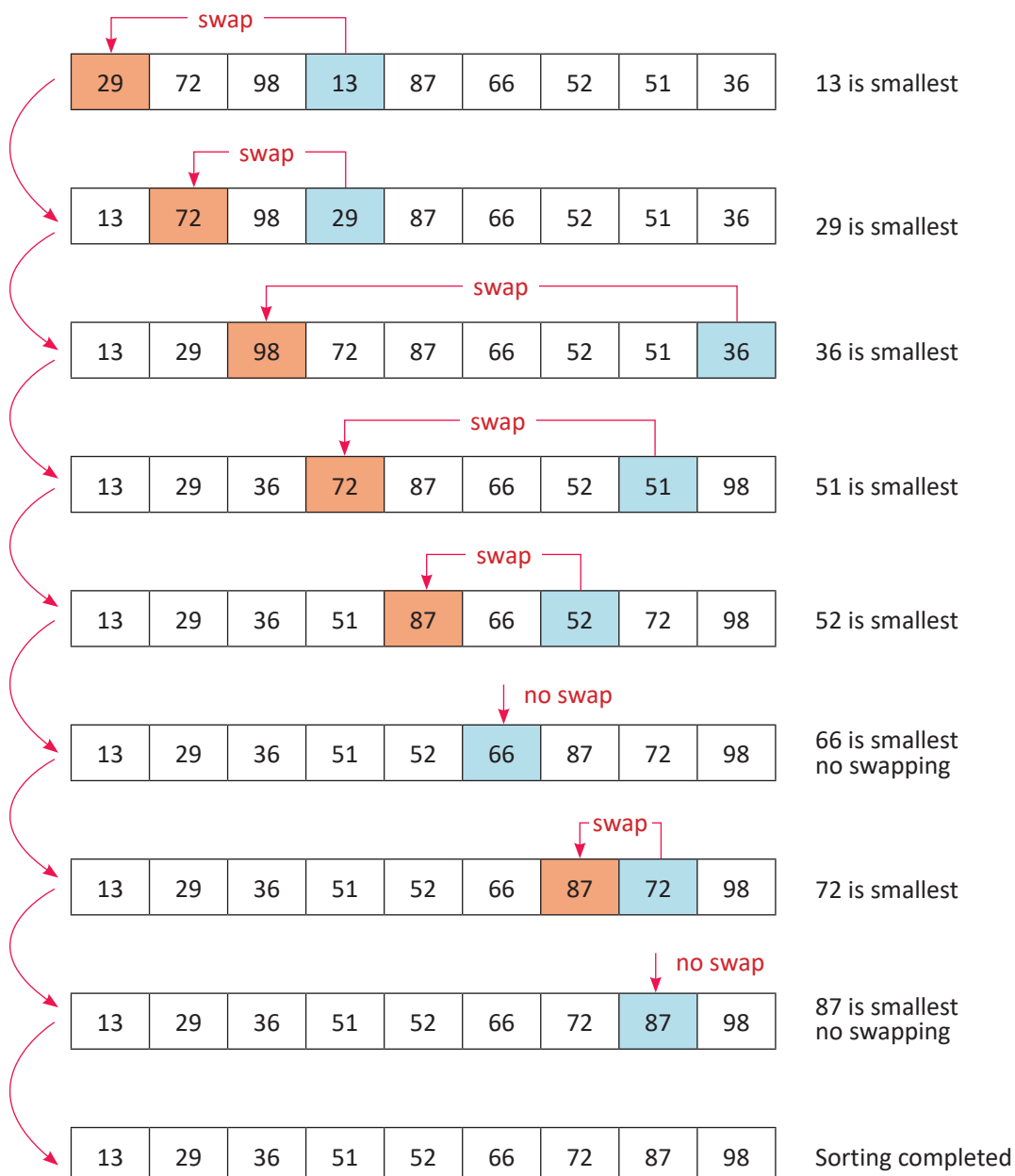


**Figure 8.12:** Selection Sort

## Program 20

Write a program to input 10 numbers in a single-dimensional array. Using the selection sort technique, arrange the elements of the array in ascending order and display the sorted array.

```
1. import java.util.*;
2. class sel_sort1
3. {
4. public static void main(String args[])
5. {
6. Scanner input=new Scanner(System.in);
7. int i,j,min=0,t;
8. int n[]=new int[10];
9. System.out.println("Enter the elements:");
10. for(i=0;i<10;i++)
11. {
12. n[i]=input.nextInt();
13. }
14. for(i=0;i<9;i++)
15. {
16. min=i;
17. for(j=i+1;j<10;j++)
18. {
19. if(n[j]<n[min])
20. min=j;
21. }
22. t=n[min];
23. n[min]=n[i];
24. n[i]=t;
25. }
26. System.out.println("Sorted array:");
27. for(i=0;i<10;i++)
28. {
29. System.out.println(n[i]);
30. }
31. }
32. }
```

### OUTPUT

Enter the elements:

25

41

8

12

43

37

81

51

66

9

Sorted array:

8

9

12

25

37

41

43

51

66

81

## Program 21

Write a program to input 10 names in a single-dimensional array. Using the selection sort technique, arrange the names of the array in ascending order and display the sorted array.

```
1. import java.util.*;
2. public class sel_sort2
3. {
4. public static void main(String args[])
5. {
```

```

6. Scanner input=new Scanner(System.in);
7. int i,j,min=0,String t;
8. String n[]=new String[10];
9. System.out.println("Enter 10 names:");
10. for(i=0;i<10;i++)
11. {
12. n[i]=input.nextLine();
13. }
14. for(i=0;i<9;i++)
15. {
16. min=i;
17. for(j=i+1;j<10;j++)
18. {
19. if(n[j].compareTo(n[min])<0)
20. min=j;
21. }
22. t=n[min];
23. n[min]=n[i];
24. n[i]=t;
25. }
26. System.out.println("Sorted array:");
27. for(i=0;i<10;i++)
28. {
29. System.out.println(n[i]);
30. }
31. }
32. }

```

#### OUTPUT

```

Enter 10 names:
Rohit
Harshit
Preeti
Santosh
Khushi
Shalini
Pankaj
Ansh
Aditya
Vishal
Sorted array:
Aditya
Ansh
Harshit
Khushi
Pankaj
Preeti
Rohit
Santosh
Shalini
Vishal

```

## Bubble Sort

In the bubble sort technique, the array is sequentially scanned multiple times and during each iteration, the pair of the consecutive elements are compared and interchanged if necessary to bring them into the specific order. In this sorting technique, every element is compared with the next element and if needed the position of the elements are swapped to make the array elements in proper order. The total number of iterations in this sorting is equal to  $1+2+3+\dots+n-1$ , where  $n$  is the number of elements in the array.

### Algorithm for Bubble Sort

**Step 1:** Input the elements in the array  $ar[ ]$  having  $n$  elements

**Step 2:** for  $i=0$  to  $n-2$  repeat steps 3 to 8

**Step 3:** for  $j=0$  to  $n-i$  repeat steps 4 to 8

**Step 4:** if( $ar[j]>ar[j+1]$ )



**Step 5:**  $t = ar[j]$

**Step 6:**  $ar[j] = ar[j+1]$

**Step 7:**  $ar[j+1] = t$

**Step 8:** End

Let us understand this technique with the help of an example. Suppose, you have an unsorted array as shown. To sort the array in an ascending order, the elements at the 0th and 1st index are compared. Their positions are swapped if the element at the 0th index is greater than the element at the 1st index. Since  $9 > 6$ , a swap occurs. This process continues until the entire array is sorted.

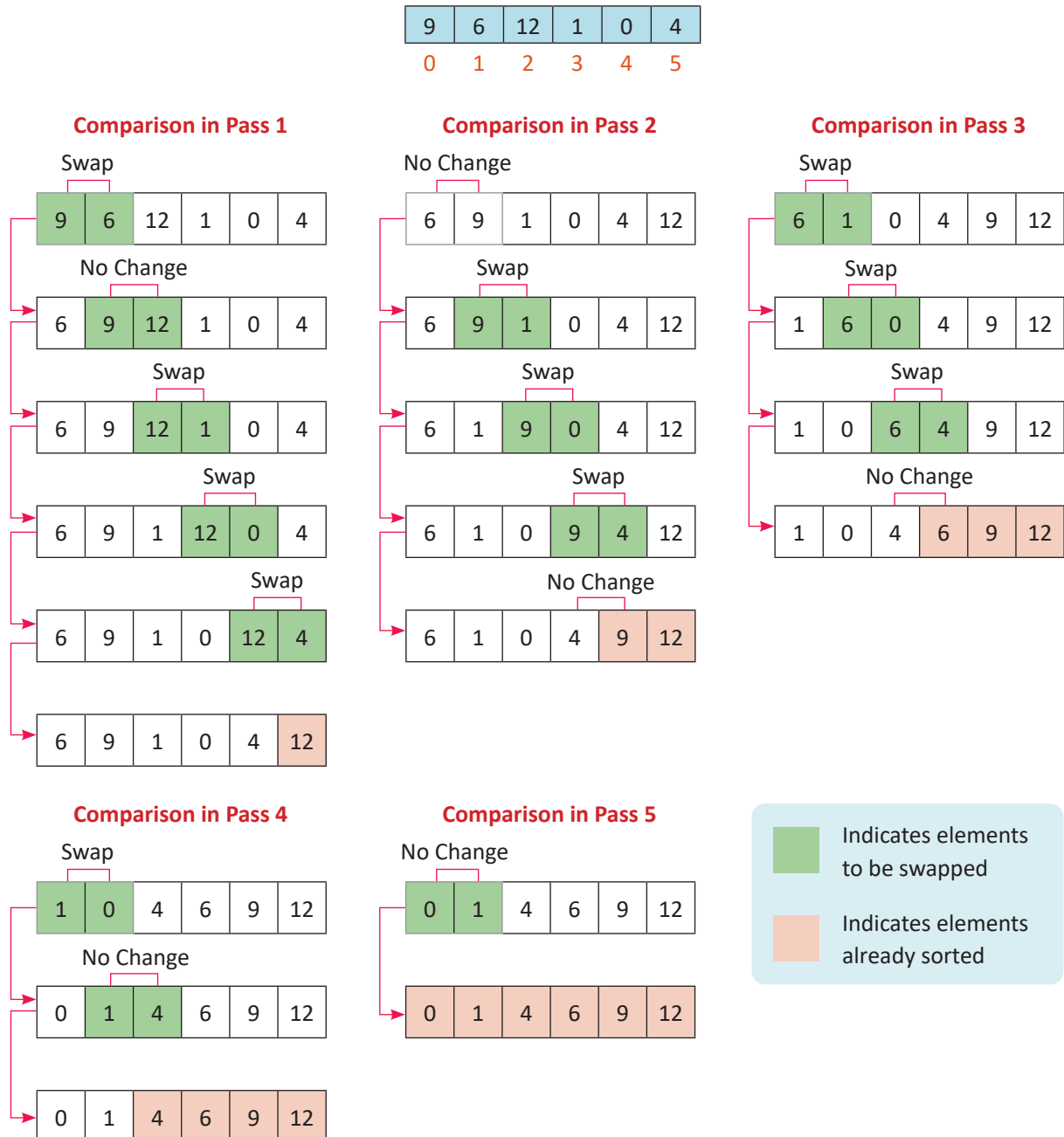


**Figure 8.13:** Bubble Sort

## Program 22

Write a program to input 10 numbers in a single-dimensional array. Using the bubble sort technique, arrange the elements of the array in ascending order and display the sorted array.

```
1. import java.util.*;
2. class bub_sort1
3. {
4. public static void main(String args[])
5. {
6. Scanner input=new Scanner(System.in);
7. int i,j,t;
8. int n[]=new int[10];
9. System.out.println("Enter the elements:");
10. for(i=0;i<10;i++)
11. {
12. n[i]=input.nextInt();
13. }
14. for(i=0;i<9;i++)
15. {
16. for(j=0;j<9-i;j++)
17. {
18. if(n[j]>n[j+1])
19. {
20. t=n[j];
21. n[j]=n[j+1];
22. n[j+1]=t;
23. }
24. }
25. }
26. System.out.println("Sorted array:");
27. for(i=0;i<10;i++)
28. {
29. System.out.println(n[i]);
30. }
31. }
32. }
```

### OUTPUT

Enter the elements:

23

4

12

35

41

29

89

47

50

11

Sorted array:

4

11

12

23

29

35

41

47

50

89

### Program 23

Write a program to input 10 names in a single-dimensional array. Using the bubble sort technique, arrange the names of the array in the descending order and display the sorted array.

```
1. import java.util.*;
2. class bub_sort2
3. {
4. public static void main(String args[])
5. {
6. Scanner input=new Scanner(System.in);
7. int i,j;
8. String t="";
9. String n[]=new String[10];
10. System.out.println("Enter 10 names:");
11. for(i=0;i<10;i++)
12. {
13. n[i]=input.nextLine();
14. }
15. for(i=0;i<9;i++)
16. {
17. for(j=0;j<9-i;j++)
18. {
19. if(n[j].compareTo(n[j+1])<0)
20. {
21. t=n[j];
22. n[j]=n[j+1];
23. n[j+1]=t;
24. }
25. }
26. }
27. System.out.println ("Sorted array in descending order:");
28. for(i=0;i<10;i++)
29. {
30. System.out.println(n[i]);
31. }
32. }
33. }
```

#### OUTPUT

```
Enter 10 names
India
Afghanistan
Russia
China
Bangladesh
Japan
NewZealand
Nepal
Holland
Pakistan
Sorted array in descending
order:
Russia
Pakistan
NewZealand
Nepal
Japan
India
Holland
China
Bangladesh
Afghanistan
```



### Let's Revise

- ◆ An array is a group of similar types of data items (homogeneous) that share a common name. It is declared using square brackets [ ].
- ◆ The elements of an array are stored in a contiguous memory location.

- ◆ An array is a composite data type. It is created using primitive data types.
- ◆ The number of elements in an array can be found by using the 'length' property of the array.
- ◆ There are two types of arrays: single-dimensional arrays and double-dimensional arrays.
- ◆ A single-dimensional array contains elements of the same data type in a contiguous manner. It represents one row of elements that can be accessed by their indices or subscripts.
- ◆ A two-dimensional array represents data in a tabular form, in which data is stored in rows and columns.
- ◆ You can access the elements of one-dimensional or two-dimensional arrays using their indices. The index of the array starts with 0 and ends with n-1, where 'n' is the size of the array.
- ◆ The two most commonly used searching techniques in Java are linear search and binary search.
- ◆ Selection sort and bubble sort are the two most commonly used sorting techniques in Java.

## Key Terms

**Index:** An index value is assigned to the array variables to keep the order in which the items are kept in the array.

**Searching:** The process of finding a particular element in an array is known as searching.

**Sorting:** Sorting is a method of arranging the elements in ascending or descending order.

**Linear Search:** Linear search or sequential search is a method for finding an element within a list. It sequentially checks each element of the list until a match is found or the whole list has been searched.

**Binary Search:** It is the process of searching a sorted array by repeatedly dividing the search interval in half.

**Selection Sort:** Selection sort is a technique in which successive elements are selected in an order and placed into their proper sorted positions.

**Bubble Sort:** A bubble sort is a sorting algorithm in which each pair of adjacent elements is compared, and the elements are swapped if they are not in order.

**Out of Bound:** When you try to access the elements of the array with an index number that is either equal to or greater than the length of the array, this situation is known as 'array out of bound'.



## SECTION 1 (Solved Questions)

### A. Fill in the blanks.

1. Given an array of `int x [ ] = {11, 22, 33, 44}`, the value of `x [1]` is \_\_\_\_\_.
2. The elements of `num[10]` are numbered from \_\_\_\_\_ to \_\_\_\_\_.
3. \_\_\_\_\_ is the process of checking whether an element is present in the array or not.
4. \_\_\_\_\_ is done to arrange the elements in ascending or descending order.

5. The \_\_\_\_\_ property of an array is used to find the number of elements of the array.
6. Each comparison in a \_\_\_\_\_ technique eliminates half of the remaining values under consideration.
7. An array that uses two subscripts is known as \_\_\_\_\_.

|             |           |                  |                          |            |
|-------------|-----------|------------------|--------------------------|------------|
| <b>Ans.</b> | 1. 22     | 2. 0, 9          | 3. Searching             | 4. Sorting |
|             | 5. length | 6. Binary search | 7. Two-dimensional array |            |

## B. Very short answer type questions.

1. What is an array?

**Ans.** An array is a structure created in the memory. It is used to store large amount of data using a single variable name.

