

As per the latest syllabus
Prescribed by the CISCE

9
ICSE

COMPUTER APPLICATIONS

(Subject Code - 86)

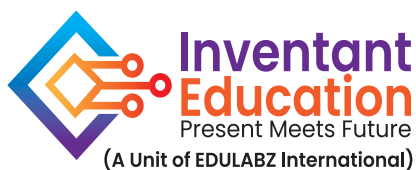
With BlueJ



Palvi Gupta

M.Tech, B.E.
(Computer Science & Engineering)
D.A.V. University, Jalandhar





D-47, Sector 2, Noida, Uttar Pradesh-201301

Email : info@inventanteducation.com

Customer care number: 18002022912

Disclaimer

This educational material, developed by Inventant Education, focuses on STEM education. While we strive for accuracy, we do not guarantee completeness or suitability for all purposes. Inventant Education is not liable for any damages resulting from the material's use or any inadvertent omissions or errors. References to products, services, or organizations are for informational purposes and do not imply endorsement.

First Edition : October, 2024

Price: ₹ 589

Copyright

© Inventant Education, a unit of Edulabz International

All rights reserved. No part of this educational material may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Inventant Education. Permission is granted to educational institutions for classroom use, provided that the material is used for non-commercial, educational purposes and is not sold or distributed for profit. For any other use, please seek written permission from Inventant Education. Inventant Education and the Inventant Education logo are registered trademarks of Inventant Education and/or its subsidiaries. All other trademarks and trade names are the property of their respective owners.

Printed at Plasticote (India), Noida

Preface

The **Class 9th and 10th ICSE Computer Applications book** is designed to introduce students to the exciting and rapidly evolving world of computer science. It serves as a comprehensive guide that lays a solid foundation in programming and other essential computing concepts. The book is structured to align with the **ICSE (Indian Certificate of Secondary Education)** syllabus, ensuring that all relevant topics are covered systematically and in-depth, preparing students for both academic success and practical application.

Key Features of the Book:

1. **Introduction to Basic Computer Concepts:** The book begins with fundamental concepts. This forms a strong foundation for understanding advanced topics.
2. **Object-Oriented Programming (OOP):** The primary focus of the book is on **Java programming**, an object-oriented language. Students are introduced to the core principles of OOP, such as classes, objects, inheritance, and polymorphism, which are critical for modern software development. The book uses simple and clear examples to explain these concepts, making it easier for students to grasp.
3. **Programming Basics:** Students learn the syntax of Java and are guided step-by-step through writing their first programs. They are taught how to use variables, data types, operators, iterative, nested and conditional statements to build simple applications.
4. **Problem-Solving Skills:** The book emphasizes **logical thinking and problem-solving** through coding exercises. It includes a variety of practical programming examples and challenges to encourage students to apply their knowledge in real-world scenarios. This approach not only enhances their programming skills but also improves their analytical thinking.
5. **Graphical User Interface (GUI) Programming:** One of the standout features of the book is its introduction to GUI programming, where students learn how to create interactive applications. They are guided through the basics of designing user interfaces with buttons, text fields, and other controls, providing them with the tools to create functional and user-friendly programs.
6. **Exercises and Projects:** At the end of each chapter, there are exercises that include both theoretical and practical questions to test students' understanding of the material. Additionally, **project work** is assigned to help students consolidate their learning by developing larger, more complex applications. These projects often mimic real-world problems, fostering creativity and deeper learning.
7. **ICSE Examination Preparation:** The book is structured to help students prepare effectively for the **ICSE exams**. It includes model questions, Java Programming, and previous year's exam questions, providing ample practice for students to master the format and style of the exam.
8. **Learning Objectives:** Clearly define what students will learn by the end of each chapter.
9. **Definition:** Provide clear and concise definitions of important terms to help students understand concepts better. It should be simple, concise, and relevant to the topic at hand.
10. **Extra Time:** Suggested activities or extensions for students who finish early or need additional challenges. It ensures that fast learners stay engaged and deepen their knowledge.
11. **Know More:** Provide additional facts or insights to encourage deeper understanding of the topic. This section sparks curiosity and provide context or trivia related to the topic.
12. **Lab Activity:** Hands-on exercises that allow students to apply theoretical concepts in practical scenarios. It enhances understanding by giving students the opportunity to experiment with the software.
13. **Teacher Notes:** Provide teachers with background information, teaching tips, and key points to emphasize during the lesson. It supports educators by offering guidance on how to deliver content effectively.

Teacher's Resource Book: Provides lesson plans and solutions to the textbook questions.

Online Support: Downloadable e-books (for teachers only)

I owe my success in this project to the unwavering support of my family. My husband's encouragement, my son Sidharth's joy, my parents' guidance, and my mother-in-law's understanding have all been crucial in helping me achieve this balance.

Suggestions for the improvement of the book are most welcome.

—Author

About the Series

Learning Objectives

After studying this chapter, you will be able to do:

- Explain ethical issues
- Understand hacking and IPR
- Discuss malicious intent and code
- Identify Good etiquette practices

Learning Objectives

Lists key takeaways for each chapter.



Definition

Abstraction means to represent the essential feature without detailing the background implementation or internal working detail.

Definition

Provides clear definitions of important terms.

EXTRA TIME

A patent is defined as an exclusive right granted for an invention.

Extra Time

Offers activities for fast learners seeking more challenges.

KEY TERMS

- **Privacy:** Privacy refers to an individual's right to control their personal information and how it is collected, used, and shared.
- **Security:** Security entails protecting data, systems, and networks from unauthorized access, misuse, or damage.
- **Data Ethics:** Data ethics refers to the principles, values, and guidelines governing the ethical use of data, including considerations of privacy, security, transparency, fairness, and accountability.
- **Intellectual Property Rights:** Intellectual property rights refer to legal protections for creative works and inventions, including patents, copyrights, and trademarks.

Key Terms

Provides meaning of important terms

Scan the QR code for more solved questions.



QR Codes

Additional solved questions related to each chapter effortlessly through QR codes

Lab Activity

1. Design a library management system. Find out the class and its objects.
2. Try to list the data members and methods of the identified class.

[Application]

Lab Activity

Hands-on exercises to apply concepts practically

SOLVED PROGRAMMING

1. Write a program to calculate the area and perimeter of the rectangle.

Ans:

```
public class Rectangle {  
    public static void main(String[] args) {
```

Java/Solved Programs

Contains additional programs related to the chapter/topic



SAMPLE PROJECTS

1. Write a program in Java to display the gas bill.

Sample Project

Contains the real-life applications to use the concepts learnt

Know More

* Objects communicate with each other by invoking methods and accessing through well-defined interfaces, facilitating interaction and collaboration

Know More

Shares extra facts to deepen understanding.

Activity

Try to think more example of abstraction in your daily life.

Activity

Interactive activities to enhance your learning experience

Exercises - 1 (Solved)

A. Multiple choice questions.

1. Directly copying text, word for word is called:

- a. Cyber security b. Cyber ethics c. Plagiarism d. All of these

[Understanding]

Exercises

Provides solved and unsolved questions of the chapter

Teacher's Notes

- Begin the chapter by asking students about the benefits of the advantages of the Internet.
- Talk about the ethical and unethical practices related to the Internet.
- List some important safety measures to be taken while using the Internet.

Teacher's Notes

Offers background info and teaching tips for educators

SYLLABUS

COMPUTER APPLICATIONS (86)

Aims:

1. To empower students by enabling them to build their own applications.
2. To introduce students to some effective tools to enable them to enhance their knowledge, broaden horizons, foster creativity, improve the quality of work and increase efficiency.
3. To develop logical and analytical thinking so that they can easily solve interactive programs.
4. To help students learn fundamental concepts of computing using object oriented approach in one computer language.
5. To provide students with a clear idea of ethical issues involved in the field of computing.

CLASS IX

There will be one written paper of two hours duration carrying 100 marks and Internal Assessment of 100 marks.

THEORY – 100 Marks

1. Introduction to Object Oriented Programming concepts

- (i) Principles of Object Oriented Programming, (Difference between Procedure Oriented and Object oriented). All the four principles of Object Oriented Programming should be defined and explained using real life examples (Data abstraction, Inheritance, Polymorphism, Encapsulation).
- (ii) Introduction to JAVA - Types of java programs – Applets and Applications, Java Compilation process, Java Source code, Byte code, Object code, Java Virtual Machine (JVM), Features of JAVA.
Definition of Java applets and Java applications with examples, steps involved in compilation process, definitions of source code, byte code, object code, JVM, features of JAVA - Simple, Robust, secured, object oriented, platform independent, etc.

2. Elementary Concept of Objects and Classes

Modelling entities and their behaviour by objects, a class as a specification for objects and as an object factory, computation as message passing/method calls between objects (many examples should be done to illustrate this). Objects encapsulate state (attributes) and have behaviour (methods). Class as a user defined data type.

A class may be regarded as a blueprint to create objects. It may be viewed as a factory that produces similar objects. A class may also be considered as a new data type created by the user, that has its own functionality.

3. Values and data types

Character set, ASCII code, Unicode, Escape sequences, Tokens, Constants and Variables, Data types, type conversions.

Escape sequences [\n, \t, \\, \", \'], Tokens and its types [keywords, identifiers, literals, punctuators, operators], primitive types and non-primitive types with examples, Introduce the primitive types with size in bits and bytes, Implicit type conversion and Explicit type conversion.

4. Operators in Java

Forms of operators, Types of operators, Counters, Accumulators, Hierarchy of operators, 'new' operator, dot (.) operator.

Forms of operators (Unary, Binary, Ternary), types of operators (Arithmetic, Relational, Logical, Assignment, Increment, Decrement, Short hand operators), Discuss precedence and associativity of operators, prefix and postfix, Creation of dynamic memory by using new operator, invoking members of class using dot operator, Introduce System.out.println() and System.out.print() for simple output. (Bitwise and shift operators are not included).

5. Input in Java

Initialization, Parameter, introduction to packages, Input streams (Scanner Class), types of errors, types of comments.

Initialization – Data before execution, Parameters – at the time of execution, input stream – data entry during execution – using methods of Scanner class [nextShort(), nextInt(), nextLong(), nextFloat (), nextDouble(), next(), nextLine(), next () .charAt(0)]

Discuss different types of errors occurring during execution and compilation of the program (syntax errors, runtime errors and logical errors). Single line comment (//) and multiline comment (/* ... */))

6. Mathematical Library Methods

Introduction to package java.lang [default], methods of Math class.

pow(x,y), sqrt(x), cbrt(x), ceil(x), floor(x), round (x), abs(a), max(a, b), min(a,b), random().

Java expressions – using all the operators and methods of Math class.

7. Conditional constructs in Java

Application of if, if else, if else if ladder, switch-case, default, break.

if, if else, if else if, Nested if, switch case, break statement, fall through condition in switch case, Menu driven programs, System.exit(0) - to terminate the program.

8. Iterative constructs in Java

Definition, Types of looping statements, entry controlled loops [for, while], exit controlled loop [do while] , variations in looping statements, and Jump statements.

Syntax of entry and exit controlled loops, break and continue, Simple programs illustrating all three loops, inter conversion from for – while – do while, finite and infinite, delay, multiple counter variables (initializations and updates). Demonstrate break and continue statements with the help of loops.

Loops are fundamental to computation and their need should be shown by examples.

9. Nested for loops

Introduce nested loops through some simple examples. Demonstrate break and continue statements with the help of nested loops.

Programs based on nested loops [rectangular, triangular [right angled triangle only] patterns], series involving single variable. (Nested while and nested do while are not included.)

10. Computing and Ethics

Ethical Issues in Computing.

Intellectual property rights; protection of individual's right to privacy; data protection on the internet; protection against Spam; software piracy, cybercrime, hacking, protection against malicious intent and malicious code. The stress should be on good etiquette and ethical practices.

INTERNAL ASSESSMENT - 100 Marks

This segment of the syllabus is totally practical oriented. The accent is on acquiring basic programming skills quickly and efficiently.

Programming Assignments (Class IX)

Students are expected to do a minimum of 20 assignments during the whole year to reinforce the concepts studied in the class.

Suggested list of Assignments:

The laboratory assignments will form the bulk of the course. Good assignments should have problems which require design, implementation and testing. They should also embody one or more concepts that have been discussed in the theory class. A significant proportion of the time has to be spent in the laboratory. Computing can only be learnt by doing.

The teacher-in-charge should maintain a record of all the assignments done as a part of practical work throughout the year and give it due credit at the time of cumulative evaluation at the end of the year.

Some sample problems are given below as examples. The problems are of varying levels of difficulty:

- (i) Programs using Assignment statements.
Example: Calculation of Area / Volume / Conversion of temperature / Swapping of values etc.
- (ii) **Programs based on**– Input through parameters.
Example: Implementation of standard formula etc.
- (iii) **Programs based on** – Input through Scanner class.
Example: Implementation of standard formula etc.
- (iv) Programs based on Mathematical methods.
Example: larger/smaller of two numbers, cube root, square root, absolute value, power, etc.
- (v) Programs based on if, if else, if else if ladder, nested if etc.
 - (a) if programs
 - Larger / smaller of two numbers
 - To check divisibility of a number, etc.
 - (b) if - else programs
 - Odd or even number
 - Eligibility to vote
 - Upper case or lower case
 - Positive or negative number
 - Vowel or Consonant
 - Buzz number etc.
 - (c) if-else-if programs
 - Programs based on discount/interest/ bonus/ taxes/ commission.
 - Programs based on slab system.
 - Programs based on Nested if.
- (vi) Programs on switch case.
 - (a) Day of a week
 - (b) Name of the month
 - (c) Names of the seasons
 - (d) Calculator
 - (e) Vowel or consonant etc.
- (vii) Programs based on Looping Statement
 - (a) Programs based on for looping statement.
 - (b) Programs based on printing simple series, summation of simple series, product of simple series.
 - (c) Prime number, perfect number, composite number, Fibonacci series. Lowest Common Multiple (LCM), Highest Common Factor (HCF) etc.

- (d) To find the biggest and smallest number from n number of entered numbers.
- (e) Program based on while loop like Armstrong number, Spy number, Niven number, Palindrome number, etc.
- (viii) Programs based on nested loops [rectangular, triangular(right angled triangle only) patterns], series involving single variable.
- (ix) Generate first n multiples of numbers from 1 to the limit input by the user.
- (x) Menu Driven programs.

Important: This list is indicative only. Teachers and students should use their imagination to create innovative and original assignments.

EVALUATION

Proposed Guidelines for Marking

The teacher should use the criteria below to judge the internal work done. Basically, four criteria are being suggested: class design, coding and documentation, variable description and execution or output. The actual grading will be done by the teacher based on his/her judgment. However, one possible way: divide the outcome for each criterion into one of 4 groups: excellent, good, fair/acceptable, poor/unacceptable, then use numeric values for each grade and add to get the total.

Class design:

- Has a suitable class (or classes) been used?
- Are all attributes with the right kinds of types present?
- Is encapsulation properly done?
- Is the interface properly designed?

Coding and Documentation:

Is the coding done properly? (choice of names, no unconditional jumps, proper organization of conditions, proper choice of loops, error handling code layout). Is the documentation complete and readable? (class documentation, variable documentation, method documentation, constraints, known bugs – if any).

Variable and Description

Format for variable description:

Name of the variable	Data Type	Purpose/Description

Evaluation of practical work (Assignments) will be done as follows:

Subject Teacher (Internal Examiner): 100 Marks

Class design (100 marks)	Class design (20 marks)	Variable description (20 marks)	Coding and Documentation (20 marks)	Execution OR Output (40 marks)
Excellent				
Good				
Fair				
Poor				

Aligned with NEP 2020 and NCF 2023

FEATURES OF NEP 2020

21st Century Skills

Learning Skills (4Cs)

- ✓ Critical Thinking
- ✓ Creativity
- ✓ Communication
- ✓ Collaboration

Literacy Skills (IMT)

- ✓ Information Literacy
- ✓ Media Literacy
- ✓ Technology Literacy

Life Skills (FLIPS)

- ✓ Flexibility
- ✓ Leadership & Responsibility
- ✓ Initiative
- ✓ Productivity & Accountability
- ✓ Social Interaction

BASED ON NCF 2023

In NCF 2023, **curriculum** means not only what is given in the books, but also how the learners learn in school, the school's environment, and more. To make learning better, we need positive changes in all these areas.

The Six Pramanas

Inference

Perception

Comparison

Verbal Testimony

Non-Apprehension

Postulation

How to Access Digital Content through QR Code

For Website Users

- ✓ "Visit "digital.inventanteducation.com"
- ✓ Click "Register" button available on the top-right.
- ✓ Select 'Teacher/Student' in 'User' Type.
- ✓ Enter your name, email, mobile number and password.
- ✓ Click 'Register', and Enter the OTP to verify your mobile/email.
- ✓ Once registered, login on to the website and go to **Scan and Learn** section. Enter the Codes printed below the QR Codes to view the required content.

For Mobile Users

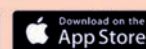
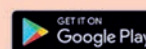
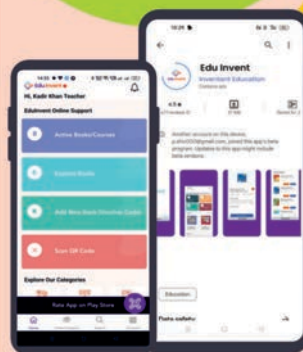
- ✓ Go to Google Play Store or Apple App Store.
- ✓ Type 'Edu Invent' in the search bar.
- ✓ Tap 'Install'. The app will take a few moments to download and install.
- ✓ Once installed, tap 'Open' to launch the app.
- ✓ Register yourself and login on the app.
- ✓ On the dashboard, click Scan QR Code button.
- ✓ Scan a QR Code printed in the book to explore the learning content associated with the QR Code.

Download Edu Invent App

Download our mobile app to avail free online support with the books published by Inventant Education.



Scan this QR code to download the app



Contents

1. Introduction to Object Oriented Programming concepts	13
• Introduction	• Defference Between Complier and Interpreters
• Procedure-Oriented Programming	• Type of Java Program
• Object-Oriented Programming	• Basic Term in Java
• Difference Between Oop And Pop	• Java API (Java Application Programming Interface)
• Elementary Concept Of Objects And Classes	• Features of Java
• Software Objects Vs Real World Objects	• Importance of Java
• Introduction to Java	
• Translators	
2. Elementary Concept of Objects and Class	32
• Introduction	• Properties of Class and Object
• Class	• Difference Between Class and Object
• Need of a Class in Java	• Relation Between Class and Objects
• Rules for Naming a Class	• Message Passing Between Objects
• Objects	
3. Values and data types	46
• Character Set	• Token
• Escape sequence	• Type conversion
4. Operators in Java	69
• Introduction	• Output Statements in Java
• Expression	• Operator Precedence
• Hierarchy of Arithmetic Operators	
5. Input in Java	98
• Introduction	• Java Scanner Class
• Common Method to Input Dates Values	• Errors in Java
• Compiling and Running Java Programs With Bluej	• Comments in Java
• Compiling (Translating) Your Java Program With Bluej	

6. Mathematical Library Methods	129
<ul style="list-style-type: none"> • Java Math Class • Basic Mathematical Methods • Arithmetic Expressions • Operation Precedence • Arithmetic Shortcuts 	
7. Conditional Constructs in Java.....	149
<ul style="list-style-type: none"> • Introduction • Flow Control Statements • If-Else vs. Switch statements • Jump • Using Break to Exit a Loop • Continue • Menu-Driven Program in Java • System.exit() In Java 	
8. Iterative constructs in Java.....	191
<ul style="list-style-type: none"> • Introduction • Loop • For Loops in Java • While Loop • Do While Loop in Java • Difference Between for Loop, While Loop, and Do-While Loop in Java 	
9. Nested loop in Java	235
<ul style="list-style-type: none"> • Nested Loop 	
10. Computing and ethics	267
<ul style="list-style-type: none"> • Introduction • Ethical Issues • Protection of Individual's Right to Privacy • Data Protection on the Internet • Intellectual Property Rights • Spam • Software Piracy • Cybercrime • Hacking • Malicious Intent • Malicious Code • Good Etiquette Practice • Good Ethical Practices 	
 Practical Exercise	283
 Project	291
 Viva Questions	293



Introduction to Object Oriented Programming concepts

Learning Objectives ♦

After studying this chapter, you will be able to:

- Understand the fundamental principles of Procedure-Oriented programming, including the structure and flow of data and procedures
- Analyze the advantages and limitations of Procedure-Oriented programming in software development
- Define the concept of Object-Oriented Programming and its significance in modern software development
- Identify real-world objects and their interactions to understand how they relate to OOP concepts
- Explain the four core principles of OOP: encapsulation, inheritance, polymorphism, and abstraction
- Difference Between Procedure-Oriented and Object-Oriented Programming
- Compare and contrast the key characteristics of Procedure-Oriented programming and Object-Oriented programming
- Describe Java as a versatile, platform-independent programming language and its role in software development
- Distinguish between Java applets and applications, understanding their use cases and execution environments
- Illustrate the role of the Java Virtual Machine (JVM) in executing Java bytecode across different platforms
- Explain the significance of Java source code, bytecode, and object code in the programming lifecycle
- Analyze the differences between these forms of code and their impact on performance and portability

INTRODUCTION

Every programming language follows a certain approach to design the software. For a software to be implemented successfully, it is vital to make it as real as possible. Real life problems are very complex and it is important to convert these requirements into a model which can be developed using a programming language. You have many programming approaches like Object-oriented, Procedure-Oriented, functional programming and many more.

