BCS-403: Object Oriented Programming With Java

70-Hour / 42-Session (100 Minutes Each) Full Course Plan

SESSION 1 — Introduction to Java

Session Objective

To introduce students to the Java programming environment, its architecture, history, features, and the process of writing, compiling, and executing Java programs.

Topics Covered

- What is Java?
- History & evolution of Java
- Features of Java (simple, secure, portable, robust, platform independent)
- Difference between JDK, JRE, JVM
- Java program execution model
- Bytecode and Java Virtual Machine
- First Java program structure
- How Java compilation works
- Setting up environment (IDE + JDK)

Detailed Explanation

Java is an object-oriented, platform-independent language developed by Sun Microsystems. Its slogan "Write Once, Run Anywhere" is achieved through bytecode execution by the JVM.
Students learn the role of JDK (development tools), JRE (execution environment), and JVM (bytecode interpreter).
A complete walkthrough of writing the first program and breaking down class, main method, statements, comments, and syntax rules will be done.

Code Example

```
public class HelloWorld {

    public static void main(String[] args) {

        System.out.println("Hello World");
    }
}
```

Hands-on Tasks

- Install Java JDK
- Configure environment variables
- Create and run first Java program
- Use command line: `javac` and `java` commands

Assignments (LeetCode)

1. Two Sum – https://leetcode.com/problems/two-sum/
2. Add Digits – https://leetcode.com/problems/add-digits/

Outcome

Students understand Java architecture, tools, and can compile & execute programs independently.

SESSION 2 — Data Types, Variables & Type Casting

Session Objective

To understand Java's primitive and reference data types, variable declaration rules, type casting, and input handling using Scanner.

Topics Covered

- Primitive types (byte, short, int, long, float, double, char, boolean)
- Reference types
- Memory allocation basics
- Type casting (implicit & explicit)
- Wrapper classes
- Taking user input through `Scanner`

Detailed Explanation

Students learn how Java stores different data types and when type conversion occurs.
Widening (automatic) and narrowing (manual) conversions are explained with memory diagrams.
Scanner class usage teaches real-world program input.

Code Example

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

int age = sc.nextInt();

double salary = sc.nextDouble();
```

Hands-on Tasks

- Create a program to input and print student details
- Demonstrate widening conversion
- Demonstrate narrowing conversion

Assignments (LeetCode)

1. Reverse Integer – https://leetcode.com/problems/reverse-integer/
2. Palindrome Number – https://leetcode.com/problems/palindrome-number/

Outcome

Students master variables, input handling, casting, and primitive/reference types.

SESSION 3 — Operators in Java

Session Objective

To teach students how Java operators work and how to apply them in expressions and real logic.

Topics Covered

- Arithmetic operators
- Relational operators
- Logical operators
- Bitwise operators
- Unary operators

- Ternary operator
- Operator precedence

Detailed Explanation

Students explore how operations are evaluated, the importance of boolean logic in conditions, and bitwise operations for optimization.
The session also covers ++i vs i++, evaluation order, and precedence rules to avoid incorrect expressions.

Code Example

```
int a = 5, b = 10;

boolean result = (a < b) && (++a > 3);
```

Hands-on Tasks

- Build a menu-driven arithmetic calculator
- Create truth tables for logical operators
- Write bitwise AND/OR/XOR examples

Assignments (LeetCode)

1. Single Number – https://leetcode.com/problems/single-number/
2. Plus One – https://leetcode.com/problems/plus-one/

Outcome

Students can apply operators in real logic and expressions confidently.

SESSION 4 — Conditional Statements

Session Objective

To help students write decision-based programs using conditional flow statements.

Topics Covered

- if statement
- if–else

- nested if
- else-if ladder
- switch-case
- fall-through concept
- Advanced switch (Java 14+)

Detailed Explanation

Students learn branching logic and decision-making in programs.
Show practical uses such as grading systems, authentication, and menu-driven applications.

Code Example