2. Which keyword is used to get memory allocated for an array?

**Ans.** The new operator is used to allocate memory to an array.

3. What is subscript?

**Ans.** The element number in an array is called the subscript.

4. What is the purpose of subscript in an array?

**Ans.** The subscript locates the element's position within the array.

5. What are the two types of arrays?

**Ans.** Single-dimensional arrays and double-dimensional arrays are the two types of arrays.

6. What is the need of arrays? Explain.

**Ans.** Arrays are needed to store a large amount of data using a single variable name so that you are able to solve the problems in a much better and more efficient way.

## C. Short answer type questions.

1. 'Array is a composite data type'. Explain this statement.

**Ans.** The data type that represents a number of similar or different data under a single declaration is called a composite data type. An array is a group of the same type of variables. Hence, an array is called a composite data type.

2. Differentiate between subscript and subscripted variables.

**Ans.** Subscript is the index of the element in the array, whereas the subscripted variable is the name of the array when it is used with a subscript to access a single element of the array.

3. Differentiate between char a[5] and int a[5].

**Ans.** The statement char a[5] defines an array a of character type that contains 5 characters in it. On the other hand, the statement int a[5] defines an array of integer type that contains 5 integers in it.

4. Define searching. List any two searching techniques.

**Ans.** The process of finding a particular element in an array is known as searching. There are two main techniques for searching in Java – linear search and binary search.

5. What is sorting? Explain the process of selection sort.

**Ans.** Sorting is a method of arranging the elements in ascending or descending order. Selection sort is a technique in which successive elements are selected in an order and placed into their proper sorted positions.

#### D. Application-based questions.

1. Consider the following code and give the output:

```
String c[]={"DEHRADUN","NAINITAL","SHIMLA","KASHMIR","RANIKHET"};
System.out.println(c[0].length()>c[3].length());
System.out.println(c[4].substring(0,4));
```

Ans. True

RANI

2. Write the statement to assign 25 to the second element of an array N[10].

Ans. N[1]=25

3. Consider the following code snippet and write the output.

```
char C[]={'j','a','v','a'};
int n=2;
System.out.println(C[n++]);
System.out.println(C[n++]);
```

Ans. v

a

## SECTION 2 (Unsolved Questions)

#### A. Very short answer type questions.

1. Find errors in the following code and rewrite them correctly:

- a. `int ar[]=new int[6];`  
`ar={6,7,8,1,2,3};`
- b. `int ar[5]={1,2,3,4,5};`
- c. `int []ar=new int[5];`
- d. `int ar[]= (4,5,6,7,8);`
- e. `int ar[]=new double[4.5];`

2. Consider the following array and answer all the questions given below:

```
int array[] = { 2,75,26,68,79,90,1,4}
```

- a. What is the subscript of 68 and 4?
- b. What is the total size of the array?
- d. What are the start and end indices?
- e. Write the elements stored at 3rd and 5th indices.

#### B. Short answer type questions.

1. Write the code statements for the following:

- a. To write the total size in bytes, of the array a[4] of char type and p[4] of float data type.
- b. To declare a single subscripted variable of float type data with size 20.
- c. To declare a single-dimensional array to store the first 5 even numbers.
- d. To store the name of 10 actresses in a single-dimensional array.
- e. To declare a matrix of 4x4.

- f. Find and print the middle index and the element present at the middle index.
  - g. Increase the value of the third element of the array by 14.
  - h. Copy the entire array in another array temp.
2. What is the difference between single-dimensional and two-dimensional arrays.
  3. Describe the 'out of bound' exception.
  4. How do you find the length of an array?
  5. Distinguish between linear and binary search techniques.
  6. Explain selection sort.
  7. How do you find the middle element of an array?
  8. Describe the bubble sort technique.

### C. Application-based questions.

1. Write the output of the following code snippets:
  - a. 

```
int x[] = { 5,6,7,8};
int y[]={1,2,3,4};
y=x;
System.out.println("x[2] = "+x[2]);
System.out.println("y[2] = "+y[2]);
```
  - b. 

```
int x[]={6,8,2,4}
for(int i=1;i<=2;i++)
System.out.println(x[i] + x[3-i]);
```
  - c. 

```
int ar[]=new int[5];
int i=0,n=48316;
while(n>0)
{
ar[i++]=n%10;
n/=10;
}
for(int j=i-1;j>=0;j--)
System.out.println(ar[j]);
```
  - d. 

```
int notes[]={ 1000, 500, 100, 50, 20, 10, 5, 2, 1};
int i, n,amt=8888;
for(i=0;i<9;i++)
{
n= amt/notes[i];
if(n!=0)
System.out.println ("The number of notes "+notes[i] +" = "+n);
amt=amt%notes[i];
}
```

- e. Consider the String `s[]={"SAMSUNG", "NOKIA", "SONY", "MICROMAX", "BLACKBERRY"};`  
Give the output of the following statements:  
`System.out.println(x[1]);`  
`System.out.println(x[3].length());`
- Write a program that accepts 10 numbers in an array and searches for a value entered by the user using linear search. If the searched item is found, print "Search successful", otherwise print "Search failed".
  - Write a program that accepts 10 numbers in an array in an ascending order and searches for a value entered by the user using the binary search technique. If the searched item is found, print "Search successful", otherwise print "Search failed".
  - Write a program to accept the year of passing from school as an integer value from the user. Find a particular year using the binary search technique on the sorted array of integers. Display "Record exists", if the input value is located in the array. If not, output the message "Record does not exist".  
Sample array: [2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015]
  - Write a program that accepts 10 numbers in an array. Arrange it in ascending order using selection sort. Print all the elements of the array.
  - Write a program that accepts the name and telephone number of 10 people. Arrange them in alphabetical ascending order using bubble sort.
  - Write a program to input 100 integers into an array named `ar[ ]`. Arrange the first 50 integers in descending order using bubble sort and the next 50 integers in ascending order using selection sort. Print the unsorted and sorted arrays.
  - Write a program to input age of 20 people and find the number of people in each category given below: 1 to 20, 21 to 40, 41 to 60, 61 and above.

### SECTION 3 (Board Questions)

- State the total size in bytes of the arrays `a[4]` of char data type and `p[4]` of float data type. (2011)  
**Ans.** Size of char `a[4]` is  $4 \times 2 = 8$  Bytes  
Size of float `p[4]` is  $4 \times 4 = 16$  Bytes
- Find the errors in the given program segment and rewrite the statements correctly to assign values to an integer array. (2014)  
`int a = new int (5);`  
`for (int i=0; i<=5; i++) a[i]=i;`  
**Ans.** Corrected code:  
`int a[] = new int[5];`  
`for (int i = 0; i < 5; i++)`  
`a[i] = i;`
- What will be the output of the following code? (2015)  
`int arr[]=new int[5];`  
`System.out.println(arr);`  
**Ans.** This code will print garbage value.
- If `int x[] = {4, 3, 7, 8, 9, 10};` what are the values of 'p' and 'q'? (2016)  
a. `p = x.length`  
b. `q=x[2]+x[5]*x[1]`



Ans. a.  $p = 6$

b.  $q = x[2] + x[5] * x[1]$

$q = 7 + 10 * 3$

$q = 7 + 30$

$q = 37$

5. `String x[] = {"SAMSUNG", "NOKIA", "SONY", "MICROMAX", "BLACKBERRY"};`

(2017)

Give the output of the following statements:

`System.out.println(x[1]);`

`System.out.println(x[3].length());`

Ans. NOKIA

8

6. `String x[] = {"Artificial intelligence", "IOT", "Machine learning", "Big data"};`

(2019)

Give the output of the following statements:

`System.out.println(x[3]);`

`System.out.println(x.length);`

Ans. Big data

4

## SECTION 4 (Case Study Based Questions)

### Case Study I

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value. To declare an array, define the variable type with square brackets. You access an array element by referring to the index number. Array indexes start with 0, [0] indicates the first element, [1] the second element, and so on. In an array, you can change the value of a specific element by using its index number. Each array has a length property. Using that, you can find how many elements an array has. You can iterate through the array elements with the for loop.

- An array is a structure created in the \_\_\_\_\_ used to store a large amount of data using a single variable name.
  - compiler
  - source code
  - memory
  - variable
- `A[10]` refers to the \_\_\_\_\_ element.
  - 9<sup>th</sup>
  - 10<sup>th</sup>
  - 11<sup>th</sup>
  - 12<sup>th</sup>
- A single-dimensional array has \_\_\_\_\_ subscript whereas a two-dimensional array has \_\_\_\_\_ subscripts.
  - 1, 2
  - 2, 1
  - 1, 3
  - 2, 2
- Array is subscripted using \_\_\_\_\_ symbol.
  - { }
  - [ ]
  - ( )
  - ||

Ans. 1. c

2. c

3. a

4. b

## Case Study II

In Java, arrays can be declared by using square brackets [ ] after the name of the array. When an array is declared, only a reference to the array is created. You can allocate memory to an array using the new operator as given: `var_name = new type [size];`

Here, `var_name` is the name of the array variable, `type` specifies the type of data being allocated, and `size` specifies the number of elements in the array.

Example: `int[] marks = new int[50];`

The above given statement creates an array 'marks' that can store 50 elements. Elements of arrays are always stored in dynamic memory.

1. The \_\_\_\_\_ operator allocates memory to an array.
  - a. create
  - b. new
  - c. inspect
  - d. allocate
2. Array is a \_\_\_\_\_ data type.
  - a. Primitive
  - b. Non-primitive
  - c. Dependent
  - d. Simple
3. \_\_\_\_\_ search is also known as sequential search.
  - a. Linear
  - b. Binary
  - c. Selection
  - d. Bubble
4. The \_\_\_\_\_ locates the element's position within the array.
  - a. Array size
  - b. Post script
  - c. Superscript
  - d. Sub script

Ans. 1. b      2. b      3. a      4. d

## Programming ONE

1. Write a program to perform a binary search on the list of integers given below, to search for an element input by the user. If it is found, display the element along with its position, otherwise display the message "Search element not found".

[5, 7, 9, 11, 15, 20, 30, 45, 89, 97]

Ans. 

```
import java.util.*;

public class BinarySearch
{
 public static void main(String args[])
 {
 Scanner in = new Scanner(System.in);
 int arr[] = {5, 7, 9, 11, 15, 20, 30, 45, 89, 97};
 System.out.print("Enter the number to search: ");
 int n=in.nextInt();
 int l=0, h=arr.length-1, index=-1;
 while (l <= h)
```

```

{
 int m = (l + h) / 2;
 if (arr[m] < n)
 l = m + 1;
 else if (arr[m] > n)
 h = m - 1;
 else
 {
 index = m;
 break;
 }
}
if (index == -1) {
 System.out.println("Search element not found");
}
else
{
 System.out.println(n + "Search element found at position " + index);
}
}
}

```

2. Write a program to store 6 elements in an array 'P' and 4 elements in an array 'Q'. Now, produce a third array 'R' containing all the elements of arrays 'P' and 'Q'. Display the resultant array.

Ans. `import java.util.*;`

```

public class merger
{
 public static void main(String args[]) {
 Scanner in = new Scanner(System.in);
 int P[] = new int[6];
 int Q[] = new int[4];
 int R[] = new int[10];
 int i = 0;
 System.out.println("Enter 6 elements of array P:");
 for (i = 0; i < P.length; i++)
 {
 P[i] = in.nextInt();
 }
 System.out.println("Enter 4 elements of array Q:");
 for (i = 0; i < Q.length; i++)
 {
 Q[i] = in.nextInt();
 }
 }
}

```

```

i = 0;
while(i < P.length)
{
 R[i] = P[i];
 i++;
}
int j = 0;
while(j < Q.length)
{
 R[i++] = Q[j++];
}
System.out.println("Elements of array R:");
for (i = 0; i < R.length; i++)
{
 System.out.print(R[i] + " ");
}
}
}

```

3. Write a program to input the weight of ten people. Sort and display them in descending order using the selection sort technique.

Ans. `import java.util.Scanner;`  
`public class SelectionSort`  
`{`  
 `public static void main(String args[]) {`  
 `Scanner in = new Scanner(System.in);`  
 `double w[] = new double[10];`  
 `System.out.println("Enter weights of 10 people: ");`  
 `for (int i = 0; i < 10; i++)`  
 `{`  
 `w[i] = in.nextDouble();`  
 `}`  
 `for (int i = 0; i < 9; i++) {`  
 `int idx = i;`  
 `for (int j = i + 1; j < 10; j++)`  
 `{`  
 `if (w[j] > w[idx])`  
 `idx = j;`  
 `}`  
 `double t = w[i];`  
 `w[i] = w[idx];`  
 `w[idx] = t;`  
 `}`  
 `System.out.println("Sorted Weights Array:");`

```

 for (int i = 0; i < 10; i++)
 {
 System.out.print(w[i] + " ");
 }
 }
}

```

4. Write a program to accept the names of 10 cities in a single-dimensional string array and their STD (Subscribers Trunk Dialling) codes in another single-dimensional integer array. Search for the name of a city input by the user in the list. If found, display "Search Successful" and print the name of the city along with its STD code, else display the message "Search unsuccessful, no such city in the list".

Ans. `import java.util.*;`

```

public class StdCodes
{
 public static void main(String args[]) {
 int SIZE = 10;
 Scanner in = new Scanner(System.in);
 String cities[] = new String[SIZE];
 String stdCodes[] = new String[SIZE];
 System.out.println("Enter " + SIZE +
 " cities and their STD codes:");
 for (int i = 0; i < SIZE; i++)
 {
 System.out.print("Enter City Name: ");
 cities[i] = in.nextLine();
 System.out.print("Enter its STD Code: ");
 stdCodes[i] = in.nextLine();
 }
 System.out.print("Enter name of city to search: ");
 String city = in.nextLine();
 int idx;
 for (idx = 0; idx < SIZE; idx++)
 {
 if (city.compareToIgnoreCase(cities[idx]) == 0)
 {
 break;
 }
 }
 if (idx < SIZE)
 {
 System.out.println("Search Successful");
 System.out.println("City: " + cities[idx]);
 System.out.println("STD Code: " + stdCodes[idx]);
 }
 }
}

```

```

 }
 else
 {
 System.out.println("Search Unsuccessful");
 }
}
}

```

5. Write a program to accept the 'year of graduation' as an integer value from the user. Search for the year entered by the user using the binary search technique on the sorted array of integers: `n[]={1982, 1987, 1993, 1996, 1999, 2003, 2006, 2007, 2009, 2010}`; Display the message "Record exists" if the year exists in the array. If not, display the message "Record does not exist".

Ans. `import java.util.*;`  
`public class year`  
`{`  
 `public static void main(String args[])`  
 `{`  
 `Scanner in = new Scanner(System.in);`  
 `int n[] = {1982, 1987, 1993, 1996, 1999, 2003, 2006, 2007, 2009, 2010};`  
 `System.out.print("Enter graduation year to search: ");`  
 `int year = in.nextInt();`  
 `int l = 0, h = n.length - 1, idx = -1;`  
 `while (l <= h)`  
 `{`  
 `int m = (l + h) / 2;`  
 `if (n[m] == year)`  
 `{`  
 `idx = m;`  
 `break;`  
 `}`  
 `else if (n[m] < year)`  
 `{`  
 `l = m + 1;`  
 `}`  
 `else`  
 `{`  
 `h = m - 1;`  
 `}`  
 `}`  
 `if (idx == -1)`  
 `System.out.println("Record does not exist");`  
 `else`  
 `System.out.println("Record exists");`  
 `}`  
`}`

6. Write a program to input 20 names in an array. Arrange these names in descending order of letters using the bubble sort technique.

Ans. `import java.util.*;`  
`public class names`  
`{`  
 `public static void main(String args[])`  
 `{`  
 `Scanner in = new Scanner(System.in);`  
 `String nm[] = new String[20];`  
 `System.out.println("Enter 20 names:");`  
 `for (int i = 0; i < nm.length; i++)`  
 `{`  
 `nm[i] = in.nextLine();`  
 `}`  
 `for (int i = 0; i < nm.length - 1; i++)`  
 `{`  
 `for (int j = 0; j < nm.length - 1 - i; j++)`  
 `{`  
 `if (nm[j].compareToIgnoreCase(nm[j + 1]) < 0)`  
 `{`  
 `String temp = nm[j + 1];`  
 `nm[j + 1] = nm[j];`  
 `nm[j] = temp;`  
 `}`  
 `}`  
 `}`  
 `System.out.println("\nSorted Names");`  
 `for (int i = 0; i < nm.length; i++)`  
 `{`  
 `System.out.println(nm[i]);`  
 `}`  
 `}`  
`}`

7. Write a program to input the seven wonders of the world along with their locations into two different arrays. Search for the name of the country entered by the user. If found, display the name of the country along with its wonder, otherwise display "Sorry not found!".