Definition

A program is a set of instructions given to a computer to perform a specific task.

PROCEDURE-ORIENTED PROGRAMMING

A program in a procedural language is a list of instructions where each statement tells the computer to do something. It focuses on procedure (function) & algorithm is needed to perform the derived computation.

When program become larger, it is divided into function & each function has clearly defined purpose. Dividing the program into functions & module is one of the cornerstones of structured programming.

Example of POP are C, VB, FORTRAN, Pascal, etc.



Programming paradigm is an approach to solve a given problem by writing a set of instructions in a programming language. It refers to the organizing principles of a program.

Characteristics of Procedure-Oriented Programming

The characteristics of Procedure-Oriented Programming are:

1. It focuses on process rather than data.
2. It takes a problem as a sequence of things to be done such as reading, calculating and printing. Hence, a number of functions are written to solve a problem.
3. A program is divided into a number of functions and each function has a clearly defined purpose.
4. Most of the functions share global data.
5. Data moves openly around the system from function to function.

Limitations of Procedure-Oriented Programming

The Limitations of Procedure-Oriented Programming are:

1. Its emphasis on doing things. Data is given a second-class status even though data is the reason for the existence of the program.
2. Since every function has complete access to the global variables, the new programmer can corrupt the data accidentally by creating a function. Similarly, if new data is to be added, all the functions needed to be modified to access the data.
3. It is often difficult to design because the components function and data structure do not model the real world.
4. It is difficult to create new data types. The ability to create the new data type of its own is called extensibility. Structured programming languages are not extensible.

To overcome these limitations, Object-Oriented programming was introduced.

OBJECT-ORIENTED PROGRAMMING

Object-Oriented Programming (OOP) is a programming paradigm centered around the concept of “objects,” which are instances of classes that encapsulate data and behaviour. At its core, OOP emphasizes the organization of code into reusable, modular components that model real-world entities.

The **examples** of OOPs are Python, Java, Basic programming and C++, etc.

The key concepts of OOP are:

1. **Abstraction:** Abstraction involves simplifying complex systems by focusing on the essential characteristics while hiding unnecessary details. In OOP, abstraction is achieved through the use of abstract classes and interfaces. Abstract classes provide a blueprint for other classes but cannot be

Definition

Abstraction means to represent the essential feature without detailing the background implementation or internal working detail.

instantiated themselves, while interfaces define a contract for classes to implement certain methods without specifying the implementation details.

It is also known as **data hiding within a program**.

Example: An ATM abstracts the banking process. Users interact with the ATM interface to perform tasks like withdrawing cash, checking account balances, or transferring funds without needing to understand the inner workings of banking systems.

2. **Encapsulation:** Encapsulation is the bundling of data and methods that operate on that data within a single unit, typically a class. It allows for the hiding of implementation details from the outside world and enables better control over access to the data. This helps in creating more modular and maintainable code.

Example: Consider a car. The engine of the car encapsulates various internal components such as pistons, crankshafts, and camshafts. These components are hidden from the outside and interact with the external world only through well-defined interfaces like the accelerator pedal or ignition key.

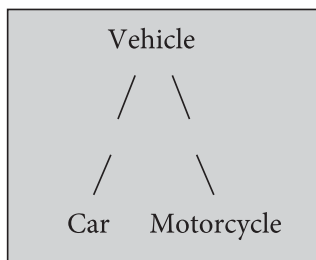
Another example is a simple Soft Drink Vending machine. It serves you the drinks whenever you request. In this case, you do not know the background of how it serves you the drink when you press a button. This is Abstraction. It contains both data (soft drinks) and functions (serving drinks). This is called Encapsulation.

3. **Inheritance:** Inheritance is a mechanism by which a class can inherit properties and behaviour from another class, known as the superclass or base class. The class inheriting from the superclass is called a subclass or derived class. Inheritance promotes code reusability and allows for the creation of hierarchical relationships between classes. For example, a “Vehicle” class may be a superclass, with subclasses like “Car” and “Truck” inheriting from it.

Example: Think about vehicles. You have a hierarchy of vehicles such as cars, trucks, and motorcycles. Each type of vehicle inherits common characteristics from a superclass “Vehicle,” such as having wheels, an engine, and the ability to move. However, each subclass may also have its own unique attributes and behaviours. For instance, a truck might inherit from the Vehicle class but also have additional features like a larger cargo capacity.

Consider a scenario where we have a superclass called Vehicle, which represents common characteristics of vehicles. You also have two subclasses: Car and Motorcycle, which inherit from the Vehicle class.

In this diagram:



Vehicle is the superclass, which contains general attributes and behaviours shared by all vehicles.

Car and Motorcycle are subclasses of Vehicle, which inherit attributes and behaviours from the superclass while also having their own unique attributes and behaviours.

Activity

Try to think more example of Polymorphism in your daily life.

4. **Polymorphism:** Polymorphism comes from the Greek words **poly** and **morphism**. **poly** means **many** and **morphism** means **form**. Polymorphism allows objects of different classes to be treated as objects of a common superclass. It enables the same method to behave differently based on the object it is called on. Polymorphism can be achieved through method overriding (where a subclass provides a

specific implementation of a method inherited from its superclass) and method overloading (where multiple methods with the same name but different parameters exist in the same class or hierarchy).

Example: In the animal kingdom, various species exhibit different behaviours, such as eating, sleeping, and moving.

Polymorphic Behaviour: Despite the diversity of species, many animals share common behaviours. For instance, carnivores, herbivores, and omnivores all eat, but they consume different types of food. Similarly, animals like birds, mammals, and fish all move, but they use different locomotion methods (e.g., flying, walking, swimming). By defining a common superclass (Animal) with methods like eat() and move(), and allowing subclasses to override these methods with species-specific behaviour, the animal kingdom demonstrates polymorphic behaviour.

By utilizing these principles, OOP enables the development of modular, maintainable, and scalable software systems, making it a widely used and essential paradigm in modern software development.



EXTRA TIME

There are two main types of polymorphism: Compile-time polymorphism and Run-time polymorphism.



Fact Time

Alan Kay invented OOP.

Characteristics of Object-Oriented Programming

The characteristics of Object-Oriented programming are:

1. OOP follows a bottom-up approach.
2. The program resulting from Object-Oriented programming is a collection of objects. Each object has its own data and a set of operations.
3. OOP restricts the free movement of data and the functions that operate on it. It uses a data/information hiding technique that allows better control over data.
4. A properly defined class can be reused, giving way to code reusability.
5. The concept of Object-Oriented programming models real-world entities very well.
6. Due to its Object-Oriented approach, it is extremely useful in solving complex problems.



Know More

- ★ The superclass is the class from which properties and behaviours are inherited.
- ★ The subclass is the class that inherits properties and behaviours from the superclass.

Limitation of Object-Oriented Programming

The limitation of Object-Oriented programming are:

1. The size of the programs created using this approach may become larger than the programs written using Procedure-Oriented programming approach.
2. Software developed using this approach requires a substantial amount of pre-work and planning.
3. OOP code is difficult to understand if you do not have the corresponding class documentation
4. In certain scenarios, these programs can consume a large amount of memory.

DIFFERENCE BETWEEN OOP AND POP

S.No	OOP	POP
1.	OOP takes a bottom-up approach in designing a program.	POP follows a top-down approach.
2.	Program is divided into objects depending on the problem.	Program is divided into small chunks based on the functions.
3.	Each object controls its own data.	Each function contains different data.
4.	Focuses on security of the data irrespective of the algorithm.	Follows a systematic approach to solve the problem.
5.	The main priority is data rather than functions in a program.	Functions are more important than data in a program.
6.	The functions of the objects are linked via message passing.	Different parts of a program are interconnected via parameter passing.
7.	Data hiding is possible in OOP.	No easy way for data hiding.
8.	Inheritance is allowed in OOP.	No such concept of inheritance in POP



EXTRA TIME

In short, Procedure-Oriented Programming focuses on procedures or functions that operate on data, while Object-Oriented Programming emphasizes the organization of code around objects that encapsulate data and behaviour. OOP provides features like encapsulation, inheritance, and polymorphism, which promote code reuse, modularity, and maintainability.

ELEMENTARY CONCEPT OF OBJECTS AND CLASSES

The main objective of Object-Oriented programming is to simulate the product as close as possible to the real world. How does it achieve that? Well, the answer is class and objects. These act as the fundamental blocks of Object-Oriented programming. Let's learn about them a little more.

1. **Class:** A class or object is a way of combining all sorts of data that relates to a single thing in one place, and a way of associating functions with that data. (For most intents and purposes, classes and objects are the same thing)
 - a. It is a template or blueprint from which objects are created.
 - b. It is a logical entity. It can't be physical.

For example, you can consider human as a class. A car can be a class.

So, let's say God made a class known as **Humans**. Now, when he created you, you became a part of that class. Similarly, your friends and family are also a part of that class. Everything in the real world is categorised into some main category. This main category is usually what you call classes in programming.

A **car** is a common real-world object that can be modeled using a class in Java.



EXTRA TIME

Java is a software bundle of related state and behaviour.

- a. **Car (Class)** The “Car” class serves as a blueprint for creating car objects. It encapsulates both data (attributes) and behaviour (methods) of cars.
 - b. **Attributes (Fields)** make, model, year, color, mileage: These are attributes of the “Car” class that represent the state or characteristics of a car object. Each car object will have its own values for these attributes.
 - c. **Behaviours (Methods)** start(), stop(), accelerate(), brake(): These are methods of the “Car” class that represent actions or behaviours that a car object can perform. Methods encapsulate the behaviour of a car, such as starting the engine, stopping the car, accelerating, and braking.
2. **Objects** An entity that is a part of a class is known as an object. An object has some behaviour and state. For example, you are a part of class Human. That makes you an object of Human class.

An object is what physically fills up the class. A class is just a blueprint but an object is an actual implementation of that blueprint. For example, God created a blueprint “Humans” which said that a human should have two legs, two arms, two ears, one mouth and the ability to talk. This is class Human. But when he created you, you were the actual implementation or result of his design of Humans. So, you are an object of class Human. Similarly, your friends and family are also objects of class Human.

Car		

	Attributes	

	- make	
	- model	
	- year	
	- color	
	- mileage	

	Behaviors	

	- start ()	
	- stop ()	
	- accelerate ()	
	- brake ()	



Know More

- ★ The term object means a combination of data and logic that represents some real time entity.

Let's take another example

A car is a road vehicle, typically with four wheels, powered by an internal combustion engine and able to carry a small number of people. Now, this is a blueprint of a car. Thus, a car is a class. Now, Maruti, Hyundai and Ford made their own versions of car. EcoSport, i20 and Brezza all have different designs and features but at the end of the day, they are all categorised into cars. Thus, these different Ford and Hyundai cars are all objects of a parent class “Car”.



EXTRA TIME

Classes and objects provide a way to model and represent real-world entities in Java programs. They enable code reuse, modularity, and abstraction, making it easier to manage and maintain complex systems.

SOFTWARE OBJECTS VS REAL WORLD OBJECTS

A real-world object is a physic body, on the other hand a software object is an abstraction of a data entity used in a programming paradigm known as Object Oriented Programming.

A software object is a logical representation of the real-world object and not the complete replica of it. For example, an employee of an organisation. Its software object can have properties like name, address, job, salary and so on but it cannot include every detail about that employee like how many degrees his elbow can bend.

In simple words, a software object tries to project itself as close as possible to the real-world object. That is how a software is created.

Characteristics of Software Object

A software object has three characteristics:

1. **State:** Represents data (value) of an object.
2. **Behaviour:** Represents the behaviour (functionality) of an object such as deposit, withdraw etc.
3. **Identity:** Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But it is used internally by the JVM to identify each object uniquely.

For example, your name, skin color, age, height, weight are the features that define you and define your state. The tasks you perform like talking, walking, working is your behaviour. You have a unique ID number that identifies you uniquely all over the country.

This was a general example of how to use concepts of Object-Oriented programming for real world problems.

INTRODUCTION TO JAVA

Java programming language was originally developed by **Sun Microsystems** which was initiated by **James Gosling** and released in **1995** as core component of **Sun Microsystems' Java platform** (Java 1.0 [J2SE]).

History of Java

The history of Java is an interesting journey that begins in the early 1990s. Here's a brief overview:

1. **Origins at Sun Microsystems:** Java was developed by **James Gosling**, Mike Sheridan, and Patrick Naughton at Sun Microsystems (later acquired by Oracle Corporation) in the early 1990s.

The project was initially named "Oak" after an oak tree outside Gosling's office. It was later renamed "Java" due to trademark issues.

2. **Public Release (1995):** Java 1.0, the first official version of Java, was released to the public by **Sun Microsystems** in 1995.

It included the **Java Development Kit** (JDK) and the **Java Runtime Environment** (JRE).

3. **Key Features:** Java was designed with the goal of being platform-independent, secure, and robust.

It introduced features such as automatic memory management (garbage collection), exception handling, and a rich set of standard libraries.

4. **"Write Once, Run Anywhere":** One of Java's most significant contributions to software development is its platform independence.

Java programs are compiled into bytecode, which can run on any device with a **Java Virtual Machine** (JVM), regardless of the underlying hardware or operating system. This principle is often summarized as **"Write Once, Run Anywhere"** (WORA).

5. **Popularity and Adoption:** Java quickly gained popularity due to its simplicity, portability, and suitability for building a wide range of applications, from desktop to web and enterprise systems.

It became the language of choice for many developers and organizations, leading to widespread adoption in various industries.

6. **Evolution and Versions:** Over the years, Java has continued to evolve with regular releases and updates introducing new features, enhancements, and improvements.

Major releases include Java 2 (renamed as Java SE), Java EE (Enterprise Edition) for enterprise applications, and Java ME (Micro Edition) for mobile and embedded devices.

Recent versions include Java 8, which introduced lambda expressions and the Stream API, and Java 11, which marked the transition to a faster release cadence with a new long-term support (LTS) model.

7. **Open Sourcing:** In 2006, Sun Microsystems open-sourced the Java platform under the GNU **General Public License** (GPL) through the **OpenJDK** (Java Development Kit).

This move further contributed to Java's widespread adoption and allowed the community to contribute to its development.

8. **Acquisition by Oracle:** In 2010, Oracle Corporation acquired Sun Microsystems, becoming the steward of the Java platform.

Oracle continues to develop and maintain Java, overseeing its evolution and providing updates and support to the community.

Throughout its history, Java has remained a versatile and powerful programming language, playing a significant role in shaping the modern software development landscape. It continues to be used extensively in a wide range of applications, from enterprise systems and web services to mobile apps and cloud computing.

Fact Time

Java was originally known as OAK.

TRANSLATORS

The computer language that is the machine code is almost impossible to be understood by the humans. So, humans write their programs in high level languages and then this is converted to low level language so that the computer can understand it. So, who does this conversion? Here comes the Translators.

Translators are software's which are used to convert high level languages to low level languages. There are two types of translators:

1. **Interpreters:** An interpreter reads and execute code line by line. In Java, the interpreter is part of JVM.
2. **Compilers:** A compiler is a Program that translates Java Source code into independent, intermediate representation of the code.



Know More

- * The Java development kit comes with a collection of tools that are used for developing and running Java programs.

DEFFERENCE BETWEEN COMPLIER AND INTERPRETERS

S.No	Feature	Compiler	Interpreter
1.	Purpose	Converts entire source code into machine code or intermediate code before execution.	Translate and execute the source code line by line.
2.	Output	Produces an output containing machine code or byte code.	Does not produce a seperate output file.
3.	Example	Jawa uses a compiler to convert source code into byte code.	Python user an interpreter to execute the code line by line.
4.	Efficiency	Generally produces faster execution as the entire code is optimized before execution.	Slower execution since the code is translated on the fly during execution.
5.	Platform Independence	Compiled code is usually platform specific unless compiled into ab intermediate code.	Interprered code is platform independent but sequences the presense inter preter on the targer machine.



Know More

- ★ In Java compilation, first, the compiler converts the source code to an intermediate language (Java Byte Code), which is further converted to machine code by the interpreter (Java Virtual Machine).

TYPE OF JAVA PROGRAM

There are two type of Java programs:

- Applications:** Java programs that run directly on your machine.
 - Applications must have a main().
 - Java applications are compiled using the javac command and run using the Java command.



EXTRA TIME

The term Java application generally refers to an application that is designed for stand-alone use. A frequently used slang term for application is app.

- Applets:** Java programs that can run over the Internet. The standard client/server model is used when the Applet is executed. The server stores the Java Applet, which is sent to the client machine running the browser, where the Applet is then run.
 - Applets do not require a main(), but in general will have a paint().
 - An Applet also requires an HTML file before it can be executed.

- c. Java Applets are also compiled using the javac command, but are run either with a browser or with the appletviewer command.

BASIC TERM IN JAVA

Before understanding the Java compilation program, let us first understand some of the basic terms used in Java.

1. **Source code:** These are the original instructions written by a programmer in Java language following the rules of Java.
 - a. Java source code refers to the human-readable text written by developers to create Java programs.
 - b. Source code files typically have a .java extension.
 - c. Java source code contains classes, methods, variables, and other elements of the Java programming language.
2. **Bytecode:** Bytecode is program code that has been compiled from source code into low-level code designed for a software interpreter. It may be executed by a Java Virtual Machine (such as a JVM) or further compiled into machine code, which is recognized by the processor.
 - a. Bytecode is an intermediate representation of Java programs generated by the Java compiler (javac) during the compilation process.
 - b. It is a set of instructions for the Java Virtual Machine (JVM) to execute.
 - c. Bytecode is platform-independent and can be executed on any device or operating system with a compatible JVM implementation.
 - d. Bytecode files have a class extension and are stored in compiled form.
3. **Object code:** Object code is code generated by a compiler or other translator, consisting of machine code, byte code, or possibly both, combined with additional metadata that will enable a linker, loader, or linker-loader to assemble it with other object code modules into executable machine code or byte code.
 - a. In the context of Java, object code is often associated with native code generated by a **Just-In-Time** (JIT) compiler or **Ahead-Of-Time** (AOT) compiler.
 - b. Object code refers to the machine-readable binary code generated by a compiler or an assembler from source code.
 - c. Unlike bytecode, which is executed by the JVM, object code is executed directly by the underlying hardware.
 - d. Object code is specific to the target platform and cannot be executed on different platforms without modification.
4. **JVM:** The Java Virtual Machine (JVM) is an abstract computing machine that enables Java bytecode to be executed on any device or Operating System.

It acts as an interpreter for Java bytecode, translating it into machine-specific instructions that can be executed by the underlying hardware.

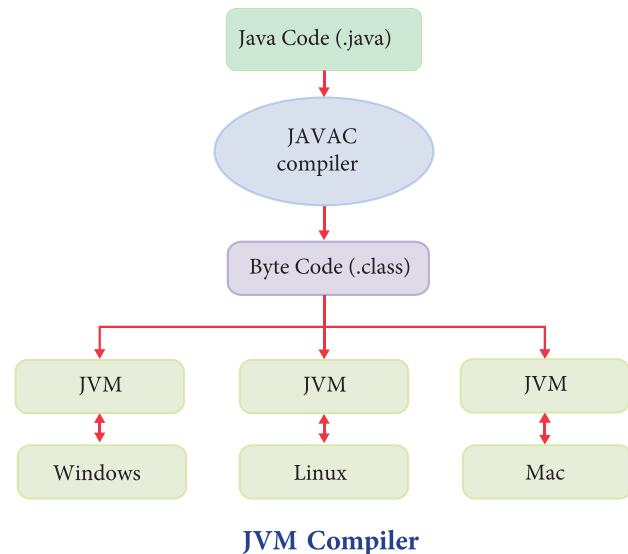
The JVM provides features like memory management, garbage collection, and security to Java programs. It ensures that Java programs run efficiently and securely on a wide range of platforms.

What does JVM performs?

It is used to load the code. It Verifies and executes the code. And it provides run time environment.

The features of JVM are as follows:

1. **Platform Independence:** One of the main features of the JVM is its platform independence. Java programs are compiled into bytecode, which is a platform-neutral intermediate representation of the program. The JVM then interprets this bytecode and executes it on any device or operating system that has a JVM implementation.
2. **Execution Environment:** The JVM provides an execution environment for Java programs, including memory management, garbage collection, and security features. It ensures that Java programs run efficiently and securely on a wide range of platforms.
3. **Just-In-Time(JIT) Compilation:** In addition to interpreting bytecode, many modern JVM implementations also use a technique called Just-In-Time (JIT) compilation. This involves compiling bytecode into native machine code at runtime, which can improve performance by executing optimized native code instead of interpreting bytecode.
4. **Class Loading and Runtime Management:** The JVM is responsible for loading classes into memory as they are needed by the program. It also manages memory allocation and deallocation, including garbage collection to reclaim memory occupied by objects that are no longer in use.
5. **Security:** The JVM includes built-in security features to protect against malicious code. It enforces access control policies, verifies bytecode to ensure it does not violate security constraints, and provides a secure execution environment for Java programs.



