# TOUCHPAD

ICSE Computer Applications-X (Ver.2.0)

(With BlueJ)

10

# 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<br>• Shows mature throwing and catching patterns<br>• Writing is now smaller and more readable<br>• Drawings are now more detailed, organised and have a sense of depth |
| Cognitive | • Attention continues to improve, becomes more selective and adaptable<br>• Recall, scripted memory, and auto-biographical memory improves<br>• Counts on and counts down, engaging in simple addition and subtraction<br>• Thoughts are now more logical |
| Language | • Vocabulary reaches about 10,000 words<br>• Vocabulary increases rapidly throughout middle childhood |
| Emotional/ Social | • Ability to predict and interpret emotional reactions of others enhances<br>• Relies more on language to express empathy<br>• Self-conscious emotions of pride and guilt are governed by personal responsibility<br>• Attends to facial and situational cues in interpreting another's feelings<br>• 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<br>• Cognitive self-regulation is now improved |
| Language | • Ability to use complex grammatical constructions enhances<br>• Conversational strategies are now more refined |
| Emotional/ Social | • Self-esteem tends to rise<br>• Peer groups emerge |

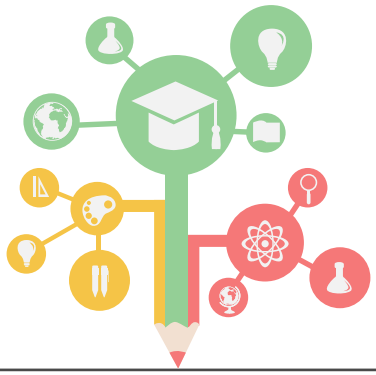
## Age 11 - 20 Years

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

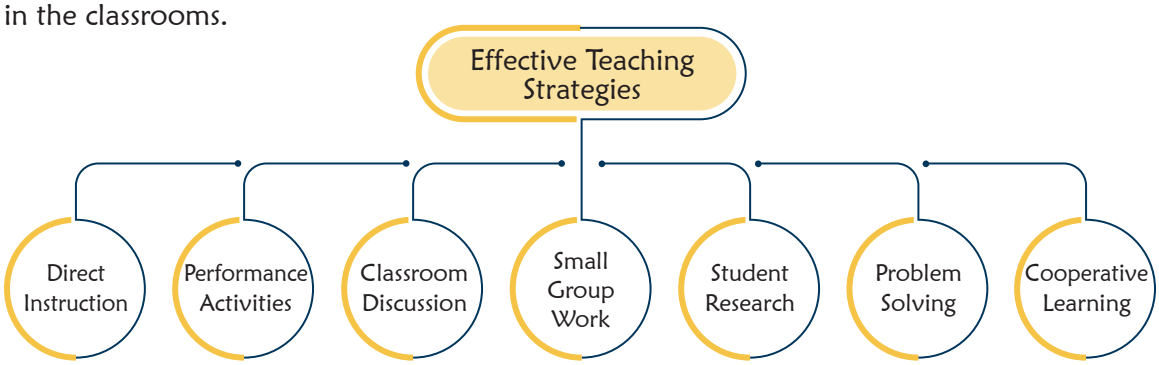
**After the class**

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

> "Knowing yourself is the beginning of all wisdom."

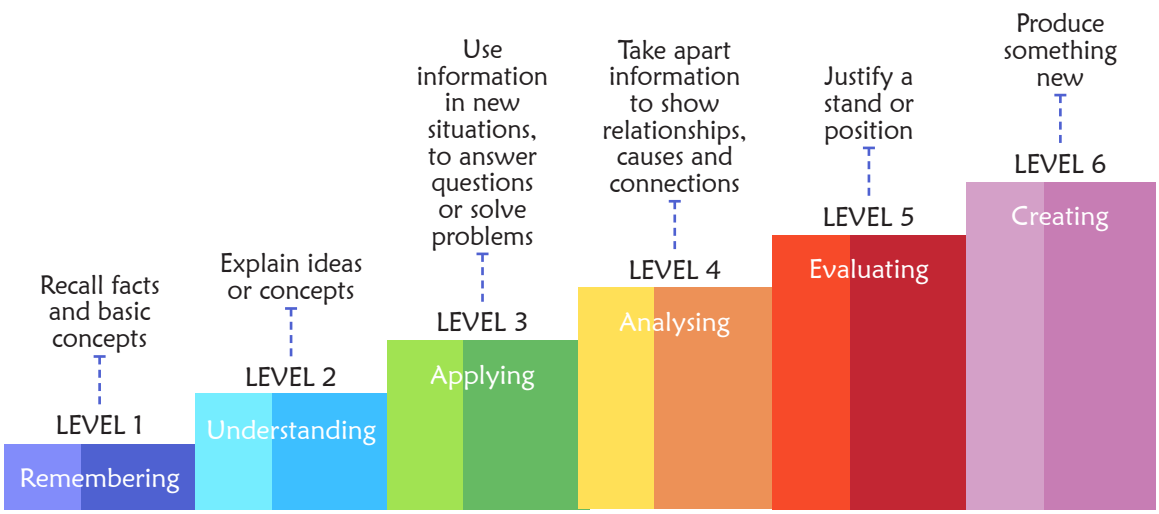
## Teaching Strategies

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 10

# LESSON PLAN

## 1 Introduction to Object-Oriented Programming Concepts

### Teaching Objectives

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

- Define and differentiate between low-level and high-level computer languages.
- Understand programming paradigms: POP vs OOP.
- Explain the core principles of Object-Oriented Programming: Encapsulation, Abstraction, Inheritance, and Polymorphism.
- Describe key terms such as Program, Programming, Programmer, and Object.
- Solve textbook-based questions (MCQs, fill-ups, short/long answers).
- Apply knowledge to case studies and real-life programming design.

| Number of Periods | |
|---|---|
| Theory | Practical |
| 5 | 2 |

### Teaching Plan

**Introduction (Engagement)**

**Strategy:** Think-Pair-Share

- Ask: "What language do humans use to communicate with computers?"
- Use analogy of different human languages vs programming languages.
- Display visuals contrasting low-level code (binary) with Java (high-level).