Seven Wonders:

Chichen Itza, Christ the Redeemer, Taj Mahal, Great Wall of China, Machu Picchu, Petra, Colosseum

Locations:

Mexico, Brazil, India, China, Peru, Jordan, Italy

Examples:

Country name: INDIA

Output: TAJ MAHAL

Country name: USA

Output: Sorry Not found!

```

Ans. import java.util.*;
public class wonderful
{
 public static void main(String args[])
 {
 String wonders[] = {"CHICHEN ITZA", "CHRIST THE REDEEMER", "TAJMAHAL",
 "GREAT WALL OF CHINA", "MACHU PICCHU", "PETRA",
 "COLOSSEUM"};
 String locations[] = {"MEXICO", "BRAZIL", "INDIA", "CHINA", "PERU", "JORDAN",
 "ITALY"};
 Scanner in = new Scanner(System.in);
 System.out.print("Enter country: ");
 String c = in.nextLine();
 int i;
 for (i = 0; i < locations.length; i++)
 {
 if (locations[i].equals(c))
 {
 System.out.println(locations[i] + " - " + wonders[i]);
 break;
 }
 }
 if (i == locations.length)
 System.out.println("Sorry Not Found!");
 }
}

```

8. Write a program to input numbers into a 4 x 5 array and display the sum of each row of the array.

```

Ans. import java.util.*;
class dda_sum_row
{
 public static void main(String args[])
 {
 Scanner input=new Scanner(System.in);
 int a[][]=new int[4][5];
 int i,j,s;
 for(i=0;i<4;i++)
 {
 for(j=0;j<5;j++)
 {
 System.out.println("Enter a number");
 a[i][j]=input.nextInt();
 }
 }
 }
}

```



```

 for(i=0;i<4;i++)
 {
 s=0;
 for(j=0;j<5;j++)
 {
 s=s+a[i][j];
 }
 System.out.println("Sum of row" +(i+1)+ "=" +s);
 }
 }
}

```

9. Write a program that allows the user to specify the size of a matrix's rows and columns. It also takes integer values as user input and displays all of the values of the matrix.

Ans. **import** java.util.\*;  
**class** dda\_sum\_column  
{  
**public static void** main(String args[])  
 {  
 Scanner input=new Scanner(System.in);  
 int r,c,i,j,s;  
 System.out.println("Enter the number of rows");  
 r=input.nextInt();  
 System.out.println("Enter the number of columns");  
 c=input.nextInt();  
**int** n[][]=new **int**[r][c];  
 for(i=0;i<r;i++)  
 {  
 for (j=0;j<c;j++)  
 {  
 System.out.println("Enter a number");  
 n[i][j]=input.nextInt();  
 }  
 }  
 for(i=0;i<c;i++)  
 {  
 s=0;  
 for (j=0;j<r;j++)  
 {  
 s=s+n[j][i];  
 }  
 System.out.println("Sum of column" +(i+1)+ "=" +s);  
 }  
 }  
}

10. Write a program to input numbers in a 4x4 array and display the sum of its left and right diagonals.

Ans. `import java.util.*;`

```
class dda_sumofd
{
 public static void main(String args[])
 {
 Scanner input=new Scanner(System.in);
 int i,j,sl=0,sr=0;
 int n[][]=new int[4][4];
 for(i=0;i<4;i++)
 {
 for (j=0;j<4;j++)
 {
 System.out.println("Enter a number");
 n[i][j]=input.nextInt();
 }
 }
 for(i=0;i<4;i++)
 {
 for (j=0;j<4;j++)
 {
 System.out.print(n[i][j]+" ");
 }
 System.out.println();
 }
 for(i=0;i<4;i++)
 {
 sl=sl+n[i][i];
 sr=sr+n[i][3-i];
 }
 System.out.println("sum of right diagonal="+sr);
 System.out.println("sum of left diagonal="+sl);
 }
}
```



**Link**

Scan the given QR Code to learn how to access the elements of an array.



## Chapter Highlights

- 9.1 Strings
- 9.2 Methods of String Class
- 9.3 Implementation of String Class Methods
- 9.4 String Array

## Learning Outcomes

## The student will be able to:

- understand about strings and how to create them.
- know how to use various string handling functions.
- create string based programs.

Java provides the String class to create and process strings. A string is a group of characters. It is a non-primitive data type, as it is a class in Java. A string is also referred to as a composite data type because it is composed of character type of data. Java has many library methods to perform various functions on strings. These methods also ease the process of string handling. The information about these methods is present in the **java.lang** package. This package is loaded by default whenever a Java program is loaded.

## 9.1 Strings

A string is a group of characters enclosed within double quotes. It may contain uppercase letters, lowercase letters, digits, blank spaces, and special characters. For example, "Programming@Java", "Computer Application", etc.

## Creating and Using String Objects

In order to use strings in your program, you need to create objects of the String class, which is a member of the **java.lang** package. You can do so by using the following two methods:

- ♦ Using the new operator
- ♦ Using String literal

```
String Word;
```

```
Word = "HELLO";
```

Index/Subscript

|   |   |   |   |   |
|---|---|---|---|---|
| H | E | L | L | O |
| 0 | 1 | 2 | 3 | 4 |

Figure 9.1: Strings in Memory

Let us discuss these methods in detail.

## Using the new Operator

As you create objects of other classes using the new operator, in a similar manner, you can create an object of the String class using the new operator. This method is known as the explicit method of creating strings.

### Let's Know More

String objects created using string literals are stored in the **String Constant Pool** and string objects created using the new operator are stored in the **Heap memory**.

### Know the Fact

Whenever you create a string object using a string literal, the JVM first checks the content of the object to be created. If the object already exists in the String Constant Pool with the same content, then it returns the reference to that object and does not create the new object. If the content is different from the existing objects, then it will only create the new object.

### Read More

String objects in Java are immutable. That means, once you create String objects, you can't modify them. If you try to modify them, a new object will be created with modifications.

### Fact File

Unlike in C and C++, strings in Java are not terminated with a null character. Strings are treated as objects in Java.

**Syntax:** `String stringName = new String( );`

Here, `String( )` is the default constructor of the String class.

You can pass any string to the `String( )` constructor and a string will be created.

**Example:** `String message = new String("Programming");`

### Using String Literal

This method is known as the implicit method of creating strings. Here, the String literal is used to create the strings like any of the other variables in Java. You can write a group of characters and enclose them in double quotes. The Java compiler then creates an object of the String class and assigns it the value of the string.

**Syntax:** `String stringName = "group_of_characters";`

**Example:** `String message = "Programming";`

## 9.2 Methods of String Class

A string is a set of characters enclosed in double-quotes. Java includes some built-in methods in String class that are used for manipulating strings as listed below:

### length( )

This function returns the number of characters present in a string, including whitespaces. Its return type is 'int'.

**Syntax:** `String_variable.length();`

**Example:** `String a="Java";  
int l=a.length();`

**Output:** 4

### charAt(int n)

This function returns the character present at the specified position (index) in a string. Its return type is 'char'.

**Syntax:** `String_variable.charAt(index position);`

**Example:** `String a="Java";  
char c=a.charAt(2);`

**Output:** v

### indexOf(char ch)

This function returns the position (index) of the first occurrence of a specified character in a string. If the character is not present in the string, the method will return -1. The return type of the method is 'int'.

**Syntax:** `String_variable.indexOf(char);`



```
Example: String a="Java";
 int c=a.indexOf('a');
```

**Output:** 1

### lastIndexOf(char ch)

This function returns the position (index) of the last occurrence of a specified character in a string. If the character is not present in the string, the method will return -1. The return type of the method is 'int'.

**Syntax:** String\_variable.lastIndexOf(char);

```
Example: String a="Java";
 int c=a.lastIndexOf('a');
```

**Output:** 3

### toUpperCase()

This function converts a string to uppercase. All the lowercase letters get converted to uppercase, but the rest of the characters remain unchanged. Its return type is 'String'.

**Syntax:** String\_variable.toUpperCase( );

```
Example: String a="Java";
 a.toUpperCase();
```

**Output:** JAVA

### toLowerCase()

This function converts a string to lowercase. All the uppercase letters get converted to lowercase, but the rest of the characters remain unchanged. Its return type is 'String'.

**Syntax:** String\_variable.toLowerCase();

```
Example: String a="Java";
 a.toLowerCase();
```

**Output:** java

### equals(String str)

This function checks whether two strings are the same or not and returns true or false accordingly. It is case-sensitive, i.e., for this method, the corresponding uppercase and lowercase characters are different. That means, 'G' and 'g' are not equal. Its return type is 'boolean'.

**Syntax:** String\_Variable\_1.equals(String\_Variable\_2);

```
Example: String a="Java";
 String b="JAVA";
 String c="COBOL";
 a.equals(b);
```

**Output:** false

```
b.equals(c);
```

**Output:** false

### **equalsIgnoreCase(String str)**

This function also checks whether two strings are the same or not, but it ignores case, i.e., uppercase or lowercase. Its return type is 'boolean'.

**Syntax:** String\_Variable\_1.equalsIgnoreCase(String\_Variable\_2);

**Example:** String a="Java";

```
String b="JAVA";
```

```
String c="COBOL";
```

```
a.equalsIgnoreCase(b);
```

```
b.equalsIgnoreCase(c);
```

**Output:** true

false

### **replace(char oldChar, char newChar)**

This function replaces all occurrences of a particular character or a group of characters with another character or group of characters in the string. Its return type is 'String'.

**Syntax:** String\_Variable.replace(char oldChar, char newChar);

**Example:** String a="bhaskar";

```
a.replace('a','p');
```

**Output:** bhpskpr

### **concat(String str)**

This function concatenates (combines) two strings together to produce a new string. Its return type is 'String'.

**Syntax:** String\_Variable\_1.concat(String\_Variable\_2);

**Example:** String a="bhaskar"; String b="bajaj";

```
a.concat(b);
```

**Output:** bhaskarbajaj

### **trim( )**

This function removes blank spaces from the beginning or end of a string. Its return type is 'String'.

**Syntax:** String\_variable.trim( );

**Example:** String a=" bhaskar ";

```
a.trim();
```

**Output:** bhaskar

### substring(int beginIndex)

This function returns a new string that is a substring of the given string. The substring begins with the character at the specified start index and extends to the end of the string. Its return type is 'String'.

**Syntax:** String\_variable.substring(beginIndex);

**Example:** String a="BHASKAR";  
a.substring(2);

**Output:** ASKAR

### substring (int beginIndex, int endIndex)

This function also returns a new string that is a substring of the given string. The substring begins with the character at the specified start index and extends to one less than the ending index. Its return type is 'String'.

**Syntax:** String\_variable.substring(beginIndex, endIndex);  
a.substring(2,5);

**Output:** ASK

### compareTo(String str)

This function lexicographically (means dictionary order) compares two strings. It not only checks the equality of strings but also checks whether a string is bigger or smaller than the other string. The comparison is based on each character's Unicode value in the strings. Its return type is 'int'. The method returns 0 if the string is equal to the other string. A negative value (less than 0) is returned if the string has fewer characters than the other string, and a positive value (a value greater than 0) is returned if the string is longer than the other string (more characters).

**Syntax:** String\_Variable\_1.compareTo(String\_Variable\_2);

**Example 1:** String a = "Hello";  
String b = "Hi";  
a.compareTo(b);

**Output 1:** -4

**Example 2:** String a = "Hello";  
String b = "Hello";  
a.compareTo(b);

**Output 2:** 0

### compareToIgnoreCase(String str)

This method also compares two strings lexicographically, but it ignores the lowercase and uppercase differences. This comparison is based on the Unicode value of each character in the string converted to lowercase. This function returns an integer value.

**Syntax:** String\_Variable\_1.compareToIgnoreCase(String\_Variable\_2);

**Example:** String a = "Hello";  
String b = "HELLO";  
a.compareToIgnoreCase(b);

**Output:** 0

## startsWith(String str)

This function checks whether a string starts with a given substring or not. Its return type is 'Boolean', i.e., the method returns true if the given string starts with the specified prefix, otherwise it returns false.

**Syntax:** String\_Variable.startsWith(substring);

**Example:** String a="BHASKAR"; a.startsWith("BH");

**Output:** true

## endsWith(String str)

This function checks whether a string ends with a given substring or not. Its return type is 'Boolean', i.e., the method returns true if the given string ends with the specified suffix, otherwise it returns false.

**Syntax:** String\_Variable.endsWith(substring);

**Example:** String a="BHASKAR";  
a.endsWith("BH");

**Output:** false

### Let's Know More

ASCII codes for uppercase letters (A–Z) are: 65–90  
ASCII codes for lowercase letters (a–z) are: 97–122  
ASCII codes for digits (0–9) are: 48–57

## valueOf(all types)

This function returns the string representation of the given value, i.e., it converts different types of values into strings. Using this function, you can convert an integer value to a string, a long integer value to a string, a boolean value to a string, a character value to a string, a float value to a string, a double value to a string, an object to a string, and a char array to a string.

**Syntax:** String\_variable.valueOf(input\_value);

**Example:** int value=30;  
String s1=String.valueOf(value);  
System.out.println(s1+10);

**Output:** 3010



## 9.3 Implementation of String Class Methods

Consider the following programs to get familiar with the use of built-in functions for handling strings.

### Program 1

Write a program to input a sentence and print all letters in different lines.

```
1. import java.util.*;
2. public class inputString
3. {
4. public static void main(String args[])
5. {
```



```

6. Scanner input=new Scanner(System.in);
7. String s;
8. int i, l;
9. char c;
10. System.out.println("Enter a string:");
11. s=input.nextLine();
12. l=s.length(); //Returning length of the string
13. for(i=0;i<l;i++)
14. {
15. c=s.charAt(i);
16. System.out.println(c);
17. }
18. }
19. }

```

#### OUTPUT

```

Enter a string:
How are you?

H
o
w
a
r
e
y
o
u
?

```

### Program 2

Write a program to input a sentence and find the number of vowels in it.

```

1. import java.util.*;
2. public class Count_Vowels
3. {
4. public static void main(String args[])
5. {
6. Scanner input=new Scanner(System.in);
7. String s;
8. int i,l,v=0;
9. char c;
10. System.out.println("Enter a sentence:");
11. s=input.nextLine();
12. l=s.length(); //Returning length of the string
13. for(i=0;i<l;i++)
14. {
15. c=s.charAt(i); //Returning character's index
16. if(c=='A' || c=='E' || c=='I' || c=='O' || c=='U' || c=='a' || c=='e' || c=='i'
17. || c=='o' || c=='u')
18. v++;
19. }
20. System.out.println("Number of vowels are:"+v);
21. }

```

#### OUTPUT

```

Enter a sentence:
Hello! How are you?

Number of vowels are: 7

```

### Program 3

Write a program to input a sentence and find the biggest word present in the sentence. Also, print the length of the biggest word.

```
1. import java.util.*;
2. public class Find_Big
3. {
4. public static void main(String args[])
5. {
6. Scanner input=new Scanner(System.in);
7. String s,w="",bw="";
8. int i,l,lw,max=0;
9. char c;
10. System.out.println("Enter a sentence:");
11. s=input.nextLine();
12. s=s+" ";
13. l=s.length();
14. for(i=0;i<l;i++)
15. {
16. c=s.charAt(i);
17. if(c!=' ')
18. w=w+c;
19. else
20. {
21. lw=w.length();
22. if(lw>max)
23. {
24. max=lw;
25. bw=w;
26. }
27. w="";
28. }
29. }
30. System.out.println("Biggest word is="+bw);
31. System.out.println("Length of the b Biggest word is="+max);
32. }
33. }
```

#### OUTPUT

Enter a sentence:

This is a test program.

