TOUCHPAD

ICSE Computer Applications-X (Ver.2.0)

With BlueJ

9

Teacher's Manual

Extended Support for Teachers

ORANGE

www.orangeeducation.in

Teacher's Time Table

Periods / Days	0	I	II	III	IV	BREAK	V	VI	VII	VIII
Monday										
Tuesday										
Wednesday										
Thursday										
Friday										
Saturday										

Development Milestones in a Child

Development milestones are a set of functional skills or age-specific tasks that most children can do at a certain age. These milestones help the teacher identify and understand how children differ in different age groups.

Age
5 - 8 Years

Physical	• First permanent tooth erupts • Shows mature throwing and catching patterns • Writing is now smaller and more readable • Drawings are now more detailed, organised and have a sense of depth
Cognitive	• Attention continues to improve, becomes more selective and adaptable • Recall, scripted memory, and auto-biographical memory improves • Counts on and counts down, engaging in simple addition and subtraction • Thoughts are now more logical
Language	• Vocabulary reaches about 10,000 words • Vocabulary increases rapidly throughout middle childhood
Emotional/ Social	• Ability to predict and interpret emotional reactions of others enhances • Relies more on language to express empathy • Self-conscious emotions of pride and guilt are governed by personal responsibility • Attends to facial and situational cues in interpreting another's feelings • Peer interaction is now more prosocial, and physical aggression declines

"If you cannot do great things, do small things in a great way."

Age 9 - 11 Years

Physical	• Motor skills develop resulting in enhanced reflexes
Cognitive	• Applies several memory strategies at once • Cognitive self-regulation is now improved
Language	• Ability to use complex grammatical constructions enhances • Conversational strategies are now more refined
Emotional/ Social	• Self-esteem tends to rise • Peer groups emerge

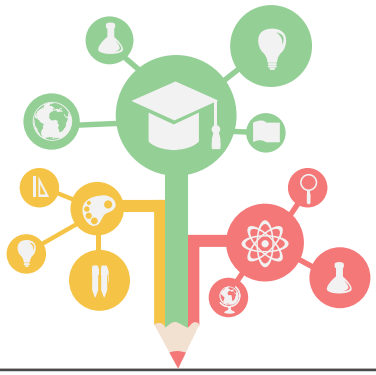
Age 11 - 20 Years

Physical	• If a girl, reaches peak of growth spurt • If a girl, motor performance gradually increases and then levels off • If a boy, reaches peak and then completes growth spurt • If a boy, motor performance increases dramatically
Cognitive	• Is now more self-conscious and self-focused • Becomes a better everyday planner and decision maker
Emotional/ Social	• May show increased gender stereotyping of attitudes and behaviour • May have a conventional moral orientation

Managing the children's learning needs according to their developmental milestones is the key to a successful teaching-learning transaction in the classroom.

"Family is the most important thing in the world."

Teaching Pedagogies



Pedagogy is often described as the approach to teaching. It is the study of teaching methods including the aims of education and the ways in which such goals can be achieved.

Lesson Plans

A lesson plan is the instructor's road map which specifies what students need to learn and how it can be done effectively during the class time. A lesson plan helps teachers in the classroom by providing a detailed outline to follow in each class.

A lesson plan addresses and integrates three key components:

- Learning objectives
- Learning activities
- Assessment to check the student's understanding

A lesson plan provides an outline of the teaching goals:

Before the class

1. Identify the learning objectives.
2. Plan the lesson in an engaging and meaningful manner.
3. Plan to assess student's understanding.
4. Plan for a lesson closure.

During the class

Present the lesson plan.

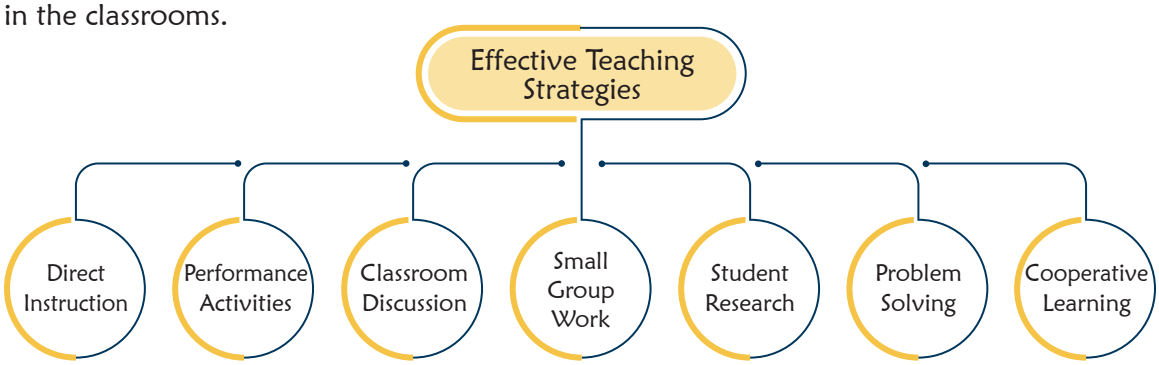
After the class

Reflect on what worked well and why. If needed, revise the lesson plan.

"Knowing yourself is the beginning of all wisdom."