**Lesson Delivery (Explanation & Demonstration)**

1. **Computer Language**
   - Define: A set of rules and symbols used to write programs.
   - Types: Low-Level Language (machine-dependent) and High-Level Language (machine-independent)
   - Examples: Assembly, Java, Python, C++
2. **Programming Paradigms**
   - POP (Procedure-Oriented Programming)
     - Top-down approach, global data, functions

- Examples: BASIC, FORTRAN, COBOL
- OOP (Object-Oriented Programming)
  - Bottom-up approach, object-based, data-centric
  - Examples: Java, C++, Python

**3. POP vs OOP Comparison**

- Tabular format (as in book)
- Focus on approach, security, modularity, and object usage

**4. Principles of OOP**

a. Encapsulation
- Bundling data and methods
- Real-life analogy: Capsule

b. Abstraction
- Showing only essential details
- Example: Car dashboard operation

c. Inheritance
- One class (child) inherits properties of another (parent)
- Example: Vertebrates → Fish, Mammals

d. Polymorphism
- One name, multiple forms
- Example: A man as father, employee, customer

Strategy: RAFT Writing
- Role: Inheritance
- Audience: Student coder
- Format: Message
- Topic: "I make your programs reusable!"

5. Key Definitions
- Program: A set of statements to complete a task
- Programming: Writing instructions using a language
- Programmer: Person who creates and tests code

**Extension (Further Exploration)**

**Strategy:** Jigsaw

- Divide class into groups to explore each principle of OOP through visuals, examples, and real-world analogies

**Discussion Prompts:**

- Which OOP principle offers better data security?

- Why is reusability important in large applications?

**Evaluation (Assessment & Review)**

**Strategy:** One Minute Paper

- Ask: "Which OOP principle is hardest to understand and why?"

**Quiz & Worksheet:** Based on textbook

- MCQs (e.g., Q1–20 from book)
- Fill in the blanks (Section B)
- Short Answer Questions (e.g., definition-based Qs)
- Long Answer Questions (e.g., POP vs OOP comparison, OOP principles explanation)
- Assertion and Reasoning (Section D)

**Suggested Activity**

**Strategy:** Concept Mapping + Case Study

- Task: Create a class structure for "Library System" using encapsulation, inheritance, etc.
- Role-play object behaviours like Book, Member, Librarian

# 2 Elementary Concepts of Objects and Classes

## 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 new and constructors.
- Understand how a class acts as an object factory.
- Explain the this keyword and properties of objects in memory.

| Number of Periods | |
|---|---|
| Theory | Practical |
| 4 | 2 |

## Teaching Plan

**Introduction (Engagement)**

**Strategy:** Think-Pair-Share

- Ask: "If you wanted to define 'a book' in a program, what things would you describe?"
- Show real-world analogies: A class as a blueprint, an object as the book itself.

**Lesson Delivery (Explanation & Demonstration)**

1. **Concept of Class and Object**
   - Real-life examples: COMPUTER class and types of computers

- Characteristics vs Behaviours

**2. Class in Java**

- Definition: A user-defined data type and object factory
- Syntax: class Square { int side; void area(); }
- Key components: Data members (attributes), Methods (behaviours)

**3. Object in Java**

- Object = instance of a class
- Syntax: Square sq = new Square();
- Constructor role: initialising data

**Strategy:** Guided Coding + Dry Run Table Creation

- Students practice instantiating objects and calling class methods

4. Properties of Class and Object

- Class as Object Factory: Creates similar objects
- Object as Instance: Allocates memory and stores object state
- Class as User-Defined Data Type: Combines attributes using predefined types
- this keyword usage with example (Bus class)

**Strategy:** Real Code + Live Compilation

- Demonstrate Bus and Story_Book class using IDE or simulator

**Extension (Further Exploration)**

**Strategy:** RAFT Writing

- Role: Object
- Audience: Programmer
- Format: Letter
- Topic: "Why I am the real-world result of your blueprint!"

**Discussion Prompts:**

- How does memory get allocated to an object?
- Why is a class called an object factory?

**Evaluation (Assessment & Review)**

**Strategy:** One Minute Paper

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

**Quiz & Worksheet:** Based on textbook

- MCQs (Solved + Unsolved Questions)
- Fill in the blanks
- Short Answer Questions (e.g., Why object is called instance?)

- Long Answer Questions (e.g., Explain 'Class is a user-defined data type')
- Assertion and Reasoning (e.g., this keyword, object creation logic)

**Suggested Activity**

**Strategy:** Object Creation Simulation + Role Play

- Role-play class and objects (e.g., class: Smartphone, objects: iPhone, Samsung)
- Visualisation of class blueprint vs object state

# 3 Values and Data Types

## Teaching Objectives

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

- Understand the purpose of Java tokens, literals, identifiers, and variables.
- Define and classify primitive and non-primitive data types.
- Apply rules for declaring, initialising, and using variables.
- Use escape sequences and recognise their outputs.
- Perform implicit and explicit type conversions (casting).

| Number of Periods | |
|---|---|
| Theory | Practical |
| 4 | 2 |

## Teaching Plan

**Introduction (Engagement)**

**Strategy:** Think-Pair-Share

- Ask: "What types of values do you store in your phone – numbers, names, photos? How do we tell the computer what kind of data to expect?"
- Discuss how programming languages differentiate and manage types of data.

**Lesson Delivery (Explanation & Demonstration)**

1. **Java Tokens**
   - Categories: Keywords, Identifiers, Literals, Operators, Separators
   - Rules for naming identifiers (valid and invalid examples)
2. **Java Data Types**
   - Primitive Types: int, float, double, boolean, char, byte, short, long
   - Non-primitive Types: String, Arrays, Classes
   - Show memory size, range, and default values
3. **Variables and Declaration**
   - Syntax for declaration and initialisation (static & dynamic)
   - Example: int x = 10; and float pi = 3.14f;

4. **Escape Sequences**
   - \n, \t, \\, \"
   - Print statements to demonstrate newline, tab, quotes, and slash
5. **Type Casting**
   - Widening (Implicit): int to float
   - Narrowing (Explicit): double to int
   - Syntax and casting examples with output

**Strategy:** Guided Coding + Dry Run

- Students run code with each data type and conversion scenario

**Extension (Further Exploration)**

**Strategy:** RAFT Writing

- Role: Data Type
- Audience: Java Learner
- Format: Poem or Tweet
- Topic: "My size and type define your program"

**Discussion Prompts:**

- Why does Java need both primitive and non-primitive types?
- What are the risks of incorrect type conversion?