Biggest word is: program

Length of the biggest word  
is: 7

#### Program 4

Write a program to input a sentence and find the words that begin and end with the same letter. Also, count the number of such words.

```
1. import java.util.*;
2. public class Count_Words
3. {
4. public static void main(String args[])
5. {
6. Scanner input=new Scanner(System.in);
7. String s,w="";
8. int i, l, f=0;
9. char c,c1,c2;
10. System.out.println("Enter a sentence:");
11. s=input.nextLine();
12. s=s.toUpperCase();
13. s=s+" ";
14. l=s.length();
15. for(i=0;i<l;i++)
16. {
17. c=s.charAt(i);
18. if(c!=' ')
19. w=w+c;
20. else
21. {
22. c1=w.charAt(0);
23. c2=w.charAt(w.length()-1);
24. if(c1==c2)
25. {
26. System.out.println(w);
27. f++;
28. }
29. w="";
30. }
31. }
32. System.out.println("The total number of words that begin and end with
 the same letter is:"+f);
33. }
34. }
```

#### OUTPUT

Enter a sentence:

mom fell from the stairs

MOM

STAIRS

The total number of words  
that begin and end with  
the same letter is: 2

### Program 5

Write a program to input a word and arrange it alphabetically.

```
1. import java.util.*;
2. public class Word_Sorting
3. {
4. public static void main(String args[])
5. {
6. Scanner input=new Scanner(System.in);
7. String s,s1="";
8. int i,j,l;
9. char c;
10. System.out.println("Enter a word:");
11. s=input.nextLine();
12. l=s.length();
13. for(i=65;i<=90;i++)
14. {
15. for(j=0;j<l;j++)
16. {
17. c=s.charAt(j);
18. if((int)c==i || (int)c==(i+32))
19. s1=s1+c;
20. }
21. }
22. System.out.println(s1);
23. }
24. }
```

### OUTPUT

Enter a word:  
Program  
agmoPrr

### Program 6

Write a program to input the following sentence:

"January 26 is celebrated as the Republic Day of India."

Reprint the sentence as:

"August 15 is celebrated as the Independence Day of India."

```
1. import java.util.*;
2. public class Chng_Sentence
3. {
4. public static void main(String args[])
5. {
6. Scanner input=new Scanner(System.in);
```

```

7. String s,w="",s1="";
8. int i,l;
9. char c;
10. System.out.println("Enter a sentence:");
11. s=input.readLine();
12. s=s+" ";
13. l=s.length();
14. for(i=0;i<l;i++)
15. {
16. c=s.charAt(i);
17. if(c!=' ')
18. w=w+c;
19. else
20. {
21. if(w.equalsIgnoreCase("January"))
22. w="August";
23. else if(w.equalsIgnoreCase("26"))
24. w="15";
25. else if(w.equalsIgnoreCase("Republic"))
26. w="Independence";
27. s1=s1+" "+w;
28. w="";
29. }
30. }
31. System.out.println("The changed sentence is:");
32. System.out.println(s1);
33. }
34. }

```

#### OUTPUT

```

Enter a sentence:
January 26 is celebrated as
the Republic Day of India
The changed sentence is:
August 15 is celebrated as
the Independence Day of
India

```

## 9.4 String Array

In Java, an array is the most common linear data structure for storing several values of the same data type under the same name. A String array is a collection of strings with a fixed number of values. The String array behaves similarly to other array data types. In a string array, the first element is placed at Index 0, the second element at Index 1, and so on, just like in an integer array. You can declare a String array without size or with size as shown below:

```

String[] stringArray1; //Declaration of the String Array without specifying the size
String[] stringArray2 = new String[2]; //Declaration by specifying the size

```

A string array has the following characteristics:

- ◆ It can be initialised either at the time of declaration or by populating the values after the declaration.
- ◆ The elements can be added to a String array after declaring it.
- ◆ The String array can be iterated using the for loop.
- ◆ The searching and sorting operations can be performed on the String array.

Consider the following program to understand how to initialise a String array and access elements from it.

### Program 7

Write a program to display elements of a string array.

```
1. import java.util.*;
2. class StringArray
3. {
4. public static void main(String args[])
5. {
6. Scanner input=new Scanner(System.in);
7. int i;
8. String States[]=new String [3];
9. for(i=0;i<3;i++)
10. {
11. System.out.println("Enter a state:");
12. States[i]=input.nextLine();
13. }
14. System.out.println("States of India");
15. for(i=0; i<States.length; i++)
16. {
17. System.out.println(States[i]);
18. }
19. }
20. }
```

### OUTPUT

```
States of India
Bihar
Delhi
Punjab
Rajasthan
```

## Character Data in Java

In programming, a character refers to any individual value used to represent text, a number, or a symbol. In Java programs, it is enclosed in single quotes. Considering it as a data type, char is a primitive data type in Java. It occupies 2 bytes of memory space. Wrapper class of char type data is Character. It can use 65536 codes, out of which the first 256 codes are the ASCII (American Standard Code for Information Interchange) codes.

### Syntax:

The syntax to define a character type data is as follows:

```
<Data type><variable > = <'character literal'>
```

**Example:** `char ch = 'B';`

## Character Based Methods in Java

There are many useful character based methods in Java. Some of them are listed below:

**Character.isLetter(ch):** This function checks whether the input character is a letter or not and returns true or false accordingly. Its return type is 'Boolean'.

**Character.isDigit(ch):** This function checks whether the input character is a digit or not and returns true or false accordingly. Its return type is 'Boolean'.

**Character.isLetterOrDigit(ch):** This function returns true if the input character is either a digit or a letter and returns true or false accordingly. Its return type is 'Boolean'.

**Character.isUpperCase(ch):** This function checks whether the input character is an uppercase letter or not and returns true or false accordingly. Its return type is 'Boolean'.

**Character.isLowerCase(ch):** This function checks whether the input character is a lowercase letter or not and returns true or false accordingly. Its return type is 'Boolean'.

**Character.isWhiteSpace(ch):** This function checks whether the input character is a blank space or not and returns true or false accordingly. Its return type is 'Boolean'.

**Character.toUpperCase(ch):** This function converts the input character to an uppercase letter. Its return type is 'char'.

**Character.toLowerCase(ch):** This function converts the input character to a lowercase letter. Its return type is 'char'.

## Character Based Programs

Consider the following programs to get familiar with the use of the above-discussed methods.

### Program 8

Write a program to input a sentence and reprint it after changing the case of each letter.

```
1. import java.util.*;
2. public class Chng_Case
3. {
4. public static void main(String args[])
5. {
6. Scanner input=new Scanner(System.in);
7. String s, s1 = "";
8. int i, l;
9. char c;
```

### OUTPUT

Enter a sentence:

Programming In Java

pROGRAMMING iN jAVA

```

10. System.out.println("Enter a sentence:");
11. s=input.nextLine();
12. l=s.length();
13. for(i=0;i<l;i++)
14. {
15. c=s.charAt(i);
16. if(Character.isUpperCase(c))
17. c=Character.toLowerCase(c);
18. else if(Character.isLowerCase(c))
19. c=Character.toUpperCase(c);
20. else
21. c=c;
22. s1=s1+c;
23. }
24. System.out.println(s1);
25. }
26. }

```

### Program 9

Write a program to input a sentence and reprint it after changing according to the following instructions:

- ◆ uppercase to lowercase
- ◆ lowercase to uppercase
- ◆ digits to #
- ◆ blank space to ?
- ◆ special character to \*

```

1. import java.util.*;
2. public class Chng_Char
3. {
4. public static void main(String args[])
5. {
6. Scanner input=new Scanner(System.in);
7. String s,s1="";
8. int i,l;
9. char c;
10. System.out.println("Enter a sentence:");
11. s=input.nextLine();
12. l=s.length();
13. for(i=0;i<l;i++)
14. {
15. c=s.charAt(i);

```

#### OUTPUT

```

Enter a sentence:
Shayam lives in Delhi
sHAYAM?LIVES?IN?dELHI

```



```

16. if(Character.isUpperCase(c)) //Checking character is uppercase or not
17. c=Character.toLowerCase(c); //Converting character into lowercase
18. else if(Character.isLowerCase(c))//Checking character is lowercase or not
19. c=Character.toUpperCase(c); //Converting character into uppercase
20. else if(c==' ')
21. c='?';
22. else if(Character.isDigit(c)) //Checking character is digit or not
23. c='#';
24. else
25. c='*';
26. s1=s1+c;
27. }
28. System.out.println(s1);
29. }
30. }

```

## Enrich Yourself

### StringBuffer Class

The **StringBuffer** is a peer class of String that has a lot of the same features as strings. String represents fixed-length, immutable character sequences, whereas **StringBuffer** represents growable and writable character sequences.

The main difference between the String and StringBuffer classes is that the **StringBuffer** class is a **mutable** class, unlike the String class, which is **immutable**. The StringBuffer can be changed dynamically. They are preferred when heavy modification of character strings is involved (appending, inserting, deleting, modifying, etc.).

### Constructors of StringBuffer Class

The StringBuffer class has three important constructors as discussed below:

```
StringBuffer m= new StringBuffer();
```

This statement allocates an empty space in the memory to store a string.

```
StringBuffer m= new StringBuffer("JAVA");
```

This statement allocates space in the memory to store a string in the variable m.

```
StringBuffer m= new StringBuffer(25);
```

This statement allocates space in the memory to store a string of maximum 25 characters.

### Functions of StringBuffer

The following are some important functions of the StringBuffer class:

**capacity():** This function returns the current capacity of the StringBuffer.

```
StringBuffer st = new StringBuffer();
System.out.println("capacity : " + st.capacity());
```

**Output:** capacity: 16

```
StringBuffer st = new StringBuffer("Bobby");
System.out.println("capacity : " + st.capacity());
```

**Output:** capacity: 21

```
StringBuffer st = new StringBuffer(50);
System.out.println("capacity : " + st.capacity());
```

**Output:** capacity: 100

**length():** This function returns the length (character count) of the StringBuffer.

```
StringBuffer st = new StringBuffer("Bobby");
System.out.println("length: " + st.length());
```

**Output:** length: 5

**append( ):** This function is used to add a string to another string.

```
StringBuffer st = new StringBuffer("Foot");
StringBuffer st1 = new StringBuffer("Ball");
System.out.println(st.append(st1));
```

**Output:** FootBall

**setCharAt( ):** This function is used to replace a character with another character at the specified index.

```
StringBuffer st = new StringBuffer("FootBall");
System.out.println(st.setCharAt(4, 'P'));
```

**Output:** FootPall

**insert( ):** This function is used to insert a string at specified index into another string.

```
StringBuffer st = new StringBuffer("Narendra Modi");
StringBuffer st1 = new StringBuffer("DamodarDas");
System.out.println(st.insert(8, st1));
```

**Output:** Narendra DamodarDas Modi

**delete( ):** This function is used to delete characters from one index position to another.

```
StringBuffer st = new StringBuffer("FootBall");
System.out.println(st.delete(3, 6));
```

**Output:** Fooll

**setLength( ):** This function is used to set the length of the StringBuffer. If there are more characters, then they are truncated. If there are fewer, then the empty space is filled with garbage value.

```
StringBuffer st = new StringBuffer("FootBall");
System.out.println(st.setLength(3));
```

**Output:** Foo

**reverse( ):** This function is used to reverse a string.

```
StringBuffer st = new StringBuffer("FootBall");
System.out.println(st.reverse());
```

**Output:** llaBttooF

**toString( ):** This function converts to a string representing the data in the current StringBuffer.

```
StringBuffer st = new StringBuffer("FootBall");
String n=st.toString()
System.out.println(n);
```

**Output:** FootBall

**replace(intstart, intend, Stringstr):** This function replaces the characters in a substring of the current StringBuffer with the characters in the specified string.

```
StringBuffer st = new StringBuffer("FootBall");
System.out.println(st.replace(1,5,"uma"));
```

**Output:** Fumall

## Difference Between String and StringBuffer

| String                                       | StringBuffer                                                                |
|----------------------------------------------|-----------------------------------------------------------------------------|
| String type object has fixed length.         | StringBuffer type object has provisions to change the length of the string. |
| New object is created to produce the change. | Changes are made in the same object.                                        |
| It is a general approach of programming.     | It is an advanced approach of programming.                                  |
| Strings are immutable.                       | StringBuffer is mutable.                                                    |

## Let's Revise

The following table summarises the methods of the String class that you have learnt in the chapter:

| Function Name                | Description                                                                                                                                  | Example                                                           |
|------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------|
| int length( )                | It finds the number of characters present in the string.<br><b>Syntax:</b><br>String variable.length( );                                     | n="Singh";<br>n.length();<br><b>Output:</b> 5                     |
| char charAt(int n)           | It finds the character present at the specified index position in the string.<br><b>Syntax:</b><br>String variable.charAt(indexposition);    | n="Singh";<br>n.charAt(3);<br><b>Output:</b> g                    |
| int indexOf<br>(char ch)     | It finds the position of the first occurrence of a specified character in the string.<br><b>Syntax:</b><br>String variable.indexOf(char);    | n="Singh";<br>n.indexOf('n');<br><b>Output:</b> 2                 |
| int lastIndexOf<br>(char ch) | It finds the position of the last occurrence of a specified character in the string.<br><b>Syntax:</b><br>String variable.lastIndexOf(char); | String a="Java";<br>int c=a.lastIndexOf('a');<br><b>Output:</b> 3 |

|                                                   |                                                                                                                                                                                                          |                                                                                                                                |
|---------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| String<br>toUpperCase( )                          | It converts all the characters of the string to uppercase.<br><b>Syntax:</b><br>String variable.toUpperCase( );                                                                                          | n="Singh";<br>n.toUpperCase();<br><b>Output:</b> SINGH                                                                         |
| String<br>toLowerCase( )                          | It converts all the characters of the string to lowercase.<br><b>Syntax:</b><br>String variable.toLowerCase( );                                                                                          | n="Singh";<br>n.toLowerCase();<br><b>Output:</b> singh                                                                         |
| boolean<br>equals(String<br>str)                  | It checks whether two strings are same or not.<br><b>Syntax:</b><br>String variable1.equals(String Variable2);                                                                                           | n1="Computer";<br>n2="Application";<br>n1.equals(n2);<br><b>Output:</b> false                                                  |
| boolean<br>equalsIgnoreCase<br>(String str)       | It checks whether two strings are equal or not but it is not case sensitive.<br><b>Syntax:</b><br>String variable1.equalsIgnoreCase(String Variable2);                                                   | n1="computer";<br>n2="COMPUTER";<br>n1.equalsIgnoreCase(n2);<br><b>Output:</b> true                                            |
| String replace<br>(char oldChar,<br>char newChar) | It replaces all occurrences of a particular character or a group of characters with another character or group of characters in a string.<br><b>Syntax:</b><br>String variable.replace(oldChar, newChar) | String n="Mr. Ajit is<br>principal of sxd";<br>n.replace("Ajit","Sanjay");<br><b>Output:</b> Mr. Sanjay is<br>principal of sxd |
| String concat<br>(String str)                     | It combines two strings together to produce the resultant new string. Since concat joins without space.<br><b>Syntax:</b><br>String variable1.concat(string variable2);                                  | n1="Bytes Of";<br>n2="Knowledge";<br>n1.concat(n2);<br><b>Output:</b> Bytes OfKnowledge                                        |
| String trim( )                                    | It removes any blank spaces from the beginning or at the end of the string.<br><b>Syntax:</b><br>String variable.trim( );                                                                                | n="      computer      ";<br>n.trim();<br><b>Output:</b> computer                                                              |
| String substring<br>(int beginIndex)              | It returns a part of the string.<br><b>Syntax:</b><br>String variable.substring( );                                                                                                                      | n="computer";<br>n.substring(2);<br><b>Output:</b> mputer                                                                      |
| int compareTo<br>(String str)                     | It compares two strings lexicographically.<br><b>Syntax:</b><br>String variable1.compareTo (String variable2);                                                                                           | a="JAVA";<br>b="PROGRAMMING";<br>c="PYTHON";<br>d="PROGRAMMING";<br>b.compareTo(d);<br><b>Output:</b> 0                        |
| startsWith(String<br>str)                         | It checks whether a string starts with a given substring or not.<br><b>Syntax:</b><br>String variable.startWith(substring);                                                                              | a="Think Out Of The Box";<br>b="Think";<br>a.startsWith(b);<br><b>Output:</b> TRUE                                             |

|                      |                                                                                                                          |                                                                                |
|----------------------|--------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------|
| endsWith(String str) | It checks whether a string ends with a given substring or not.<br><b>Syntax:</b><br>String variable.endsWith(substring); | a="Think Out Of The Box";<br>b="Box";<br>a.endsWith(b);<br><b>Output:</b> TRUE |
|----------------------|--------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------|

## Key Terms

**String:** A string is a term that refers to a set of characters.

**Java.lang( ):** It is the package that contains the various string functions. It is loaded by default whenever a Java program is loaded.

**ASCII Code:** Every character on the keyboard has an integer code for its interconversion to binary form. This integer code is called the ASCII code (American Standard Code for Information Interchange).



## SECTION 1 (Solved Questions)

### A. Fill in the blanks.

- \_\_\_\_\_ are immutable, their values cannot be changed after they are created.
- The \_\_\_\_\_ method returns the character at the specified index.
- The \_\_\_\_\_ method returns the index of the first occurrence of the specified character in the given string.
- The \_\_\_\_\_ method combines the string at the end of another string.