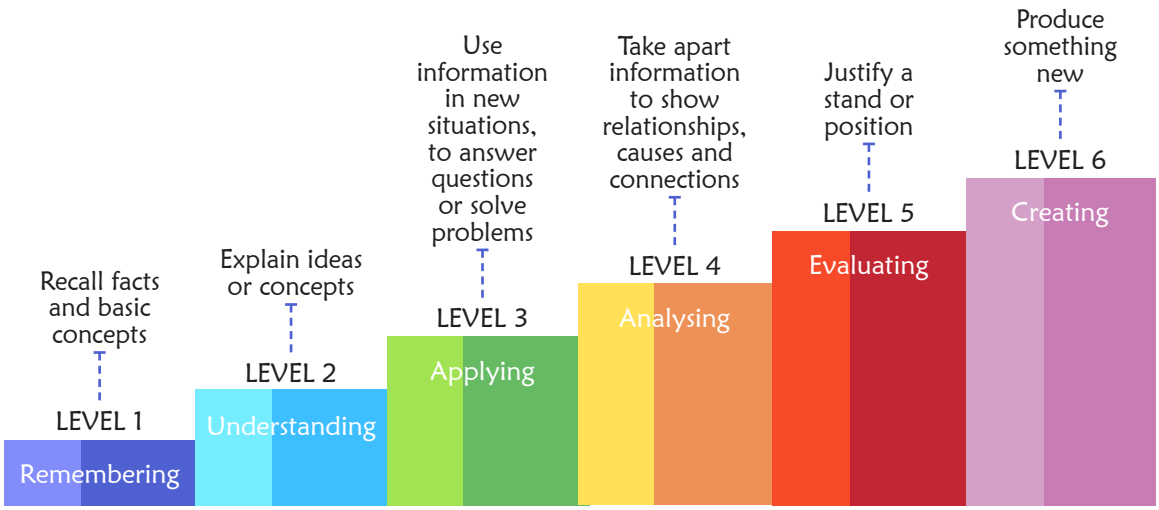
Teaching Strategies

Numerous strategies have evolved over the years to facilitate the teaching-learning process in the classrooms.



Bloom's Taxonomy

Bloom's Taxonomy was created by Dr Benjamin Bloom and several of his colleagues, to promote higher forms of thinking in education instead of rote learning. There are three domains of learning: cognitive (mental), affective (emotional), and psychomotor (physical). However, when we refer to Bloom's Taxonomy we speak of the cognitive domain. Bloom's Taxonomy is a list of cognitive skills that is used by teachers to determine the level of thinking their students have achieved. As a teacher, one should attempt to move students up the taxonomy as they progress in their knowledge.



Teachers should focus on helping students to remember information before expecting them to understand it, helping them understand it before expecting them to apply it to a new situation, and so on.

> "If you have no confidence in self, you are twice defeated in the race of life."

CLASS 9

LESSON PLAN

1 Principles of Object-Oriented Programming

Teaching Objectives

By the end of this lesson, students will be able to:

- Identify and classify different types of computer languages.
- Differentiate between low-level and high-level languages.
- Compare Procedure-Oriented Programming (POP) and Object-Oriented Programming (OOP).
- Define and explain the four key principles of OOP: Encapsulation, Abstraction, Inheritance, and Polymorphism.
- Illustrate OOP concepts using real-life analogies and examples.

Number of Periods	
Theory	Practical
3	2

Teaching Plan

Introduction

- Strategy: Brainstorming
  - Ask students to brainstorm: "What languages do computers understand?"
  - Write responses on the board (e.g., Binary, English, C++, Java).
- Present visual examples of binary vs high-level syntax to demonstrate ease of use.
- Use a quick analogy: Comparing a car's dashboard (abstraction) to its engine (hidden details).

Lesson Delivery

1. **Types of Computer Languages**
   - Machine Language, Assembly Language, High-Level Languages (3GL, 4GL, 5GL).
   - Role of compilers, assemblers, and interpreters.

     Strategy: Numbered Heads Together.
   - Form groups of four; assign numbers.
   - Ask questions such as:
     - Which language is machine-dependent?

- What is the role of a compiler?
- Give examples of 3GL and 4GL.

2. **POP vs OOP**
   - Explain characteristics, approach (top-down vs bottom-up), focus (functions vs data).
   - Present tabular comparison.

     Strategy: Round Robin.
   - In groups, students take turns listing advantages of POP and OOP.
   - Share findings with class.
3. **OOP Principles**
   - **Encapsulation:** Grouping data and methods into a class.
   - **Abstraction:** Hiding unnecessary details.
   - **Inheritance:** One class inheriting from another.
   - **Polymorphism:** One method, multiple behaviours.

     Strategy: Think-Share
   - **Pose questions:** "What happens if we don't hide data?" "Can a child have features of a parent in code?"
   - Students think individually, then share answers in pairs and then with the whole class.

     Code Demonstration: Java class with encapsulation and inheritance.

Extension

Strategy: One Sentence Summary

- Ask students to summarise: "What is OOP and why is it important?" in one sentence.

RAFT Writing

- **Role:** An OOP Principle
- **Audience:** A beginner programmer
- **Format:** Diary entry
- **Topic:** "My day in a programmer's code"

Discussion Prompts

- Why is high-level language preferred today?
- How does OOP help in solving real-world problems?

Evaluation

Strategy: Muddiest Point

- Ask students to write down the part they found most confusing.

Worksheet:

- Fill in the blanks and multiple-choice questions based on textbook exercises.

Student Activity: Application Cards

- Each student writes one real-world application of OOP on a card and shares it with the class.

Suggested Activity

Strategy: Send-a-Problem

- Create a folder with a scenario: "Design an OOP-based solution for a Library System".
- Each group adds their solution, then passes it to the next group who adds theirs.
- Final group selects the best elements and explains.

2 Introduction to Java

Teaching Objectives

By the end of this lesson, students will be able to:

- Explain the evolution and history of Java.
- Distinguish between Java Applications and Java Applets.
- Understand the Java compilation process including the roles of javac and JVM.
- Identify and explain the key features, advantages, and disadvantages of Java.
- Use BlueJ IDE to create, compile, and execute a Java class.

Number of Periods	
Theory	Practical
3	2

Teaching Plan

Introduction

- **Strategy:** Think-Pair-Share
  - Ask students: "Where have you seen Java used?" or "Have you heard of any games or websites that use Java?"
  - Allow them to think and share with a partner before discussing as a class.
- Use a world map to highlight Java's origin and its global evolution.
- Show logos or software interfaces (e.g., Minecraft, Eclipse IDE) that involve Java.