EXTRA TIME

The process of converting the source code into machine code is called compilation. The compilation produces the machine code. For different platforms different machine codes are produced.

JAVA API (JAVA APPLICATION PROGRAMMING INTERFACE)

Java API is a library of compiled code (small built-in programs) that can be used in our program in order to lessen our program effort.

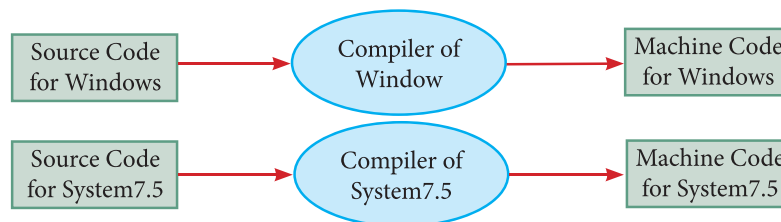
Java Platform

The Java Compiler, JVM combined with Java APIs makes Java platform. JDK (Java Development Kit). It contains Java compiler, Java Virtual Machine (JVM), Java API (libraries) etc.

Java Compilation

Ordinary Compilation Process

The compiler for Windows converts the source program directly to machine code for Windows only. The compiler for System7.5 compiles to machine code only for System7.5. The compiler for a particular system software can convert only for that system software.



Ordinary Compilation Process

Java Compilation Process

Java Compiler compiles the source code to Java Byte Code, then this byte code is interpreted by Java Interpreter into Machine Code for a specific platform i.e., if the operating system is Windows then to the machine code for Windows, if the operating system is System7.5 then to the machine code for System7.5

Diagrammatic Representation of Java Compilation Process



Java Compilation Process

FEATURES OF JAVA

The feature of Java are as follow:

1. **Object Oriented:** In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
2. **Platform Independent:** Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Java Virtual Machine (JVM) on whichever platform it is being run on.
3. **Simple:** Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.
4. **Secure:** With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
5. **Portable:** Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable.
6. **Robust:** Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.

7. **Multithreaded:** With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.
8. **Interpreted:** Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.
9. **High Performance:** With the use of Just-In-Time compilers, Java enables high performance.



Know More

- ★ The class which other classes are derived through the process of inheritance is called base class or superclass.
- ★ The class that inherits properties from a base class is called a derived class.

IMPORTANCE OF JAVA

1. **Java is Platform Independent:** The program written in Java can be run on any platform that means it can be executed by any operating system and any processors; because the source code is compiled by Java Compiler to byte code which is suited to any platform and then interpreted by JVM to native executable code. So, Java is platform independent.
2. **Java is a Programming Language as well as a Platform:** Java can be used to develop various application programs like MS Word, PowerPoint, Excel etc. So, Java is a programming language. Java includes some programs like compiler, interpreter-generally which are parts of a system software. So, Java is a platform also.



Know More

- ★ Java is owned by Oracle, and more than 3 billion devices run Java.

KEY TERMS

- **Class:** In Java, a class is a blueprint or template for creating objects.
- **Objects:** Objects are instances of classes.
- **Polymorphism:** Polymorphism is a concept in Object-Oriented programming that allows Objects of different types to be treated as objects of a common superclass.
- **Encapsulation:** Encapsulation is the bundling of data (attributes) and methods (behaviours) that operate on that data into a single unit, typically a class.
- **JVM (Java Virtual Machine):** JVM is an abstract computing machine that enables Java bytecode to be executed on any device or operating system.

Summary ♦

- ❖ Object oriented programming is an approach which uses data more than functions. This data is called object.
- ❖ Procedure-Oriented programming put more emphasis on functions rather than data.
- ❖ There are 4 principles of Object-Oriented programming.
- ❖ Abstraction refers to the practice of showing only the relevant details and hiding the unimportant details.
- ❖ Encapsulation means binding the data and functions together in one solution.
- ❖ Inheritance is one of the most important characteristics of Object-Oriented languages where one module borrows the features of some other module and hence helps in reusing the same module at multiple places.
- ❖ Polymorphism is derived from two words- poly and morph. Poly means many and morph means forms. It refers to the concept of one thing doing multiple tasks.
- ❖ Java is an Object-Oriented language developed by Sun Microsystems.
- ❖ Source code is the original instructions generated by human programmer.
- ❖ Bytecode is program code that has been compiled from source code into low-level code designed for a software interpreter.
- ❖ Object code is the code output of compiler.
- ❖ A Java virtual machine (JVM) is a virtual machine that enables a computer to run Java programs as well as programs written in other languages and compiled to Java bytecode.

Exercises - 1 (Solved)

A. Multiple choice questions.

[Understanding]

1. In OOP, what is the process of bundling data and methods that operate on the data into a single unit known as?
a. Polymorphism b. Inheritance c. Encapsulation d. Abstraction
2. Which OOP principle allows objects of different classes to be treated as objects of a common superclass?
a. Inheritance b. Encapsulation c. Polymorphism d. Abstraction
3. What is the term for the ability of a method to have multiple forms of implementation?
a. Inheritance b. Encapsulation c. Polymorphism d. Abstraction
4. Name the programming technique that specifies a series of well-structured steps and procedures within its programming context to compose a program.
a. Procedure-Oriented Programming b. Modular Programming
c. Object-Oriented Programming d. None of these
5. Which OOP principle focuses on hiding the internal implementation details of a class from the outside world?
a. Inheritance b. Encapsulation c. Polymorphism d. Abstraction
6. Which OOP principle promotes code reuse by allowing a class to inherit attributes and methods from another class?
a. Inheritance b. Encapsulation c. Polymorphism d. Abstraction

7. Name the programming technique that implements programs as an organized collection of interactive objects.
 - a. Procedure-Oriented Programming
 - b. Modular Programming
 - c. Object-Oriented Programming
 - d. None of these
8. What is the primary benefit of using OOP concepts in software development?
 - a. Faster execution speed
 - b. Easier debugging
 - c. Better code organization and reusability
 - d. Lower memory usage
9. **Assertion:** Inheritance promotes code reusability in Object-Oriented programming.
Reason: Subclasses inherit attributes and behaviours from their superclass.
 - a. Both Assertion and Reason are true, and Reason is the correct explanation of Assertion
 - b. Both Assertion and Reason are true, but Reason is NOT the correct explanation of Assertion
 - c. Assertion is true, but Reason is false
 - d. Assertion is false, but Reason is true
10. **Assertion:** Encapsulation enhances security and reduces coupling between components.
Reason: It allows restricting access to certain components of an object.
 - a. Both Assertion and Reason are true, and Reason is the correct explanation of Assertion
 - b. Both Assertion and Reason are true, but Reason is NOT the correct explanation of Assertion
 - c. Assertion is true, but Reason is false
 - d. Assertion is false, but Reason is true

ANSWERS

1. c. 2. a. 3. c. 4. a. 5. b. 6. a. 7. c. 8. c. 9. a. 10. a.

B. Fill in the blanks.

1. Java is a _____ language.
2. _____ developed Java programming language.
3. Java compiler converts Java source code into an intermediate binary code called _____.
4. The full form of JVM is _____.
5. _____ translates the source program into target program one line at a time.

ANSWERS

1. Case-Sensitive 2. James Gosling 3. Bytecode 4. Java Virtual Machine 5. Interpreter

C. State whether the following statements are True or False.

1. Java is a procedural programming.
2. Wrapping of data and functions together as a single unit is known as encapsulation.
3. Polymorphism comes from the Latin words.
4. Class is the process by which one object can acquire the properties of another.
5. Alan Kay invented OOP.

ANSWERS

1. False 2. True 3. False 4. False 5. True

D. Very short answer type questions.

[Understanding]

1. Expand the following terms:

- a. JDK b. JVM c. IDE.

- Ans.** a. JDK: Java Development Kit
b. JVM: Java Virtual Machine
c. IDE: Integrated Development Environment

2. What was the earlier name of Java?

- Ans.** The earlier name of Java was Oak.

3. Who developed Java?

- Ans.** Java was developed by James Gosling.

4. Define source code.

- Ans.** Source code is a set of instructions and statements written by a programmer using a computer programming language.

5. Define objects. Also, give a real-life example.

- Ans.** An object is a unique entity, which has some characteristics and behaviours. For example, a student can be considered as an object because he/she has few characteristics like name, class, age, etc.

6. Why is Java referred to as platform-independent?

- Ans.** Java is referred to as platform-independent because the Java compiled code (byte code) can run on all operating systems.

E. Short answer type questions.

[Recall]

1. How is Java platform independent?

- Ans:** Java achieves platform independence by compiling source code into bytecode, which can run on any system with a Java Virtual Machine (JVM). The JVM interprets bytecode into machine code specific to the underlying hardware and operating system. Java's standard libraries abstract away platform-specific details, ensuring consistency across different environments. This allows Java programs to be developed and executed seamlessly across various platforms without modification.

2. What is inheritance and how it is useful in Java? Give examples.

- Ans:** It is process by which objects of one class acquire the properties of objects of another class. Inheritance supports the concepts of hierarchical representation. In OOP the concepts of inheritance provides the idea of reusability. e.g. A class car inherits some property from the class Automobiles, which in turn inherits properties from Vehicle class. The capability to pass down properties and so allows us to describe things efficiently.

3. What role does polymorphism play as Java feature?

- Ans:** It means the ability for a message or data to take more than one form. For example, an operation, many types of data used in the operation. E.g. a child behaves like a student in school, customer in a shopping mall, passenger in a bus and like a son/daughter at home. Same person behave differently in various circumstances.

4. What does a class encapsulate?

- Ans:** A class encapsulated the data (instance variables) and methods.

5. Explain the two types of Java programs.

Ans: There are two type of Java programs:

- a. Applications: Java programs that run directly on the machine.
- b. Applets: Java programs that can run over the Internet. The standard client/server model is used when the Applet is executed. The server stores the Java Applet, which is sent to the client machine running the browser, where the Applet is then run.

6. State the Java concept that is implemented through:

- a. super class and a subclass.
- b. the act of representing essential features without including background details.

Ans: a. Inheritance

b. Abstraction

7. Write the differences between Procedural Programming and Object-Oriented Programming.

Ans: Procedural programming aims more at procedures. The emphasis is a doing thing rather than the data being used. In procedural Programming paradigm data are shared among all the functions participating thereby risking data safety and security. Object Oriented Programming is based on principles of data hiding, abstraction, inheritance and polymorphism. It implements programs using classes and objects, In OOP's data and procedure both given equal importance. Data and functions are encapsulated to ensure data safety and security.

8. Class is termed as an object factory. Explain.

Ans: A class is known as an object factory because it contains all the statements needed to create an object, its attributes as well as the statements to describe the operations that the object will be able to perform.

F. Application based questions.

[Analysis/Understanding]

1. Give a real-life example to explain the term encapsulation.

Ans: A capsule is a real-life example of encapsulation. Basically, a capsule encapsulates several combinations of medicine. If combinations of medicine are variables and methods, then the capsule will act as a class and the whole process is called encapsulation.

2. Explain the concept of class and objects by providing a real-life example.

Ans: A class is a group of objects that have some common characteristics and behaviours. A class is a blueprint from which individual objects are created. In other words, a class is a concept, and the object is the implementation of that concept. We need to have a class before creating the objects. Let us consider a car as a class that has characteristics, like steering wheel, seats, brakes, etc., and its behaviour is mobility. However, you can say BMW, having a registered number of 4284, is an 'object' that belongs to the class 'car'.

3. Imagine you are designing a banking application where security and scalability are crucial. Explain how OOP principles like abstraction and interface implementation can contribute to designing a secure and scalable system. Discuss how these principles facilitate code maintenance and future enhancements in the banking application.

Ans: In a banking application, abstraction via OOP isolates sensitive data and operations, reducing exposure and enhancing security. Interface implementation ensures modular, interchangeable components, facilitating scalability and adaptability to evolving demands. These principles streamline

code maintenance by compartmentalizing functionality and dependencies, enabling easier updates without widespread impact. Future enhancements benefit from this structured approach, allowing seamless integration of new features while maintaining robust security and scalability measures.

4. Consider a scenario where you have two classes, Vehicle and Car, where Car inherits from Vehicle. Explain how inheritance in this scenario promotes code reuse and simplifies maintenance. Discuss the relationship between Vehicle and Car, highlighting the concept of inheritance.

Ans: Inheritance allows Car to inherit behaviours and attributes from Vehicle, such as speed and fuel consumption, promoting code reuse by eliminating redundant code for common functionalities. This hierarchical relationship simplifies maintenance by centralizing shared methods and properties in Vehicle, ensuring consistent updates across all derived classes like Car.

Exercises - 2 (Unsolved)

A. Multiple choice questions.

[Recall]

- Which of the following is a Procedure-Oriented programming language?
a. Pascal b. C++ c. Java d. Python
- Name the Object-Oriented language among the following.
a. Pascal b. Fortran c. C++ d. All of these
- What is the most important entity in Object-Oriented programming languages?
a. Objects b. Functions c. Variables d. Expressions
- Name the concept of showing only important details and hiding non-essential details.
a. Abstraction b. Encapsulation c. Inheritance d. Polymorphism
- What is used to run bytecode?
a. JVM b. JPM c. JRE d. JDK
- Name the Java programs that run directly on a machine.
a. Applets b. Applications c. Programs d. Websites
- What are the type of Java programs that run on the Internet?
a. Applets b. Applications c. Programs d. Websites
- Name the characteristic of Java to run on any platform.
a. Robust b. Portability c. Simple d. Multithreaded
- What is the feature of Java to run parallel tasks at the same time called?
a. Robust b. Portability c. Simple d. Multithreaded
- Java was originally known as:
a. OAK b. ROSE c. WALNUT d. All of these

B. Fill in the blanks.

[Understanding]

- _____ was the first purely Object-Oriented programming language.
- OOP first came into picture in _____ by Alan and his team.
- _____ indicates code reusability in OOPs.
- Java is a platform_____.

5. The full form of JIT is _____.
6. Java program comes into _____ main types.
7. The concept of deriving one class from another class is called _____.
8. The Java compiler translates Java source code into Java _____.
9. A _____ is a program that behaves like a computer.
10. _____ is a high-level programming language.

C. State whether the following statements are True or False.

[Understanding]

1. Java is a Procedure-Oriented language.
2. An object is a real-world entity.
3. Pascal is a Procedure-Oriented language.
4. Byte code is the original code written by a programmer.
5. JVM stands for Java Virtual Machine.
6. Object code is the output of a compiler.
7. Java supports multithreading.
8. Java is a platform dependent language.
9. Sun Microsystems developed Java.
10. JVM is Java compiler.

Lab Work

[Analysis]

1. Try to find about the location where a byte code is generated.
2. Compare and discuss the difference between OOP and POP.

Project Work

[Application]

1. Write a program in Java to define a Student class with attributes like name, roll number, and marks in different subjects.
2. Write a Java program that defines a class Car with properties like color, model, and speed. Then create an object from this class.

Scan the QR code for more solved questions.



Teacher's Notes

- Help the students understand the significance of Object-Oriented programming
- Help them understand the different features of Java

As per the latest syllabus
Prescribed by the CISCE

10
ICSE

COMPUTER APPLICATIONS

(Subject Code - 86)

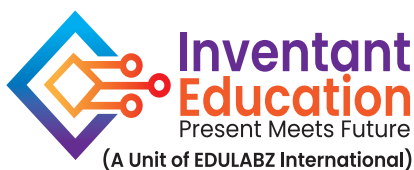
With BlueJ



Palvi Gupta

M.Tech, B.E.
(Computer Science & Engineering)
D.A.V. University, Jalandhar





D-47, Sector 2, Noida, Uttar Pradesh-201301

Email : info@inventanteducation.com

Customer care number: 18002022912

Disclaimer

This educational material, developed by Inventant Education, focuses on STEM education. While we strive for accuracy, we do not guarantee completeness or suitability for all purposes. Inventant Education is not liable for any damages resulting from the material's use or any inadvertent omissions or errors. References to products, services, or organizations are for informational purposes and do not imply endorsement.

First Edition : October, 2024

Price: ₹ 649

Copyright

© Inventant Education, a unit of Edulabz International

All rights reserved. No part of this educational material may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Inventant Education. Permission is granted to educational institutions for classroom use, provided that the material is used for non-commercial, educational purposes and is not sold or distributed for profit. For any other use, please seek written permission from Inventant Education. Inventant Education and the Inventant Education logo are registered trademarks of Inventant Education and/or its subsidiaries. All other trademarks and trade names are the property of their respective owners.

Printed at

SRG Traders Pvt. Ltd., Noida

Preface

The **Class 9th and 10th ICSE Computer Applications book** is designed to introduce students to the exciting and rapidly evolving world of computer science. It serves as a comprehensive guide that lays a solid foundation in programming and other essential computing concepts. The book is structured to align with the **ICSE (Indian Certificate of Secondary Education)** syllabus, ensuring that all relevant topics are covered systematically and in-depth, preparing students for both academic success and practical application.

Key Features of the Book:

1. **Introduction to Basic Computer Concepts:** The book begins with fundamental concepts. This forms a strong foundation for understanding advanced topics.
2. **Object-Oriented Programming (OOP):** The primary focus of the book is on **Java programming**, an object-oriented language. Students are introduced to the core principles of OOP, such as classes, objects, inheritance, and polymorphism, which are critical for modern software development. The book uses simple and clear examples to explain these concepts, making it easier for students to grasp.
3. **Programming Basics:** Students learn the syntax of Java and are guided step-by-step through writing their first programs. They are taught how to use variables, data types, operators, iterative, nested and conditional statements to build simple applications.
4. **Problem-Solving Skills:** The book emphasizes **logical thinking and problem-solving** through coding exercises. It includes a variety of practical programming examples and challenges to encourage students to apply their knowledge in real-world scenarios. This approach not only enhances their programming skills but also improves their analytical thinking.
5. **Java Programming:** One of the standout features of the book is its introduction to Java programming, where students learn how to create interactive applications. They are guided through the basics of designing user interfaces with buttons, text fields, and other controls, providing them with the tools to create functional and user-friendly programs.
6. **Exercises and Projects:** At the end of each chapter, there are exercises that include both theoretical and practical questions to test students' understanding of the material. Additionally, **project work** is assigned to help students consolidate their learning by developing larger, more complex applications. These projects often mimic real-world problems, fostering creativity and deeper learning.
7. **ICSE Examination Preparation:** The book is structured to help students prepare effectively for the **ICSE exams**. It includes model questions, Java Programming, and previous year's exam questions, providing ample practice for students to master the format and style of the exam.
8. **Learning Objectives:** Clearly define what students will learn by the end of each chapter.
9. **Definition:** Provide clear and concise definitions of important terms to help students understand concepts better. It should be simple, concise, and relevant to the topic at hand.
10. **Extra Time:** Suggested activities or extensions for students who finish early or need additional challenges. It ensures that fast learners stay engaged and deepen their knowledge.
11. **Know More:** Provide additional facts or insights to encourage deeper understanding of the topic. This section sparks curiosity and provide context or trivia related to the topic.
12. **Lab Activity:** Hands-on exercises that allow students to apply theoretical concepts in practical scenarios. It enhances understanding by giving students the opportunity to experiment with the software.
13. **Teacher Notes:** Provide teachers with background information, teaching tips, and key points to emphasize during the lesson. It supports educators by offering guidance on how to deliver content effectively.

Teacher's Resource Book: Provides lesson plans and solutions to the textbook questions.

Online Support: Downloadable e-books (for teachers only)

I owe my success in this project to the unwavering support of my family. My husband's encouragement, my son Sidharth's joy, my parents' guidance, and my mother-in-law's understanding have all been crucial in helping me achieve this balance.

Suggestions for the improvement of the book are most welcome.

—Author

About the Series

Learning Objectives

After studying this chapter, you will be able to:

- Understand the key concepts of OOP, including classes, objects, and the four principles of OOP (Encapsulation, Abstraction, Inheritance, and Polymorphism)
- Differentiate between procedural and object-oriented programming approaches
- Explain what a class and an object are in Java
- Understand the different data types in Java (primitive types like int, float, char, boolean, etc., and reference types like objects)

Learning Objectives

Lists key takeaways for each chapter.

Definition

Data Abstraction is the process of defining a data structure by providing a simplified view of its interface, hiding the internal implementation details and complexities. It emphasizes the "what" an object does rather than "how" it does it.

Definition

Provides clear definitions of important terms.



EXTRA TIME

Data abstraction helps in reducing complexity by allowing the use of a higher level without needing to understand the internal workings.

Extra Time

Offers activities for fast learners seeking more challenges.

KEY TERMS

- Class:** A class is a blueprint for creating objects. It defines a data type by bundling data and methods that work on that data into one single unit.
- Object:** An object is an instance of a class. It is a concrete realization of a class that occupies memory and can have attributes and methods.
- Inheritance:** Inheritance is a mechanism where one class (subclass) inherits the properties and methods of another class (superclass).
- Polymorphism:** Polymorphism allows objects of different classes to be treated as objects of a common superclass. It includes method overriding and method overloading.
- Abstraction:** Abstraction is the concept of hiding the implementation details and showing only the functionality to the user. It is achieved using abstract classes and interfaces.

Key Terms

Provides meaning of important terms

Scan the QR code for more solved questions.