**Ans.** 1. Strings      2. charAt()      3. indexOf()      4. concat( )

### B. Very short answer type questions.

- Which package contains information about the String methods?

**Ans.** Java.lang

- What is the return type of the equals() function?

**Ans.** Boolean

- Name the method used to compare two strings lexicographically.

**Ans.** compareTo()

- Name the package which is imported into every Java program by default.

**Ans.** Java.lang

- Write a statement to assign a special character '#' to a variable x.

**Ans.** char x='#'

- Name the operator which allows concatenation of strings.

**Ans.** +

7. Name the function used to convert all the characters of a given string to uppercase.

Ans. toUpperCase( )

8. Name the function which removes the blank spaces provided in the prefix and suffix of a string.

Ans. trim( )

### C. Short answer type questions.

1. State the purpose and return data type of the following String functions:

- a. indexOf( )                      b. compareTo( )

Ans. a. The indexOf( ) method returns the index within the string of the first occurrence of the specified character or -1 if the character is not present. Its return type is int.

b. The compareTo( ) method compares two strings lexicographically. Its return type is int.

2. Write a statement each to perform the following tasks on a string:

- a. Extract the second-last character of a word stored in the variable wd.  
b. Check if the second character of a string str is in uppercase.

Ans. a. `char ch = wd.charAt(wd.length() - 2);`

b. `boolean res = Character.isUpperCase(str.charAt(1));`

3. Write the output of the following:

- a. `System.out.println(Character.isUpperCase('R'));`  
b. `System.out.println(Character.toUpperCase('j'));`

Ans. a. true

b. J

4. Write a statement for each to perform the following tasks on a string:

- a. Find and display the position of the last space in a string s.  
b. Convert a number stored in a string variable x to double data type.

Ans. a. `System.out.println(s.lastIndexOf(" "));`

b. `double num = Double.parseDouble(x);`

5. State the values of n and ch.

```
char c = 'A';
int n = c + 1;
char ch = (char)n;
```

Ans. Value of n is 66 and ch is B.

6. State the output of the following program segment:

```
String s = "Examination";
int n = s.length();
System.out.println(s.startsWith(s.substring(5, n)));
System.out.println(s.charAt(2) == s.charAt(6));
```

Ans. **Output:**

false

true

7. State the method that:

- a. Converts a string to a primitive float data type  
b. Determines if the specified character is an uppercase character

Ans. a. `Float.parseFloat( )`  
b. `Character.isUpperCase( )`

8. State the data type and values of a and b after the following segment is executed:

```
String s1 = "Computer", s2 = "Applications"
a = (s1.compareTo(s2));
b = (s1.equals(s2));
```

Ans. Data type of a is int and the value is 2. Data type of b is boolean and the value is false.

9. What will the output of the following code snippet?

```
String s = "malayalam";
System.out.println(s.indexOf('m'));
System.out.println(s.lastIndexOf('m'));
```

Ans. Output:

0  
8

10. State the values stored in the variables str1 and str2:

```
String s1 = "good";
String s2 = "world matters";
String str1 = s2.substring(5).replace('t', 'n');
String str2 = s1.concat(str1);
```

Ans. Value stored in str1 is "manners" and str2 is "goodmanners". (Note that str1 begins with a space.)

11. Distinguish between `charAt(i)` and `indexOf(i)`.

Ans. The `charAt(i)` method returns a character at the  $i^{\text{th}}$  position, whereas the `indexOf(i)` method returns the index of the first occurrence of the specified character.

12. Differentiate between `isUpperCase( )` and `toUpperCase( )` functions.

Ans. The `isUpperCase( )` function checks whether the given character is an uppercase letter or not, whereas the `toUpperCase( )` function converts the character to an uppercase letter.

The return type of `isUpperCase( )` is Boolean whereas the return type of `toUpperCase( )` is char.

## D. Application-based questions.

1. Write output of the following program:-

```
class abc
{
 public static void main(String args[])
 {
 String str1, str2;
 str1="Multimedia"; str2="Technology";
 System.out.println(str1.lastIndexOf('i'));
 System.out.println(str1.indexOf('m'));
 System.out.println(str2.concat(str1.substring(5)));
 System.out.println(str1.concat(str2.substring(0,5)));
 System.out.println(str2.concat(str1.toUpperCase()));
 System.out.println(str1.indexOf('i'));
```

```

 System.out.println(str2.lastIndexOf('g'));
 System.out.println(Math.sqrt(str2.length()-1));
 System.out.println((str1.substring(2,5).concat(str2.substring(4,8))).
 length());
 System.out.println('A'+ 'B'+2);
 }
}

```

Ans. 4

5

Technologymedia

MultimediaTechn

TechnologyMULTIMEDIA

4

9

8

3.0

7

133

2. What do the following functions return for?

String x = "hello";

String y = "world";

a. System.out.println(x+y);

b. System.out.println(x.length());

c. System.out.println(x.charAt(3));

d. System.out.println(x.equals(y));

Ans. a. Helloworld

b. 5

c. l

d. false

3. What will be the output of the following code?

String city = "Madras";

String str="MADAM";

city=city.concat("Central");

System.out.println(city);

System.out.println(str.replace('M', 'A'));

Ans. MadrasCentral

AADAA

4. Write the output, if String k ="Malayalam";

a. k.lastIndexOf('a');

b. for(int i =k.length()-1; k>=0; k--)

System.out.print(k.charAt(i));



- Ans. a. 7  
b. malayalam

## SECTION 2 (Unsolved Questions)

### A. Very short answer type questions.

1. Consider the String `str="Write ONCE run ANYWHERE";`  
Write a statement for the following:
  - a. Convert the given string into uppercase.
  - b. Find the length of the given string.
  - c. Extract "ONCE" from the given string.
  - d. Find the location of the last character.
  - e. Insert "WORA" at index number 6.
  - f. Find the position of the last location of Y.
  - g. Find the square root of the length of the given string.
  - h. Replace all ONCE with TWICE.
  - i. Check if the character at index number 11 is whitespace or not.
  - j. Convert the given string into reverse order.
  - k. Check if the given string ends with "WHERE" or not.
  - l. Combine the first 10 characters with the last 3 characters.
  - m. Extract the third last character.
  - n. Replace all occurrences of 'T' with 'Z'.
  - o. Find the last occurrence of 'U'.
  - p. Remove extra space on the left side of the string `str`.
  - q. Compare string `str` with "Writing" to get numeric output.
2. Write the output of the following:
  - a. `"VIRTUAL BRAINPOWER".indexOf('e')`
  - b. `"Micro"+"TechNoSoft".startsWith("Micro")? "wrong": "right"`
  - c. `"Super MIndGame ".lastIndexOf('I')+'D'`
  - d. `(int)"Reliability".charAt(3) +'A'`
  - e. `"VERSATILITY".substring(2,4).concat("PROGRAM")`
3. Consider the String `s1="MISSISIPI", s2="MISSOURI";`  
Write the output of the following statements:
  - a. `s2.substring(3,8).concat(s1)`
  - b. `s2.length()+'a'`
  - c. `s1.startsWith(s2.substring(0,4))`
  - d. `s2.reverse().concat(s1)`
  - e. `s2.charAt(7)`
  - f. `s2.lastIndexOf('o')`
  - g. `s1.indexOf('S',4)`
  - h. `s1.compareTo(s2)`

- i. `s1.replace('S', 'X')`
- j. `Math.sqrt(s2.length() +1)`

## B. Short answer type questions.

1. Write the output of the following statements:

- a. `System.out.println("OUT = " +4+2);`
- b. `System.out.println("OUT = " +(4+2));`
- c. `System.out.println("OUT = " +4*2);`
- d. `System.out.println("OUT = " +4+'2');`
- e. `System.out.println("OUT = " +'A'+'b');`

2. Write the output of the following code snippet:

```
String str1="great", str2="minds";
System.out.println(str1.substring(0,2).concat(str2.substring(1)));
System.out.println("WH"+str1.substring(2).toUpperCase());
```

3. Write the output of the following code segment:

```
class test
{
 public static void main(String args[])
 {
 int i=1;
 while(++i<=4)
 {
 i++;
 System.out.print(i+" "+(char)(96+i));
 }
 System.out.println(i);
 }
}
```

## C. Application-based questions.

- 1. Write a program that accepts a line of text and prints each word in separate lines.
- 2. Write a program that accepts a line of text and prints all palindrome words.
- 3. Write a program that accepts a line of text and prints each word along with the number of vowels in it.
- 4. Write a program that accepts a line of text. Find and print the word containing the maximum number of vowels.

**Sample Input:** Education is a progressive discovery of our own ignorance

**Output:** Education

- 5. Write a program that takes two strings (a sentence and a word). Find the number of times the input word appears in the sentence.

**Sentence:** THE BIG FAT AND THE BEAUTIFUL THE OLD LADY.

**Word:** THE

**Output:** 3

- 6. Write a program that accepts a sentence, a word to search, and a word to replace with. Search for the word in the sentence and replace it with the word taken. Print the new sentence.

**Input Sentence:** "aaa boy with aaa pen on aaa cycle."

**Search Word:** aaa

**Replace With:** the

**Output:** the boy with the pen on the cycle.

7. Write a menu-driven program that accepts the full name of a person and prints as per the option given below:

**Sample Input:** VIR PRATAP SINGH

- a. short name : V.P. SINGH
- b. bibliography name. : SINGH, V.P.
- c. Initials : V.P.S.

8. Write a program that accepts full name and prints the surname followed by first name and last name.

Sample Input: AMIT KUMAR SINHA

Output: SINHA AMIT KUMAR

9. Write a program that takes a line of text and uses the menu to print the first letter of each word or last letter of each word.

**Example:** EMBRACE CHALLENGES AND PROBLEM.

**Option 1:** The first letter of each word

**Option 2:** Last letter of each word

**Choice:** 1

**Output:** E C A P

### SECTION 3 (Board Questions)

1. State the value of the characteristic and mantissa when the following code is executed: (2014)

```
String s = "4.3756";
int n = s.indexOf('.');
int characteristic=Integer.parseInt(s.substring(0,n));
int mantissa=Integer.valueOf(s.substring(n+1));
```

**Ans.** The value of characteristic is 4 and mantissa is 3756.

2. State the output of the following program: (2015)

```
String a ="Smartphone", b="Graphic Art";
String h=a.substring(2, 5);
String k=b.substring(8).toUpperCase();
System.out.println(h);
System.out.println(k.equalsIgnoreCase(h));
```

**Ans.** The output of the above code is:

art  
true

3. Write the output of the following program code: (2015)

```
char ch;
int x=97;
```

```

do
{
 ch=(char)x;
 System.out.print(ch + " ");
 if(x%10 == 0)
 break;
 ++x;
} while(x<=100);

```

**Ans.** The output of the above code is:

a b c d

4. Give the output of the following string functions: (2016)

```

"MISSISSIPPI".indexOf('S') + "MISSISSIPPI".lastIndexOf('I')
"CABLE".compareTo("CADET")

```

**Ans.** 2 + 10 = 12

ASCII Code of B - ASCII Code of D

66 - 68

-2

5. Write the return type of the following library functions: (2016)

```

isLetterOrDigit(char)
replace(char, char)

```

**Ans.** boolean

String

6. Write the output of the following code: (2017)

```

char ch = 'F';
int m = ch;
m=m+5;
System.out.println(m + " " + ch);

```

**Ans.** Output of the above code is:

75 F

7. Give the output of the following code: (2017)

```

String A ="26", B="100";
String D=A+B+"200";
int x= Integer.parseInt(A);
int y = Integer.parseInt(B);
int d = x+y;
System.out.println("Result 1 = " + D);
System.out.println("Result 2 = " + d);

```

**Ans.** Output of the above code is:

Result 1 = 26100200

Result 2 = 126

8. Consider the given statements and answer the question that follows:

```
System.out.print("BEST ");
```

(2018)

```
System.out.println("OF LUCK");
```

Choose the correct option for the output of the above statements:

a. BEST OF LUCK

b. BEST

OF LUCK

Ans. Option b is the correct answer.

9. Give the output of the following string functions:

(2018)

```
"ACHIEVEMENT".replace('E', 'A')
```

```
"DEDICATE".compareTo("DEVOTE")
```

Ans. ACHIAVAMANT

-18

10. Write the output for the following:

(2019)

```
String s1 = "phoenix"; String s2 = "island";
```

```
System.out.println(s1.substring(0).concat(s2.substring(2)));
```

```
System.out.println(s2.toUpperCase());
```

Ans. The output of above code is:

phoenixland

ISLAND

## SECTION 4 [Case Study Based Questions]

### Case Study I

Strings are a group of characters. A string is not a primitive data type, as it is a class in Java, and hence classified as a non-primitive data type. A string is also referred to as a composite data type because it is composed of the primitive data type char. Java has a library of methods to perform various functions on strings. Information about these methods is present in the java.lang package. It is loaded by default whenever a Java program is loaded.

- The \_\_\_\_\_ method counts the number of characters in a string.
  - length
  - length( )
  - count( )
  - counts( )
- What will be the output of: "JAVA".charAt(3)?
  - J
  - A
  - V
  - VA
- The String methods are available in the \_\_\_\_\_ package.
  - java.function
  - java.lang
  - java.char
  - java.string

- Ans.1. b                      2. b                      3. b                      4. b

## Case Study II

The `java.lang.String` class provides a lot of built-in methods that are used to manipulate strings in Java. With the help of these methods, you can perform operations on `String` objects such as trimming, concatenating, converting, comparing, replacing strings, etc.

Java String is a powerful concept because everything is treated as a String if you submit any form in Window based, web based, or mobile application environment.

- Ans. 1. a                      2. c                      3. c                      4. a

# Programming ZONE

1. Write a program to accept a word and convert it into lowercase, if it is in uppercase. Display the new word by replacing only the vowels with the letter following it.

Sample Input: computer

Sample Output: cpmptvfr

```
Ans. import java.util.*;

public class Replace_Vowels
{
 public static void main(String args[])
 {
 Scanner input = new Scanner(System.in);
```

```

System.out.print("Enter a word: ");
String str = input.nextLine();
str = str.toLowerCase();
String newStr = "";
int len = str.length();
for (int i = 0; i < len; i++)
{
 char ch = str.charAt(i);
 if (str.charAt(i) == 'a' || str.charAt(i) == 'e' || str.charAt(i) == 'i'
 || str.charAt(i) == 'o' || str.charAt(i) == 'u')
 {
 char nextChar = (char)(ch + 1);
 newStr = newStr + nextChar;
 }
 else
 {
 newStr = newStr + ch;
 }
}
System.out.println(newStr);
}
}

```

2. Write a program to accept a string. Convert the string into uppercase letters. Count and output the number of double letter sequences that exist in the string.

Sample Input: "SHE WAS FEEDING THE LITTLE RABBIT WITH AN APPLE"

Sample Output: 4

Ans. `import java.util.*;`

```

class Pair_Letters
{
 public static void main(String args[])
 {
 Scanner input = new Scanner(System.in);
 System.out.print("Enter string: ");
 String s = input.nextLine();
 String str = s.toUpperCase();
 int count = 0;
 int len = str.length();
 }
}

```

```

 for (int i = 0; i < len - 1; i++)
 {
 if (str.charAt(i) == str.charAt(i + 1))
 count++;
 }
 System.out.println("Double Letter Sequence Count = " + count);
 }
}

```

3. Write a program that encodes a word into PIGLATIN. To translate the word into a PIGLATIN word, convert the word into uppercase and then place the first vowel of the original word as the start of the new word along with the remaining letters. The letters present before the vowel are shifted towards the end, followed by "AY".

Sample Input 1: London

Output: ONDONLAY

Sample Input 2: Olympics

Output: OLYMPICSAY

Ans. `import java.util.*;`

`class ICSE_2013`

```

{
 public static void main(String args[])
 {
 Scanner input = new Scanner(System.in);
 System.out.print("Enter word: ");
 String word = input.next();
 int len = word.length();
 word=word.toUpperCase();
 String piglatin=" ";
 int flag=0;
 for(int i = 0; i < len; i++)
 {
 char x = word.charAt(i);
 if(x=='A' || x=='E' || x=='I' || x=='O' || x=='U')
 {
 piglatin=word.substring(i) + word.substring(0,i) + "AY";
 flag=1;
 break;
 }
 }
 if(flag == 0)
 {

```



```

 piglatin = word + "AY";
 }
 System.out.println(word + " in Piglatin format is " + piglatin);
}
}

```

4. Write a program in Java to accept a string in lowercase and change the first letter of every word to uppercase. Display the new string.