```
int marks = 85;

if(marks >= 90) System.out.println("A");

else if(marks >= 80) System.out.println("B");

else System.out.println("C");
```

Hands-on Tasks

- Build a grading system
- Menu-driven program using switch
- Validate login using conditional checks

Assignments (LeetCode)

1. Number of Steps – https://leetcode.com/problems/number-of-steps-to-reduce-a-number-to-zero/
2. Power of Two – https://leetcode.com/problems/power-of-two/

Outcome

Students understand decision-making constructs essential for all real programs.

SESSION 5 — Loops (for, while, do-while)

Session Objective

To teach looping constructs and apply them in repetitive program logic.

Topics Covered

- for loop
- while loop
- do-while loop
- break and continue
- infinite loops
- nested loops

Detailed Explanation

Students explore loops through real practical examples—patterns, mathematical functions, series, and frequency calculations.

Code Example

```
for(int i = 1; i <= 10; i++) {

    System.out.println(i);

}
```

Hands-on Tasks

- **Print multiplication table**
- **Print star patterns**
- **Generate Fibonacci series**

Assignments (LeetCode)

1. **Fizz Buzz – https://leetcode.com/problems/fizz-buzz/**
2. **Count Primes – https://leetcode.com/problems/count-primes/**

Outcome

Students master repetitive logic and pattern-based programming.

SESSION 6 — Arrays (1D & 2D Arrays)

Session Objective

To understand how arrays store sequential data, access elements, perform operations, and implement 2D (matrix) structures.

Topics Covered

- Introduction to arrays
- Declaring & initializing arrays
- Traversing arrays
- Updating, searching, deleting
- Array length property
- Introduction to 2D arrays (matrix)
- Matrix operations (add, subtract, iterate)

Detailed Explanation

Arrays store homogenous, fixed-size data. Students learn how memory is allocated, how indexes work, common pitfalls (ArrayIndexOutOfBounds), and essential operations.
2D arrays are introduced as matrix-like structures ideal for grid-based problems.

Code Example

```
int[] arr = {10, 20, 30};

for(int i : arr){

    System.out.println(i);

}



int[][] mat = {

  {1,2,3},

  {4,5,6}

};

System.out.println(mat[1][2]);
```

Hands-on Tasks

- Find max/min from array

- Rotate array elements
- Add two matrices

Assignments (LeetCode)

1. Maximum Subarray – https://leetcode.com/problems/maximum-subarray/
2. Best Time to Buy and Sell Stock – https://leetcode.com/problems/best-time-to-buy-and-sell-stock/

Outcome

Students can manipulate, process, and perform computations on arrays and matrices.

SESSION 7 — Strings & StringBuilder

Session Objective

To work with Java Strings, understand immutability, and use StringBuilder for efficient modifications.

Topics Covered

- String literal & new String()
- String methods (length, charAt, substring, indexOf)
- String comparison: equals(), compareTo()
- Mutable vs immutable objects
- StringBuilder & StringBuffer
- Efficient concatenation

Detailed Explanation

Strings are immutable; every modification creates a new object. StringBuilder is used when frequent modifications are required.
Students also explore substring extraction, splitting, joining, and character manipulation.

Code Example

```
String name = "Java";

String rev = new StringBuilder(name).reverse().toString();

System.out.println(rev);
```

Hands-on Tasks

- Reverse a string
- Count vowels, consonants, digits
- Check palindrome

Assignments (LeetCode)

1. Valid Anagram – https://leetcode.com/problems/valid-anagram/
2. First Unique Character in a String – https://leetcode.com/problems/first-unique-character-in-a-string/

Outcome

Students understand string processing and when to use StringBuilder for performance.

SESSION 8 — Classes & Objects

Session Objective

To introduce object-oriented fundamentals including class design, object creation, and method invocation.

Topics Covered

- What is a class?
- Fields & methods
- Creating objects
- Accessing members
- this keyword introduction
- Object memory representation

Detailed Explanation

A class acts as a blueprint containing properties (variables) and behaviors (methods). Objects are instances created in heap memory.
Students understand encapsulation at a conceptual level.

Code Example

```
class Student {

    String name;
    int age;

    void display(){

        System.out.println(name + ", " + age);