**Evaluation (Assessment & Review)**

**Strategy:** One Minute Paper

- Ask: "Which data type is the most useful in your opinion, and why?"

**Quiz & Worksheet:**

- MCQs and Fill in the blanks
- Identify valid identifiers and literals
- Type casting scenarios: correct or incorrect
- Match data types to their sizes
- Debug variable declarations

**Suggested Activity**

**Strategy:** Chart Making + Code Simulation

- Students create a chart comparing data types with real-world values (e.g., age → int, price → float)
- Write a Java program using all primitive data types with comments

# 4 Operators in Java

## Teaching Objectives

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

- Understand what operators are and why they are used in Java.
- Identify and categorise Java operators: Arithmetic, Relational, Logical, Assignment, Unary, and Ternary.
- Apply operators correctly in expressions and programs.
- Predict outcomes of expressions involving multiple operators.

| Number of Periods | |
|---|---|
| Theory | Practical |
| 5 | 5 |

## Teaching Plan

**Introduction**

**Strategy:** Brainstorming

- Ask: "How do we perform operations in Maths? What signs do we use?"
- Introduce the concept that operators in Java are symbols used to perform actions on variables and values.

**Lesson Delivery (Explanation & Demonstration)**

1. **Definition and Role of Operators**
   - Operators are symbols that perform operations on variables/values.
   - Syntax of basic expressions: a + b, x > y, a += 5.
2. **Arithmetic Operators**
   - +, -, *, /, %
   - Show how arithmetic operations work on variables. Strategy: Guided Practice
   - Write sample code and evaluate results together.
3. **Relational Operators**
   - , <, >=, <=, ==, !=
   - Used to compare values; result is a boolean (true/false). Strategy: Think-Pair-Share
   - Students come up with real-life comparison scenarios.
4. **Logical Operators**
   - && (AND), || (OR), ! (NOT)
   - Explain truth tables. Strategy: Table Completion
   - Students complete tables for different boolean inputs.

5. **Assignment Operators**
   - =, +=, -=, *=, /=, %=
   - Difference between simple and compound assignments. Strategy: Code Tracing
   - Predict results of compound expressions.
6. **Unary Operators**
   - ++ (increment), -- (decrement)
   - Prefix vs Postfix: ++a vs a++ Strategy: Role Play
   - One student acts as a variable, and others as pre/post operators modifying them.
7. **Ternary Operator**
   - Syntax: condition ? true_result : false_result
   - Compact if-else statement. Strategy: Example Substitution
   - Provide values and ask students to substitute them into ternary format.

**Extension RAFT Writing**

- Role: Logical Operator
- Audience: Beginner Programmer
- Format: Message
- Topic: "Why you need me in decision making"

**Discussion Prompts:**

- Why should we understand operator precedence?
- How are relational and logical operators related?

## Evaluation

**Strategy:** One Minute Paper

- Ask: "Which operator was most confusing to you and why?"

**Worksheet:**

- Fill in the blanks, Match the pair (operator with function), MCQs.
- Code snippet evaluations.

**Student Activity:** Application Cards

- Write a Java statement using each category of operator.

## Suggested Activity

**Strategy:** Send-a-Problem

- Write a real-world scenario. Ask students to write Java code with at least 3 operator types used in the solution.

# 5 Input in Java

## Teaching Objectives

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

- Understand the importance of input in Java programs.
- Use different methods of input: Scanner class and BufferedReader.
- Write programs that accept and display user input.
- Apply type conversion methods when accepting input.

| Number of Periods | |
|---|---|
| Theory | Practical |
| 12 | 20 |

## Teaching Plan

**Introduction**

**Strategy:** Think-Pair-Share

- Ask: "Have you ever filled out a form online? What happens when you enter data?"
- Introduce the idea of taking user input into a Java program.

**Lesson Delivery (Explanation & Demonstration)**

1. **Why Input is Important in Programming**
   - Programs become dynamic when they take input.
   - Examples: Calculators, forms, quizzes. Strategy: Analogy
   - Compare static programs vs interactive ones.
2. **Taking Input using Scanner Class**
   - import java.util.Scanner;
   - Syntax: Scanner sc = new Scanner
   - Methods: nextInt(), nextLine(), nextFloat(), etc. Strategy: Guided Practice
   - Sample code and live demonstration.
3. **Taking Input using BufferedReader**
   - import java.io.*;
   - Syntax: BufferedReader br = new BufferedReader
   - Method: br.readLine(); + type conversion. Strategy: Role Play.
   - One student plays computer, another the user. They enact input and reading.
4. **Parsing Input to Desired Type**
   - Integer.parseInt(), Double.parseDouble()
   - Exception handling basics with try-catch. Strategy: Code Tracing (Strategy #19)
   - Students predict output of small code blocks.

**Extension RAFT Writing**

- Role: Scanner Object
- Audience: Java Beginner
- Format: Letter
- Topic: "How I help you get user data into your program"

**Discussion Prompts:**

- Why do we need type conversion with input?
- What are the differences between Scanner and BufferedReader?

### Evaluation

**Strategy: One Minute Paper**

- Ask: "Which input method do you find easiest and why?"

**Worksheet:**

- Write code snippets using Scanner and BufferedReader.
- Fill in the blanks with appropriate methods.
- Match class names with their functions.

**Student Activity: Application Cards**

- Students write a small program taking name and age from user and print a message.

### Suggested Activity

**Strategy: Send-a-Problem**

- Challenge: "Design a program that takes three types of input: string, integer, and float, then prints a formatted output."