QR Codes

Additional solved questions related to each chapter effortlessly through QR codes

Lab Activity

- Design a library management system. Find out the class and its objects.
- Try to list the data members and methods of the identified class.

[Application]

Lab Activity

Hands-on exercises to apply concepts practically

SOLVED PROGRAMMING

1. Write a program to calculate the area and perimeter of the rectangle.

Ans:

```
public class Rectangle {  
    public static void main(String[] args) {
```

Java/Solved Programs

Contains additional programs related to the chapter/topic



Know More

* Java allows you to represent special characters using escape sequences, such as `\n` for newline, `\t` for tab, and `\uXXXX` for Unicode characters, where XXXX is the hexadecimal representation of the Unicode code point.

Know More

Shares extra facts to deepen understanding.

Exercises - 1 (Solved)

A. Multiple choice questions.

[Understanding]

1. What is the main principle of OOP that involves hiding internal state and requiring all interaction to be performed through an object's methods?
a. Inheritance b. Polymorphism c. Encapsulation d. Abstraction
2. Which OOP concept allows one class to inherit the properties and behaviours of another class?
a. Polymorphism b. Encapsulation c. Inheritance d. Abstraction

Exercises

Provides solved and unsolved questions of the chapter



Teacher's Notes

- Encourage students to write Java code to solve each exercise and run their programs to see the output.
- Encourage students to continue practicing loops on their own and explore more complex loop patterns and applications.

Teacher's Notes

Offers background info and teaching tips for educators

SAMPLE PROJECT

1. Write a program in Java to calculate the total grocery bill for a customer based on the quantity of items purchased and their prices.

Options
Enter Customer Name:
PARTIAL Quota

Sample Project

Contains the real-life applications to use the concepts learnt

Activity

Create a class with one integer instance variable. Initialize the variable using:
a. Default constructor
b. Parameterized constructor

Activity

Interactive activities to enhance your learning experience

SYLLABUS

CLASS X

There will be one written paper of two hours duration carrying 100 marks and Internal Assessment of 100 marks.

THEORY – 100 Marks

1. Revision of Class IX Syllabus

(i) Introduction to Object Oriented Programming concepts, (ii) Elementary Concept of Objects and Classes, (iii) Values and Data types, (iv) Operators in Java, (v) Input in Java, (vi) Mathematical Library Methods, (vii) Conditional constructs in Java, (viii) Iterative constructs in Java, (ix) Nested for loops.

2. Class as the Basis of all Computation

Objects and Classes

Objects encapsulate state and behaviour – numerous examples; member variables; attributes or features. Variables define state; member methods; Operations/methods/messages/ methods define behaviour.

Classes as abstractions for sets of objects; class as an object factory; primitive data types, composite data types. Variable declarations for both types; difference between the two types. Objects as instances of a class.

Consider real life examples for explaining the concept of class and object.

3. User - defined Methods

Need of methods, syntax of methods, forms of methods, method definition, method calling, method overloading, declaration of methods,

Ways to define a method, ways to invoke the methods – call by value [with programs] and call by reference [only definition with an example], Object creation - invoking the methods with respect to use of multiple methods with different names to implement modular programming, using data members and member methods, Actual parameters and formal parameters, Declaration of methods - static and non-static, method prototype / signature, - Pure and impure methods, - pass by value [with programs] and pass by reference [only definition with an example], Returning values from the methods , use of multiple methods and more than one method with the same name (polymorphism - method overloading).

4. Constructors

Definition of Constructor, characteristics, types of constructors, use of constructors, constructor overloading.

Default constructor, parameterized constructor, constructor overloading., Difference between constructor and method. of Constructor, characteristics, types of constructors, use of constructors, constructor overloading.

Default constructor, parameterized constructor, constructor overloading., Difference between constructor and method.

5. Library classes

Introduction to wrapper classes, methods of wrapper class and their usage with respect to numeric and character data types. Autoboxing and Unboxing in wrapper classes.

Class as a composite type, distinction between primitive data type and composite data type or class types. Class may be considered as a new data type created by the user, that has its own functionality. The distinction between primitive and composite types should be discussed through examples. Show how classes allow user defined types in programs. All primitive types have corresponding class wrappers. Introduce Autoboxing and Unboxing with their definition and simple examples.

The following methods are to be covered:

int parseInt(String s),

float parseFloat(String s),

boolean isDigit(char ch),

boolean isLetterOrDigit(char ch),

long parseLong(String s),

double parseDouble(String s),

boolean isLetter(char ch),

boolean isLowerCase(char ch),

boolean isUpperCase(char ch),	boolean isWhitespace(char ch),
char toLowerCase (char ch)	char toUpperCase(char ch)

6. Encapsulation

Access specifiers and its scope and visibility.

Access specifiers – private, protected and public. Visibility rules for private, protected and public access specifiers. Scope of variables, class variables, instance variables, argument variables, local variables.

7. Arrays

Definition of an array, types of arrays, declaration, initialization and accepting data of single and double dimensional arrays, accessing the elements of single dimensional and double dimensional arrays.

Arrays and their uses, sorting techniques - selection sort and bubble sort; Search techniques – linear search and binary search, Array as a composite type, length statement to find the size of the array (sorting and searching techniques using single dimensional array only).

Declaration, initialization, accepting data in a double dimensional array, sum of the elements in row, column and diagonal elements [right and left], display the elements of two-dimensional array in a matrix format.

8. String handling

String class, methods of String class, implementation of String class methods, String array

The following String class methods are to be covered:

String trim ()	String toLowerCase()
String toUpperCase()	int length()
char charAt (int n)	int indexOf(char ch)
int lastIndexOf(char ch)	String concat(String str)
boolean equals (String str)	boolean equalsIgnoreCase(String str)
int compareTo(String str)	int compareToIgnoreCase(String str)
String replace (char oldChar,char newChar)	String substring (int beginIndex)
String substring (int beginIndex, int endIndex)	boolean startsWith(String str)
boolean endsWith(String str)	String valueOf(all types)

Programs based on the above methods, extracting and modifying characters of a string, alphabetical order of the strings in an array [Bubble and Selection sort techniques], searching for a string using linear search technique.

INTERNAL ASSESSMENT - 100 Marks

This segment of the syllabus is totally practical oriented. The accent is on acquiring basic programming skills quickly and efficiently.

Programming Assignments (Class X)

The students should complete a minimum of 20 laboratory assignments during the whole year to reinforce the concepts studied in class.

Suggested list of Assignments:

The laboratory assignments will form the bulk of the course. Good assignments should have problems which require design, implementation and testing. They should also embody one or more concepts that have been discussed in the theory class. A significant proportion of the time has to be spent in the laboratory. Computing can only be learnt by doing.

The teacher-in-charge should maintain a record of all the assignments done by the student throughout the year and give it due credit at the time of cumulative evaluation at the end of the year.

Some sample problems are given below as examples. The problems are of varying levels of difficulty:

(i) User defined methods

- (a) Programs depicting the concept of pure, impure, static, non- static methods.

- (b) Programs based on overloaded methods.
- (c) Programs involving data members, member methods invoking the methods with respect to the object created.

(ii) Constructors

- (a) Programs based on different types of constructors mentioned in the scope of the syllabus.
- (b) Programs / outputs based on constructor overloading

(iii) Library classes

- (a) Outputs based on all the methods mentioned in the scope of the syllabus.
- (b) Programs to check whether a given character is an uppercase/ lowercase / digit etc.

(iv) Encapsulation

Questions based on identifying the different variables like local, instance, arguments, private, public, class variable etc.

(v) Arrays

- (a) Programs based on accessing the elements of an array.
- (b) Programs based on sort techniques mentioned in the scope of the syllabus.
- (c) Programs based on search techniques mentioned in the scope of the syllabus.
- (d) Programs on Double dimensional arrays as given in the scope of the syllabus.

(vi) String handling

- (a) Outputs based on all the string methods mentioned in the scope of the syllabus.
- (b) Programs based on extracting the characters from a given string and manipulating the same.
- (c) Palindrome string, pig Latin, alphabetical order of characters, etc.

Important: This list is indicative only. Teachers and students should use their imagination to create innovative and original assignments.

INTERNAL ASSESSMENT - 100 Marks

EVALUATION

The teacher-in-charge shall evaluate all the assignments done by the student throughout the year [both written and practical work]. He/she shall ensure that most of the components of the syllabus have been used appropriately in the assignments. Assignments should be with appropriate list of variables and comment statements. The student has to mention the output of the programs.

Proposed Guidelines for Marking

The teacher should use the criteria below to judge the internal work done. Basically, four criteria are being suggested: class design, coding and documentation, variable description and execution or output. The actual grading will be done by the teacher based on his/her judgment. However, one possible way: divide the outcome for each criterion into one of 4 groups: excellent, good, fair/acceptable, poor/unacceptable, then use numeric values for each grade and add to get the total.

Class design:

- Has a suitable class (or classes) been used?
- Are all attributes with the right kinds of types present?
- Is encapsulation properly done?
- Is the interface properly designed

Coding and documentation:

Is the coding done properly? (Choice of names, no unconditional jumps, proper organization of conditions, proper choice of loops, error handling, code layout) Is the documentation complete and readable? (class documentation, variable documentation, method documentation, constraints, known bugs - if any).

Variable description:

Format for variable description:

Name of the Variable	Data Type	Purpose/description

Execution or Output:

Does the program run on all sample input correctly?

Evaluation of practical work will be done as follows:

Subject Teacher (Internal Examiner)			50 marks	
External Examiner			50 marks	
Criteria (Total-50 marks)	Class design (10 marks)	Variable description (10 marks)	Coding and Documentation (10 marks)	Execution OR Output (20 marks)
Excellent	10	10	10	20
Good	8	8	8	16
Fair	6	6	6	12
Poor	4	4	4	8

An External Examiner shall be nominated by the Head of the School and may be a teacher from the faculty, but not teaching the subject in the relevant section/class. For example, A teacher of Computer Science of class VIII may be deputed to be the External Examiner for class X.

The total marks obtained out of 100 are to be sent to CISCE by the Head of the school.

The Head of the school will be responsible for the online entry of marks on CISCE's CAREERS portal by the due date.

EQUIPMENT

There should be enough computer systems to provide for a teaching schedule where at least three-fourth of a time available is used for programming and assignments/practical work. The course shall require at least 4 periods of about 40 minutes duration per week. In one week, out of 4 periods the time should be divided as follows:

- 2 periods – Lecture cum demonstration by the Instructor.
- 2 periods – Assignments/Practical work.

The hardware and software platforms should be such that students can comfortably develop and run programs on those machines.

Since hardware and software evolve and change very rapidly the schools shall need to upgrade them as required. Following are the minimal specifications as of now.

RECOMMENDED FACILITIES:

- A lecture cum demonstration room with a MULTIMEDIA PROJECTOR/ an LCD and Overhead Projector (OHP) attached to the computer.
- A white board with white board markers should be available.
- A fully equipped Computer Laboratory that allows one computer per student.
- The computers should have a minimum of 1 GB RAM and at least a P - IV or Equivalent Processor.
- Good Quality printers.
- A scanner, a web cam/a digital camera (Should be provided if possible).

SOFTWARE FOR CLASSES IX & X

Any suitable Operating System can be used.

For teaching fundamental concepts of computing using object oriented approach, Blue J environment (3.2 or higher version) compatible with JDK (5.0 or higher version) as the base or any other editor or IDE, compatible with JDK (5.0 or higher version) as the base may be used. Ensure that the latest versions of software are used.

Aligned with NEP 2020 and NCF 2023

FEATURES OF NEP 2020

21st Century Skills

Learning Skills (4Cs)

- ✓ Critical Thinking
- ✓ Creativity
- ✓ Communication
- ✓ Collaboration

Literacy Skills (IMT)

- ✓ Information Literacy
- ✓ Media Literacy
- ✓ Technology Literacy

Life Skills (FLIPS)

- ✓ Flexibility
- ✓ Leadership & Responsibility
- ✓ Initiative
- ✓ Productivity & Accountability
- ✓ Social Interaction

BASED ON NCF 2023

In NCF 2023, **curriculum** means not only what is given in the books, but also how the learners learn in school, the school's environment, and more. To make learning better, we need positive changes in all these areas.

The Six Pramanas

Inference

Perception

Comparison

Verbal Testimony

Non-Apprehension

Postulation

How to Access Digital Content through QR Code

For Website Users

- ✓ "Visit "digital.inventanteducation.com"
- ✓ Click "Register" button available on the top-right.
- ✓ Select 'Teacher/Student' in 'User' Type.
- ✓ Enter your name, email, mobile number and password.
- ✓ Click 'Register', and Enter the OTP to verify your mobile/email.
- ✓ Once registered, login on to the website and go to **Scan and Learn** section. Enter the Codes printed below the QR Codes to view the required content.

For Mobile Users

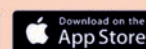
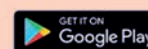
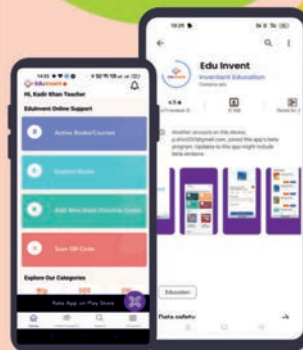
- ✓ Go to Google Play Store or Apple App Store.
- ✓ Type 'Edu Invent' in the search bar.
- ✓ Tap 'Install'. The app will take a few moments to download and install.
- ✓ Once installed, tap 'Open' to launch the app.
- ✓ Register yourself and login on the app.
- ✓ On the dashboard, click Scan QR Code button.
- ✓ Scan a QR Code printed in the book to explore the learning content associated with the QR Code.

Download Edu Invent App

Download our mobile app to avail free online support with the books published by Inventant Education.



Scan this QR code to download the app



Contents

1. Revision of class IX 13

- Introduction to OOPs
- Elementary concepts of objects and classes
- Real-World Vs. software classes and objects
- Values and data type
- Encoding of characters
- Escape sequence
- Tokens
- Errors
- Input in Java
- Mathematical Library Method
- Conditional statements in Java
- Unusual termination of a program(System.exit(0))
- Iterative construct in Java
- Break statement in Java
- Continue statement
- Entry and exit controlled loop
- Nested loop in Java

2. Class as the basic of all computation 65

- Introduction
- Object
- Class
- Difference between class and object
- Creating objects of a class
- Different components of a class
- Nested class
- Class as an object factory
- Using this keyword

3. User Defined Method..... 106

- Introduction
- Method
- Actual and formal parameters
- Invoke a method
- Static and non-static methods
- Pure and impure methods
- Method overloading

4. Constructor.....141

- Introduction
- Constructor
- Call a constructor
- Characteristics of a constructor
- Different types of constructors
- Constructor overloading
- Difference between constructor and method
- Similarity between constructor and method

5. Library Classes 172

- Introduction
- Need of wrapper classes
- Primitive and composite data types in Java
- How to use wrapper class?
- Methods used in wrapper classes and their usages
- Autoboxing and unboxing
- Parse and character functions
- Conversion from characters to ASCII values
- Conversion from ASCII values to characters

6. Encapsulation and Inheritance 196


- Introduction
- Encapsulation
- Access specifier
- Scope of variables
- Scope of variable related to block
- Inheritance

7. Arrays 219

- Introduction
- Array real-life examples
- Array
- Sorting
- Searching

8. String Handling 265

- Introduction
- String class in Java
- Methods of the string class
- String array in Java
- Graphical representation of a string array

 Practical exercise	298
 Project	343
 Viva questions	346



Revision of class IX

Learning Objectives



After studying this chapter, you will be able to:

- Understand the key concepts of OOP, including classes, objects, and the four principles of OOP (Encapsulation, Abstraction, Inheritance, and Polymorphism)
- Differentiate between procedural and object-oriented programming approaches
- Explain what a class and an object are in Java
- Understand the different data types in Java (primitive types like int, float, char, boolean, etc., and reference types like objects)
- Identify the difference between primitive and reference data types
- Perform type conversions between different data types
- Recognize the different types of operators in Java (arithmetic, relational, logical, bitwise, assignment, etc.)
- Understand the precedence and associativity of operators in Java expressions
- Understand how to take user input in Java using the Scanner class
- Differentiate between various input methods in Java, such as Scanner, BufferedReader, and command-line arguments
- Use common mathematical methods (like Math.sqrt(), Math.pow(), Math.abs(), Math.max(), etc.) in Java programs
- Understand the syntax and use of iterative constructs (for, while, do-while) for repeated execution of code
- Apply loops and conditions together to create efficient Java programs
- Understand the structure and working of nested for loops in Java

INTRODUCTION TO OOPS

As the name suggests, Object-Oriented Programming or Java OOPs concept refers to languages that use objects in programming, they use objects as a primary source to implement what is to happen in the code. Objects are seen by the viewer or user, performing tasks you assign.

Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc. in programming. The main aim of OOPs is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

Benefits of OOPs

The key benefits of OOP are as follows:

1. **Code Reusability:** Enables developers to reuse existing code by inheriting from existing classes. This reduces redundancy and accelerates development by leveraging pre-existing, tested code.
2. **Modularity:** Encourages the organization of code into distinct, self-contained objects or modules. This separation improves code readability, maintainability, and debugging by isolating functionality.
3. **Improved Maintainability:** The modular nature of OOP makes it easier to update or fix parts of the system without affecting other parts. This leads to more maintainable and scalable code.

4. **Enhanced Productivity:** By reusing existing components and working with well-defined abstractions, developers can work more efficiently and reduce development time.
5. **Better Data Management:** Encapsulation ensures that data is managed and protected within objects, leading to better control over data access and modification.
6. **Design Flexibility:** OOP facilitates the use of design patterns and principles, which provide flexible solutions to common design problems. This results in adaptable systems that can evolve with changing requirements.
7. **Scalability:** The modular and reusable nature of OOP makes it easier to scale applications as new features or functionalities can be added without overhauling the existing system.
8. **Enhanced Collaboration:** OOP's modular approach allows multiple developers to work on different parts of the system simultaneously, improving team collaboration and efficiency.

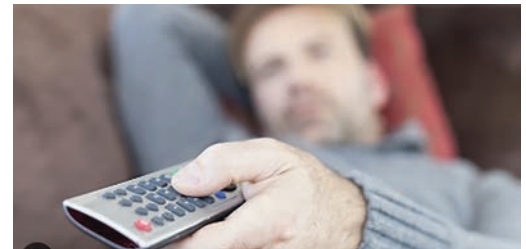
Principles of OOPs

The principles of Object-Oriented Programming (OOP) using everyday examples:

1. **Encapsulation:** Encapsulation is like putting things inside a box with a lock. The box keeps everything inside safe and secure, and you only have a few keys (methods) to access what's inside.

Example: Imagine a television remote control. The remote has buttons (methods) for turning the TV on and off, changing channels, and adjusting volume. You don't need to know how the remote works internally (how the buttons send signals to the TV). You just use the remote to interact with the TV.

```
public class RemoteControl {
    private boolean power; // State (encapsulated)
    // Method to turn TV on or off
    public void togglePower() {
        power = !power;
        System.out.println("TV is " + (power ? "On" : "Off"));
    }
}
```



2. **Abstraction:** Abstraction is like using a smartphone. You interact with apps through a simple interface without needing to understand the complex technology behind them.

Example: Think of a car's dashboard.

The dashboard shows you essential information like speed and fuel level. You don't need to know the complex mechanics of the engine to drive the car; you just use the dashboard to get what you need.

```
abstract class Vehicle {
    abstract void start(); // Abstract method (no body)
}
class Car extends Vehicle {
    void start() {
        System.out.println("Car engine starts with a key.");
    }
}
```



Definition

Data Abstraction is the process of defining a data structure by providing a simplified view of its interface, hiding the internal implementation details and complexities. It emphasizes the "what" an object does rather than "how" it does it.



EXTRA TIME

Data abstraction helps in reducing complexity by allowing the user to interact with the system at a higher level without needing to understand the internal workings.

3. **Data Hiding:** It is a concept in Object-Oriented Programming (OOP) that refers to restricting access to the internal state or implementation details of an object. It ensures that the object's data is protected from unauthorized access and modification, promoting encapsulation and increasing the robustness of the code.

Definition

Data Hiding is the practice of making the internal details of an object private or inaccessible to the outside world. This is achieved by defining access modifiers for the class members (fields and methods) to control visibility and access.

Example: Imagine you have a secret diary that contains your personal thoughts and feelings. To keep your diary private, you use a lock and key. Only you, with the key, can open the diary and read or write inside it.

- Diary (Class):** The diary represents an object, and the information inside (your thoughts) is the data.
- Lock and Key (Data Hiding):** The lock and key represent data hiding. They prevent anyone else from accessing the diary without your permission.

Example:

```
public class SecretDiary {  
    // Private data (just like your personal diary content)  
    private String diaryEntry;  
    // Constructor to initialize the diary entry  
    public SecretDiary(String entry) {  
        this.diaryEntry = entry;  
    }  
    // Public method to read the diary entry  
    public String readDiary() {  
        return diaryEntry;  
    }  
    // Public method to update the diary entry  
    public void updateDiary(String newEntry) {  
        this.diaryEntry = newEntry;  
    }  
}
```

Explanation:

- Private Data (diaryEntry):** Just like the secret content in your diary, the diaryEntry is private and cannot be accessed directly from outside the SecretDiary class.
 - Public Methods (readDiary, updateDiary):** These methods are like you deciding who can read or update your diary. They control how the data is accessed and modified, ensuring that others can only interact with the data in ways you allow.
4. **Inheritance:** Inheritance is like a child inheriting traits from their parents. If a parent has a trait, the child automatically gets that trait.