    }

}


Student s = new Student();

s.name = "Amit";

s.age = 20;

s.display();
```

Hands-on Tasks

- Create Student, Car, Employee classes
- Call methods using objects

Assignments (LeetCode)

1. Merge Two Sorted Lists – https://leetcode.com/problems/merge-two-sorted-lists/
2. Move Zeroes – https://leetcode.com/problems/move-zeroes/

Outcome

Students understand real-world modeling using classes and objects.

SESSION 9 — Constructors & Overloading

Session Objective

To learn about constructors, constructor overloading, and method overloading as part of early OOP development.

Topics Covered

- What is a constructor?
- Default vs parameterized constructors

- Constructor overloading
- Method overloading
- this() and constructor chaining

Detailed Explanation

Constructors initialize objects automatically. Method overloading provides polymorphism by allowing same method name with different parameters.
Examples of practical constructors in real-world models are covered.

Code Example

```
class Box {

    int length, width;


    Box() { length = width = 1; }

    Box(int l, int w) { length = l; width = w; }


    int area() { return length * width; }

}
```

Hands-on Tasks

- Create multiple constructors for Student class
- Overload methods for calculating area

Assignments (LeetCode)

1. Search Insert Position – https://leetcode.com/problems/search-insert-position/
2. Remove Element – https://leetcode.com/problems/remove-element/

Outcome

Students understand initialization logic and polymorphism via overloading.

SESSION 10 — Static & Final (Variables, Methods, Blocks)

Session Objective

To teach the usage of static (class-level) and final (constant) constructs.

Topics Covered

- static variable
- static method
- static block
- static memory behavior
- final variable
- final method
- final class

Detailed Explanation

Static members belong to the class, not an object. Final keyword restricts changes:

- final variable = constant
- final method = cannot override
- final class = cannot be inherited

Memory diagrams illustrate shared static allocation.

Code Example

```
class Counter {
    static int count = 0;
    Counter(){ count++; }
}


final class ConstantDemo {
    final int MAX = 100;
}
```

Hands-on Tasks

- Create static counter
- Demonstrate static block execution
- Create a final class and test inheritance restriction

Assignments (LeetCode)

1. Length of Last Word – https://leetcode.com/problems/length-of-last-word/
2. Richest Customer Wealth – https://leetcode.com/problems/richest-customer-wealth/

Outcome

Students differentiate between instance vs class-level members and use final for safety.

SESSION 11 — Inheritance & Polymorphism

Session Objective

To understand how inheritance promotes code reuse and how polymorphism enables dynamic behavior in object-oriented programming.

Topics Covered

- What is inheritance?
- Types of inheritance (single, multilevel, hierarchical)
- super keyword (calling parent constructor/method)
- Method overriding
- Dynamic Method Dispatch (Runtime Polymorphism)
- Upcasting & Downcasting
- Real-world examples

Detailed Explanation

Inheritance allows a class to acquire properties and methods of another class, reducing redundancy.
Polymorphism allows the same method call to behave differently based on the object's actual type.
Students see real OOP behavior through method overriding and runtime binding.

Code Example

```
class Animal {

    void sound(){ System.out.println("Animal makes sound"); }
}
class Dog extends Animal {
    void sound(){ System.out.println("Dog barks"); }
}

Animal a = new Dog();
a.sound();  // Runtime polymorphism
```

Hands-on Tasks

- Create Vehicle → Car → ElectricCar hierarchy
- Demonstrate overriding using super keyword
- Upcasting demonstration

Assignments (LeetCode)

1. Linked List Cycle – https://leetcode.com/problems/linked-list-cycle/
2. Middle of the Linked List – https://leetcode.com/problems/middle-of-the-linked-list/

Outcome

Students can build hierarchical models and use polymorphism to achieve dynamic behavior.

SESSION 12 — Abstraction & Interfaces

Session Objective

To understand abstraction using abstract classes and interfaces and apply them to design modular programs.

Topics Covered

- Abstract class & abstract methods
- Partial abstraction
- Interface and its characteristics
- Multiple inheritance using interfaces

- Interface default & static methods (Java 8)
- Real-world use cases

Detailed Explanation

**Abstraction hides internal details and exposes only required operations.
Interfaces allow a class to implement multiple behaviors.
Students learn when to use abstract classes vs interfaces.**

Code Example