## 6 Mathematical Library Methods

### Teaching Objectives

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

- Understand the purpose of Java's Math class.
- Use various predefined methods of the Math class.
- Apply methods like Math.sqrt(), Math.pow(), Math.abs(), Math.max(), Math.min(), Math.round(), Math.ceil(), Math.floor() etc.
- Write programs that use Math methods for calculations.

| Number of Periods | |
|---|---|
| Theory | Practical |
| 10 | 15 |

## Teaching Plan

**Introduction**

**Strategy:** Real-life Connection

- Ask: "How do calculators work internally? What if we want to calculate a square root in code?"
- Introduce the Math class in Java as a set of built-in tools to do mathematical tasks.

**Lesson Delivery (Explanation & Demonstration)**

1. **Introduction to Math Class**
   - Java provides a Math class with many useful functions for mathematical operations.
   - No need to import explicitly; it's part of java.lang package.
2. **Commonly Used Methods**
   - Math.sqrt(double) – Square root
   - Math.pow(double, double) – Power function
   - Math.abs() – Absolute value
   - Math.max(), Math.min() – Maximum and Minimum
   - Math.round() – Rounding to nearest whole number
   - Math.ceil() – Round up
   - Math.floor() – Round down Strategy: Guided Practice
   - Demonstrate each method with simple examples and output.
3. **Understanding Return Types**
   - Emphasise return types: double/int based on the method.
   - Highlight differences between Math.round() (returns long) and others returning double. Strategy: Think-Pair-Share
   - Students pair up, discuss possible outputs for expressions like Math.ceil(3.4) or Math.
4. **Use in Expressions**
   - Combine multiple Math functions in a single expression.
   - For example: double result = Math.sqrt(Math.pow(3,2) + Math.pow(4,2)); Strategy: Code Tracing
   - Predict the output of multi-method expressions.

**Extension RAFT Writing**

- Role: A Math method (e.g. Math.sqrt())
- Audience: A beginner coder
- Format: Message
- Topic: "Why I'm useful in your program"

**Discussion Prompts:**

- How are Math methods different from operators?
- Why do we use rounding methods in real-life billing applications?

### Evaluation

**Strategy:** One Minute Paper

- Ask: "Which Math method do you see yourself using most and why?"

**Worksheet:**

- Match the Math method with its function.
- Fill in the blanks and MCQs.
- Short code writing using Math class methods.

**Student Activity: Application Cards**

- Each student writes a short program using at least three Math methods for any calculation (e.g. area, compound interest, physics formula).

**Suggested Activity**

**Strategy:** Send-a-Problem

- Create a problem statement like "Calculate distance between two points using Math.pow and Math.sqrt."
- Pass it to peers to solve with different inputs.

## 7 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.
- Apply conditional statements (if, if-else, if-else-if, nested if) for decision making.
- Use ternary operators for compact conditional expressions.
- Use switch case for multi-branching decisions.
- Differentiate between if and switch constructs.
- Implement menu-driven programs and terminate programs using System.exit().

| Number of Periods | |
|---|---|
| Theory | Practical |
| 6 | 4 |

### Teaching Plan

**Introduction (Engagement)**

**Strategy:** Think-Pair-Share

- Ask: "Have you ever made a decision based on a condition? What if your code could do the same?"

- Introduce the idea of conditional constructs controlling program flow.

**Lesson Delivery (Explanation & Demonstration)**

**1. Flow of Control**

- Normal Flow (sequential)
- Conditional Flow (based on decisions)
- Multi-branching (selective paths)

**2. Conditional Statements**

- if, if-else, if-else-if, if and only if, nested if
- Syntax, explanation, and sample programs for:
  - Income tax calculation
  - Positive/Negative number detection
  - Triangle type (equilateral, isosceles, scalene)

**3. Ternary Operator**

- Syntax: result = (condition) ? value1 : value2;
- Use nested ternary operators to simplify decisions

4. Switch Case Statement

- Syntax, usage, and examples
- Case blocks with break and the fall-through problem
- Menu-driven examples: area/perimeter, calculator, interest calculation

**5. System.exit() Function**

- Normal and abnormal termination: System.exit(0) and System.exit(1)
- Program examples: stop program on invalid input

**6. Real-world Problem Solving**

- ABC Electricity Bill Generator
- Telecom billing with call slabs
- Seasonal shopping discount systems
- Sales commission and gift logic

**Strategy:** Guided Coding & Code Tracing

- Demonstrate logic tracing on flowchart and dry run tables

**Extension (Further Exploration)**

**Strategy:** RAFT Writing

- Role: switch statement
- Audience: if-else ladder
- Format: Letter
- Topic: "Why I'm your organised alternative"

**Discussion Prompts:**

- What are the advantages of using switch over if-else?
- In what situations is ternary operator better than full if-else?

**Evaluation (Assessment & Review)**

**Strategy:** Exit Ticket

- Ask: "What is one new type of flow control you learned today, and how would you use it?"

**Quiz & Worksheet:**

- MCQs: Flow control types, switch behavior, output prediction
- Fill in the blanks: Data types, logical expressions, operator symbols
- Short Answer Questions: Differences between statements, syntax recall
- Long Answer Questions: Real-life program writing with if-else, switch
- Assertion and Reasoning Questions
- Ternary ↔ if-else ↔ switch conversion

**Suggested Activity**

**Strategy:** Menu-Driven Coding Challenge + Case-Based Group Discussion

- Students create a Java menu-driven program with at least 3 options
- Debug & convert between ternary and if-else constructs

# 8 Iterative Constructs in Java

## Teaching Objectives

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

- Understand the purpose and types of loops in Java.
- Distinguish between entry-controlled and exit-controlled loops.
- Implement for, while, and do-while loops in Java.
- Recognise and apply different forms of loops: finite, infinite, null, step, and delay loops.
- Convert one type of loop into another.
- Use break, continue, and return statements in loop control.
- Apply loops to solve practical problems and coding exercises.

| Number of Periods | |
|---|---|
| Theory | Practical |
| 6 | 4 |

## Teaching Plan

**Introduction (Engagement)**

**Strategy:** Think-Pair-Share

- Ask: "What would you do if you had to repeat something 100 times in a program?"