Lesson Delivery

1. **Evolution of Java**
   - Discuss the story of OAK and how it became Java
   - Present version timeline chart (from 1995 to 2024)

Strategy: Timeline Activity / Visual Chart Presentation.

- Students collaboratively build a Java version timeline using cards or a visual digital board.

2. **Types of Java Programming**
   - Explain Java Applications (standalone) vs Java Applets (web-based).
   - Compare with real-life examples (MS Word vs online games).

     Strategy: Role Play
   - Students act out scenes representing Java Application and Java Applet functionalities.
3. **Java Compilation Process**
   - Explain step-by-step: source code > bytecode > machine code.
   - Components: Java Compiler (javac), Java Virtual Machine (JVM).

     Strategy: Picture Study & Discussion
   - Use diagram of Java Compilation process from the textbook.
   - Ask students to label and explain each stage in groups.
4. **Features, Advantages and Disadvantages of Java**
   - **Features:** Object-oriented, Robust, Platform-independent, etc.
   - **Advantages:** Portable, Economical, Secure, etc.
   - **Disadvantages:** Slower than C++, GUI issues, large memory requirement.

     Strategy: Four Corners
   - Assign corners: Features, Advantages, Disadvantages, Applications.
   - Let groups rotate and list points for each category, then present.
5. **Working with BlueJ**
   - Demonstrate installation steps and starting the IDE.
   - Steps to create new project, class, compile, and run.

     Strategy: Guided Demonstration + Hands-on Task.
   - **Teacher demonstrates;** students follow along on systems.
   - Students type and run a sample program to print a message.

Extension

Strategy: RAFT Writing

- **Role:** Java Bytecode.
- **Audience:** Beginner Java Student.
- **Format:** Journal Entry.
- **Topic:** "My journey from source code to execution".

Discussion Prompts

- Why is Java called a platform-independent language?
- How does automatic garbage collection help in performance?
- Where is Java used today beyond computers?

Evaluation

Strategy: One Minute Paper

- Ask: "What part of Java did you find most interesting today?"

Quiz / Worksheet

- Multiple choice, fill in the blanks, case study questions (from book).

Student Activity: Picture Labelling.

- Students label compilation diagram and explain roles of javac and JVM.

Strategy: Application Cards

- Students write down where they think Java is applied in daily life.

Suggested Activity

Strategy: Jigsaw

- **Expert Groups:** History, Compilation Process, Features, IDE (BlueJ).
- After discussion, mix and teach peers in new groups.
- Summarise learning through a class display or peer quiz.

3 Elementary Concept of Classes and Objects

Teaching Objectives

By the end of this lesson, students will be able to:

- Understand the concept of classes and objects through real-life analogies.
- Define and explain class structure and object instantiation in Java.
- Identify the components of a class: data members and methods.
- Create and use Java objects using BlueJ.
- Distinguish between classes and objects based on properties and memory.

Number of Periods	
Theory	Practical
3	2

Teaching Plan

Introduction

- **Strategy:** Think-Pair-Share
  - Ask: "What do you think a car, a mobile phone, and a student have in common?"
  - Let students pair up and think of how these relate to programming.
- Show visuals of a car as a class and different models as objects.
- Introduce terms like blueprint, instance, attribute, and method using household examples.

Lesson Delivery

1. **Concept of Classes and Objects**
   - Define Class as a blueprint with characteristics (attributes) and behaviour (methods).
   - Explain Object as an instance of a class with real values.

     Strategy: Real-Life Analogy + Picture Study.
   - Use examples from the book: student and car classes with object values.
   - Students describe attributes and methods for each object in groups.
2. **Class in Java**
   - Show Java syntax: accessSpecifier class ClassName { ... }
   - Components: access specifier, data members, methods, constructor (if any).

     Strategy: Fussing with Definitions.
   - Students define components (e.g., "method") in their own words, share, and refine.
3. **Object in Java**
   - Syntax to create an object: ClassName obj = new ClassName();
   - Explain instantiation and constructor (default vs user-defined).

     Strategy: Guided Practice + Code Execution
   - Students write and execute simple Java class and object using BlueJ.
   - Example: Circle class with radius and area methods.
4. **Message Passing Between Objects**
   - Concept of dot operator (obj.method()).
   - Demonstrate using example: Car and Driver class (start, stop)

     Strategy: Role Play.
   - Two students act as objects; one sends a method call message to the other.

5. **Properties of Class and Object**
   - Class is blueprint (logical, no memory until object is created).
   - Object is physical (exists in memory after instantiation).
   - Class is a user-defined data type.

     Strategy: Application Cards.
   - Students write how a class can be used as a blueprint in real systems (e.g., Inventory).
6. **Difference between Class and Object**
   - Tabular comparison: logical vs physical, memory allocation, usage.

     Strategy: Jigsaw
   - E**xpert groups:** Class definition, Object creation, Object properties, Real-world examples.
   - Groups teach peers and summarise key differences.

Extension

Strategy: RAFT Writing

- **Role:** Java Object.
- **Audience:** New Java Programmer.
- **Format:** Short letter.
- **Topic:** "Why I need my class to exist".

Discussion Prompts

- Why does object need to be created from a class?
- Can we run code without creating objects?

Evaluation

Strategy: One Sentence Summary

- Ask: "What is the key difference between a class and an object?"

Worksheet

- Multiple choice, fill in the blanks, and code analysis based on textbook exercises.

Case Study Discussion

- Use examples from book like Product class or Course management system.

Suggested Activity

Strategy: Role Play + Simulation

- Create "object cards" with different attributes (e.g., Student with name, roll).
- Assign students to act as objects of a class and perform method-based tasks in sequence.

4 Values and Types

Teaching Objectives

By the end of this lesson, students will be able to:

- Identify the character set used in Java and differentiate between ASCII and Unicode.
- Understand escape sequences and their usage in Java.
- Define and categorise Java tokens, keywords, identifies, and literals.
- Understand data types in Java and their classification.
- Define and declare variables using static and dynamic initialisation.
- Explain and demonstrate type conversion in Java (implicit and explicit).