- a. **Base Class:** A base class (also known as a parent class or superclass) is a class that provides common attributes and methods that other classes can inherit. It serves as the foundation for creating other classes.

The base class encapsulates common functionality that can be shared among multiple derived classes, promoting code reuse and reducing redundancy.

- b. **Derived Class:** A derived class (also known as a child class or subclass) is a class that inherits attributes and methods from a base class. It extends or modifies the functionality of the base class.

The derived class builds on top of the base class, allowing for specialization and additional features while reusing the base class's existing functionality.

Example: Imagine you have a base class called Animal. You have specific animals like Dog and Cat that inherit traits from Animal (like eating and sleeping) but also have their unique traits (like barking or meowing).

```
// Base Class
class Animal {
    // Base class attributes
    String name;
    // Constructor
    public Animal(String name) {
        this.name = name;
    }
    // Base class method
    void eat() {
        System.out.println(name + " is eating.");
    }
}

// Derived Class
class Dog extends Animal {
    // Additional attribute specific to Dog
    String breed;
    // Constructor
    public Dog(String name, String breed) {
        super(name); // Call the constructor of the base class
        this.breed = breed;
    }
    // Derived class method
    void bark() {
        System.out.println(name + " is barking.");
    }
}

// Main class to test the above classes
public class Main {
    public static void main(String[] args) {
        // Create an object of the derived class
        Dog myDog = new Dog("Buddy", "Golden Retriever");
        // Call methods from both base and derived classes
        myDog.eat(); // Inherited from Animal
        myDog.bark(); // Defined in Dog
    }
}
```

Explanation:

a. Base Class (Animal):

Has a common attribute name and a method eat().

Provides a common interface for all animals.

b. Derived Class (Dog):

Inherits the name attribute and eat() method from the Animal class.

Adds its own attribute breed and method bark().

Uses super(name) to initialize the base class attribute.

Benefits of base and derived classes

- i. **Code Reusability:** Derived classes reuse code from the base class, reducing duplication.
- ii. **Extensibility:** New functionality can be added to derived classes without modifying the base class.
- iii. **Maintainability:** Changes in the base class automatically propagate to derived classes, making maintenance easier.

Example:

```
// Base Class
class Animal {
    void eat() {
        System.out.println("This animal eats food.");
    }
}

// Derived Class (Single Inheritance)
class Dog extends Animal {
    void bark() {
        System.out.println("The dog barks.");
    }
}

// Further Derived Class (Multilevel Inheritance)
class Bulldog extends Dog {
    void bulldogSpecificBehaviour() {
        System.out.println("The bulldog has specific behaviours.");
    }
}

public class Main {
    public static void main(String[] args) {
        Bulldog myBulldog = new Bulldog();
        // Accessing methods from the base class
        myBulldog.eat(); // Output: This animal eats food.
        // Accessing methods from the immediate base class
        myBulldog.bark(); // Output: The dog barks.
        // Accessing methods from the derived class
        myBulldog.bulldogSpecificBehaviour(); // Output: The bulldog has specific
        behaviours.
    }
}
```

Advantages of Inheritance

The advantages of inheritance are as follows:

- i. **Code Reusability:** Inheritance allows you to reuse existing code, reducing redundancy and improving maintainability.
- ii. **Method Overriding:** Derived classes can provide specific implementations of methods already defined in the base class.
- iii. **Hierarchical Organization:** It helps in organizing classes into a hierarchy, making the code more intuitive and manageable.



Know More

- ★ Inheritance in OOP allows you to create new classes that reuse, extend, or modify the behaviour defined in other classes. It promotes code reuse, hierarchical organization, and flexibility in managing and extending code.

5. **Polymorphism:** Polymorphism is like a Swiss Army knife that can be used in many ways. It allows the same tool (method) to perform different tasks depending on how you use it.

Example:

Imagine you have a device called a printer that can perform different tasks. The printer can be of different types, such as:

- i. Inkjet Printer
- ii. Laser Printer

Even though these printers are different, they all have a common function: `print()`. Polymorphism allows you to use the `print()` function in a way that is appropriate for each type of printer.

Example

```
// Base class
class Printer {
    // Common method
    void print() {
        System.out.println("Printing...");
    }
}

// Derived class: InkjetPrinter
class InkjetPrinter extends Printer {
    // Overridden method
    @Override
    void print() {
        System.out.println("Printing in colour with Inkjet Printer...");
    }
}

// Derived class: LaserPrinter
class LaserPrinter extends Printer {
    // Overridden method
    @Override
    void print() {
        System.out.println("Printing quickly with Laser Printer...");
    }
}
```

```

    }
}
// Main class to test polymorphism
public class Main {
    public static void main(String[] args) {
        // Create objects of different types
        Printer myPrinter1 = new InkjetPrinter();
        Printer myPrinter2 = new LaserPrinter();
        // Call the print method on both objects
        myPrinter1.print(); // Output: Printing in colour with Inkjet Printer...
        myPrinter2.print(); // Output: Printing quickly with Laser Printer...
    }
}

```

Explanation:

Base Class (Printer): Has a general method print() that performs a common task.

Derived Classes (InkjetPrinter and LaserPrinter):

Each derived class overrides the print() method to provide a specific implementation suited to its type.

InkjetPrinter provides a detailed print statement for colour printing.

LaserPrinter provides a detailed print statement for fast printing.

Polymorphism in Action: When you call print() on an InkjetPrinter object, it uses the print() method defined in the InkjetPrinter class.

When you call print() on a LaserPrinter object, it uses the print() method defined in the LaserPrinter class.

Despite both objects being referred to by the Printer type, each object uses its specific version of the print() method.

ELEMENTARY CONCEPTS OF OBJECTS AND CLASSES

Class: A class in programming is like a template or a blueprint used to create objects. Think of it as a recipe for making something. Just as a recipe tells you what ingredients you need and how to mix them to make a dish, a class defines what properties (attributes) and actions (methods) an object will have.

Example:

Imagine you want to create various types of sandwiches.

The recipe is a detailed guide on how to make a sandwich. It tells you what ingredients to use (like bread, cheese, and ham) and how to assemble them (like spreading butter on the bread and layering the ingredients).

The class is like this recipe. It outlines the properties (what ingredients or attributes are needed) and methods (how to prepare the sandwich or actions the object can perform).

Creating an object of a class in Java

In Java, creating an object of a class involves the following steps:

1. **Define the Class:** Create a class that serves as a template for objects. The class includes attributes (fields) and methods (functions).
2. **Create an Object:** Use the new keyword followed by the class constructor to create an instance of the class.

3. **Use the Object:** Access the attributes and methods of the object using the dot (.) operator.

Example

Let's walk through a complete example.

Define the Class:

```
// Define the class
public class Car {
    // Attributes
    String colour;
    String model;
    int year;
    // Constructor
    Car(String colour, String model, int year) {
        this.colour = colour;
        this.model = model;
        this.year = year;
    }
    // Method to display car details
    void displayInfo() {
        System.out.println("Car Model: " + model);
        System.out.println("Car Colour: " + colour);
        System.out.println("Car Year: " + year);
    }
}

Create an Object:
public class Main {
    public static void main(String[] args) {
        // Create an object of the Car class
        Car myCar = new Car("Red", "Toyota Corolla", 2020);
        // Use the object's method
        myCar.displayInfo();
    }
}
```

Explanation

Class Definition:

Attributes: colour, model, and year are attributes of the Car class.

Constructor: Car(String colour, String model, int year) initializes the attributes with the values provided when creating the object.

Method: displayInfo() prints the car's details.

Creating an Object:

Car myCar = new Car("Red", "Toyota Corolla", 2020); creates an object of the Car class. The new keyword allocates memory for the object, and the constructor initializes it with the provided values.

Using the Object: myCar.displayInfo(); calls the displayInfo() method on the myCar object, which prints the car's details.

Example: Write a program in Java to calculate the area of a circle.

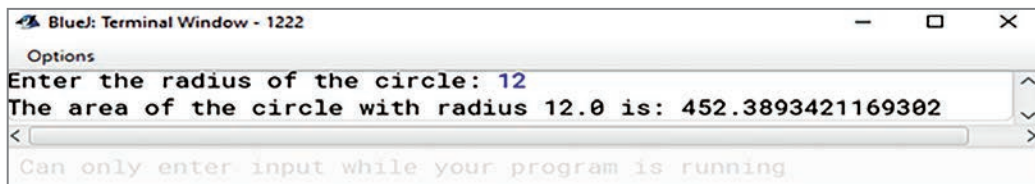
```
import java.util.Scanner;
public class CircleArea {
    // Method to calculate the area of a circle
```

```

    public static double calculateArea(double radius) {
        return Math.PI * radius * radius;
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // Prompt the user to enter the radius
        System.out.print("Enter the radius of the circle: ");
        double radius = scanner.nextDouble();
        // Calculate the area
        double area = calculateArea(radius);
        // Display the result
        System.out.println("The area of the circle with radius " + radius + " is: " + area);
        // Close the scanner
        scanner.close();
    }
}

```

Output



Features of class

1. **Class as an Object Factory:** A class serves as a blueprint or factory for creating objects. It defines the structure and behaviour that the objects (instances) will have.
Example: In Java, new Car() creates an instance of the Car class using its blueprint.
2. **Object as an Instance of a Class:** An object is a concrete instance of a class. While the class defines the structure and behaviour, an object represents a specific implementation of that class, holding actual values for its fields.

Example: Car myCar = new Car("Toyota", 2020); creates an object myCar of type Car.

3. **Class as a User-Defined Data Type:** A class allows you to define a new data type that can include fields (attributes) and methods (functions). This custom data type can be used to model real-world entities or concepts in your program.

Example: class Person { String name; int age; } defines a new data type Person with name and age fields.

// Define a class named Person (User-Defined Data Type)

```

public class Person {
    // Fields (Attributes)
    String name;
    int age;
    // Constructor
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    // Method to display person information
    public void displayInfo() {
        System.out.println("Name: " + name);
    }
}

```



```

        System.out.println("Age: " + age);
    }
    // Main method to test the Person class
    public static void main(String[] args) {
        // Create instances of Person (Objects)
        Person person1 = new Person("Alice", 30);
        Person person2 = new Person("Bob", 25);
        // Display information of each person
        person1.displayInfo(); // Output: Name: Alice, Age: 30
        person2.displayInfo(); // Output: Name: Bob, Age: 25
    }
}

```

Objects

In programming, especially in Object-Oriented Programming (OOP), an object is an instance of a class. It encapsulates both data (attributes) and behaviours (methods) that are defined in the class. Objects are fundamental to understanding how OOP works, as they represent real-world entities or abstract concepts within a program.

Characteristics of an Object

The characteristics of an object are as follows:

- i. **State:** Defined by the attributes of the object. These are variables that hold data specific to the object.
- ii. **Behaviour:** Defined by the methods of the object. These are functions or operations that the object can perform or that can be performed on it.
- iii. **Identity:** Each object has a unique identity, which differentiates it from other objects, even if they have the same state and behaviour.

Example

Consider a Car class. Here's how you define and use objects in Java:

```

// Define a class named Car
public class Car {
    // Fields (Attributes)
    String make;
    String model;
    int year;
    // Constructor
    public Car(String make, String model, int year) {
        this.make = make;
        this.model = model;
        this.year = year;
    }
    // Method to display car information
    public void displayInfo() {
        System.out.println("Make: " + make);
        System.out.println("Model: " + model);
        System.out.println("Year: " + year);
    }
    // Main method to create and use Car objects
    public static void main(String[] args) {
        // Create instances (objects) of Car

```

```

        Car car1 = new Car("Toyota", "Camry", 2022);
        Car car2 = new Car("Honda", "Civic", 2021);
        // Use methods on objects
        car1.displayInfo();
        car2.displayInfo();
    }
}

```

Explanation:

Objects created: car1 and car2 are objects of the Car class.

State: Each object has its own make, model, and year.

Behaviour: The displayInfo method operates on each Car object to display its details.

REAL-WORLD VS. SOFTWARE CLASSES AND OBJECTS

1. **Real-World Classes and Objects:** In the real world, a class is a blueprint or prototype for creating objects (instances). Here's how the concept translates to real-world examples:

Class: Think of a class as a general concept or category. For example, "Vehicle" is a class. It represents a category that includes various types of vehicles.

Object: An object is an actual instance of a class. For example, "Toyota Camry" is a specific instance of the "Vehicle" class. Each Toyota Camry has specific attributes and can perform certain actions.

Real-World Example:

Class: Vehicle

Attributes: Type, Colour, Brand

Behaviour: Drive, Stop

Object: Toyota Camry

Attributes: Type: Sedan, Colour: Blue, Brand: Toyota

Behaviour: Drive, Stop

2. **Software Classes and Objects:** In software development, the concepts of classes and objects are similar but applied in the context of programming:

Class: A class in software is a blueprint that defines attributes (fields) and behaviours (methods) for objects. It's a template from which objects are created.

Object: An object in software is an instance of a class. It encapsulates data and methods defined in the class.

Software Example:

Class: Vehicle

Attributes: type, colour, brand

Methods: drive(), stop()

Object: toyotaCamry

Attributes: type: Sedan, colour: Blue, brand: Toyota

Methods: drive(), stop()

Comparison between real-world example and software example

Aspect	Real-World Example	Software Example
Class	Blueprint of a general category (e.g., Vehicle)	Template for creating objects (e.g., Vehicle class)
Object	A specific instance of a class (e.g., Toyota Camry)	An instance of a class with specific attributes (e.g., toyotaCamry)
Attributes	Characteristics or properties (e.g., Colour, Brand)	Fields or properties in a class (e.g., colour, brand)
Behaviour	Actions or functionalities (e.g., Drive, Stop)	Methods in a class (e.g., drive(), stop())
Instantiation	Physical entity (e.g., a car)	Creation of an object in memory (e.g., new Vehicle())

VALUES AND DATA TYPE

Have you ever wondered what the world would be like if people and things had no names? Think about it, every time you refer to a person or thing, you would have to describe their specific physical appearance because they have no name you can identify them with. Moreover, how do you think contacts on your phone would appear if your contacts had no name in the first place? Strange, right?

Naming is as vital in programming languages as it is in our everyday life, and that's where identifiers in Java have a role to play. Just like naming people is a way to identify them, Java identifiers allow the programmer to refer to different items in a program.

Character Set

In Java, a character set is essentially a collection of characters that can be used in programming. This includes letters, digits, operators, and delimiters. Each of these components plays a crucial role in writing and structuring Java code.

1. **Letters:** Java uses letters from the alphabet for variable names, method names, class names, and more. Both uppercase and lowercase letters are used.

Uppercase Letters: A, B, C, ..., Z

Lowercase Letters: a, b, c, ..., z

Examples:

```
int age; // 'a' and 'g' are letters
```

```
String name; // 'n', 'a', 'm', and 'e' are letters
```

2. **Digits:** Digits are used to represent numbers in Java. They are crucial for defining integer and floating-point values.

Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Examples:

```
int number = 123; // '1', '2', '3' are digits
```

```
double price = 19.99; // '1', '9', '9', '9' are digits
```

3. **Operators:** Operators are symbols used to perform operations on variables and values. Java includes several types of operators:

- a. Arithmetic Operators: +, -, *, /, %
- b. Relational Operators: ==, !=, >, <, >=, <=
- c. Logical Operators: &&, ||, !
- d. Assignment Operators: =, +=, -=, *=, /=, %=

Examples:

```
int sum = 5 + 3; // '+' is an arithmetic operator
```

```
boolean result = (5 > 3) && (3 < 4); // '&&' is a logical operator
```

4. **Delimiters:** Delimiters are symbols used to separate and organize code. They include punctuation marks that help define the structure of Java programs:

- a. Semicolon (;): Used to terminate statements.
- b. Comma (,): Used to separate items in lists.
- c. Parentheses (()): Used to group expressions and parameters.
- d. Braces ({}): Used to define the beginning and end of a block of code.
- e. Brackets ([]): Used for arrays and indexing.

Examples:

```
int[] numbers = {1, 2, 3, 4, 5}; // '{}' and '[]' are delimiters
public void display() { // '{}' denotes the start and end of the method body
    System.out.println("Hello, World!"); // ';' denotes the end of the statement
}
```

ENCODING OF CHARACTERS

It refers to the process of representing characters in a digital format so that they can be understood and processed by computers. In programming languages like Java, character encoding plays a critical role in ensuring that text data is correctly interpreted and displayed.

Two common encoding systems are ASCII and Unicode.

Definition

Character encoding is the process of converting characters into a format that can be stored and manipulated by computers.

1. **ASCII (American Standard Code for Information Interchange):** ASCII is one of the earliest character encoding standards, used primarily for representing English characters and basic symbols.

- i. Character Set: ASCII includes 128 characters. These consist of:
- ii. 33 control characters (e.g., newline, carriage return)
- iii. 95 printable characters (e.g., letters, digits, punctuation)
- iv. Encoding: Each character is represented by a 7-bit binary number.
- v. Range: The values range from 0 to 127.

Example Encoding:

1. The letter 'A' is encoded as 65 (binary: 01000001).
2. The digit '0' is encoded as 48 (binary: 00110000).

ASCII Example in Java:

```
public class ASCIIDemo {  
    public static void main(String[] args) {  
        char character = 'A';  
        int asciiValue = (int) character;  
        System.out.println("ASCII value of '" + character + "' is: " + asciiValue);  
    }  
}
```

ASCII Table Example:

Character	ASCII Code (Decimal)	Binary
'A'	65	01000001
'B'	66	01000010
'0'	48	00110000
'1'	49	00110001

2. **Unicode:** Unicode is a comprehensive character encoding standard designed to include characters from virtually all writing systems in use today.
 - a. Character Set: Unicode includes a vast number of characters—over 143,000 from various languages and symbol sets.
 - b. Encoding Forms:
 - i. UTF-8: Variable-length encoding; uses 1 to 4 bytes per character.
 - ii. UTF-16: Variable-length encoding; uses 2 or 4 bytes per character.
 - iii. UTF-32: Fixed-length encoding; uses 4 bytes per character.

Example Encoding:

- i. The letter 'A' is encoded as U+0041 in Unicode.
- ii. The smiley face '😊' is encoded as U+1F60A in Unicode.

Example:

```
public class UnicodeDemo {  
    public static void main(String[] args) {  
        char unicodeChar = '😊'; // Unicode character  
        int unicodeValue = (int) unicodeChar;  
1    System.out.println("Unicode code point of '" + unicodeChar + "' is: " + unicodeValue);  
    }  
}
```

ESCAPE SEQUENCE

There are some characters which when used with a special character (\) brings out a different meaning to itself. Such combination of characters is called an escape sequence. An escape sequence is a combination of characters that has a meaning other than the literal characters contained therein, and is marked by one or more preceding (and possibly terminating) characters.

For example, when you want to enter a new line while printing, you use '\n' to print a newline. Notice how '\n' is contained within quotes. Whenever Java compiler comes across forward slash \, it reads the next character as well because it knows this is an escape sequence.

Several other escape sequences in Java are:

1. Backspace is replaced with \b
2. Newline is replaced with \n
3. Tab is replaced with \t
4. Carriage return is replaced with \r
5. Form feed is replaced with \f
6. Double quote is replaced with \"
7. Backslash is replaced with \\
8. Single quote is replaced by \'
9. Horizontal tab is replaced by \t

Escape sequences enable you to include special characters in strings and format text in a way that is meaningful and readable.

Example:

```
public class EscapeSequenceDemo {
    public static void main(String[] args) {
        // New Line
        System.out.println("Hello\nWorld");
        // Carriage Return
        System.out.println("Hello\rWorld");
        // Tab
        System.out.println("Hello\tWorld");
        // Backslash
        System.out.println("This is a backslash: \\");
        // Single Quote
        System.out.println("It\'s a test.");
        // Double Quote
        System.out.println("He said, \"Hello!\"");
        // Unicode Character
        System.out.println("Unicode Character: \u0041"); // Prints 'A'
    }
}
```



Know More

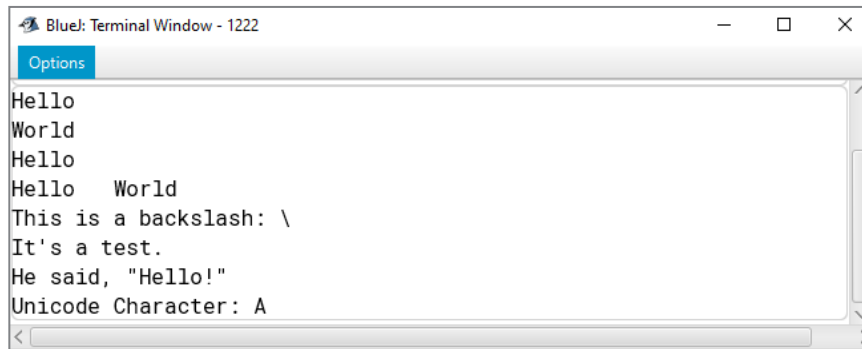
- ★ Java allows you to represent special characters using escape sequences, such as \n for newline, \t for tab, and \uXXXX for Unicode characters, where XXXX is the hexadecimal representation of the Unicode code point.