- Use analogy of daily tasks (e.g., brushing teeth daily = loop)

**Lesson Delivery (Explanation & Demonstration)**

1. **Loop Basics**
   - Definition of loop, iteration, loop control variable
   - 4 parts of a loop: Initialization, Condition, Update, Body
2. **Loop Categories**
   - Entry Controlled: for, while
   - Exit Controlled: do-while
3. **The for Loop**
   - Syntax, dry run, example: print even numbers to 10
   - Multiple expressions in one loop
4. **The while Loop**
   - Syntax, dry run, use-case: unknown number of repetitions
   - Example: extract digits from a number
5. **The do-while Loop**
   - Syntax, executes at least once
   - Example: sum of first 5 natural numbers
6. **Forms of Loops**
   - Infinite loop, Null loop, Delay loop
   - Finite loop: Continuous & Step loop
7. **Loop Interconversion**
   - Convert for to while, do-while and vice versa
   - Syntax side-by-side conversion chart
8. **Jump Statements**
   - break: exit from loop
   - continue: skip to next iteration
   - return: go back to the calling method
   - Examples and dry runs of each

   **Strategy:** Guided Coding + Dry Run Table
   - Students simulate 3 loops with break/continue and dry-run results
9. **Loop-Based Programs**
   - Prime check, even/odd sum-product, Fibonacci, Armstrong, Palindrome, etc.

**Extension (Further Exploration)**

**Strategy:** RAFT Writing

- Role: Infinite Loop

- Audience: Programmer
- Format: Tweet
- Topic: "Why you should fear me in your code!"

**Discussion Prompts:**

- Why does loop conversion matter?
- When is do-while the only suitable loop?

**Evaluation (Assessment & Review)**

**Strategy:** One Minute Paper

- Ask: "Which loop is most useful and why?"

**Quiz & Worksheet: Based on textbook**

- MCQs
- Fill in the blanks
- Short/Long Answer Questions
- Assertion and Reasoning Questions
- Program Output Prediction and Correction

**Suggested Activity**

**Strategy:** Loop Simulation Game + Debugging Challenge

- Simulate each loop as a group movement
- Students fix broken loop logic or predict output
- Menu-driven programs using iterative logic

# 9 Nested Loops

## Teaching Objectives

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

- Understand the structure and use of nested loops in Java.
- Differentiate between outer and inner loops and trace their flow.
- Implement nested loops using for, while, and do-while.
- Use nested loops to create various pattern-based programs.
- Apply break and continue within nested loops appropriately.

| Number of Periods | |
|---|---|
| Theory | Practical |
| 4 | 4 |

## Teaching Plan

**Introduction (Engagement)**

**Strategy:** Real-Life Analogy Discussion

- Ask: "Have you ever seen a matrix, chessboard, or seating chart? How are they structured?"

- Introduce nested loops using the concept of rows and columns.

**Lesson Delivery (Explanation & Demonstration)**

1. **What are Nested Loops?**
   - Loop inside another loop
   - Flow control: outer loop controls rows, inner loop controls columns
2. **Syntax & Structure**
   - Nested for loop
   - Nested while loop
   - Nested do-while loop
   - Dry run example for 3x3 matrix
3. **Pattern Logic Building**
   - Use of loop counters for character and number placement
   - Row and column control via nested structures
4. **Common Pattern Programs**
   - Solid square and rectangle
   - Right-angled triangle (ascending and descending)
   - Pyramid patterns (half and full)
   - Number triangle
   - Floyd's Triangle
   - Hollow triangle
   - Alphabet pyramid
   - Diamond and hourglass
5. **Applying Break and Continue**
   - Break inner loop on condition
   - Continue to skip current inner iteration
   - Examples with output

**Strategy:** Guided Coding + Group Practice

- Students write and debug pattern code in pairs

**Extension**

**Strategy:** RAFT Writing

- Role: Inner Loop
- Audience: Java Student
- Format: Message
- Topic: "Why I repeat inside the structure"

**Discussion Prompts:**

- How does the number of iterations multiply in nested loops?

- Why are nested loops important in data processing and graphics?

**Evaluation (Assessment & Review)**

**Strategy:** One Minute Paper

- Ask: "Which pattern program did you enjoy most and why?"

**Quiz & Worksheet:**

- Pattern prediction questions (output-based)
- Fill in the blanks on syntax
- Trace and dry-run nested loops
- Short Answer Questions: Explain nested flow
- Program completion: partially written pattern logic

**Practical Coding Tasks:**

- Create your own 3-pattern challenge (alphabet, number, special character)
- Build a menu-driven Java program for pattern selection

**Suggested Activity**

**Strategy:** Pattern Coding Contest + Debugging Task

- Small groups compete to write the cleanest logic for given patterns
- Bonus: fix incorrect nested pattern code shared by teacher

# 10 User-Defined Methods

## Teaching Objectives

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

- Understand the concept, structure, and types of methods in Java.
- Differentiate between predefined and user-defined methods.
- Define, invoke, and apply static, non-static, pure, and impure methods.
- Explain actual vs formal parameters.
- Demonstrate method overloading.
- Apply methods to build modular Java programs.

| Number of Periods | |
|---|---|
| Theory | Practical |
| 5 | 5 |

## Teaching Plan

**Introduction (Engagement)**

**Strategy:** Think-Pair-Share

- Ask: "If you had to calculate interest in multiple programs, would you rewrite the same logic each time or reuse it?"

- Introduce the concept of methods as reusable code blocks.