Number of Periods	
Theory	Practical
4	2

Teaching Plan

Introduction

- **Strategy:** Think-Pair-Share
  - Ask: "What kinds of characters do you think a computer understands?"
  - Share ASCII and Unicode examples with common keyboard input.
- Display code snippets with escape sequences and observe their output.

Lesson Delivery

1. **Character Sets in Java**
   - Explain alphabets, digits, operators, and delimiters.
   - Highlight ASCII and Unicode with examples and code.

     Strategy: Picture Study.
   - Use table and code from book for ASCII and Unicode display programs.
   - Discuss difference between the two standards.
2. **Escape Sequences**
   - Define escape sequences with meaning (e.g., \t, \n, " etc.).
   - Show sample programs for each sequence and their output.

     Strategy: Guided Demonstration + Role Play.
   - Assign escape sequences to students and ask them to act out their effects (e.g., tab, new line).
3. **Java Tokens**
   - Define tokens as smallest meaningful element in a Java program.

- **Types:** keywords, identifiers, literals, operators, special symbols.

  Strategy: Round Robin.
- Students name as many keywords or tokens as they can in sequence.

4. **Identifiers and Literals**
   - Define rules and conventions for naming identifiers.
   - Classify and explain different types of literals (e.g., int, float, char, string, boolean, null).

Strategy: RAFT Writing

- **Role:** Literal.
- **Audience:** Beginner Java Programmer.
- **Format:** Tweet.
- **Topic:** "Why I matter in a program".

5. **Operators, Punctuators and Separators**
   - Define and demonstrate their roles using examples.
   - Classify operators: arithmetic, logical, relational, assignment, unary.
6. **Java Data Types**
   - Distinguish between primitive and non-primitive data types.
   - Explain int, float, char, boolean, long etc. with memory size and default values.

     Strategy: Application Cards.
   - Students write example use of each data type in real-world context.
7. **Variables and Initialisation**
   - Define variable and difference between identifier and variable.
   - Explain static and dynamic initialisation with syntax.

     Strategy: Jigsaw.
   - **Groups:** static initialisation, dynamic initialisation, declaration vs assignment.
8. **Type Conversion**
   - Define type conversion and its syntax: datatype var = (datatype)value.
   - Discuss implicit (widening) and explicit (narrowing) conversions.

     Strategy: Role Play + Debugging Exercise.
   - Provide code snippets with and without proper conversion.
   - Ask students to correct the errors and explain why.

Extension

Strategy: One Sentence Summary

- **Ask:** "What is the key idea behind type conversion?"

Discussion Prompts

- Why are Unicode and escape sequences important in Java?
- How do identifiers impact the readability of a program?

Evaluation

Quiz

- Use MCQs, fill in the blanks, true/false and match the column from the textbook.

Worksheet Activity

- Label data types, categorise tokens, complete variable declarations.

Code Activity

- Write Java program with all primitive data types and literals, escape sequences and conversions.

Suggested Activity

Strategy: Case Study + RAFT Presentation

- Provide real-world Java scenario (e.g., calculator, student marks system).
- Students write short role-based summaries for keyword, variable, or token used in program.

5 Operators in Java

Teaching Objectives

By the end of this lesson, students will be able to:

- Define and identify operators and operands in Java expressions.
- Understand the forms (unary, binary, ternary) and types of operators in Java.
- Explain the usage of arithmetic, relational, logical, assignment, conditional, bitwise, and special operators.
- Demonstrate the concept of hierarchy and associativity of operators through examples.
- Differentiate between counters and accumulators and write programs using both.
- Understand the purpose and syntax of output statements in Java.

Number of Periods	
Theory	Practical
4	2

Teaching Plan

Introduction

- **Strategy:** Brainstorming
  - Ask: "What symbols have you seen in maths or Java like +, -, =, *? What do they do?"
  - Gather responses and display on board as 'Operators'.
- Relate to real-life expressions like shopping bill, profit-loss, comparisons, etc.

Lesson Delivery

1. **Operators and Expressions**
   - **Define:** Operand, Operator, Expression, Arithmetic Statement.
   - Examples and explanation of c = a + b;

     Strategy: Picture Study + Guided Demonstration.
   - Show breakdown of code: operand, operator, expression types.
2. **Forms of Operators**
   - Unary, Binary, Ternary (with examples: a++, a+b, (a>b)?a:b).

     Strategy: Role Play.
   - Students enact operand + operator scenes to simulate operations.
3. **Arithmetic Operators**
   - Explain symbols: +, -, *, /, %, ++, --, unary +/−.
   - Demonstrate code snippets for each.

     Strategy: Coding Lab Activity + Pair Debugging.
   - Students execute given arithmetic programs and correct errors.
4. **Counters and Accumulators**
   - Define both with example loops.
   - Show programs for prime number (counter) and sum (accumulator).

     Strategy: Jigsaw.
   - Expert groups study programs using counter and accumulator and teach peers.
5. **Relational & Logical Operators**
   - **Relational:** >, <, >=, <=, ==, !=.
   - **Logical:** &&, ||, !.

     Strategy: Think-Share.
   - Give real-world conditions and ask students to simulate expressions.

6. **Assignment & Shorthand Operators**
   - Explain =, +=, -=, *=, etc. with coding examples.
   - Use visual flow of shorthand expansion.
7. **Conditional Operator (Ternary)**
   - Syntax and usage (condition)?truePart:falsePart.
   - Rewrite if-else code as ternary.
8. **Bitwise Operators**
   - Explain AND, OR, XOR, NOT, Shift operations with binary visualisation.