Know More

- ★ Escape sequences in Java allow you to include special characters in strings that might otherwise be difficult to represent. They are essential for formatting text and managing string content effectively.

Output



```
Blue: Terminal Window - 1222
Options
Hello
World
Hello
Hello World
This is a backslash: \
It's a test.
He said, "Hello!"
Unicode Character: A
```

TOKENS

In Java, tokens are the smallest units of a program that have meaning. Tokens are categorised based on their role in the syntax of the language. They are the building blocks of Java code. Here's a breakdown of the different types of tokens in Java:

Types of Tokens in Java

1. **Keywords:** Reserved words that have a special meaning in Java. They cannot be used as identifiers (e.g., variable names, function names).

Examples: class, public, static, void, if, else, for, while, int, return.

2. **Identifiers:** Names given to variables, methods, classes, and other user-defined elements. Identifiers must start with a letter, underscore, or dollar sign, and can be followed by letters, digits, underscores, or dollar signs.

Examples: myVariable, calculateSum, Student, MAX_VALUE.

3. **Literals:** Constants that represent fixed values in your code.

Types of Literals:

- i. **Integer Literals:** 10, 0xA (hexadecimal), 077 (octal).
 - ii. **Floating-Point Literals:** 3.14, 0.5e2 (scientific notation).
 - iii. **Character Literals:** 'a', '\n' (newline), '\u0041' (Unicode for 'A').
 - iv. **String Literals:** "Hello World", "Java".
4. **Operators:** Symbols used to perform operations on variables and values.

Types of Operators:

- i. Arithmetic Operators: +, -, *, /, %.
- ii. Relational Operators: ==, !=, <, >, <=, >=.
- iii. Logical Operators: &&, ||, !.
- iv. Assignment Operators: =, +=, -=, *=, /=.
- v. Unary Operators: +, -, ++, --.
- vi. Bitwise Operators: &, |, ^, ~, <<, >>, >>>.

5. Separators (Delimiters): Symbols that separate or group parts of the code.

Examples:

```
Parentheses: ()  
Braces: {}  
Brackets: []  
Comma: ,  
Semicolon: ;  
Dot: .
```

6. Comments: Annotations in the code that are not executed but provide explanations or notes.

Types of Comments:

1. Single-Line Comments: `// This is a comment`
2. Multi-Line Comments: `/* This is a comment */`
3. Documentation Comments: `/** This is a documentation comment */`

Example:

```
public class Example {  
    public static void main(String[] args) {  
        // Keywords and Identifiers  
        int number = 10; // int and number are identifiers, 10 is an integer literal  
        // Operators  
        int result = number * 2; // * is an arithmetic operator  
        // String Literal  
        String message = "The result is: " + result;  
        // "The result is: " and result are part of the string literal  
        // Printing  
        System.out.println(message); // println is a method call  
    }  
}
```

ERRORS

Errors in Java are issues that occurs during the execution of a program, Causing it to stop unexpectedly.

Types of errors

In Java, errors can be categorized into several types, each representing different issues that can occur during the compilation or execution of a program. Understanding these errors helps in diagnosing and fixing problems in Java applications.

1. **Syntax Errors:** These occur when the code does not conform to the syntax rules of the Java programming language.

Examples:

- i. Missing semicolons (`;`)
- ii. Mismatched parentheses, braces, or brackets
- iii. Incorrect usage of keywords or operators

Example:

```
public class SyntaxErrorExample {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!") // Missing semicolon  
    }  
}
```

2. **Compile-Time Errors:** Errors detected by the Java compiler during the compilation process.

Types:

- i. Syntax Errors
- ii. Type Errors: Incorrect data types used.
- iii. Missing Return Statement: Methods that are supposed to return a value but do not.

Example:

```
public class CompileTimeErrorExample {  
    public static void main(String[] args) {  
        int a = "string"; // Type mismatch error  
    }  
}
```

3. **Runtime Errors:** Errors that occur during the execution of the program, after successful compilation.

Examples:

- i. `ArithmeticException`: Division by zero
- ii. `NullPointerException`: Attempting to use an object reference that is null
- iii. `ArrayIndexOutOfBoundsException`: Accessing an invalid index in an array

Example:

```
public class RuntimeErrorExample {  
    public static void main(String[] args) {  
        int[] arr = new int[5];  
        System.out.println(arr[10]); // ArrayIndexOutOfBoundsException  
    }  
}
```

4. **Logical Errors:** Errors that do not cause the program to crash but produce incorrect results. These are errors in the logic of the program.

Examples:

- i. Incorrect algorithm or calculations
- ii. Wrong conditions in loops or conditional statements

Example:

```
public class LogicalErrorExample {  
    public static void main(String[] args) {  
        int a = 10;  
        int b = 20;  
        System.out.println(a - b); // Logical error if the expected result was a + b  
    }  
}
```

INPUT IN JAVA

In Java, there are several ways to take input from the user. The most common method is using the Scanner class, which is part of the java.util package. Here is an overview of how to use the Scanner class and other methods for taking input in Java.

1. **Using the Scanner Class:** The Scanner class is commonly used to take input from the console.

Example:

```
import java.util.Scanner;
public class InputExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // Taking integer input
        System.out.print("Enter an integer: ");
        int intValue = scanner.nextInt();
        System.out.println("You entered: " + intValue);
        // Taking double input
        System.out.print("Enter a double: ");
        double doubleValue = scanner.nextDouble();
        System.out.println("You entered: " + doubleValue);
        // Taking string input
        scanner.nextLine(); // Consume newline left-over
        System.out.print("Enter a string: ");
        String stringValue = scanner.nextLine();
        System.out.println("You entered: " + stringValue);
        scanner.close();
    }
}
```

2. **Using BufferedReader and InputStreamReader:** The BufferedReader class is used to read text from a character-input stream, buffering characters to provide efficient reading of characters, arrays, and lines.

Example:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
public class BufferedReaderExample {
    public static void main(String[] args) {
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        try {
            // Taking integer input
            System.out.print("Enter an integer: ");
            int intValue = Integer.parseInt(reader.readLine());
            System.out.println("You entered: " + intValue);
            // Taking double input
            System.out.print("Enter a double: ");
            double doubleValue = Double.parseDouble(reader.readLine());
            System.out.println("You entered: " + doubleValue);
            // Taking string input
            System.out.print("Enter a string: ");
            String stringValue = reader.readLine();
            System.out.println("You entered: " + stringValue);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

4. **Using Function Arguments:** Function arguments are values passed to methods when they are called. These are used to provide inputs to methods and functions.

Example:

```

public class FunctionArgumentsExample {
    // Method to display a message with a name and age
    public static void displayInfo(String name, int age) {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
    public static void main(String[] args) {
        // Calling the method with arguments
        displayInfo("Alice", 25);
    }
}

```

In this example, the displayInfo method accepts name and age as parameters and prints them. The values for name and age are passed when calling the method from main.

5. **Command-Line Arguments:** Command-line arguments are values provided to the program when it is executed from the command line. These are passed to the main method as an array of String objects.

Example:

```

public class CommandLineArgumentsExample {
    public static void main(String[] args) {
        // Check if any command-line arguments are provided
        if (args.length > 0) {
            System.out.println("Command-line arguments provided:");
            // Iterate through the arguments and print them
            for (int i = 0; i < args.length; i++) {
                System.out.println("Argument " + (i + 1) + ": " + args[i]);
            }
        } else {
            System.out.println("No command-line arguments provided.");
        }
    }
}

```

MATHEMATICAL LIBRARY METHOD

In Java, the Math class provides a set of static methods to perform basic numeric operations and mathematical functions. These methods include calculations for trigonometric functions, logarithms, exponentiation, and more.

Methods of Math class

The Math class in Java is a part of the java.lang package, and it provides a collection of methods for performing basic numeric operations such as exponentiation, logarithms, square roots, and trigonometric functions.

Some commonly used methods in the Math class along with examples:

1. **abs():** Returns the absolute value of a number.

```
int absValue = Math.abs(-10); // absValue is 10
```

2. **max():** Returns the maximum of two numbers.

```
int maxValue = Math.max(5, 10); // maxValue is 10
```

3. **min():** Returns the minimum of two numbers.

```
int minValue = Math.min(5, 10); // minValue is 5
```

4. **sqrt():** Returns the square root of a number.

```
double sqrtValue = Math.sqrt(16); // sqrtValue is 4.0
```

5. **pow():** Returns the value of the first argument raised to the power of the second argument.

```
double powerValue = Math.pow(2, 3); // powerValue is 8.0
```

6. **exp():** Returns Euler's number e raised to the power of a double value.

```
double expValue = Math.exp(1); // expValue is approximately 2.718
```

7. **log():** Returns the natural logarithm (base e) of a double value.

```
double logValue = Math.log(10); // logValue is approximately 2.302
```

8. **log10():** Returns the base 10 logarithm of a double value.

```
double log10Value = Math.log10(100); // log10Value is 2.0
```

9. **sin():** Returns the sine of the specified double value (angle in radians).

```
double sinValue = Math.sin(Math.PI / 2); // sinValue is 1.0
```

10. **cos():** Returns the cosine of the specified double value (angle in radians).

```
double cosValue = Math.cos(0); // cosValue is 1.0
```

11. **tan():** Returns the tangent of the specified double value (angle in radians).

```
double tanValue = Math.tan(Math.PI / 4); // tanValue is 1.0
```

12. **asin():** Returns the arc sine of a value (result in radians).

```
double asinValue = Math.asin(1); // asinValue is PI/2
```

13. **acos():** Returns the arc cosine of a value (result in radians).

```
double acosValue = Math.acos(0); // acosValue is PI/2
```

14. **atan():** Returns the arc tangent of a value (result in radians).

```
double atanValue = Math.atan(1); // atanValue is PI/4
```

15. **cbrt():** Returns the cube root of a double value.

```
double cbrtValue = Math.cbrt(27); // cbrtValue is 3.0
```


16. **ceil():** Returns the smallest integer that is greater than or equal to the argument.

```
double ceilValue = Math.ceil(2.3); // ceilValue is 3.0
```

17. **floor():** Returns the largest integer that is less than or equal to the argument.

```
double floorValue = Math.floor(2.7); // floorValue is 2.0
```

18. **round():** Returns the closest long or int to the argument.

```
long roundValue = Math.round(2.5); // roundValue is 3
```

19. **random():** Returns a pseudorandom double value between 0.0 (inclusive) and 1.0 (exclusive).

```
double randomValue = Math.random(); // randomValue is between 0.0 and 1.0
```

Example:

```
public class MathExamples {
    public static void main(String[] args) {
        // Example of Math.abs()
        int absValue = Math.abs(-10);
        System.out.println("Absolute value of -10: " + absValue);
        // Example of Math.max() and Math.min()
        int maxValue = Math.max(5, 10);
        int minValue = Math.min(5, 10);
        System.out.println("Maximum value: " + maxValue);
        System.out.println("Minimum value: " + minValue);
        // Example of Math.sqrt()
        double sqrtValue = Math.sqrt(16);
        System.out.println("Square root of 16: " + sqrtValue);
        // Example of Math.pow()
        double powerValue = Math.pow(2, 3);
        System.out.println("2 raised to the power 3: " + powerValue);
        // Example of Math.sin()
        double sinValue = Math.sin(Math.PI / 2);
        System.out.println("Sine of PI/2: " + sinValue);
        // Example of Math.random()
        double randomValue = Math.random();
        System.out.println("Random value: " + randomValue);
    }
}
```

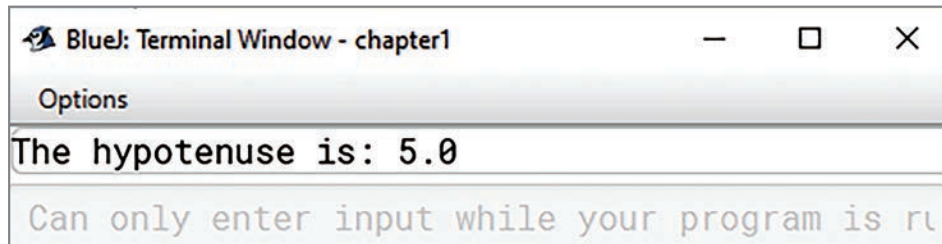
SOLVED PROGRAM

1. Write a program in Java to calculate the hypotenuse of a right-angled triangle.

Ans:

```
public class HypotenuseCalculator {
    public static void main(String[] args) {
        double sideA = 3.0;
        double sideB = 4.0;
        double hypotenuse = Math.sqrt(Math.pow(sideA, 2) + Math.pow(sideB, 2));
        System.out.println("The hypotenuse is: " + hypotenuse);
    }
}
```

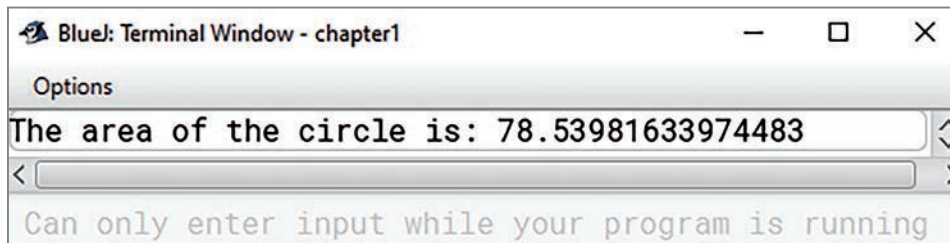
Output



2. Write a program in Java to calculate the area of a circle.

Ans:

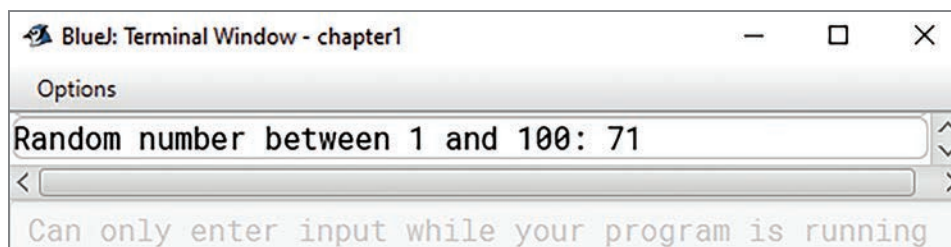
```
public class CircleAreaCalculator {
    public static void main(String[] args) {
        double radius = 5.0;
        double area = Math.PI * Math.pow(radius, 2);
        System.out.println("The area of the circle is: " + area);
    }
}
```



3. Write a program in Java to Generate a Random Number within a Range

Ans:

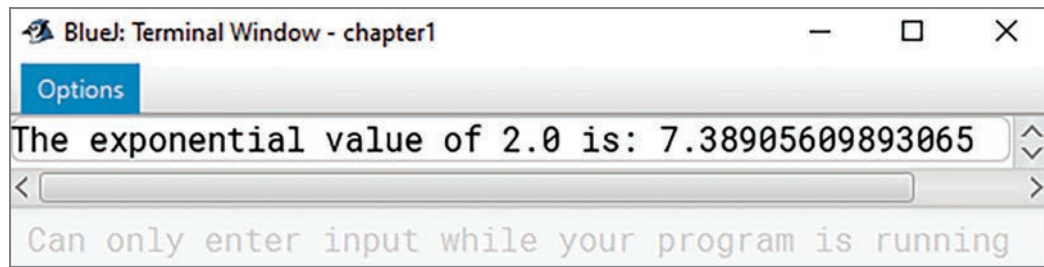
```
public class RandomNumberGenerator {
    public static void main(String[] args) {
        int min = 1;
        int max = 100;
        int randomNumber = (int)(Math.random() * (max - min + 1) + min);
        System.out.println("Random number between " + min + " and " + max + ": " + randomNumber);
    }
}
```



4. Write a program in Java to calculate the exponential value.

Ans:

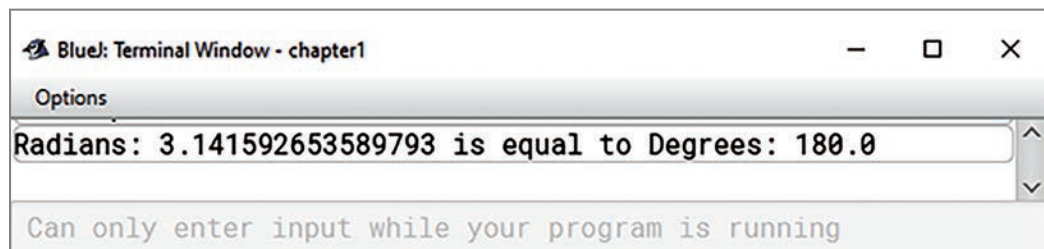
```
public class ExponentialCalculator {
    public static void main(String[] args) {
        double value = 2.0;
        double result = Math.exp(value);
        System.out.println("The exponential value of " + value + " is: " + result);
    }
}
```



5. Write a program in Java to convert radians to degrees.

Ans:

```
public class RadiansToDegreesConverter {
    public static void main(String[] args) {
        double radians = Math.PI;
        double degrees = Math.toDegrees(radians);
        System.out.println("Radians: " + radians + " is equal to Degrees: " + degrees);
    }
}
```



CONDITIONAL STATEMENTS IN JAVA

Conditional statements in Java are used to execute specific blocks of code based on certain conditions. These statements allow the program to make decisions and control the flow of execution. Here's a summary of the main conditional statements in Java:

1. **if Statement:** The if statement evaluates a condition, and if the condition is true, the code block inside the if statement is executed.

Syntax of if Statement:

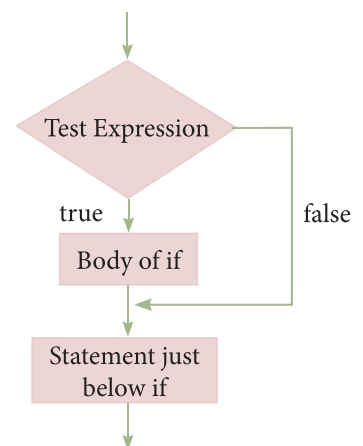
```
if(condition) {
    block of code // is executed if the condition is true
}
```

Here condition will either be evaluated as true or false. If the value is true, then a block of code will be executed. Otherwise, it will ignore it and skip to the statement just below the if command.

The test expression is first evaluated as indicated in the above diagram, and if the evaluation yields a true result, the body of the If statement is then executed; otherwise, the 'if' block is skipped and execution takes place below it.

Example:

```
class if_condition {
    public static void main(String[] args) {
        double a = 0.5;
```



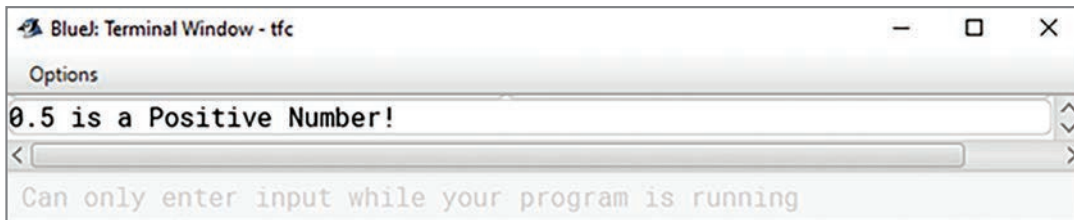
Flowchart of If-statement

```

    if (a>0)
    {
        System.out.println(a + " is a Positive Number!");
    }
}

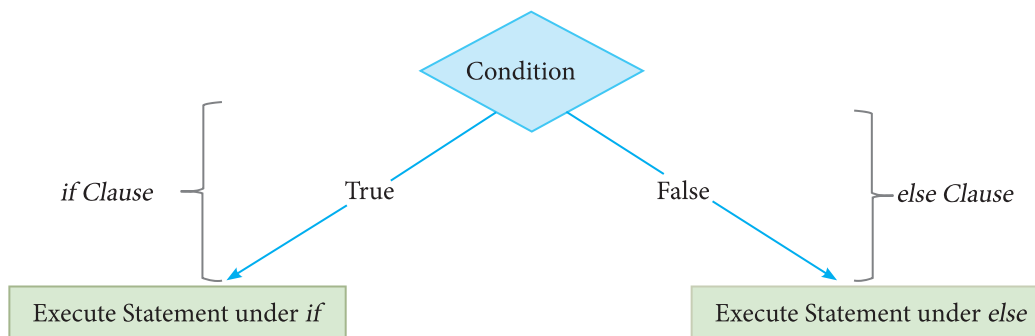
```

Output



- If-Else Statement:** If-Else statement is a control structure that selects or chooses a set of statements depending upon certain conditions.

If statements are like a subset of if-else statements.



Flowchart of if-else statement

Syntax:

```

if (condition)
{
    //Statements to be executed if condition satisfies
}
else
{
    //Statements to be executed if the condition is not satisfied
}

```

Working of if-else statements

Here, the if clause evaluates the expression. If it comes out as true, statements under if block gets executed. Else, statements under the else block get executed. As you can observe here, depending upon the condition some sets of statements are executed and some are bypassed.