**Lesson Delivery (Explanation & Demonstration)**

1. Definition & Need for Methods
   - A method is a reusable block of code that performs a specific task.
   - Enhances modularity, debugging, and code reuse.
2. **Forms of Methods**
   - Predefined Methods: Built-in Java methods like next(), charAt()
   - User-defined Methods: Custom methods created for specific tasks
3. **Defining a Method**
   - Syntax: <AccessSpecifier> <ReturnType> <MethodName>(parameters)
   - Components: Header, Signature, Parameters, Return type, Body
4. **Ways to Define a Method**
   - With/without parameters
   - With/without return values
   - Primitive and non-primitive return types
5. **Actual and Formal Parameters**
   - Actual: Passed during method call
   - Formal: Received in method definition
   - Pass by Value vs Pass by Reference with examples
6. **Invoking Methods**
   - Static and Non-static
   - Call by value: basic types
   - Call by reference: arrays and objects
7. **Static vs Non-Static Methods**
   - Static: Belongs to class, no object needed
   - Non-static: Belongs to object, accessed via instance
8. **Pure vs Impure Methods**
   - Pure: Does not modify object state, returns value
   - Impure: Changes object state, may or may not return value
9. **Method Overloading**
   - Same method name, different parameter type/order/number
   - Supports polymorphism and increases flexibility

**Strategy:** Guided Coding + Dry Run

- Show method examples for palindrome, factorial, tax calculation, pattern printing

**Extension (Further Exploration)**

**Strategy:** RAFT Writing

- Role: Overloaded Method
- Audience: Java Student
- Format: Note
- Topic: "Why I'm flexible enough to do many things at once!"

**Discussion Prompts:**

- When would you use method overloading?
- Why are pure methods safer in larger programs?

**Evaluation (Assessment & Review)**

**Strategy:** One Minute Paper

- Ask: "Which part of defining methods do you find most important?"

**Quiz & Worksheet:**

- MCQs & Fill in the blanks
- Short Answers: Differences, syntax rules
- Long Answers: Explain with code examples
- Assertion & Reasoning: Method types, overloading
- Code Tracing: Identify outputs and errors

**Suggested Activity**

**Strategy:** Menu-Driven Coding Project + Case-Based Debugging

- Students create a Java program with multiple overloaded methods for shapes and numbers
- Peer-review each other's code for pure/impure method usage

## 11 Class as the Basis of all Computation

### Teaching Objectives

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

- Define and differentiate between classes and objects in Java.
- Understand characteristics (attributes) and behaviours (methods) of objects.
- Write Java programs to define classes, create objects, and invoke methods.
- Understand access specifiers and data types in a class.
- Demonstrate instantiation and use of the this keyword.
- Explain and apply nested classes.
- Recognise Java constructors and their role in object creation.

| Number of Periods | |
|---|---|
| Theory | Practical |
| 6 | 4 |

## Teaching Plan

**Introduction (Engagement)**

**Strategy:** Real-Life Analogy Discussion

- Ask: "What common properties do all books, vehicles, or mobiles have? How are they different?"
- Introduce the concept of classes as blueprints and objects as real-world instances.

**Lesson Delivery (Explanation & Demonstration)**

1. Concept of Object
   - Defined as a real-world entity with characteristics and behaviour.
   - Example: Computer with attributes: Speed, Accuracy; behaviour: Processing, Storing
2. Concept of Class
   - Class is a blueprint to create objects
   - All objects of a class share same properties and behaviours
   - Example: Dog, Computer, Shape
3. Class vs Object

| Class | Object |
|---|---|
| Logical entity | Real-world entity |
| Blueprint | Instance of class |
| No memory | Takes memory |

4. **Creating Objects in Java**
   - Syntax: <ClassName> <objectName> = new <Constructor>();
   - Example: Computer mainframe = new Computer();
5. Components of a Class
   - Access Specifiers (public, private, protected)
   - Data Members (instance, class/static, local variables)
   - Member Methods (static and non-static)
   - Constructor: Special method with same name as class
6. Access Specifiers

| Access Specifier | Same Class | Inherited Class | Other Class |
|---|---|---|---|
| private | Yes | No | No |
| protected | Yes | Yes | No |
| public | Yes | Yes | Yes |

7. **Data Members & Methods**
   - Instance Variables: Declared in class, different for each object
   - Static/Class Variables: Shared across all objects
   - Local Variables: Inside method, limited scope
8. **Constructors**
   - Automatically called when object is created
   - No return type, same name as class
   - Used for initialisation
9. **Nested Class**
   - Class declared inside another class
   - Example: class Employee contains inner class class Salary
10. **Using this Keyword**
    - Resolves ambiguity between instance variables and parameters
    - this.variable = parameter;

**Strategy:** Guided Code Writing

- Live demonstration of multiple programs: Rectangle, Sphere, Student, Banking System, Library, Telephone, etc.

**Extension (Further Exploration)**

**Strategy:** RAFT Writing

- Role: Java Object
- Audience: Programmer
- Format: Diary Entry
- Topic: "A day in my life inside a class"

**Discussion Prompts:**

- Why does each object need memory but a class doesn't?
- When and why should we use constructors?

**Evaluation (Assessment & Review)**

**Strategy:** One Minute Paper

- Ask: "What's the difference between a class and an object in your own words?"

**Quiz & Worksheet:**

- MCQs: Object creation, access specifiers, method types
- Fill in the blanks: Constructor use, this keyword, instantiation
- Short Answer: Define class/object, list access specifiers
- Long Answer: Write programs to define classes and objects

- Assertion and Reasoning Questions
- Case Study Analysis: Ajay's Bookstore example

**Suggested Activity**

**Strategy:** Code Lab + Roleplay Simulation

- Create a Book class with constructor and methods, simulate object behaviour
- Students act as instance variables/methods of class 'School'

# 12 Constructors

## Teaching Objectives

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

- Define a constructor and state its key properties.
- Understand the concept of constructor overloading.
- Identify similarities and differences between constructors and methods.
- Use this keyword in constructor definitions.

| Number of Periods | |
|---|---|
| Theory | Practical |
| 3 | 2 |

## Teaching Plan

**Introduction (Engagement)**

**Strategy:** Real-Life Analogy

- Ask: "What happens the moment a new object is created in a program?"
- Relate constructor to a first-time setup process.

**Lesson Delivery (Explanation & Demonstration)**

1. **What is a Constructor?**
   - Constructor is a member method having the same name as the class.
   - It is invoked automatically at the time of object creation.
   - It does not return any value and can be overloaded.
2. **Syntax of Constructor**

```
ClassName() {
    // constructor body
}
```

   - Called using the new keyword: ClassName object = new ClassName();
3. **Characteristics of Constructor**
   - Same name as class.
   - No return type.

- Automatically invoked on object creation.
- Can be overloaded.