     Strategy: Application Cards.
   - Students give use cases of where bitwise operators might be useful.
9. **Special Operators: new and dot (.)**
   - Explain with example: creating object, accessing members.
   - Use System.out.println() breakdown.
10. **Hierarchy and Associativity of Operators**
    - Explain precedence and associativity rules.
    - Provide solved examples from book and work them out together.

      Strategy: Debugging Challenge.
    - Present faulty expression, students correct using precedence and brackets.
11. **Output Statements**
    - Explain System.out.print() vs System.out.println().
    - Importance of semicolon, syntax rules.

Extension

Strategy: RAFT Writing

- **Role:** Unary Operator.
- **Audience:** Classmate.
- **Format:** Letter.
- **Topic:** "Why I work alone but make a big difference".

Discussion Prompts

- Why are shorthand operators useful in loops?
- How does Java evaluate complex expressions with many operators?
- When is the conditional operator better than if-else?

Evaluation

Strategy: One Minute Paper

- Ask: "What operator did you find hardest and why?"

Quiz & Worksheet

- MCQs, Fill in the blanks, Expressions rewriting, Output prediction from book.

Practical Coding Tasks

- Create Java programs using all three forms of operators.
- Solve problems using counters and accumulators.

Suggested Activity

Strategy: Case Study + Role Simulation

- Create scenario: Website visit counter & sales tracker.
- Students develop logic using variables, operators, output statements.

6 – Input and Exception Handling in Java

Teaching Objectives

By the end of this lesson, students will be able to:

- Understand the concept and types of comments in Java.
- Define and use packages in Java, including import statements.
- Identify and use different types of input methods: initialisation, parameters, Scanner class, and InputStreamReader class.
- Distinguish between different types of errors: syntax, logical, and runtime errors.
- Explain the purpose of testing, debugging, and exception handling.
- Apply the try-catch block for handling exceptions in Java.

Number of Periods	
Theory	Practical
4	2

Teaching Plan

Introduction

- **Strategy:** Think-Pair-Share
  - Ask: "Have you ever written notes in your book to remember something? Why do you think programmers do the same in code?"
  - Introduce the idea of comments in programming.
- Discuss how programs interact with users via input and handle unexpected inputs.

Lesson Delivery

1. **Comments in Java**
   - **Define:** Single-line (//), Multi-line (/* */), and Documentation (/** */) comments.

     Strategy: Coding Exercise + Highlight & Explain.
   - Students add all three types of comments in a simple Java program.
2. **Packages in Java**
   - Define package, built-in packages, and sub-packages.
   - Demonstrate the use of import keyword with Calendar class example.

     Strategy: Visual Demonstration + Syntax Lab
   - Students import different classes from java.util and use them in programs.
3. **Input in Java**
   - Initialisation: Direct input in code.
   - Parameters: Input via command line or method arguments.
   - Scanner class: Various nextXXX() methods (int, double, float, etc.)
   - InputStreamReader class: readLine() with type conversion.

     Strategy: Jigsaw
   - Expert groups work on one method of input and demonstrate to peers.
   - Examples: Swapping, area of circle, name/roll input, etc.
4. **Errors in Java**
   - Syntax, Logical, and Runtime errors.

     Strategy: Debugging Challenge + Case Study.
   - Students are given faulty programs and asked to find and correct errors.
5. **Testing and Debugging**
   - Define testing and debugging.
   - Emphasise importance of verifying and validating code.

     Strategy: Pair & Share
   - Students review each other's programs and test/debug collaboratively.
6. **Exception Handling in Java**
   - Define exception, try-catch block, throw and throws keywords.
   - Syntax and application using input programs.

Strategy: RAFT Writing.

- **Role:** Java Exception

- **Audience:** New programmer.
- **Format:** Note
- **Topic:** "Why I disrupt your code, and how you can handle me".

Extension

Discussion Prompts

- What happens if you divide by zero in Java?
- Why do we need exception handling if we can check for errors?
- How are Scanner and InputStreamReader different in how they work and are used?

Strategy: Role Play

- Act out roles of input sender, receiver, and error handler using everyday analogies (e.g., message sent, misunderstood, fixed).

Evaluation

Strategy: One Minute Paper

- Ask: "Which input method did you find most useful and why?"

Worksheet / MCQ / Coding Questions

- Identify type of comment, package syntax, input methods used.
- Classify errors and fill in blanks.
- Implement Java program using Scanner/InputStreamReader with try-catch block.

Suggested Activity

Strategy: Scenario-based Simulation

- Simulate a real-time system like a railway ticket counter where users give inputs, and errors occur if input is wrong
- Students build logic to handle input and exceptions in code

7 Mathematical Library Methods in Java

Teaching Objectives

By the end of this lesson, students will be able to:

- Understand the difference between user-defined methods and built-in (library) methods.
- Identify and use various methods from the Math class in Java.
- Use Java syntax for mathematical methods such as Math.pow(), Math.sqrt(), Math.abs(), etc.

- Apply mathematical methods to solve real-life programming problems.
- Write Java programs demonstrating use of trigonometric and exponential methods.

Number of Periods	
Theory	Practical
4	2

Teaching Plan

Introduction

- **Strategy:** Think-Pair-Share
  - Ask: "Have you used a calculator to find square roots or powers? How would you do that in Java?"
  - Discuss similarities between calculator functions and Java Math methods.
- Introduce the java.lang package and the concept of Math class with built-in methods.

Lesson Delivery

1. **User-defined vs Library Methods**
   - Define and differentiate user-defined and built-in methods.
   - Example: A method to calculate area vs using Math.sqrt().
2. **The Math Class and Syntax**
   - Format: Math.functionName(arguments);
   - Explain automatic import of java.lang package.

     Strategy: Visual Guide + Guided Syntax Writing.
   - Students copy example syntax and label parts: class, method, arguments, return type.
3. **Common Mathematical Methods in Java**
   - min(), max(), sqrt(), cbrt(), pow(), abs(), round(), ceil(), floor(), rint(), log(), exp(), random().
   - Explain usage, return types, examples and expected outputs.