Example:

```

class if_else_condition {
    public static void main(String[] args) {
        double a = -0.5;
    }
}

```



EXTRA TIME

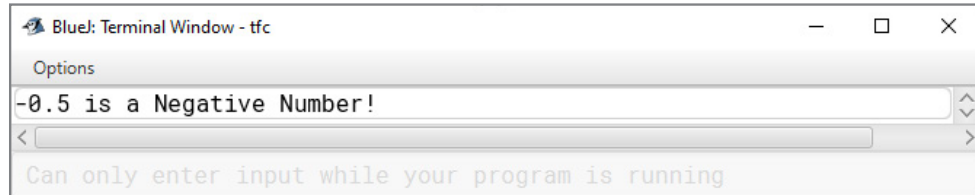
An else clause should always come after the if clause. If not, then the compiler will generate an error stating “misplaced else”.

```

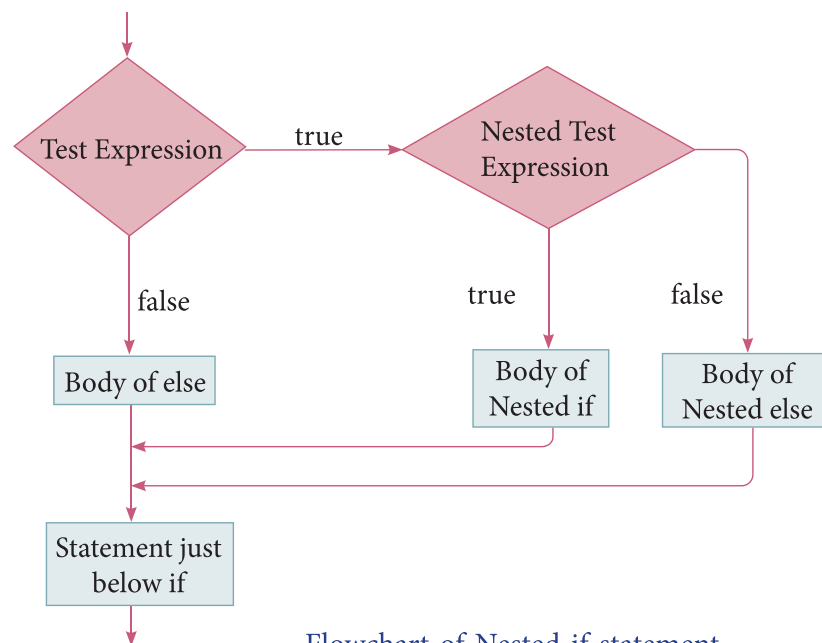
    if (a>0)
    {
        System.out.println(a + " is a Positive Number!");
    }
    else
    {
        System.out.println(a + " is a Negative Number!");
    }
}

```

Output



3. **Nested If-Else Statements:** Nested means within. Nested if condition means if-within-if. Nested if condition comes under decision-making statement in Java. There could be infinite if conditions inside an if condition. The below syntax represents the Nested if condition.



Flowchart of Nested if statement

Syntax:

```

if condition1 {
    // Executes when condition1 is true

    if condition2 {
        // Executes when condition2 is true
    }
}

```

Example:

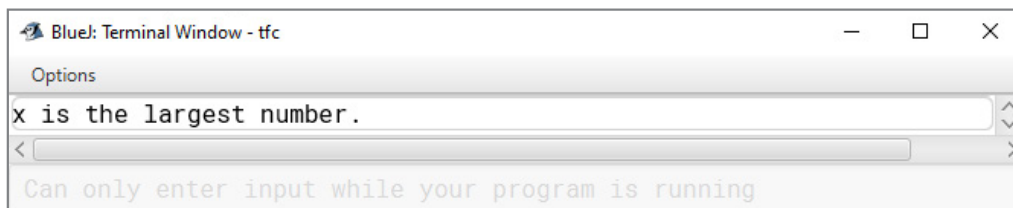
```

// Write a program to illustrate the

```

```
// use of nested if statement
public class NestedIfElseExample {
    public static void main(String[] args) {
        int x = 30;
        int y = 10;
        int z = 20;
        if (x > y) {
            if (x > z) {
                System.out.println("x is the largest number.");
            } else {
                System.out.println("z is the largest number.");
            }
        } else {
            if (y > z) {
                System.out.println("y is the largest number.");
            } else {
                System.out.println("z is the largest number.");
            }
        }
    }
}
```

Output



- Switch Statement:** The switch statement is used to select one of many code blocks to be executed based on the value of an expression. It is a more readable alternative to using multiple if-else statements when dealing with many possible values of a variable.

```
int day = 3;
String dayName;
switch (day) {
    case 1:
        dayName = "Monday";
        break;
    case 2:
        dayName = "Tuesday";
        break;
    case 3:
        dayName = "Wednesday";
        break;
    case 4:
        dayName = "Thursday";
        break;
    case 5:
        dayName = "Friday";
        break;
    case 6:
        dayName = "Saturday";
}
```



```

        break;
    case 7:
        dayName = "Sunday";
        break;
    default:
        dayName = "Invalid day";
        break;
}

```

System.out.println("Day of the week: " + dayName);

UNUSUAL TERMINATION OF A PROGRAM(SYSTEM.EXIT(0))

The System.exit(0) method in Java is used to terminate the currently running Java Virtual Machine (JVM) and end the program. The 0 argument indicates a normal termination, while any non-zero value indicates an abnormal termination.

Example:

```

public class UnusualTermination {
    public static void main(String[] args) {
        System.out.println("Program started.");
        // Perform some tasks
        for (int i = 0; i < 5; i++) {
            System.out.println("Task " + (i + 1));
        }
        // Terminate the program
        System.out.println("Terminating the program...");
        System.exit(0);
        // The following line will never be executed
        System.out.println("This line will not be printed.");
    }
}

```

SOLVED PROGRAMS

1. Write a program in Java to checks if a number is positive.

Ans:

```

public class IfStatementExample {
    public static void main(String[] args) {
        int number = 10;
        // If statement
        if (number > 0) {
            System.out.println("The number is positive.");
        }
    }
}

```

2. Write a program in Java to checks if a number is positive or negative/zero.

Ans:

```

public class IfElseStatementExample {
    public static void main(String[] args) {
        int number = -10;
        // If-Else statement
        if (number > 0) {
            System.out.println("The number is positive.");
        } else {

```

```

        System.out.println("The number is negative or zero.");
    }
}

```

3. Write a program in Java to checks if a number is positive, negative, or zero using nested if statements.

Ans:

```

public class NestedIfStatementExample {
    public static void main(String[] args) {
        int number = 0;
        // Nested If statement
        if (number >= 0) {
            if (number == 0) {
                System.out.println("The number is zero.");
            } else {
                System.out.println("The number is positive.");
            }
        } else {
            System.out.println("The number is negative.");
        }
    }
}

```

4. Write a program in Java to prints the name of the day based on the day of the week.

Ans:

```

public class SwitchStatementExample {
    public static void main(String[] args) {
        int dayOfWeek = 3;
        // Switch statement
        switch (dayOfWeek) {
            case 1:
                System.out.println("Monday");
                break;
            case 2:
                System.out.println("Tuesday");
                break;
            case 3:
                System.out.println("Wednesday");
                break;
            case 4:
                System.out.println("Thursday");
                break;
            case 5:
                System.out.println("Friday");
                break;
            case 6:
                System.out.println("Saturday");
                break;
            case 7:
                System.out.println("Sunday");
                break;
            default:
                System.out.println("Invalid day of the week.");
        }
    }
}

```

```

        break;
    }
}

```

5. Write a program in Java to calculate the volume of solids, viz. cuboid, cylinder and cone can be calculated by the formula:

Volume of a cuboid ($v = l \times b \times h$)

Volume of a cylinder ($v = \pi \times r^2 \times h$)

Volume of a cone ($v = (1/3) \times \pi \times r^2 \times h$)

Using a switch case statement, write a program to find the volume of different solids by taking suitable variables and data types.

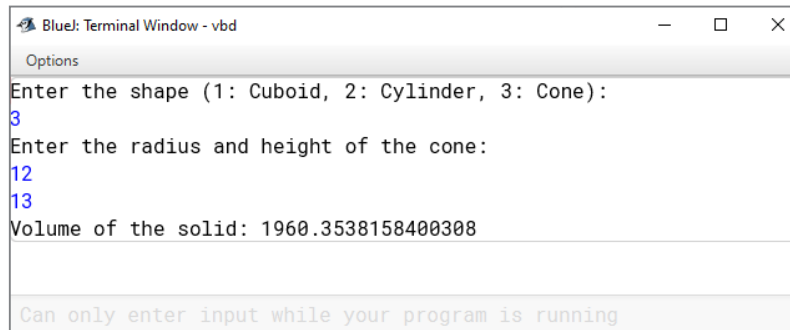
Ans:

```

import java.util.Scanner;
public class VolumeCalculator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the shape (1: Cuboid, 2: Cylinder, 3: Cone):");
        int shape = scanner.nextInt();
        double volume = 0;
        switch (shape) {
            case 1:
                System.out.println("Enter the length, width, and height of the cuboid:");
                double length = scanner.nextDouble();
                double width = scanner.nextDouble();
                double height = scanner.nextDouble();
                volume = length * width * height;
                break;
            case 2:
                System.out.println("Enter the radius and height of the cylinder:");
                double radiusCylinder = scanner.nextDouble();
                double heightCylinder = scanner.nextDouble();
                volume = Math.PI * Math.pow(radiusCylinder, 2) * heightCylinder;
                break;
            case 3:
                System.out.println("Enter the radius and height of the cone:");
                double radiusCone = scanner.nextDouble();
                double heightCone = scanner.nextDouble();
                volume = (1.0 / 3.0) * Math.PI * Math.pow(radiusCone, 2) * heightCone;
                break;
            default:
                System.out.println("Invalid shape choice!");
                System.exit(0);
        }
        System.out.println("Volume of the solid: " + volume);
        scanner.close();
    }
}

```

Output



```
BlueJ: Terminal Window - vbd
Options
Enter the shape (1: Cuboid, 2: Cylinder, 3: Cone):
3
Enter the radius and height of the cone:
12
13
Volume of the solid: 1960.3538158400308
Can only enter input while your program is running
```

ITERATIVE CONSTRUCT IN JAVA

Iterative constructs in Java are used to execute a block of code repeatedly based on certain conditions. These constructs are essential for tasks that require repeated execution, such as processing items in a list or performing operations until a certain condition is met. Here's an overview of the main iterative constructs in Java:

1. **for Loop:** The for loop is used when the number of iterations is known beforehand. It consists of an initialization, a condition, and an increment/decrement operation.

Syntax:

```
for (initialization; condition; update) {
    // Code to be executed
}
```

Example:

```
for (int i = 0; i < 5; i++) {
    System.out.println("Iteration: " + i);
}
This loop will print the numbers 0 to 4.
```

2. **while Loop:** The while loop is used when the number of iterations is not known and the loop needs to continue until a certain condition becomes false. It checks the condition before executing the loop body.

Syntax:

```
while (condition) {
    // Code to be executed
}
```

Example:

```
int i = 0;
while (i < 5) {
    System.out.println("Iteration: " + i);
    i++;
}
```

This loop will also print the numbers 0 to 4.

3. **do-while Loop:** The do-while loop is similar to the while loop, but it guarantees that the loop body is executed at least once because the condition is checked after the execution of the loop body.

Syntax:

```
do {  
    // Code to be executed  
} while (condition);
```

Example:

```
int i = 0;  
do {  
    System.out.println("Iteration: " + i);  
    i++;  
} while (i < 5);
```

This loop will print the numbers 0 to 4 as well.

BREAK STATEMENT IN JAVA

The break statement is used to exit a loop or switch statement prematurely. When break is encountered, the control is transferred to the statement immediately following the loop or switch.

Example:

```
public class BreakExample {  
    public static void main(String[] args) {  
        for (int i = 1; i <= 10; i++) {  
            if (i == 5) {  
                break; // Exit the loop when i is 5  
            }  
            System.out.println(i);  
        }  
        System.out.println("Loop terminated.");  
    }  
}
```

CONTINUE STATEMENT

The continue statement skips the current iteration of a loop and proceeds to the next iteration. Unlike break, continue does not terminate the loop; it simply skips the remaining code in the current iteration and moves to the next iteration of the loop.

Example:

```
public class ContinueExample {  
    public static void main(String[] args) {  
        for (int i = 1; i <= 10; i++) {  
            if (i == 5) {  
                continue; // Skip the rest of the loop body when i is 5  
            }  
            System.out.println(i);  
        }  
        System.out.println("Loop completed.");  
    }  
}
```

ENTRY AND EXIT CONTROLLED LOOP

In Java, loops can be categorized as either entry-controlled or exit-controlled based on when the condition is evaluated.

Entry-Controlled Loop

An entry-controlled loop is one where the condition is checked before the loop body is executed. If the condition is true, the loop body is executed; otherwise, the loop is skipped entirely. Common entry-controlled loops in Java are for and while loops.

Example: for Loop (Entry-Controlled)

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

Example: while Loop (Entry-Controlled)

```
public class WhileLoopExample {
    public static void main(String[] args) {
        int i = 1;
        while (i <= 5) {
            System.out.println("Iteration " + i);
            i++;
        }
    }
}
```

Exit-Controlled Loop

An exit-controlled loop is one where the condition is checked after the loop body is executed. This means the loop body is executed at least once, regardless of whether the condition is true or false. The do-while loop in Java is an example of an exit-controlled loop.

Example: do-while Loop (Exit-Controlled)

```
public class DoWhileLoopExample {
    public static void main(String[] args) {
        int i = 1;
        do {
            System.out.println("Iteration " + i);
            i++;
        } while (i <= 5);
    }
}
```

SOLVED PROGRAMS

1. Write a program in Java to print the sum of the first 10 natural numbers using for loop.

Ans:

```
public class SumForLoop {
    public static void main(String[] args) {
        int sum = 0;
        for (int i = 1; i <= 10; i++) {
            sum += i;
        }
        System.out.println("Sum of the first 10 natural numbers: " + sum);
    }
}
```


2. Write a program in Java to print factorial of a number using while loop.

Ans:

```
public class FactorialWhileLoop {
    public static void main(String[] args) {
        int number = 5;
        int factorial = 1;
        int i = 1;
        while (i <= number) {
            factorial *= i;
            i++;
        }
        System.out.println("Factorial of " + number + " is: " + factorial);
    }
}
```

3. Write a program in Java to print reverse a number using do-while loop.

Ans:

```
public class ReverseNumberDoWhile {
    public static void main(String[] args) {
        int number = 12345;
        int reverse = 0;
        do {
            int digit = number % 10;
            reverse = reverse * 10 + digit;
            number /= 10;
        } while (number != 0);
        System.out.println("Reversed number is: " + reverse);
    }
}
```

4. Write a program in Java to print fibonacci series up to n terms using for loop.

Ans:

```
public class FibonacciForLoop {
    public static void main(String[] args) {
        int n = 10;
        int firstTerm = 0, secondTerm = 1;
        System.out.print("Fibonacci Series up to " + n + " terms: ");
        for (int i = 1; i <= n; ++i) {
            System.out.print(firstTerm + " ");
            // compute the next term
            int nextTerm = firstTerm + secondTerm;
            firstTerm = secondTerm;
            secondTerm = nextTerm;
        }
    }
}
```

5. Write a program in Java to print sum of digits of a number using while loop.

Ans:

```
public class SumOfDigitsWhileLoop {
    public static void main(String[] args) {
        int number = 12345;
        int sum = 0;
        while (number != 0) {
            sum += number % 10;
            number /= 10;
        }
    }
}
```

```

        System.out.println("Sum of digits: " + sum);
    }
}

```

NESTED LOOP IN JAVA

Nested loops in Java are loops within loops. They are useful for dealing with multi-dimensional data structures, such as matrices, or for performing repetitive tasks that require multiple levels of iteration.

Syntax

Example 1: Nested for Loops

```

public class NestedLoopsExample {
    public static void main(String[] args) {
        int rows = 5;
        int columns = 5;
        for (int i = 1; i <= rows; i++) {
            for (int j = 1; j <= columns; j++) {
                System.out.print(i * j + "\t"); // Print the product of i and j
            }
            System.out.println(); // Move to the next line after each row
        }
    }
}

```

Example 2: Nested while Loops

```

public class NestedWhileLoops {
    public static void main(String[] args) {
        int rows = 4;
        int i = 1;
        while (i <= rows) {
            int j = 1;
            while (j <= i) {
                System.out.print("*"); // Print star
                j++;
            }
            System.out.println(); // Move to the next line
            i++;
        }
    }
}

```

Output

```

*
**
***
****

```

Example 3: Nested for and while Loops

```

public class NestedForWhileLoops {
    public static void main(String[] args) {
        int rows = 3;
        for (int i = 1; i <= rows; i++) {
            int j = 1;

```

```

        while (j <= i) {
            System.out.print(j + " "); // Print numbers in ascending order
            j++;
        }
        System.out.println(); // Move to the next line
    }
}

```

Output

```

1
1 2
1 2 3

```

SOLVED PROGRAMS

1. Write a program in Java to print a table of 2 using nested for loop.

Ans:

```

public class Tables {
    public static void main(String[] args)
    {
        System.out.println("Display Table: ");
        // Outer for loop.
        for(int i = 2; i <= 2; i++)
        {
            // Inner for loop.
            for(int j = 1; j <= 10; j++) {
                System.out.println(i+ " * " +j+" = "+ (i*j));
            }
            System.out.println(" ");
        }
    }
}

```

Output

2. Write a program in Java to print inverted pyramid star pattern.

```

*
**
***
****
*****

```

Ans:

```

public class Main {
    public static void main(String[] args) {
        // Outer loop for number of rows
        for (int i = 1; i <= 5; i++) {
            // Inner loop for number of stars in each row
            for (int j = 1; j <= i; j++) {
                System.out.print("*");
            }
            // Move to the next line after printing stars in each row
            System.out.println();
        }
    }
}

```

3. Write a program in Java to generate star pyramid pattern using nested for loop.

```

*
***
*****
*****
*****
*****

```

Ans:

```

public class Pyramid {
    public static void main(String[] args) {
        int rows = 5;
        for (int i = 1; i <= rows; i++) {
            for (int j = 1; j <= rows - i; j++) {
                System.out.print(" ");
            }
            for (int k = 1; k <= 2 * i - 1; k++) {
                System.out.print("*");
            }
            System.out.println();
        }
    }
}

```

4. Write a program to print the following pattern.

```

*
***
*****
*****
*****
*****
*****
***
*

```

Ans:

```

import java.util.Scanner;
public class Main
{
    public static void main(String[] args)
    {
        //scanner class declaration
        Scanner sc=new Scanner(System.in);
        //taking user input
        System.out.print("Enter the number of row ");
        int n=sc.nextInt();
        //declare for loop for print first pyramid
        for(int i=1;i<=n;i++)

```

```

        {
            for(int j=1;j<=n-i;j++)
            {
                System.out.print(" ");
            }
            for(int j=1;j<=i*2-1;j++)
            {
                System.out.print("*");
            }
            System.out.println();
        }
        //declare for loop for print reverse pyramid
        for(int i=n-1;i>0;i--)
        {
            for(int j=1;j<=n-i;j++)
            {
                System.out.print(" ");
            }
            for(int j=1;j<=i*2-1;j++)
            {
                System.out.print("*");
            }
            System.out.println();
        }
    }
}

```

5. Write a program to print the following pattern.

```

    A
  B C D
E F G H I

```

Ans:

```

public class Alpha {
    public static void main(String[] args) {
        int k = 65;
        System.out.println("Displaying alphabet pattern: ");
        // Outer for loop.
        for(int i = 65; i <= 69; i += 2)
        {
            // Inner for loop.
            for(int j = 69; j >= 65; j--)
            {
                if(j > i)
                    System.out.print(" ");
                else
                    System.out.format("%c ", k++);
            }
            System.out.println(" ");
        }
    }
}

```

JAVA PROGRAMS

1. Write a program in Java to calculate the square root of a number.

Ans:

```
import java.util.Scanner;
public class SquareRootCalculator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number: ");
        double num = scanner.nextDouble();
        double sqrt = Math.sqrt(num);
        System.out.println("Square root of " + num + " is: " + sqrt);
        scanner.close();
    }
}
```

2. Write a program in Java to calculate the power of a number.

Ans:

```
import java.util.Scanner;
public class PowerCalculator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the base: ");
        double base = scanner.nextDouble();
        System.out.print("Enter the exponent: ");
        double exponent = scanner.nextDouble();
        double result = Math.pow(base, exponent);
        System.out.println(base + " raised to the power of " + exponent + " is: " + result);
        scanner.close();
    }
}
```

3. Write a program in Java to calculate the absolute value of a number

Ans:

```
import java.util.Scanner;
public class AbsoluteValueCalculator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number: ");
        double num = scanner.nextDouble();
        double absValue = Math.abs(num);
        System.out.println("Absolute value of " + num + " is: " + absValue);
        scanner.close();
    }
}
```

4. Write a program in Java to generate random numbers.

Ans:

```
public class RandomNumberGenerator {
    public static void main(String[] args) {
        for (int i = 0; i < 5; i++) {
            double randomNumber = Math.random(); // Generates a random number between 0.0 and 1.0
            System.out.println("Random number " + (i + 1) + ": " + randomNumber);
        }
    }
}
```