**4. Constructor Overloading**

- Creating multiple constructors in the same class with different parameter lists.

```
class Student {
    Student() { }
    Student(String name) { }
    Student(String name, int age) { }
}
```

**5. Similarities between Constructor and Method**

| Constructor | Method |
|---|---|
| Belongs to class | Belongs to class |
| Can have parameters | Can have parameters |
| Can be overloaded | Can be overloaded |

**6. Differences between Constructor and Method**

| Constructor | Method |
|---|---|
| Invoked automatically | Invoked by calling |
| No return type | Has return type |
| Same name as class | Different name from class |

**7. Use of this Keyword**

- This refers to the current class instance.
- Used to distinguish between instance variables and parameters.

```
this.name = name;
```

**Evaluation (Assessment & Review)**

**Worksheet / Quiz:**

- MCQs: Choose the correct definition or usage of constructors.
- Fill in the blanks: Characteristics and syntax-based questions.
- Short Answer Questions: Explain constructor overloading.
- Long Answer Question: Differentiate between constructor and method.
- Output-Based Questions: Identify constructor being invoked.

**Suggested Activity**

**Strategy:** Guided Code Writing

- Write a Java program with multiple constructors (overloaded)
- Demonstrate use of this keyword

# 13 Library Classes

## Teaching Objectives

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

- Define library classes and wrapper classes.
- Identify different Java packages and their uses.
- Distinguish between primitive and composite data types.
- Use wrapper classes to convert between primitive types and strings.
- Apply character methods from the Character class.
- Understand autoboxing and unboxing.

| Number of Periods | |
|---|---|
| Theory | Practical |
| 5 | 3 |

## Teaching Plan

**Introduction (Engagement)**

**Strategy:** Think-Pair-Share

- Ask: "Have you ever reused someone else's code to make your task easier?"
- Discuss: Java's library classes and how they simplify coding by providing predefined solutions.

**Lesson Delivery (Explanation & Demonstration)**

1. **Library Classes and Java Packages**
   - Library classes: Predefined classes provided by Java.
   - Packages: Collection of similar types of classes (e.g., java.lang, java.io, java.util, java.math).
   - import java.util.*; imports all classes from java.util package.
2. **Primitive and Composite Data Types**
   - Primitive data types: byte, short, int, long, float, double, char, boolean
   - Composite data types: Defined by the user (e.g., class, array, interface, string)
3. **Class as a Composite Data Type**
   - A class contains data members (primitive) and methods.
   - Considered a user-defined data type.
4. **Wrapper Classes**
   - In-built classes in java.lang to convert primitive types into objects.
   - Example:
     - int → Integer, float → Float, double → Double, char → Character
   - Purposes:
     - Convert between types
     - Use primitives as objects

**5. Using Wrapper Classes**

- Importing wrapper classes: import java.lang.*; or specific ones like import java.lang.Integer;
- Syntax: `WrapperClass.method()`

**a. Converting Primitive to String**

- Integer.toString(int)
- Float.toString(float)
- Double.toString(double)
- Long.toString(long)

**b. Converting String to Primitive**

- Integer.parseInt(String) or Integer.valueOf(String)
- Float.parseFloat(String) or Float.valueOf(String)
- Double.parseDouble(String) or Double.valueOf(String)
- Long.parseLong(String) or Long.valueOf(String)

**c. Character Class Methods**

- isLetter(), isDigit(), isLetterOrDigit(), isWhitespace()
- isUpperCase(), isLowerCase(), toUpperCase(), toLowerCase()

**6. Autoboxing and Unboxing**

- Autoboxing: Primitive → Wrapper object (e.g., Integer intObj = 10;)
- Unboxing: Wrapper → Primitive (e.g., int i = intObj;)

**Evaluation (Assessment & Review)**

**Worksheet / Quiz:**

- MCQs: Wrapper class usage, conversions, method functions
- Fill in the blanks: Syntax, method names
- Short Answer Questions: Define wrapper, differentiate data types
- Long Answer: Use and importance of wrapper classes
- Assertion and Reasoning Questions
- Case-based Questions: Application of wrapper methods

**Suggested Activity**

**Strategy:** Code Practice + Role-based Demonstration

- Students write code using Character, Integer, Float classes
- Demonstrate conversion using wrapper methods, including autoboxing/unboxing

# 14 Encapsulation and Inheritance

## Teaching Objectives

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

- Define encapsulation and explain its purpose in object-oriented programming.
- Identify and use access specifiers (private, public, protected).
- Understand the types and scope of variables in Java.
- Define inheritance and explain its types and purpose.

| Number of Periods | |
|---|---|
| Theory | Practical |
| 6 | 4 |

## Teaching Plan

**Introduction (Engagement)**

**Strategy:** Real-Life Analogy Discussion

- Ask: "How does a capsule keep medicine protected inside it?" to relate with data protection.
- Link this concept to encapsulation and inheritance in Java.

**Lesson Delivery (Explanation & Demonstration)**

**1. Encapsulation**

- Definition: Bundling of data members and member methods into a single unit.
- Promotes data hiding and safety.
- Example: Capsule analogy and encapsulation_example class.

2. Access Specifiers

- Used to control access to class members.

**a. private:**

- Accessible only within the class.
- Example: private_access_sp class.

**b. public:**

- Accessible from anywhere.
- Example: public_access_sp and other_class.

**c. protected:**

- Accessible within the same package and through inheritance.
- Example: derived_class extends protected_access_sp.

**3. Scope of Variables**

a. Local Variables:

- Declared inside methods; accessible only within the method.
- Example: f and i in factorial class.

b. Class Variables:
   - Declared with static; shared among all objects.
   - Example: c in count_object class.

c. Instance Variables:
   - Declared in a class but outside methods; unique for each object.
   - Example: n, t, and s in instance_variable class.

d. Argument Variables:
   - Passed to methods when called.
   - Example: radius in argument_variable class.

e. Block Scope:
   - Variables defined within braces {} and limited to that block.
   - Example: s in scope_variable class.

4. Inheritance
   - Reusing properties and methods of one class in another.
   - Base class (superclass) and Derived class (subclass).