     Strategy: Coding Lab with Pair Debugging.
   - Provide half-written code with missing method calls for students to complete.
4. **Real-life Program Examples**
   - Kinetic energy calculation using Math.pow().
   - Equilateral triangle area using Math.sqrt().
   - Compound interest with Math.pow().

     Strategy: Jigsaw (Strategy #13).
   - Each group explains one program and how the Math method works.
5. **Trigonometric Methods**
   - Math.sin(), Math.cos(), Math.tan().

- Explain need to convert degrees to radians using formula: radian = (Math.PI * degree)/180.

  Strategy: Simulation + Application Cards (Strategy #27).
- Use real scenarios like calculating slopes, waves, oscillations.
- Students describe where these functions might be used in real-world scenarios.

Extension

Strategy: RAFT Writing

- **Role:** Math.pow()
- **Audience:** New Programmer.
- **Format:** Note.
- **Topic:** "Why I'm your power tool in Java!"

Discussion Prompts

- When should you use Math.ceil() vs Math.round()?
- Why are values returned by Math.random() different every time?
- What is the difference between floor and rint?

Evaluation

Strategy: One Minute Paper

- Ask: "Which Math method did you find most useful today and why?"

Quiz / Worksheet

- MCQs on method outputs, syntax errors.
- Fill in the blanks (e.g. Math.ceil(3.2) = ?).
- Java expression writing (e.g. \u221a(a^2 + b^2) as Math.sqrt(Math.pow(a,2) + Math.pow(b,2))).

Code Exercises

- Write programs using min, max, pow, sqrt, floor, ceil.
- Trigonometric applications using radians.

Suggested Activity

Strategy: Case Study + Role Simulation

- Students are given a real-world scenario (e.g. physics lab simulation).
- They choose relevant Math methods and design Java code to compute the outputs.

8 Conditional Constructs in Java

Teaching Objectives

By the end of this lesson, students will be able to:

- Understand the concept of flow of control in Java and its types.
- Apply various conditional statements like if, if-else, if-else-if ladder, nested if, and switch.
- Use ternary operator as a compact alternative to simple if-else.
- Distinguish between if-else and switch statements.
- Terminate Java programs using System.exit().
- Write real-world Java programs using different types of conditional constructs

Number of Periods	
Theory	Practical
5	3

Teaching Plan

Introduction

- **Strategy:** Think-Pair-Share
  - Ask: "Do all instructions in a program always run one after another? What if we want to make a decision in the middle?"
  - Discuss basic idea of control flow in real life (e.g., road traffic signals, menu-based decisions).

Lesson Delivery

1. **Flow of Control in Java**
   - Types: Normal, Conditional, and Multiple Branching.
   - Demonstrate sequential flow using a simple sum and average program.
2. **if Statement**
   - Syntax and one-line vs block.
   - **Example:** Checking if a number is positive.

     Strategy: Role Play.
   - Students act as a decision tree – one condition, one path.
3. **if-else Statement**
   - Syntax with dual branches.
   - Example: Voter eligibility check.
4. **Multiple if Statements**
   - Used when all conditions may need to be checked.
   - **Example:** Displaying numbers in alphabets for 1-5.

Strategy: Jigsaw (Strategy #13)

- Each group explains one type of if construct with examples

5. **if-else-if Ladder**
   - Used when one among many conditions must be satisfied
   - Example: Grade evaluation program
6. **Ternary Operator**
   - Compact form of simple if-else
   - Syntax: result = (condition) ? value1 : value2;
   - Example: Pass or Fail based on marks

Strategy: RAFT Writing

- **Role:** Ternary operator.
- **Audience:** Beginner.
- **Format:** Postcard.
- **Topic:** "I am the shortcut decision-maker".

7. Nested if Statement
   - When one condition is embedded within another.
   - Example: Checking range (greater than 40, then 80).
8. Dangling else
   - Explain ambiguity of unmatched else without proper braces.
9. switch Statement
   - Used for menu-based choices or exact value matching.
   - Syntax and flow diagram explanation.

     Strategy: Simulation Activity.
   - Students input choices for menu-driven programs like Grade messages, SI/CI calculator, etc.
10. Difference Between if and switch
    - Present a comparison chart:
      - Condition types.
      - Supported data types.
      - Use cases.
11. System.exit()
    - Use of System.exit(0) and System.exit(1) for normal/abnormal termination.

Extension

Strategy: Case-Based Simulation

- Real-world problems:
  - Income tax calculator
  - Stream selection based on marks
  - Electricity bill calculator
  - Admission eligibility check

Discussion Prompts

- When should we use if-else over switch?
- Why might nested ifs lead to logical errors?
- What happens if break is missing in a switch?

Evaluation

Strategy: One Minute Paper

- Ask: "Which conditional construct is most useful and why?"

Quiz & Worksheet

- MCQs on flow types, syntax, output prediction
- Fill in the blanks and true/false from book exercises
- Rewrite if as ternary and switch as if-else

Coding Practice

- Create Java programs using all three forms of operators.
- Solve problems using counters and accumulators.

Suggested Activity

Strategy: Debugging Challenge + Role Simulation

- Provide error-based conditional logic for correction.
- Students simulate different cases using decision trees.

8 Conditional Constructs in Java

Teaching Objectives

By the end of this lesson, students will be able to:

- Understand the concept of flow of control in Java and its types.

- Apply various conditional statements like if, if-else, if-else-if ladder, nested if, and switch.
- Use ternary operator as a compact alternative to simple if-else.
- Distinguish between if-else and switch statements.
- Terminate Java programs using System.exit().
- Write real-world Java programs using different types of conditional constructs

Number of Periods	
Theory	Practical
5	3

Teaching Plan

Introduction

- **Strategy:** Think-Pair-Share
  - Ask: "Do all instructions in a program always run one after another? What if we want to make a decision in the middle?"
  - Discuss basic idea of control flow in real life (e.g., road traffic signals, menu-based decisions).