Sample input: we are in cyber world

Sample output: We Are In Cyber World

Ans. `import java.util.*;`

`class Chng_Upper`

```

{
 public static void main(String args[])
 {
 Scanner input = new Scanner(System.in);
 System.out.println("Enter a sentence:");
 String str = in.nextLine();
 String word = " ";
 for (int i = 0; i < str.length(); i++)
 {
 if (i == 0 || str.charAt(i-1) == ' ')
 {
 word += Character.toUpperCase(str.charAt(i));
 }
 else
 {
 word += str.charAt(i);
 }
 }
 System.out.println(word);
 }
}

```

5. Write a program to input a sentence, convert it into uppercase, and count and display the total number of words starting with a letter 'A'. (2019)

Sample Input: ADVANCEMENT AND APPLICATION OF INFORMATION TECHNOLOGY ARE EVER CHANGING

Sample Output: Total number of words starting with letter 'A' = 4

Ans. `import java.util.*;`

`class Count_Vowel`

```

{

```

```

public static void main(String args[])
{
 Scanner input = new Scanner(System.in);
 System.out.println("Enter a string: ");
 String str = in.nextLine();
 str = " " + str;
 int c = 0;
 int len = str.length();
 str = str.toUpperCase();
 for (int i = 0; i < len - 1; i++)
 {
 if (str.charAt(i) == ' ' && str.charAt(i + 1) == 'A')
 c++;
 }
 System.out.println("Total number of words starting with letter 'A' = " + c);
}
}

```

6. Write a program to input a sentence and find the word that has the highest potential. Also, print the potential of that word. (The potential of a word is the sum of the ASCII codes of the letters of the word.)

**Sample Input:** LOOK BEFORE YOU LEAP

**Output:** Word with the highest potential=BEFORE

Potential=435

Ans. 

```

import java.util.*;
public class Chk_Potential
{
 public static void main(String args[])
 {
 Scanner input=new Scanner(System.in);
 String s,w="",bw="";
 int i,l,p=0,max=0;
 char c;
 System.out.println("Enter a sentence:");
 s=input.nextLine();
 s=s+" ";
 l=s.length();
 for(i=0;i<l;i++)
 {
 c=s.charAt(i);
 if(c!=' ')
 {

```

```

 w=w+c;
 p=p+(int)c;
 }
 else
 {
 if(p>max)
 {
 max=p;
 bw=w;
 }
 w="";
 p=0;
 }
}
System.out.println("Word with the highest potential="+bw);
System.out.println("Potential="+max);
}
}

```

7. Write a program to input a sentence and reprint the sentence after exchanging the first and last letters of each word.

**Input String:** ALL THE BEST

**Output String:** LLA EHT TESB

**Ans.** `import java.util.*;`  
`public class Char_Exachnge`  
`{`

```

 public static void main(String args[])
 {
 Scanner input=new Scanner(System.in);
 String s, w="", s1="", w1="";
 int i, l, lw;
 char c, c1, c2;
 System.out.println("Enter a sentence:");
 s=input.nextLine();
 s=s+" ";
 l=s.length();
 for(i=0;i<l;i++)
 {
 c=s.charAt(i);

```

```

 if(c!=' ')
 w=w+c;
 else
 {
 lw=w.length();
 c1=w.charAt(0);
 c2=w.charAt(lw-1);
 w1=c2+w.substring(1,lw-1)+c1;
 s1=s1+" "+w1;
 w="";
 w1="";
 }
 }
 System.out.println(s1);
}
}

```

8. Write a program to input two words and check whether they are anagrams or not.  
For example, CAT and ACT are anagrams.

Ans. `import java.util.*;`  
`public class Chk_Anagram`  
`{`  
 `public static void main(String args[])`  
 `{`  
 `Scanner input=new Scanner(System.in);`  
 `String s1,s2,s3="",s4="";`  
 `int i,j,l,l1;`  
 `char c;`  
 `System.out.println("Enter two words:");`  
 `s1=input.nextLine();`  
 `s2=input.nextLine();`  
 `l=s1.length();`  
 `for(i=65;i<=90;i++)`  
 `{`  
 `for(j=0;j<l;j++)`  
 `{`  
 `c=s1.charAt(j);`  
 `if((int)c==i || (int)c==(i+32))`

```

 s3=s3+c;
 }
}
l1=s2.length();
for(i=65;i<=90;i++)
{
 for(j=0;j<l1;j++)
 {
 c=s1.charAt(j);
 if((int)c==i || (int)c==(i+32))
 s4=s4+c;
 }
}
if(s3.equalsIgnoreCase(s4))
 System.out.println("Anagrams");
else
 System.out.println("Not Anagrams");
}
}

```

9. Write a program to input a number and check whether it is a Kaprekar number or not. (A number is said to be Kaprekar, if the square of the number is divided into two parts and their sum results in the same number).

Example:  $9^2=81$  ( $8+1=9$ )

$45^2=2025$  ( $20+25=45$ )

$99^2=9801$  ( $98+01=99$ )

$297^2=88209$  ( $88+209=297$ )

Ans. `import java.util.*;`  
`public class Kaprekar`  
`{`  
 `public static void main(String args[])`  
 `{`  
 `Scanner input=new Scanner(System.in);`  
 `int n,sq,l,p,n1,n2,t;`  
 `String s,s1,s2;`  
 `char c;`  
 `System.out.println("Enter a number:");`  
 `n=input.nextInt();`  
 `sq=n*n;`  
 `s=Integer.toString(sq);`

```

l=s.length();
p=l/2;
s1=s.substring(0,p);
s2=s.substring(p);
n1=Integer.parseInt(s1);
n2=Integer.parseInt(s2);
t=n1+n2;
if(t==n)
 System.out.println("Kaprekar Number");
else
 System.out.println("Not Kaprekar Number");
}
}

```



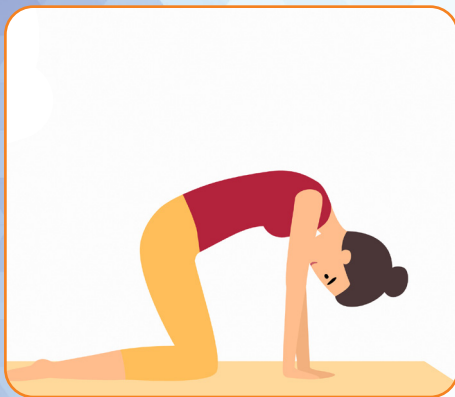
**Link**

Scan the given QR Code to learn the use of the String arrays.



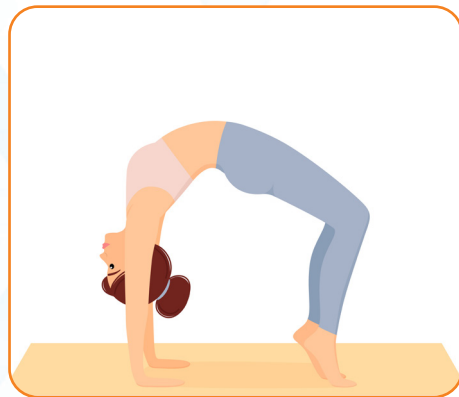
## Yoga Poses for Relieving Stress

### Breathe out the stress, breathe in the rest!



#### Cat Pose

Soothes and stretches the lower back  
Massages spine and relieves stress  
Creates emotional balance



#### Bridge Pose

Reduces anxiety, fatigue, backaches,  
and insomnia Provides gentle stretching of the legs and back

# Practice Questions

## Question 1

- What is an array? If `int arr[ ] = {3, 5, 7, 9};` what is the value of `arr.length`?
- State the value of 'y' after the following is executed:  
`char x= '7';`  
`y=Character.isLetter(x);`
- What are the wrapper classes? Name the package that has the wrapper classes in Java.
- What value will be returned on executing the following statement?  
`"KNOWLEDGE".substring(2,6);`  
Also, write the data type of the `substring( )` method.
- What do you mean by boxing and unboxing?



Scan the QR  
Code for the latest  
sample paper

## Question 2

- Differentiate between if-else and switch-case statements.
- Name any four tokens used in Java.
- What is the purpose of the new operator?
- Mention the types of access specifiers.
- What is constructor overloading?

## Question 3

- Define the precedence of operators.
- What are escape sequences? Give any two examples.
- What are the values stored in variables R1 and R2?  
`double R1=Math.abs(Math.min(-2.83,-5.83));`  
`double R2=Math.sqrt(Math.floor(16.3));`
- Differentiate between `=` and `==` operators.
- Describe the use of the `indexOf( )` method. Also, give an appropriate example.

## Question 4

- Convert the following code segment into its equivalent 'for' loop.  

```
int amount=51;
i=0;
while(amount>0)
{
 System.out.println(amount);
 amount=amount-1;
}
```
- Write a statement to create an object 'best' of the class BigB.

- c. What will be the output of the following program segment when executed?

```
int d=5000,i=20;
System.out.println(d>2000?d-=5000:(i>=20?d+=5000:20000));
System.out.println(3-6>-2 && 7+2<16 ||2>=4-2);
```

- d. What will be the output of the following code segment?

```
System.out.println("NINE="+7+'2');
System.out.println(('P'+ 'Q'+2));
```

- e. Give the output of the following:

```
int a[]={3,5,7,9};
int s=0;
for(i=0;i<=1;i++)
{
 S=a[i]+a[3-i];
 System.out.println(s);
}
```

### Question 5

Define a class 'ElectricBill' with the following specifications:

#### Instance variables / data members:

- **String n:** To store the name of the customer
- **int units:** To store the number of units consumed
- **double bill:** To store the amount to be paid

#### Member methods:

- **void accept( ):** To accept the name of the customer and number of units consumed
- **void calculate( ):** To calculate the bill as per the following tariff:

|                 |       |
|-----------------|-------|
| First 100 units | ₹2.00 |
| Next 200 units  | ₹3.00 |
| Above 300 units | ₹5.00 |

A surcharge of 2.5% is charged if the number of units consumed is above 300 units.

- **void print ( ):** To print the details as given below:

Name of the customer: Anu

Number of units consumed: 300

Bill amount: 800

Write a main ( ) method to create an object of the class and call the above member methods.

### Question 6

Write a program to input and store roll numbers, names, and marks in three subjects of 'n' number of students in five single-dimensional arrays. Search for the name entered by the user using the binary search technique. If found, display the details along with a remark based on average marks as given below. If the searched name is not found, display the message "Name does not exist in the array".





| Average Marks | Remark      |
|---------------|-------------|
| 85 – 100      | Distinction |
| 75 – 84       | Excellent   |
| 60 – 74       | First Class |
| 40 – 59       | Pass        |
| Less Than 40  | Poor        |

### Question 7

Write a program to calculate the sum of all the prime numbers between the range of 1 and 100.

### Question 8

Using a switch statement, write a menu-driven program to input a number and check whether it is palindrome or perfect.

- **Palindrome:** A number is said to be a palindrome if you get the same number even after reversing the digits, for example: 121)
- **Perfect:** A number is said to be perfect if the sum of the factors (including 1 and excluding the number itself) is equal to the same number. For example,  $6=1+2+3$ . As a result, 6 is a perfect number.

### Question 9

Define a class to overload a function series( ) to perform the following:

- **series(int n):** To print Fibonacci series up to n terms 0,1,1,2,3,5.....n
- **series(int m, int n):** To print odd numbers in a series from m to n (where m is always less than n)
- **series( ):** To print all English alphabets in upper case

### Question 10

Write a program to input the names of your 10 friends and store them in a single-dimension array. Arrange the array alphabetically in descending order using the bubble sort technique. Print the sorted array.

# Internal Assessment

1. A trigonometrical expression is given as:  $(\tan A - \tan B) / (1 + \tan A * \tan B)$ ; Write a program to calculate the value of the given expression by taking the values of angles A and B (in degrees) as input. [Hint: A radian is equal to  $(22/7 * 180) * \text{degree}$ ]

Ans.

```
import java.util.*;
public class angles
{
 public static void main(String args[])
 {
 Scanner input=new Scanner(System.in);
 System.out.print("Enter angle A in degrees: ");
 double angA=input.nextDouble();
 System.out.print("Enter angle B in degrees: ");
 double angB=input.nextDouble();
 double angA1=(22 * angA) / (7 * 180);
 double angB1=(22 * angB) / (7 * 180);
 double numerator=Math.tan(angA1) - Math.tan(angB1);
 double demoninator=1 + Math.tan(angA1) * Math.tan(angB1);
 double t=numerator / demoninator;
 System.out.println("tan(A - B) = " + t);
 }
}
```

Variable Description Table

| Variable    | Type   | Description                           |
|-------------|--------|---------------------------------------|
| angA        | double | To input the first angle in degree    |
| angB        | double | To input the second angle in degree   |
| angA1       | double | To convert the first angle in radian  |
| angB1       | double | To convert the second angle in radian |
| numerator   | double | To find the numerator                 |
| denominator | double | To find the denominator               |
| t           | double | To find the final result              |

**Output:**

```
Enter angle A in degrees: 30
Enter angle B in degrees: 45
tan(A - B) = -0.2680621352464786
```

2. Write a program in Java that accepts the seconds as input and converts them into the corresponding number of hours, minutes, and seconds.

Ans.

```
import java.util.*;
public class time
{
 public static void main(String args[])
 {
 Scanner input = new Scanner(System.in);
 long h,m,s;
```

```

 System.out.println("Enter time in seconds: ");
 s=input.nextLong();
 h=s/3600;
 s=s%3600;
 m=s/60;
 s=s%60;
 System.out.println(h + "Hour(s)" + m + "Minute(s)" + s + "Second(s)");
 }
}

```

**Variable Description Table**

| Variable | Type | Description          |
|----------|------|----------------------|
| h        | long | To store the hours   |
| m        | long | To store the minutes |
| s        | long | To store the seconds |

**Output:**

```

Enter time in seconds:
4530
1 Hour(s) 15 Minute(s) 30 Second(s)

```

3. Write a program to input a number and find out if the number entered by the user is a two-, three-, or four-digits number.

**Sample input: 235**

**Sample output: 235 is a 3-digit number**

**Ans.**

```

import java.util.*;
public class digits
{
 public static void main(String args[])
 {
 int n;
 Scanner input=new Scanner(System.in);
 System.out.print("Enter number: ");
 n=input.nextInt();
 if (n >=10 && n<=99)
 System.out.println("Two-digit number");
 else if (n>=100 && n<=999)
 System.out.println("Three-digit number");
 else if (n>=1000 && n<=9999)
 System.out.println("Four-digit number");
 else
 System.out.println("Number not in range ");
 }
}

```

**Variable Description Table**

| Variable | Type | Description         |
|----------|------|---------------------|
| n        | int  | To input the number |

**Output:**

```
Enter number: 345
Three-digit number
Enter number: 23456
Number not in range
```

**4. Write a program in Java to read three integers and display them in descending order.**

**Ans.** `import java.util.*;`  
`public class descending_order`  
`{`  
 `public static void main(String args[])`  
 `{`  
 `Scanner input=new Scanner(System.in);`  
 `int a,b,c,max,min,mid;`  
 `System.out.print("Enter first number: ");`  
 `a=input.nextInt();`  
 `System.out.print("Enter second number: ");`  
 `b=input.nextInt();`  
 `System.out.print("Enter third number: ");`  
 `c=input.nextInt();`  
 `max=-1;`  
 `mid=-1;`  
 `min=-1;`  
 `if (a > b) {`  
 `if (a > c)`  
 `{`  
 `max = a;`  
 `if (b > c)`  
 `{`  
 `mid = b;`  
 `min = c;`  
 `}`  
 `} else`  
 `{`  
 `mid = c;`  
 `min = b;`  
 `}`  
 `}`  
 `else`  
 `{`  
 `max=c;`  
 `mid=a;`  
 `min=b;`  
 `}`  
 `}`  
 `else`  
 `{`  
 `if (b > c)`  
 `{`  
 `max = b;`  
 `if (a > c)`

```

 {
 mid = a;
 min = c;
 }
 else
 {
 mid = c;
 min = a;
 }
 }
 else
 {
 max = c;
 mid = b;
 min = a;
 }
}
System.out.println("Numbers in descending order:");
System.out.print(max + " " + mid + " " + min);
}
}

```

**Variable Description Table**

| Variable | Type | Description                 |
|----------|------|-----------------------------|
| a        | int  | To input a number           |
| b        | int  | To input a number           |
| c        | int  | To input a number           |
| max      | int  | To find the biggest number  |
| min      | int  | To find the smallest number |
| mid      | int  | To find the middle number   |