5. Write a program in Java to calculate the trigonometric values (sine, cosine, and tangent) of an angle.

Ans:

```
import java.util.Scanner;

public class TrigonometricCalculator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter an angle in degrees: ");
        double degrees = scanner.nextDouble();
        double radians = Math.toRadians(degrees);
        double sinValue = Math.sin(radians);
        double cosValue = Math.cos(radians);
        double tanValue = Math.tan(radians);
        System.out.println("Sine of " + degrees + " degrees is: " + sinValue);
        System.out.println("Cosine of " + degrees + " degrees is: " + cosValue);
        System.out.println("Tangent of " + degrees + " degrees is: " + tanValue);
        scanner.close();
    }
}
```

6. Write a program in Java to check if a number is prime.

Ans:

```
import java.util.Scanner;

public class PrimeNumberCheck {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = scanner.nextInt();
        boolean isPrime = true;
        if (num <= 1) {
            isPrime = false;
        } else {
            for (int i = 2; i <= num / 2; i++) {
                if (num % i == 0) {
                    isPrime = false;
                    break;
                }
            }
        }
        if (isPrime) {
            System.out.println(num + " is a prime number.");
        } else {
            System.out.println(num + " is not a prime number.");
        }
        scanner.close();
    }
}
```

7. Write a program in Java to sum of digits of a number.

Ans:

```
import java.util.Scanner;

public class SumOfDigits {
    public static void main(String[] args) {
```

```

Scanner scanner = new Scanner(System.in);
System.out.print("Enter a number: ");
int num = scanner.nextInt();
int sum = 0;
while (num != 0) {
    sum += num % 10;
    num /= 10;
}
System.out.println("Sum of the digits is: " + sum);
scanner.close();
}
}

```

8. Write a program in Java to factorial of a number.

Ans:

```

import java.util.Scanner;
public class Factorial {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = scanner.nextInt();
        int factorial = 1;
        for (int i = 1; i <= num; i++) {
            factorial *= i;
        }
        System.out.println("Factorial of " + num + " is: " + factorial);
        scanner.close();
    }
}

```

9. Write a program in Java to reverse a number.

Ans:

```

import java.util.Scanner;
public class ReverseNumber {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = scanner.nextInt();
        int reverse = 0;
        while (num != 0) {
            int digit = num % 10;
            reverse = reverse * 10 + digit;
            num /= 10;
        }
        System.out.println("Reversed number is: " + reverse);
        scanner.close();
    }
}

```

10. Write a program in Java to print fibonacci series.

Ans:

```

import java.util.Scanner;
public class FibonacciSeries {
    public static void main(String[] args) {

```

```

Scanner scanner = new Scanner(System.in);
System.out.print("Enter the number of terms: ");
int terms = scanner.nextInt();
int first = 0, second = 1;
System.out.print("Fibonacci Series: " + first + " " + second);
for (int i = 3; i <= terms; i++) {
    int next = first + second;
    System.out.print(" " + next);
    first = second;
    second = next;
}
scanner.close();
}
}

```

KEY TERMS

- **Class:** A class is a blueprint for creating objects. It defines a data type by bundling data and methods that work on that data into one single unit.
- **Object:** An object is an instance of a class. It is a concrete realization of a class that occupies memory and can have attributes and methods.
- **Inheritance:** Inheritance is a mechanism where one class (subclass) inherits the properties and methods of another class (superclass).
- **Polymorphism:** Polymorphism allows objects of different classes to be treated as objects of a common superclass. It includes method overriding and method overloading.
- **Abstraction:** Abstraction is the concept of hiding the implementation details and showing only the functionality to the user. It is achieved using abstract classes and interfaces.

Summary ♦

- ❖ OOPs is a programming paradigm based on the concept of objects, which can contain data in the form of fields and code in the form of methods. The core principles are Encapsulation, Inheritance, Polymorphism and Abstraction.
- ❖ Class: A blueprint or template for creating objects. It defines a data type by bundling data (attributes) and methods (functions) that operate on the data.
- ❖ Object: An instance of a class. It represents a specific realization of the class with concrete values for its attributes.
- ❖ Primitive Data Types: Basic types provided by Java. Includes int, char, boolean, double, etc.
- ❖ Non-Primitive Data Types: Also known as reference types. Includes String, arrays, classes, and interfaces.
- ❖ Arithmetic Operators: Used for mathematical operations. Examples include +, -, *, /, %.
- ❖ Relational Operators: Used to compare values. Examples include ==, !=, >, <, >=, <=.
- ❖ Logical Operators: Used for logical operations. Examples include && (AND), || (OR), ! (NOT).
- ❖ Assignment Operators: Used to assign values to variables. Examples include =, +=, -=, *=, /=.
- ❖ Using Scanner: A common way to get user input in Java.
- ❖ Command Line Arguments: Data passed to the program at runtime via the command line.
- ❖ For Loop: Repeats a block of code a specific number of times.
- ❖ While Loop: Repeats a block of code as long as a condition is true.
- ❖ Do-While Loop: Similar to the while loop, but guarantees that the code block is executed at least once.

Exercises - 1 (Solved)

A. Multiple choice questions.

[Understanding]

1. What is the main principle of OOP that involves hiding internal state and requiring all interaction to be performed through an object's methods?
a. Inheritance b. Polymorphism c. Encapsulation d. Abstraction
2. Which OOP concept allows one class to inherit the properties and behaviours of another class?
a. Polymorphism b. Encapsulation c. Inheritance d. Abstraction
3. Which OOP principle enables one interface to be used for a general class of actions?
a. Encapsulation b. Abstraction c. Inheritance d. Polymorphism
4. What is the term for a class that is designed to be inherited by other classes?
a. Derived class b. Subclass c. Base class d. Super class
5. What is a class in Java?
a. A blueprint for creating objects b. A data type
c. An object itself d. A method in Java
6. Which of the following is a correct way to create an object in Java?
a. Object obj = new Object(); b. Object obj = Object.new();
c. new Object obj = new Object(); d. Object obj = Object.create();
7. Which method is used to initialize an object in Java?
a. finalize() b. initialize() c. constructor d. setup()
8. What is the default value of an instance variable of type int in Java?
a. 0 b. null c. 1 d. -1
9. How do you access a method from an object in Java?
a. object.methodName(); b. methodName.object();
c. object.callMethod(); d. object.method();
10. Which keyword is used to access members of a superclass from a subclass?
a. this b. super c. base d. parent

ANSWERS

1. c. 2. c. 3. d. 4. c. 5. a. 6. a. 7. c. 8. a. 9. a. 10. b.

B. Fill in the blanks.

[Recall]

1. In Java, the keyword _____ is used to execute a block of code if a specified condition is true.
2. The _____ statement allows you to choose between multiple blocks of code based on the value of a variable.
3. The _____ keyword is used to specify an alternative block of code to be executed if the condition in an if statement is false.
4. To test multiple conditions in a single if statement, you can use _____ operators such as && (AND) and || (OR).

5. The _____ keyword can be used within a switch statement to exit from the switch block and prevent fall-through.
6. In a switch statement, if no matching case is found, the _____ block will be executed if it is present.
7. The _____ keyword is used to exit from a loop or switch statement prematurely.
8. To skip the remaining statements in the current iteration of a loop and proceed to the next iteration, use the _____ keyword.
9. The _____ statement is used to execute a block of code repeatedly based on a condition being true.
10. The _____ loop is guaranteed to execute its block of code at least once, regardless of the condition.

ANSWERS

- | | | | | |
|------------|-----------|-------------|------------|--------------|
| 1. if | 2. switch | 3. else | 4. logical | 5. break |
| 6. default | 7. break | 8. continue | 9. while | 10. do-while |

C. State the following statements are True or False.

[Recall]

1. Scanner class is used for input in Java.
2. System.out.println() is used to read input from the user.
3. You can use readInt() method of the Scanner class to read integer values.
4. Scanner can only read data from the console.
5. The Math.sqrt() method returns the square root of a number.
6. The Math.pow() method is used to calculate the power of a number.
7. The Math.abs() method can only be used with integer values.
8. The Math.random() method generates a random integer value.
9. The for loop in Java can only be used for iteration with a predefined number of iterations.
10. The while loop continues executing as long as its condition evaluates to true.

D. Very short answer type questions.

[Understanding]

1. What does OOP stand for?

Ans: The full form of OOP is Object-Oriented Programming.

2. What keyword is used to create a class in Java?

Ans: class.

3. What is an object in Java?

Ans: An instance of a class.

4. Which keyword is used to inherit a class in Java?

Ans: extends.

5. What is encapsulation?

Ans: Hiding the internal state of an object and requiring all interaction to be done through methods.

6. What is the result of $4 + 2$ in Java?

Ans: 6

7. What operator is used for logical AND in Java?

Ans: &&.

8. How do you declare a variable as a constant in Java?

Ans: Use the final keyword.

9. What is the purpose of the ++ operator?

Ans: To increment a variable by 1.

10. What is the result of $10 \% 3$ in Java?

Ans: 1

E. Short answer type questions.

[Analysis]

1. Describe the switch statement in Java. How does it work? Provide an example.

Ans: The switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

```
int day = 2;
switch (day) {
    case 1:
        System.out.println("Monday");
        break;
    case 2:
        System.out.println("Tuesday");
        break;
    default:
        System.out.println("Invalid day");
}
```

2. How do you declare a variable in Java?

Ans: `int x = 10;` (variable type, name, and initial value).

3. Explain the different types of control statements in Java with examples.

Ans: Control statements in Java manage the flow of execution. They include:

- i. Selection statements: if, if-else, switch.
- ii. Iteration statements: for, while, do-while.
- iii. Jump statements: break, continue

Example:

```
int number = 10;
if (number > 5) {
    System.out.println("Number is greater than 5");
} else {
    System.out.println("Number is 5 or less");
}
switch (number) {
    case 10:
        System.out.println("Number is 10");
        break;
```

```

        default:
            System.out.println("Number is not 10");
    }

```

4. How do you write an if-else statement in Java?

Ans:

```

if (condition) {
    // code
} else {
    // code
}

```

5. Explain the structure and use of a for loop in Java with an example.

Ans: The for loop is used to execute a block of statements repeatedly until a specified condition is true. The structure includes initialization, condition, and increment/decrement.

```

for (int i = 0; i < 5; i++) {
    System.out.println("Iteration: " + i);
}

```

This loop will print "Iteration: 0" to "Iteration: 4".

6. Compare and contrast while and do-while loops with examples.

Ans: while loop: Checks the condition before executing the loop body.

```

int i = 0;
while (i < 5) {
    System.out.println("Iteration: " + i);
    i++;
}

```

do-while loop: Executes the loop body at least once before checking the condition.

```

int i = 0;
do {
    System.out.println("Iteration: " + i);
    i++;
} while (i < 5);

```

7. Write a program using a nested for loop to print a multiplication table up to 5x5.

Ans:

```

public class MultiplicationTable {
    public static void main(String[] args) {
        for (int i = 1; i <= 5; i++) {
            for (int j = 1; j <= 5; j++) {
                System.out.print(i * j + "\\t");
            }
            System.out.println();
        }
    }
}

```

8. How do you use the break statement within a loop? Provide an example.

Ans: The break statement terminates the loop immediately.

```

for (int i = 0; i < 10; i++) {
    if (i == 5) {
        break;
    }
    System.out.println("Iteration: " + i);
}

```


This loop will print “Iteration: 0” to “Iteration: 4” and then stop.

9. Explain the use of the continue statement in a loop with an example.

Ans: The continue statement skips the current iteration of the loop and proceeds to the next iteration.

```
for (int i = 0; i < 10; i++) {  
    if (i % 2 == 0) {  
        continue;  
    }  
    System.out.println("Odd number: " + i);  
}
```

This loop will print odd numbers between 0 and 9.

10. Describe how to read user input from the console using the Scanner class. Provide a program that reads a name and age from the user.

Ans:

```
import java.util.Scanner;  
public class UserInput {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
        System.out.print("Enter your name: ");  
        String name = scanner.nextLine();  
        System.out.print("Enter your age: ");  
        int age = scanner.nextInt();  
        System.out.println("Name: " + name);  
        System.out.println("Age: " + age);  
        scanner.close();  
    }  
}
```

F. Application based questions.

[Application]

- 1. You are developing an inventory management system for a small store. The system needs to calculate the total cost of items in stock. Each item has a quantity and a price per unit. Write a Java program to read the quantity and price for each item and calculate the total inventory value using a loop.**

Ans:

```
import java.util.Scanner;  
public class InventoryManagement {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
        double totalValue = 0;  
        for (int i = 0; i < 5; i++) {  
            System.out.print("Enter quantity for item " + (i + 1) + ": ");  
            int quantity = scanner.nextInt();  
            System.out.print("Enter price per unit for item " + (i + 1) + ": ");  
            double price = scanner.nextDouble();  
            totalValue += quantity * price;  
        }  
        System.out.println("Total inventory value: $" + totalValue);  
        scanner.close();  
    }  
}
```

2. A bank offers different interest rates based on the type of account. Using input in Java, write a program to calculate the interest for a given principal amount, rate of interest, and time period using the mathematical library method `Math.pow()` to compute compound interest.

Ans:

```
import java.util.Scanner;

public class BankInterestCalculation {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter principal amount: ");
        double principal = scanner.nextDouble();
        System.out.print("Enter rate of interest: ");
        double rate = scanner.nextDouble();
        System.out.print("Enter time period (in years): ");
        double time = scanner.nextDouble();
        System.out.print("Enter number of times interest is compounded per year: ");
        int n = scanner.nextInt();
        double amount = principal * Math.pow((1 + rate / (n * 100)), n * time);
        double interest = amount - principal;
        System.out.println("Compound interest: $" + interest);
        scanner.close();
    }
}
```

3. Develop a grade processing system for a class of students. The program should read the marks of each student, calculate the average, and determine the highest and lowest marks using loops and conditional statements.

Ans:

```
import java.util.Scanner;

public class GradeProcessingSystem {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] marks = new int[10];
        int sum = 0;
        int highest = Integer.MIN_VALUE;
        int lowest = Integer.MAX_VALUE;
        for (int i = 0; i < marks.length; i++) {
            System.out.print("Enter marks for student " + (i + 1) + ": ");
            marks[i] = scanner.nextInt();
            sum += marks[i];
            if (marks[i] > highest) {
                highest = marks[i];
            }
            if (marks[i] < lowest) {
                lowest = marks[i];
            }
        }
        double average = sum / (double) marks.length;
        System.out.println("Average marks: " + average);
        System.out.println("Highest marks: " + highest);
        System.out.println("Lowest marks: " + lowest);
        scanner.close();
    }
}
```

4. Write a Java program that reads temperatures in Celsius from the user and converts them to Fahrenheit until the user decides to stop. Use a loop to continuously read input and the mathematical library method for the conversion.

Ans:

```
import java.util.Scanner;
public class TemperatureConversion {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String choice;
        do {
            System.out.print("Enter temperature in Celsius: ");
            double celsius = scanner.nextDouble();
            double fahrenheit = (celsius * 9 / 5) + 32;
            System.out.println("Temperature in Fahrenheit: " + fahrenheit);
            System.out.print("Do you want to convert another temperature? (yes/no): ");
            choice = scanner.next();
        } while (choice.equalsIgnoreCase("yes"));
        scanner.close();
    }
}
```

5. A company wants to calculate the commission for its sales employees based on their sales. Write a program that reads the sales amount for each employee and calculates the commission using different rates based on the sales amount. Use loops to process multiple employees and conditional statements to determine the commission rate.

Ans:

```
import java.util.Scanner;
public class SalesCommission {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of employees: ");
        int numEmployees = scanner.nextInt();
        for (int i = 0; i < numEmployees; i++) {
            System.out.print("Enter sales amount for employee " + (i + 1) + ": ");
            double sales = scanner.nextDouble();
            double commission;
            if (sales >= 10000) {
                commission = sales * 0.1;
            } else if (sales >= 5000) {
                commission = sales * 0.05;
            } else {
                commission = sales * 0.02;
            }
            System.out.println("Commission for employee " + (i + 1) + ": $" + commission);
        }
        scanner.close();
    }
}
```

Exercises - 2 (Unsolved)

A. Multiple choice questions.

[Understanding]

1. What is the size of an int data type in Java?
a. 8 bits b. 16 bits c. 32 bits d. 64 bits
2. Which of the following is a non-primitive data type in Java?
a. int b. char c. boolean d. String
3. What will be the result of 10 / 3 in Java?
a. 3.0 b. 3 c. 3.33 d. Error
4. Which data type can hold a decimal number in Java?
a. int b. char c. float d. boolean
5. What is the default value of a boolean variable in Java?
a. true b. false c. 0 d. Null
6. What does the + operator do when used with Strings in Java?
a. Performs addition b. Concatenates the Strings
c. Performs subtraction d. Multiplies the Strings
7. Which operator is used to compare two values in Java?
a. = b. == c. := d. =>

B. Fill in the blanks.

[Recall]

1. The _____ loop uses a condition to determine whether to continue looping and is used when the number of iterations is not known in advance.
2. The _____ loop is typically used when you know in advance how many times you need to execute a block of code.
3. In a for loop, the _____ statement initializes the loop control variable.
4. The _____ part of a for loop specifies the condition that must be true for the loop to continue executing.
5. The _____ part of a for loop is executed after each iteration of the loop.
6. In a while loop, if the condition is always true, the loop will become an _____ loop.
7. The _____ keyword is used to skip the rest of the current loop iteration and move on to the next iteration.

C. State whether the following statements are True or False.

[Understanding]

1. The do-while loop executes its block of code at least once, regardless of the condition.
2. A break statement can be used to exit a for, while, or do-while loop.
3. The if-else statement can only have two branches.
4. The switch statement in Java can only be used with int and char types.
5. The ternary operator is a shorthand for the if-else statement.
6. The else part of an if-else statement is mandatory.
7. In Java, String is a primitive data type.

D. Very short answer type questions.**[Understanding]**

1. Which class is used to read input from the console in Java?
2. How do you create a Scanner object?
3. What method is used to read an integer from the console using Scanner?
4. How do you read a string input from the user?
5. What is the default delimiter for Scanner?
6. What loop is used when the number of iterations is known?
7. How do you exit a loop early in Java?

E. Short answer type questions.**[Analysis]**

1. Explain the difference between next() and nextLine() methods of the Scanner class.
2. How do you handle exceptions while reading input in Java? Provide an example.
3. Write a Java program that calculates and prints the area of a circle given its radius using the Math.PI and Math.pow methods.
4. Explain the Math.abs() method with an example program.
5. Write a program to demonstrate the usage of the Math.random() method to generate a random number between 1 and 100.
6. Describe the Math.max() and Math.min() methods with examples.
7. Write a program that uses Math.ceil() and Math.floor() methods to round a number up and down.

F. Previous year questions and answers**[Analysis/Application]**

1. Identify the type of operator &&: **[ICSE 2023]**
a. Ternary b. Unary c. Logical d. Relational
2. What value will Math.sqrt(Math.ceil(15.3)) return? **[ICSE 2023]**
a. 16.0 b. 16 c. 4.0 d. 5.0
3. The absence of which statement leads to fall through situation in switch case statement? **[ICSE 2023]**
a. continue b. break c. return d. System.exit(0)
4. int x = (int)32.8; is an example of _____ typecasting. **[ICSE 2023]**
a. implicit b. automatic c. explicit d. coercion
5. The code obtained after compilation is known as: **[ICSE 2023]**
a. Source code b. Object code c. Machine code d. Java byte code
6. Missing a semicolon in a statement is what type of error? **[ICSE 2023]**
a. Logical b. Syntax c. Runtime d. No error
7. The number of bits occupied by the value 'a' are: **[ICSE 2023]**
a. 1 bit b. 2 bits c. 4 bits d. 16 bits
8. Consider the following program segment and select the output of the same when n = 10 : **[ICSE 2023]**

```
switch(n)
{
    case 10 : System.out.println(n*2);
```

```

        case 4 : System.out.println(n*4); break;
        default : System.out.println(n);
    }

```

- a. 20,80 b. 10, 4 c. 20, 40 d. 10, 10

9. The number of bits occupied by the value 'a' are: [ICSE 2023]

- a. 1 bit b. 2 bits c. 4 bits d. 16 bits

10. Convert the following do...while loop to for loop: [ICSE 2023]

```

int x = 10;
do
{
    x--;
    System.out.print(x);
}

```

11. Convert the following do...while loop to for loop: [ICSE 2023]

```

int x = 10;
do
{
    x--;
    System.out.print(x);
}

```

12. Name the following: [ICSE 2023]

- What is an instance of the class called?
- The method which has same name as that of the class name.

Project Work

[Application]

Create a simple library management system to keep track of books, members, and transactions (issuing and returning books).

Lab Work

[Application]

- Write a program in Java to print a diamond shape star.
- Write a program in Java to print a pyramid of numbers.

Scan the QR code for more solved questions.



ICODE-tzFe



Teacher's Notes

- Encourage students to write Java code to solve each exercise and run their programs to see the output
- Encourage students to continue practicing loops on their own and explore more complex loop patterns and applications