Lesson Delivery

1. **Flow of Control in Java**
   - Types: Normal, Conditional, and Multiple Branching.
   - Demonstrate sequential flow using a simple sum and average program.
2. **if Statement**
   - Syntax and one-line vs block.
   - **Example:** Checking if a number is positive.

     Strategy: Role Play.
   - Students act as a decision tree – one condition, one path.
3. **if-else Statement**
   - Syntax with dual branches.
   - Example: Voter eligibility check.
4. **Multiple if Statements**
   - Used when all conditions may need to be checked.
   - **Example:** Displaying numbers in alphabets for 1-5.

     Strategy: Jigsaw (Strategy #13)
   - Each group explains one type of if construct with examples
5. **if-else-if Ladder**
   - Used when one among many conditions must be satisfied
   - Example: Grade evaluation program

6. **Ternary Operator**
    - Compact form of simple if-else
    - Syntax: result = (condition) ? value1 : value2;
    - Example: Pass or Fail based on marks

Strategy: RAFT Writing

- **Role:** Ternary operator.
- **Audience:** Beginner.
- **Format:** Postcard.
- **Topic:** "I am the shortcut decision-maker".

7. **Nested if Statement**
    - When one condition is embedded within another.
    - Example: Checking range (greater than 40, then 80).
8. **Dangling else**
    - Explain ambiguity of unmatched else without proper braces.
9. **switch Statement**
    - Used for menu-based choices or exact value matching.
    - Syntax and flow diagram explanation.

      Strategy: Simulation Activity.
    - Students input choices for menu-driven programs like Grade messages, SI/CI calculator, etc.
10. **Difference Between if and switch**
    - Present a comparison chart:
        - Condition types.
        - Supported data types.
        - Use cases.
11. **System.exit()**
    - Use of System.exit(0) and System.exit(1) for normal/abnormal termination.

Extension

Strategy: Case-Based Simulation

- Real-world problems:
    - Income tax calculator
    - Stream selection based on marks

- Electricity bill calculator
- Admission eligibility check

Discussion Prompts

- When should we use if-else over switch?
- Why might nested ifs lead to logical errors?
- What happens if break is missing in a switch?

Evaluation

Strategy: One Minute Paper

- Ask: "Which conditional construct is most useful and why?"

Quiz & Worksheet

- MCQs on flow types, syntax, output prediction
- Fill in the blanks and true/false from book exercises
- Rewrite if as ternary and switch as if-else

Coding Practice

- Create Java programs using all three forms of operators.
- Solve problems using counters and accumulators.

Suggested Activity

Strategy: Debugging Challenge + Role Simulation

- Provide error-based conditional logic for correction.
- Students simulate different cases using decision trees.

9 Iterative Constructs in Java

Teaching Objectives

By the end of this lesson, students will be able to:

- Understand the purpose and structure of loops in Java.
- Differentiate between entry-controlled and exit-controlled loops.
- Identify and implement different loop types: for, while, and do-while.
- Understand and create null, infinite, finite, and step loops.
- Interconvert between different types of loops.
- Understand and use jump statements: break, continue, and return.
- Apply looping concepts in real-world Java programs.

Number of Periods	
Theory	Practical
5	3

Teaching Plan

Introduction

- **Strategy:** Think-Pair-Share
  - Ask: "If you wanted the computer to count from 1 to 100, how would you do it without writing 100 print statements?"
  - Show a basic for loop on screen and explain how it saves effort.

Lesson Delivery

1. **Introduction to Loops**
   - Define loop, iteration, and purpose
   - Parts of a loop: Initialisation, Condition, Increment/Decrement, Body
2. **Categories of Loops**
   - Entry Control Loop (condition before body): for, while.
   - Exit Control Loop (condition after body): do-while.
3. **Types of Loops in Java**
   a. **for Loop**
      - Syntax and example: print numbers 1 to 10.
      - Dry run explanation.

   b. **while Loop**
      - Used when number of iterations is not known.
      - **Example:** sum of digits.

   c. **do-while Loop**
      - Executes at least once.
      - **Example:** print sum of first 5 numbers.

        Strategy: Role Play + Flowchart Walkthrough.
      - Students act as control points in flow of a loop (init, condition, body, increment).
4. **Infinite, Null and Finite Loops**
   - Infinite Loop: no termination (e.g., while(true), for(;;)).
   - Null Loop: loop with empty body (delay or skip).
   - Finite Loop: terminates after condition becomes false.
5. **Step Loop and Continuous Loop**
   - **Step:** Increment/decrement by >1 (e.g., i+=5).

- Continuous: i++ or i--.

6. **Loop Interconversion**
   - Convert for ⟷ while ⟷ do-while.

     Strategy: Jigsaw (Strategy #13).
   - Each group rewrites the same logic using a different loop type and presents.
7. **Jump Statements**
   - break: exit the loop immediately.
   - continue: skip current iteration.
   - return: exit from method.

     Strategy: Debugging Challenge.
   - Students fix logical errors in loop code with incorrect jump statements.

Extension

Strategy: Case-Based Problem Solving

- Use of loops in:
  - Factorial program
  - Fibonacci series
  - Prime number check
  - Armstrong and Perfect numbers
  - Table generation

Discussion Prompts

- When is do-while a better option?
- How can we avoid infinite loops?

Evaluation

Strategy: One Sentence Summary

- "How would you explain a 'for' loop to someone new to programming?"

Quiz & Worksheet

- MCQs, Fill in the blanks, Loop tracing questions, Output prediction.
- Convert one type of loop to another.

Coding Practice

- Programs using for, while, do-while, null loop, infinite loop
- Use of break, continue, and return

Suggested Activity

Strategy: Simulation + Menu-Driven Program Building.

- Students build a Java menu-based program using switch and loops for series generation, sum/ product calculation, etc.