**Output:**

```

Enter first number: 65
Enter second number: 43
Enter third number: 81
Numbers in descending order: 81 65 43

```

**5. Write a program to input three numbers and find the biggest number.**

**Ans.**

```

import java.util.*;
class number
{
 public static void main(String args[])
 {
 Scanner input=new Scanner(System.in);
 int a,b,c,max=0;
 System.out.println("Enter three numbers");
 a=input.nextInt();
 b=input.nextInt();
 c=input.nextInt();
 }
}

```

```

 if(a>b && a>c)
 max=a;
 else if(b>a && b>c)
 max=b;
 else if (c>a && c>b)
 max=c;
 System.out.println("Biggest number is: "+max);
 }
}

```

**Variable Description Table**

| Variable | Type | Description                |
|----------|------|----------------------------|
| a        | int  | To input the first number  |
| b        | int  | To input the second number |
| c        | int  | To input the third number  |
| max      | int  | To find the biggest number |

**Output:**

```

Enter three numbers
76
54
89
Biggest number is: 89

```

6. Write a program to input the marks obtained by a student and print his grade as per the following:

| Marks Range | Grade |
|-------------|-------|
| 90 & above  | A     |
| 75-89       | B     |
| 60-74       | C     |
| 40-59       | D     |
| Below 40    | E     |

**Ans.**

```

import java.util.*;
class marks
{
 public static void main(String args[])
 {
 Scanner input=new Scanner(System.in);
 int m;
 System.out.println("Enter marks");
 m=input.nextInt();
 if(m>=90)
 System.out.println("A");
 else if(m>=75 && m<90)
 System.out.println("B");
 else if(m>=60 && m<75)
 System.out.println("C");
 else if(m>=40 && m<60)
 System.out.println("D");
 else

```

```

 System.out.println("E");
 }
}

```

**Variable Description Table**

| Variable | Type | Description    |
|----------|------|----------------|
| m        | int  | To input marks |

**Output:**

```

Enter marks
86
A

```

7. Write a program to input a letter and check whether it is an uppercase vowel, lowercase vowel, or not a vowel.

**Ans.**

```

import java.util.*;
class letter_case
{
 public static void main(String args[])
 {
 Scanner input=new Scanner(System.in);
 char m;
 System.out.println("Enter a character");
 m=input.next().charAt(0);
 switch(m)
 {
 case 'A': case 'E':case 'I':case 'O':case 'U':
 System.out.println("Uppercase vowel ");
 break;
 case 'a': case 'e':case 'i':case 'o':case 'u':
 System.out.println("Lowercase vowel ");
 break;
 default:
 System.out.println("Not a vowel");
 }
 }
}

```

**Variable Description Table**

| Variable | Type | Description                          |
|----------|------|--------------------------------------|
| m        | char | To input the character to be checked |

**Output:**

```

Enter a character
d
Lowercase vowel

```

8. Write a program to input a sentence and count how many times the same letters have occurred concurrently.

**Ans.**

```

import java.util.*;

```

```

class count_occurrence
{
 public static void main(String args[])
 {
 Scanner input=new Scanner(System.in);
 String s;
 int i,l,v=0;
 char c1,c2;
 System.out.println("Enter a sentence");
 s=input.nextLine();
 s=s.toUpperCase();
 l=s.length();
 for(i=0;i<(l-1);i++)
 {
 c1=s.charAt(i);
 c2=s.charAt(i+1);
 if(c1==c2)
 v++;
 }
 System.out.println("Total occurrence of same letters is: "+v);
 }
}

```

**Variable Description Table**

| Variable | Type   | Description                                |
|----------|--------|--------------------------------------------|
| s        | String | To input a sentence                        |
| l        | int    | To find the length of the sentence         |
| i        | int    | To use in the loop                         |
| c1       | char   | To extract the character                   |
| c2       | char   | To extract the second character            |
| v        | int    | To find the occurrence of repeated letters |

**Output:**

```

Enter a sentence
He went to the market to get green apples
Total occurrence of same letters is: 2

```

9. Write a program to input a sentence and find the longest word present in it. Also, count the length of the longest word.

**Ans.**

```

import java.util.*;
class longest_word
{
 public static void main(String args[])
 {
 Scanner input=new Scanner(System.in);
 String s,w="",bw="";
 int i,l,lw,max=0;
 char c;
 System.out.println("Enter a sentence");
 s=input.nextLine();
 s=s+" ";
 }
}

```



```

 l=s.length();
 for(i=0;i<l;i++)
 {
 c=s.charAt(i);
 if(c!=' ')
 w=w+c;
 else
 {
 lw=w.length();
 if(lw>max)
 {
 max=lw;
 bw=w;
 }
 w="";
 }
 }
 System.out.println("Longest word is: "+bw);
 System.out.println("Length of longest word is: "+ max);
 }
}

```

**Variable Description Table**

| Variable | Type   | Description                            |
|----------|--------|----------------------------------------|
| s        | String | To input a sentence                    |
| l        | int    | To find the length of the sentence     |
| i        | int    | To use in the loop                     |
| c        | char   | To extract the character               |
| w        | String | To extract the word                    |
| lw       | int    | To store the length of the word        |
| bw       | String | To store the longest word              |
| max      | int    | To find the length of the longest word |

**Output:**

```

Enter a sentence
This is a beautiful story
Longest word is: beautiful
Length of longest word is: 9

```

10. Write a menu-driven program to convert a given temperature from Fahrenheit to Celsius and vice-versa. For an incorrect choice, an appropriate message should be displayed. [Hint:  $c = 5/9*(f-32)$  and  $f = 1.8*c+32$ ]

**Ans.**

```

import java.util.*;
class temperature
{
 public static void main(String args[])
 {
 Scanner input = new Scanner(System.in);
 System.out.println("Press 1 to convert from Fahrenheit to Celsius");
 }
}

```

```

 System.out.println("Press 2 to convert from Celsius to Fahrenheit");
 int choice = input.nextInt();
 double f = 0.0, c = 0.0;
 switch (choice)
 {
 case 1:
 System.out.print("Enter temperature in Fahrenheit: ");
 f = input.nextDouble();
 c = 5 / 9.0 * (f - 32);
 System.out.println("Temperature in Celsius: " + c);
 break;
 case 2:
 System.out.print("Enter temperature in Celsius: ");
 c = input.nextDouble();
 f = 1.8 * c + 32;
 System.out.println("Temperature in Fahrenheit: " + f);
 break;
 default:
 System.out.println("Incorrect choice");
 break;
 }
 }
}

```

**Variable Description Table**

| Variable | Type   | Description                        |
|----------|--------|------------------------------------|
| ch       | int    | To input the choice                |
| c        | double | To input temperature in Celsius    |
| f        | double | To input temperature in Fahrenheit |

**Output:**

```

Press 1 to convert from Fahrenheit to Celsius
Press 2 to convert from Celsius to Fahrenheit
2
Enter temperature in Celsius: 50
Temperature in Fahrenheit: 122.0

```

11. Write a menu-driven program that asks for the length and breadth of a rectangle and calculates the following as per user choice. Also, display an appropriate message if an incorrect choice is entered by the user.

Press 1 to calculate the area

Press 2 to calculate the perimeter

Press 3 to calculate the diagonal

**Ans.**

```
import java.util.*;
class rectangle
{
 public static void main(String args[])
 {
```

```

Scanner input=new Scanner(System.in);
int l,b,a,p,ch;
double d;
System.out.println("Enter length and breadth of the rectangle");
l=input.nextInt();
b=input.nextInt();
System.out.println("Enter choice 1-Area/2-Perimeter/3-Diagonal");
ch=input.nextInt();
switch(ch)
{
 case 1:
 a=l*b
 System.out.println("Area="+a);
 break;
 case 2:
 p=2*(l+b);
 System.out.println("Perimeter="+p);
 break;
 case 3:
 d=Math.sqrt(l*l+b*b);
 System.out.println("Diagonal="+d);
 break;
 default:
 System.out.println("Invalid choice");
}
}
}

```

**Variable Description Table**

| Variable | Type   | Description            |
|----------|--------|------------------------|
| l        | int    | To input the length    |
| b        | int    | To input the breadth   |
| a        | int    | To store the area      |
| p        | int    | To store the perimeter |
| d        | int    | To store the diagonal  |
| ch       | double | To input the choice    |

**Output:**

Enter length and breadth of  
the rectangle

6

8

Enter choice

1-Area, 2-Perimeter, 3 Diagonal

1

Area=48

Enter length and breadth of  
the rectangle

5

7

Enter choice

1-Area,2-Perimeter,3-  
Diagonal

2

Perimeter=24

Enter length and breadth of  
the rectangle

6

8

Enter choice

1-Area,2-Perimeter,3-Diagonal

3

Diagonal=10.0

12. Write a program to input a sentence. Search for the word entered by the user. Also, count how many times it has occurred in the sentence.

```
Ans. import java.util.*;
class word_occurrence
{
 public static void main(String args[])
 {
 Scanner input=new Scanner(System.in);
 String s,w="",wrd;
 int i,l,f=0;
 char c;
 System.out.println("Enter a sentence");
 s=input.nextLine();
 System.out.println("Enter a word");
 wrd=input.next();
 s=s+" ";
 l=s.length();
 for(i=0;i<l;i++)
 {
 c=s.charAt(i);
 if(c!=' ')
 w=w+c;
 else
 {
 if(w.equalsIgnoreCase(wrd))
 f++;
 w="";
 }
 }
 System.out.println("Total occurrence of word is: "+f);
 }
}
```

**Variable Description Table**

| Variable | Type   | Description                        |
|----------|--------|------------------------------------|
| s        | String | To input a sentence                |
| wrd      | String | To input a word                    |
| w        | String | To form the word                   |
| i        | int    | To use in the loop                 |
| l        | int    | To find the length of the sentence |
| f        | int    | To find the frequency of the word  |

**Output:**

```
Enter a sentence
This is the book which I had lost in the class
Enter a word
the
Total occurrence of word is: 2
```

13. Write a program to input a sentence. Find those words that begin and end with a vowel. Also, count its total occurrences in the sentence.

```
Ans. import java.util.*;
class begin_end_vowel
{
 public static void main(String args[])
 {
 Scanner input=new Scanner(System.in);
 String s,w="";
 int i,l,f=0;
 char c,c1,c2;
 System.out.println("enter a sentence");
 s=input.nextLine();
 s=s.toUpperCase();
 s=s+" ";
 l=s.length();
 for(i=0;i<l;i++)
 {
 c=s.charAt(i);
 if(c!=' ')
 w=w+c;
 else
 {
 c1=w.charAt(0);
 c2=w.charAt(w.length()-1);
 if((c1=='A' || c1=='E' || c1=='I' || c1=='O' || c1=='U') && (c2=='A' || c2=='E' || c2=='I' || c2=='O' || c2=='U'))
 {
 System.out.println(w);
 f++;
 }
 w="";
 }
 }
 System.out.println("Occurrences of the words that begin and end with a vowel is: "+f);
 }
}
```

**Variable Description Table**

| Variable | Type   | Description                                 |
|----------|--------|---------------------------------------------|
| s        | String | To input a sentence                         |
| w        | String | To extract the word                         |
| i        | int    | To use in the loop                          |
| l        | int    | To find the length of the sentence          |
| f        | int    | To find the frequency of the required words |
| c        | char   | To extract the character                    |
| c1       | char   | To extract the first character              |
| c2       | char   | To extract the last character               |

**Output:**

```
Enter a sentence
An eye for an eye
EYE
EYE
Occurrences of the words that begin and end with a vowel is:2
```

**14. Write a program to input a word and convert it to its Piglatin form.**

```
Ans. import java.util.*;
class piglatin
{
 public static void main(String args[])
 {
 Scanner input=new Scanner(System.in);
 String s,s1,s2,s3;
 int i,l,p=0;
 char c;
 System.out.println("Enter a word");
 s=input.next();
 l=s.length();
 for(i=0;i<l;i++)
 {
 c=s.charAt(i);
 if(c=='A' || c=='E' || c=='I' || c=='O' || c=='U' || c=='a' || c=='e' ||
 c=='i' || c=='o' || c=='u')
 {
 p=i;
 break;
 }
 }
 s1=s.substring(p);
 s2=s.substring(0,p);
 s3=s1.concat(s2.concat("AY"));
 System.out.println("Piglatin form is:"+s3);
 }
}
```

**Variable Description Table**

| Variable | Type   | Description                             |
|----------|--------|-----------------------------------------|
| s        | String | To input a sentence                     |
| s1       | String | To extract a part of the word           |
| s2       | String | To extract a part of the word           |
| s3       | String | To find the final word                  |
| i        | int    | To use in the loop                      |
| l        | int    | To find the length of the sentence      |
| p        | int    | To find the position of the first vowel |
| c        | char   | To extract the character                |

**Output:**

```
Enter a word
TROUBLE
Piglatin form is: OUBLETRAY
```

**15. Write a program to input a word and arrange the letters of the word alphabetically.**

```
Ans. import java.util.*;
class arranging_letters
{
 public static void main(String args[])
 {
 Scanner input=new Scanner(System.in);
 String s,s1="";
 int i,j,l;
 char c;
 System.out.println("Enter a word");
 s=input.next();
 l=s.length();
 for(i=65;i<=90;i++)
 {
 for(j=0;j<l;j++)
 {
 c=s.charAt(j);
 if((int)c==i || (int)c==i+32)
 s1=s1+c;
 }
 }
 System.out.println("Letters of a word in alphabetical order is:" + s1);
 }
}
```

**Variable Description Table**

| Variable | Type   | Description                    |
|----------|--------|--------------------------------|
| s        | String | To input a word                |
| s1       | String | To find the new word           |
| i        | int    | To use in the loop             |
| j        | int    | To use in the loop             |
| l        | int    | To find the length of the word |
| c        | char   | To extract the character       |

**Output:**

```
Enter a word
COMPUTER
Letters of a word in alphabetical order is: CEMOPRTU
```

**16. Write a program to input a sentence. Find the letter that occurs the maximum number of times. Also, show the total number of occurrences of that letter.**

```
Ans. import java.util.*;
class string1
{
 public static void main(String args[])
 {
 Scanner input=new Scanner(System.in);
 String s;
 int i,j,l,f,max=0;
 char c,cm=' ';
```

```

System.out.println("Enter a sentence");
s=input.nextLine();
l=s.length();
for(i=65;i<=90;i++)
{
 f=0;
 for(j=0;j<l;j++)
 {
 c=s.charAt(j);
 if((int)c==i || (int)c==i+32)
 f++;
 }
 if(f>max)
 {
 max=f;
 cm=(char)i;
 }
}
System.out.println("Maximum occurring letter is: " + cm);
System.out.println("Occurrences of that letter: " + max);
}
}

```

**Variable Description Table**

| Variable | Type   | Description                                       |
|----------|--------|---------------------------------------------------|
| s        | String | To input a word                                   |
| s1       | String | To find the new word                              |
| i        | int    | To use in the loop                                |
| j        | int    | To use in the loop                                |
| l        | int    | To find the length of the word                    |
| c        | char   | To extract the character                          |
| f        | int    | To find the frequency of the letter               |
| cm       | char   | To find the character occurring maximum times     |
| max      | int    | To find the frequency of maximum occurring letter |

**Output:**

```

Enter a sentence
INDIA IS INDEPENDENT COUNTRY
Maximum occurring letter is: N
Occurrences of that letter: 5

```

- 17. Write a program to input a sentence. Search for a word entered by the user and replace it with the new word entered by him/her.**

**Ans.**

```

import java.util.*;
class find_replace
{
 public static void main(String args[])
 {
 Scanner input=new Scanner(System.in);
 String s,w1,w2,w="",s1="";
 int i,l;
 }
}

```



```

char c;
System.out.println("Enter a sentence");
s=input.nextLine();
System.out.println("Enter a word that you want to replace");
w1=input.nextLine();
System.out.println("Enter a new word");
w2=input.nextLine();
s=s+" ";
l=s.length();
for(i=0;i<l;i++)
{
 c=s.charAt(i);
 if(c!=' ')
 w=w+c;
 else
 {
 if(w.equalsIgnoreCase(w1))
 w=w2;
 else
 w=w;
 s1=s1+" "+w;
 w="";
 }
}
System.out.println(s1);
}
}

```

**Variable Description Table**

| Variable | Type   | Description                        |
|----------|--------|------------------------------------|
| s        | String | To input a sentence                |
| w1       | String | To input the word to search        |
| w2       | String | To input the word to replace with  |
| w        | String | To extract the word                |
| i        | int    | To use in the loop                 |
| l        | int    | To find the length of the sentence |
| c        | char   | To extract the character           |