**Types of Inheritance:**

- Single
- Multiple (Note: Java doesn't support multiple inheritance)
- Multilevel
- Hierarchical
- Hybrid

**Evaluation (Assessment & Review)**

**Worksheet / Quiz:**

- MCQs: Definitions, uses, and access specifiers
- Fill in the blanks: Variable types, scope, inheritance
- Short Answer Questions: Definitions of encapsulation, access specifiers
- Long Answer: Explain inheritance and its types
- Assertion and Reasoning: Based on encapsulation, access, and inheritance
- Case-based Questions: Program interpretation and output prediction

**Suggested Activity**

**Strategy:** Program Writing and Role-play

- Write Java programs using all three access specifiers and identify output
- Create a simulation where students role-play base and derived class properties

# 15 Arrays

## Teaching Objectives

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

- Define and explain the concept of arrays.
- Identify the types of arrays: single and multi-dimensional.
- Perform array operations: creation, initialization, accessing, insertion, deletion, searching, sorting, and merging.
- Implement programs using array concepts.

| Number of Periods | |
|---|---|
| Theory | Practical |
| 7 | 6 |

## Teaching Plan

**Introduction (Engagement)**

**Strategy:** Think-Pair-Share

- Ask: "What if we need to store marks of 100 students? Will declaring 100 variables be efficient?"
- Lead to the idea of arrays for efficient data management.

**Lesson Delivery (Explanation & Demonstration)**

1. **What is an Array?**
   - Set of values of same data type stored under one variable name.
   - Composite data type and reference type.
2. **Array as Composite Type**
   - Composite types store a number of values using a single declaration.
   - Arrays are collections of similar data type.
3. **Types of Arrays**
   - Single Dimensional Array
   - Multi-Dimensional Array
4. **Declaring and Initializing Arrays**
   - Syntax:
     - int a[] = new int[10];
     - String[] names = new String[40];
   - Initializing:
     - Direct assignment: int ar[] = {1, 2, 3, 4, 5};
     - Using loops or scanner.

5. **Accessing Array Elements**
   - Accessed via index starting from 0.
   - Example: ar[0], ar[1] etc.
6. **Length of an Array**
   - Use .length attribute: arr.length
7. Taking Values from User
   - Using method argument
   - Using Scanner class
8. **Multi-Dimensional Arrays**
   - Syntax: int a[][] = new int[2][3];
   - Declaration, initialization and accessing elements
9. **Array Operations**
   - Searching:
     - Linear Search
     - Binary Search
   - Sorting:
     - Bubble Sort
     - Selection Sort
   - Insertion:
     - Shifting elements to insert a new value
   - Deletion:
     - Shifting elements left and setting last to 0
   - Merging:
     - Combining two arrays into a third

**Evaluation (Assessment & Review)**

**Worksheet / Quiz:**

- MCQs: Array declaration, type, operations
- Fill in the blanks: Index, .length, data types
- Short Answer: Array definition, types, advantages
- Long Answer: Explain array operations
- Code-Based: Identify output, errors, and tracing

**Suggested Activity**

**Strategy:** Code Writing Lab + Sorting Simulation

- Students write programs using array operations
- Group sort simulation for Bubble and Selection Sort

# 16 String Handling

## Teaching Objectives

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

- Understand what a String is and how to declare it using implicit and explicit ways.
- Use various String methods such as length(), charAt(), indexOf(), substring(), concat(), equals(), compareTo() etc.
- Apply string operations like searching, extracting substrings, replacing, comparison, and case conversion.
- Create and use String arrays.
- Implement real-world problems using string manipulation and logic.

| Number of Periods | |
|---|---|
| Theory | Practical |
| 7 | 6 |

## Teaching Plan

**Introduction (Engagement)**

**Strategy:** Real-Life Analogy + Think-Pair-Share

- Ask: "Have you ever stored a full name or an address in a variable using Java?"
- Discuss the challenges with character arrays and introduce the efficiency of String class.

**Lesson Delivery (Explanation & Demonstration)**

1. **String Basics**
   - Definition: A string is a sequence of characters enclosed in double quotes.
   - Package: java.lang
   - Declaration:
     - Implicit: String s = "Java";
     - Explicit: String s = new String("Java");
2. **String Storage in Memory**
   - Each character is indexed (like an array), starting from 0.
   - Use .length() to find total number of characters.
3. **Important String Methods**

| Method | Purpose |
|---|---|
| length() | Returns number of characters |
| charAt(index) | Character at a specific index |
| indexOf(ch) | First index of character |
| lastIndexOf(ch) | Last index of character |
| toLowerCase() | Converts to lowercase |
| toUpperCase() | Converts to uppercase |

| | |
|---|---|
| concat(str) | Joins two strings |
| substring(index) | Extract from index to end |
| substring(start, end) | Extract between indices |
| replace(ch1, ch2) | Replaces characters |
| equals(str) | Compares strings (case-sensitive) |
| equalsIgnoreCase(str) | Case-insensitive comparison |
| compareTo(str) | Lexical comparison |
| compareToIgnoreCase(str) | Lexical comparison ignoring case |
| startsWith(str) | Checks prefix |
| endsWith(str) | Checks suffix |
| trim() | Removes leading/trailing spaces |
| valueOf(primitive) | Converts primitive to string |

**4. String Arrays**

- Declaration: String[] cities = {"Delhi", "Mumbai"};
- Accessing elements: cities[0], cities[1].charAt(2)
- cities.length gives total elements
- cities[i].length() gives length of element string

**Evaluation (Assessment & Review)**

**Worksheet / Quiz:**

- MCQs: Syntax, method use, output predictions
- Fill in the blanks: Method names, definitions
- Short Answer: Define string, compare == and .equals()
- Long Answer: Difference between methods, usage scenarios
- Output-based Questions: Predict results of method calls
- Assertion and Reasoning: Based on concepts of method behaviour
- Case Study: Use of substring() and compareTo()

**Suggested Activities**

**Strategy:** Code Writing + Menu-based Program

- Write programs to:
  - Count vowels, digits, and words
  - Toggle case
  - Replace characters/words
  - Abbreviate full names
  - Identify palindromes
  - Create PigLatin transformation
  - Sort names alphabetically
  - Print pattern using substrings