10 Nested Loops in Java

Teaching Objectives

By the end of this lesson, students will be able to:

- Understand the structure and concept of nested loops in Java.
- Differentiate between nested for, while, and do-while loops.
- Write Java programs using nested loops for various patterns and number series.
- Apply break and continue statements within nested loops.
- Analyse and debug nested loop logic through dry runs.

Number of Periods	
Theory	Practical
5	3

Teaching Plan

Introduction

- **Strategy:** Think-Pair-Share
  - Ask: Have you ever used one loop inside another in your program? Why might we need to repeat things within repeated actions?"
  - Show real-life analogy: Rows and columns in a table.

Lesson Delivery

1. **Introduction to Nested Loops**
   - Define nested loop and explain inner vs outer loop.
   - Syntax and logic flow for all types.
2. **Nested for Loop**
   - Syntax demonstration.
   - Example: factorial of even numbers between 1 to 5.
3. **Nested while Loop**
   - Syntax and working.
   - Example: print prime numbers between 3 and 6.

4. **Nested do-while Loop**
   - Syntax walkthrough.
   - Example: factors of numbers from 2 to 5.

     Strategy: Guided Coding + Dry Run Table Creation.
   - Students code and manually trace nested loop execution using dry run table.
5. **Using break in Nested Loops**
   - Use case in outer loop and inner loop.
   - Example: stop loop on condition match.
6. **Using continue in Nested Loops**
   - Skip current iteration in inner or outer loop.
   - Example programs to demonstrate difference.
7. **Pattern Programming Using Nested Loops**
   - Number and character patterns.
   - Floyd's triangle, Pascal's triangle, and various other shapes.

     Strategy: Jigsaw (Strategy #13).
   - Assign one pattern to each group; they explain logic and share dry run.
8. **Dry Run and Debugging**
   - Students create dry run table for factorial, prime, and pattern programs.

Extension

Strategy: Role Simulation + Pattern Workshop

- Students become loops and simulate iterations for chosen patterns

Discussion Prompts

- What happens when break is used in outer vs inner loop?
- How is continue different in nested structure?
- What types of problems are solved better with nested loops?

Evaluation

Strategy: One Sentence Summary

- Ask: "How do nested loops help in pattern generation?"

Quiz & Worksheet

- Tick the correct answer, fill in the blanks, pattern tracing, and output prediction from the textbook.
- Rewrite if as ternary and switch as if-else

Coding Practice

- Nested loops for:
  - Factorial generation
  - Prime number checking
  - Strong number check
  - Patterns using stars, numbers, and characters

Suggested Activity

Strategy: Case-Based Implementation

- **Real-world simulation:** Table calculator (print table of 1 to 5), character and pyramid patterns, triangle of symbols.
- Student groups design and code one real-world pattern with nested loop logic.

11 Computing and Ethics

Teaching Objectives

By the end of this lesson, students will be able to:

- Understand the concept of ethics and its significance in everyday life and in the digital world.
- Recognise different types of intellectual property and associated rights.
- Identify ethical and unethical behaviour related to software, privacy, and online communication.
- Explain cybercrimes and learn ways to protect against them.
- Demonstrate good etiquette and responsible behaviour in digital environments.

Number of Periods	
Theory	Practical
5	2

Teaching Plan

Introduction

- **Strategy:** Think-Pair-Share
  - Ask: "What does it mean to do the right thing online?"
  - Students share examples of online ethical or unethical behaviour.
- Connect discussion to the idea of ethics as a moral compass in the digital age.

Lesson Delivery

1. **Introduction to Nested Loops**
   - Meaning, origin (Greek ethos), role in life and technology.
   - Real-life examples of ethical decisions.

2. Intellectual Property & Rights
    - **Definition and types:** patents, trademarks, copyrights, trade secrets.
    - **Examples:** software, music, art, scientific discoveries.

Strategy: RAFT Writing

- **Role:** Copyright
- **Audience:** New coder
- **Format:** Diary entry
- **Topic:** "How I protect your creation"

3. Data Privacy and Data Protection

- Meaning and examples of breaches (e.g., social media misuse, password theft).
- Ways to protect data (firewalls, antivirus, safe browsing, backups).

  Strategy: Case Study Discussion.
- Discuss a real-life example of data breach and analyse what could have prevented it.

4. Spam and Software Piracy

- What is spam? Why it is dangerous? How to identify it?
- Types of software piracy: counterfeiting, internet piracy, hard disk loading,

5. Cybercrime and Hacking

- Define cybercrime and list types: phishing, cyberstalking, cyber abuse, grooming, cyberwarfare.
- What is hacking and why it is done?

  Strategy: Simulation Activity.
- Students role-play a cybercrime scenario and present steps to report and prevent it.

6. Malicious Code

- Meaning and examples: viruses, worms, trojans
- Safety measures: antivirus, VPN, firewalls, avoid pop-ups

7. Good Etiquette Practices (Netiquette)
    - Respectful communication online.
    - Proper use of language and emoticons.
    - Avoiding spam, misinformation, and inflammatory comments.

8. Ten Commandments of Computer Ethics

- Overview and classroom discussion on each principle.

Extension

Strategy: Jigsaw

- Divide students into groups to research and present:
  - Intellectual Property
  - Cybercrime
  - Online Privacy
  - Netiquette

Discussion Prompts

- Why are ethics more important in the age of the internet?
- How can you ensure you are an ethical digital citizen?

Evaluation

Strategy: One Minute Paper

- Ask: "What is one ethical rule you will follow while using the internet?"

Quiz & Worksheet

- MCQs, Fill in the blanks, definitions, short and long answers based on textbook.

Coding Practice

- Create a poster on "Dos and Don'ts of Digital Ethics".

Suggested Activity

Strategy: Poster Making + Ethical Debate

- Poster creation on cyber safety, followed by classroom debate on controversial ethical situations in IT.