**Output:**

```

Enter a sentence
A girl wearing red dress was going to buy red apples
Enter a word that you want to replace
RED
Enter a new word
GREEN
A girl wearing green dress was going to buy green apples

```

**18. Write a program to input a 15-digit code and check whether it is a valid IMEI code or not.**

```
Ans. import java.util.*;
class IMEI
{
 public static void main(String args[])
 {
 Scanner input=new Scanner(System.in);
 System.out.print("Enter a 15 digit IMEI code: ");
 long n=input.nextLong();
 String s=Long.toString(n);
 int l=s.length();
 if(l!=15)
 System.out.println("Output: Invalid Input");
 else
 {
 int d=0, sum=0;
 for(int i=1; i<=15; i++)
 {
 d=(int)(n%10);
 if(i%2==0)
 {
 d=2*d;
 }
 int a=0;
 while(d>0)
 {
 a=a + d%10;
 d=d/10;
 }
 sum=sum + a;
 n=n/10;
 }
 System.out.println("Sum is: "+sum);
 if(sum%10==0)
 System.out.println("Valid IMEI Code");
 else
 System.out.println("Invalid IMEI Code");
 }
 }
}
```

**Variable Description Table**

| Variable | Type   | Description                   |
|----------|--------|-------------------------------|
| n        | long   | To input the IMEI code        |
| s        | String | To convert the code to string |
| l        | int    | To find the length            |
| d        | int    | To extract the digit          |
| sum      | int    | To find the sum               |

|   |     |                    |
|---|-----|--------------------|
| i | int | To use in the loop |
| a | int | To find the digit  |

#### Output:

Enter a 15-digit IMEI code: 123456789234567

Sum is: 77

Invalid IMEI Code

#### 19. Write a program to input a number and check whether it is a Kaprekar number or not.

```
Ans. import java.util.*;
class Kaprekar_Number
{
 public static void main(String args[])
 {
 Scanner input=new Scanner(System.in);
 System.out.print("Enter a Number : ");
 int n=input.nextInt();
 int sq=n*n;
 String s=Integer.toString(sq);
 if(sq<=9)
 s="0"+s;
 int l=s.length();
 int mid=l/2;
 String left=s.substring(0,mid);
 String right=s.substring(mid);
 int x=Integer.parseInt(left);
 int y=Integer.parseInt(right);
 if(x+y == n)
 System.out.println(n+" is a kaprekar number");
 else
 System.out.println(n+" is Not a kaprekar number");
 }
}
```

**Variable Description Table**

| Variable | Type   | Description                          |
|----------|--------|--------------------------------------|
| n        | int    | To input a number                    |
| sq       | int    | To find the square of the number     |
| s        | String | To convert the number to String      |
| l        | int    | To find the length of the String     |
| mid      | int    | To find the mid point                |
| left     | String | To find the left part                |
| right    | String | To find the right part               |
| x        | int    | To convert the left part to integer  |
| y        | int    | To convert the right part to integer |

#### Output:

Enter a number: 45

45 is a Kaprekar Number

**20. Write a program to input 10 numbers in an array and find the biggest number.**

```
Ans. import java.util.*;
public class biggest_number
{
 public static void main(String args[])
 {
 Scanner input=new Scanner(System.in);
 int i,big=0;
 int n[]=new int[10];
 System.out.println("Enter numbers for an array");
 for(i=0;i<10;i++)
 {
 n[i]=input.nextInt();
 if(n[i]>big)
 big=n[i];
 }
 System.out.println("Biggest Number is"+ big);
 }
}
```

**Variable Description Table**

| Variable | Type          | Description                            |
|----------|---------------|----------------------------------------|
| n[ ]     | Integer array | Integer array                          |
| i        | int           | To use in the loop                     |
| big      | int           | To find the biggest number in the loop |

**Output:**

```
Enter numbers for an array
43 32 54 67 38 89 43 65 87 98
Biggest number is: 98
```

**21. Write a program to input the number of students in a class. Create a single-dimensional array to store the marks of the students and find the average marks obtained in the class.**

```
Ans. import java.util.*;
public class single_dimensional_array
{
 public static void main(String args[])
 {
 Scanner input=new Scanner(System.in);
 int n,i,s=0;
 double av;
 System.out.println("Enter the number of students");
 n=input.nextInt();
 int m[]=new int[n];
 System.out.println("Enter the marks");
 for(i=0;i<n;i++)
 {
 m[i]= input.nextInt();
 s=s+m[i];
 }
 av=s/n;
 }
}
```

```

 System.out.println("Average marks="+av);
 }
}

```

**Variable Description Table**

| Variable | Type          | Description                        |
|----------|---------------|------------------------------------|
| n        | int           | To input the number of students    |
| m[]      | Integer array | To store the marks of the students |
| i        | int           | To use in the loop                 |
| s        | int           | To find the sum                    |
| av       | double        | To find the average                |

**Output:**

```

Enter the number of students
5
Enter the marks
34 56 43 61 59
Average marks is: 50.0

```

- 22. Write a program to input 10 numbers in a single-dimensional array. Find the biggest, second biggest, and the smallest element of the array.**

**Ans.**

```

import java.util.*;
public class array_elements
{
 public static void main(String args[])
 {
 Scanner input=new Scanner(System.in);
 int i,max,min,smax;
 int n[]=new int[10];
 System.out.println("Enter 10 numbers:");
 for(i=0;i<10;i++)
 {
 n[i]= input.nextInt();
 }
 max=smax=min=n[0];
 for(i=0;i<10;i++)
 {
 if(n[i]>max)
 {
 smax=max;
 max=n[i];
 }
 if(n[i]>smax && n[i]<max)
 {
 smax=n[i];
 }
 if(n[i]<min)
 {
 min=n[i];
 }
 }
 }
}

```

```

 System.out.println("Biggest number is: "+max);
 System.out.println("Second biggest number is: "+smax);
 System.out.println("Smallest number is: "+min);
 }
}

```

**Variable Description Table**

| Variable | Type          | Description                       |
|----------|---------------|-----------------------------------|
| i        | int           | To use in the loop                |
| n[]      | Integer array | To store the numbers              |
| max      | int           | To find the biggest number        |
| min      | int           | To find the smallest number       |
| smax     | int           | To find the second biggest number |

**Output:**

```

Enter 10 numbers:
43 76 45 87 55 23 76 99 70 55
Biggest number is: 99
Second biggest number is: 87
Smallest number is: 23

```

- 23. Write a program to input 10 numbers into a single-dimensional array. Use the linear search technique to search for a number entered by a user to check whether the element exists in the array or not.**

**Ans.**

```

import java.util.*;
public class serach_number
{
 public static void main(String args[])
 {
 Scanner input=new Scanner(System.in);
 int i,ns,x=0;
 int n[]=new int [10];
 System.out.println("Enter 10 numbers");
 for(i=0;i<10;i++)
 {
 n[i]= input.nextInt();
 }
 System.out.println("Enter a number that you want to search: ");
 ns=input.nextInt();
 for(i=0;i<10;i++)
 {
 if(n[i]==ns)
 {
 x=1;
 break;
 }
 }
 if(x==1)
 System.out.println("Number exists");
 else
 System.out.println("Number does not exist");
 }
}

```

**Variable Description Table**

| Variable | Type          | Description                           |
|----------|---------------|---------------------------------------|
| n[]      | Integer array | To input numbers in the SDA           |
| i        | int           | To use in the loop                    |
| ns       | int           | To input the number to search         |
| x        | int           | To confirm the presence of the number |

**Output:**

Enter 10 numbers:

11    22    33    44    55    66    77    88    99    110    55

Enter a number that you want to search: 55

Number exists

- 24. Write a program to input 10 numbers into a single-dimensional array. Use the binary search technique to search for a number entered by a user to check whether the element exists in the array or not.**

**Ans.** `import java.util.*;`

`public class assign_24`

`{`

`public static void main(String args[])`

`{`

`Scanner input=new Scanner(System.in);`

`int i,lb=0,ub=9,m,ns,k=0;`

`int n[]=new int[10];`

`System.out.println("Enter 10 numbers");`

`for(i=0;i<10;i++)`

`{`

`n[i]=input.nextInt();`

`}`

`System.out.print("Enter a number that you want to search: ");`

`ns= input.nextInt();`

`while(lb<=ub)`

`{`

`m=(lb+ub)/2;`

`if(ns<n[m])`

`ub=m-1;`

`if(ns>n[m])`

`lb=m+1;`

`if(ns==n[m])`

`{`

`k=1;`

`break;`

`}`

`}`

`if(k==1)`

`System.out.println("Number exists");`

`else`

`System.out.println("Number does not exist");`

`}`

`}`

**Variable Description Table**

| Variable | Type          | Description                           |
|----------|---------------|---------------------------------------|
| n[]      | Integer array | To input numbers in the SDA           |
| i        | int           | To use in the loop                    |
| ns       | int           | To input the number to search         |
| x        | int           | To confirm the presence of the number |
| lb       | int           | To mark the lower base of the array   |
| ub       | int           | To mark the upper base of the array   |
| m        | int           | To mark the mid position of the array |

**Output:**

```
Enter 10 numbers
12 24 36 48 60 72 84 96 108 120
Enter a number that you want to search:65
Number does not exist
```

25. Write a program to store 6 elements in an array P and 4 elements in an array Q. Concatenate both the arrays to create the third array R that contains all the elements of array P and Q. Also, display the resultant array.

**P []= {4, 6,1,2,3,10}**

**Q[]={19, 23, 7, 8}**

**Output: R[]={ 4, 6, 1, 2, 3, 10, 19, 23, 7, 8}**

**Ans.**

```
import java.util.*;
public class concatenate
{
 public static void main(String args[]) {
 Scanner input=new Scanner(System.in);
 int P[]=new int[6];
 int Q[]=new int[4];
 int R[]=new int[10];
 int i=0;
 System.out.println("Enter 6 elements of array P:");
 for (i=0; i < P.length; i++)
 {
 P[i]=input.nextInt();
 }
 System.out.println("Enter 4 elements of array Q:");
 for (i=0; i < Q.length; i++)
 {
 Q[i]=input.nextInt();
 }
 i=0;
 while(i < P.length)
 {
 R[i]=P[i];
 i++;
 }
 }
}
```



```

 int j=0;
 while(j < Q.length)
 {
 R[i++]=Q[j++];
 }
 System.out.println("Elements of Array R:");
 for (i=0; i < R.length; i++)
 {
 System.out.print(R[i] + " ");
 }
 }
}

```

**Variable Description Table**

| Variable | Type          | Description                 |
|----------|---------------|-----------------------------|
| p[]      | Integer array | To input numbers in the SDA |
| q[]      | Integer array | To input numbers in the SDA |
| r[]      | Integer array | To input numbers in the SDA |
| i        | int           | To use in the loop          |

**Output:**

```

Enter 6 elements of array P:
45 34 54 65 55 45
Enter 4 elements of array Q:
38 87 56 40
Elements of Array R:
45 34 54 65 55 45 38 87 56 40

```

# Viva Questions

**1. What do you understand by the term access specifier?**

Ans. These are keywords that define the accessibility of the function, they can either be public, private, or protected.

**2. What are the actual parameters?**

Ans. Actual parameters are those that appear during function invocation or function call.

Example: `ob.sum(4,5);`

**3. What is the API (Application Programming Interface)?**

Ans. API consists of libraries of pre-compiled code that programmers can use in their applications and programs for designing software.

**4. What are arguments?**

Ans. The values that are passed to the method during its invocation from the caller module are called arguments.

**5. What is a block of statements?**

Ans. It is a sequence of one or more statements that are enclosed within curly brackets. A block statement is generally used to group together several statements, so they can be used as a single statement anywhere in the program.

**6. What is BlueJ?**

Ans. BlueJ is an Integrated Development Environment (IDE) used for writing, editing, compiling, testing, executing, and debugging Java programs.

**7. What is the use of break statement?**

Ans. It is a jump statement that is used to terminate the loop or switch-case when the condition is satisfied.

**8. What do you understand by the term buffer?**

Ans. It is a high-speed temporary storage memory location. Buffer is also sometimes referred to as cache.

**9. What do you understand by bytecode?**

Ans. Java bytecode consists of a set of pseudo machine language instructions that are understood by the JVM and are independent of the underlying hardware. Java bytecode is obtained when compilation is done by the compiler.

**10. What do you understand by the data type character?**

Ans. It is a data type that may contain letters (small or capital), digits, blank space, or special characters.

**11. What is a class?**

Ans. It is a blueprint that represents a set of objects that share some common characteristics and behaviours.

**12. What is coercion?**

Ans. Implicit data type conversion is termed as "coercion" in which the data gets converted from a lower type to its higher type without any intervention from the user.

**13. What is a compiler?**

Ans. A compiler is a language processor that converts high-level language instructions into machine-level language in one go.

**14. What is a constructor?**

Ans. It is a member method of a class that has the same name as that of a class and has no return type, not

even void. It is used to initialise the instance variables of a class and gets invoked automatically as soon as the object of the class is created.

**15. What is the use of continue statement?**

Ans. The continue statement is used to skip the current iteration and go to the next iteration when the condition is satisfied within the program.

**16. What do you understand by the "dangling else" problem?**

Ans. The "dangling else" problem arises when the number of 'if' is more than the number of 'else' in a nested if statement.

**17. What do you understand by data abstraction?**

Ans. The act of representing only essential features without including their background details is called data abstraction.

**18. What is debugging?**

Ans. Debugging is the process of identifying and correcting errors that occur during the compilation of the program.

**19. What is the use of the dot (.) operator?**

Ans. It is a special type of operator that is used to access instance members of an object or class members of a class.

**20. What do you understand by dynamic binding?**

Ans. It is the process of linking the function call with the function signature during the execution of the program.

**21. What is encapsulation?**

Ans. The process of binding up the data members and member functions (methods) together into a single unit is called encapsulation.

**22. What is an escape sequence?**

Ans. It is a character preceded by a backslash (\) that has a special meaning to the compiler. For example, the newline character ("\\n" ) is used within print statements to display the text in new line.

**23. What is an expression?**

Ans. It is any valid and legal combination of operators, constants, and variables.

**24. What is a fall-through situation?**

Ans. When you miss the break statement in a switch-case structure, it leads to a "fall of control" in which all the cases are executed till the break statement arrives, such a situation is known as a 'fall-through' situation.

**25. What is the significance of the final keyword?**

Ans. The final keyword is used as a prefix to a variable, which makes it a constant.

**26. What are the formal parameters?**

Ans. Formal parameters are used in the function definition. They contain the variable name along with the variable data type.

**27. What do you understand by function prototype?**

Ans. It is the first line of a function that consists of access specifier, a modifier, a return type, a function name, and a list of parameters is called a function prototype.

**28. What are identifiers?**

Ans. Identifiers are the fundamental building blocks of a program. They are also known as variables. Their values change during the execution of the program.

**29. What is an impure expression?**

Ans. Impure expressions are those that have several operands of distinct data types.

**30. What is an infinite loop?**

Ans. An infinite loop is a loop that never ends.

**31. What is inheritance?**

Ans. It is the ability of a class to derive the properties from another class.

**32. What do you understand by the iteration statements?**

Ans. Statements that allow a set of instructions to be performed repeatedly until a certain condition is met are referred to as iteration statements.

**33. Define JDK.**

Ans. The JDK is a software development tool used for developing Java applications and applets.

**34. Define JVM.**

Ans. JVM stands for Java Virtual Machine. It is the interpreter of Java that converts Java bytecode into machine code.

**35. What are the keywords?**

Ans. Keywords are the reserved words that convey a special meaning to the Java program. They cannot be used as variable names.

**36. What are the literals?**

Ans. Literals are the fixed values that are assigned to variables that do not change their values during program execution.

**37. What are the local variables?**

Ans. Local variables are variables declared inside a method or block.

**38. What is a nested loop?**

Ans. A loop that contains another loop inside its body is known as a nested loop.

**39. What is the use of the new keyword?**

Ans. It is a keyword that allocates memory and instantiates an object.

**40. What are non-primitive data types?**

Ans. These data types depend on other data types and are also known as reference data types. For example, array, classes, objects, interfaces, etc.

**41. What is Oak?**

Ans. It is the earlier name of Java.

**42. Who developed Java?**

Ans. James Gosling developed Java.

**43. What is an object?**

Ans. An object is a unique entity that has some characteristics and behaviours.

**44. What is polymorphism?**

Ans. It is the ability of a method to behave in more than one form.

**45. What is the precedence of operators?**

Ans. The sequence in which operators are evaluated in a Java expression is termed as precedence of operators.