```
abstract class Shape {

    abstract void area();

}



interface Drawable {

    void draw();

}



class Circle extends Shape implements Drawable {

    void area(){ System.out.println("Area = πr²"); }

    public void draw(){ System.out.println("Drawing Circle"); }

}
```

Hands-on Tasks

- Shape abstract class with Circle, Square
- Implement Multiple interfaces in one class

Assignments (LeetCode)

1. Remove Linked List Elements – https://leetcode.com/problems/remove-linked-list-elements/
2. Reverse Linked List – https://leetcode.com/problems/reverse-linked-list/

Outcome

Students master abstraction and interface usage in designing reusable architectures.

SESSION 13 — Exception Handling Basics

Session Objective

To learn exception categories, handling techniques, and writing safer code using try/catch.

Topics Covered

- **Errors vs Exceptions**
- **Exception hierarchy**
- **Checked vs Unchecked exceptions**
- **try, catch blocks**
- **Multiple catch blocks**
- **Importance of error handling**

Detailed Explanation

Java exceptions help prevent abrupt program termination.
Students explore common exceptions, how to prevent them, and how to gracefully catch errors.

Code Example

```
try {

    int a = 10 / 0;

} catch(ArithmeticException e) {

    System.out.println("Cannot divide by zero");

}
```

Hands-on Tasks

- **Handle divide-by-zero**
- **Handle array index out of bounds**
- **Use multiple catch blocks**

Assignments (LeetCode)

1. Valid Parentheses – https://leetcode.com/problems/valid-parentheses/
2. Min Stack – https://leetcode.com/problems/min-stack/

Outcome

Students write safe and stable programs with proper error-handling.

SESSION 14 - finally, throw, throws & Custom Exceptions (Part 1)

Session Objective

To understand advanced exception handling keywords and create meaningful custom exceptions.

Topics Covered

- finally block
- throw keyword
- throws keyword
- Propagation of exceptions
- Basics of custom exception design

Detailed Explanation

finally block executes regardless of error, ideal for closing resources.
throw manually throws exceptions, while throws indicates potential exceptions in method signature.
Students learn where to use custom exceptions in real applications.

Code Example

```
void validate(int age) throws Exception {

    if(age < 18) throw new Exception("Not eligible");

    else System.out.println("Eligible");

}
```

Hands-on Tasks

- Build AgeValidation custom exception

- Demonstrate throw & throws with file operations
- Practice finally block

Assignments (LeetCode)

1. Implement Queue Using Stacks – https://leetcode.com/problems/implement-queue-using-stacks/
2. Implement Stack Using Queues – https://leetcode.com/problems/implement-stack-using-queues/

Outcome

Students can throw, propagate, and handle exceptions professionally.

SESSION 15 — Custom Exceptions (Part 2)

Session Objective

To design fully custom exception classes for domain-specific validations.

Topics Covered

- Extending Exception class
- Creating meaningful messages
- Using constructors in exceptions
- Throwing custom exceptions in real use cases
- Exception best practices in projects

Detailed Explanation

Students build multiple domain-level exceptions (LoginException, PasswordException).
They learn how custom exceptions improve code readability and correctness.

Code Example

```
class LoginException extends Exception {

    LoginException(String msg){ super(msg); }

}



void login(String user, String pass) throws LoginException {
    if(!user.equals("admin")) throw new LoginException("User invalid");
}
```

Hands-on Tasks

- LoginException
- PasswordStrengthException
- InvalidProductException

Assignments (LeetCode)

1. Climbing Stairs – https://leetcode.com/problems/climbing-stairs/
2. Pascal's Triangle – https://leetcode.com/problems/pascals-triangle/

Outcome

Students confidently design structured exception hierarchies.

SESSION 16 — Byte Streams (FileInputStream & FileOutputStream)

Session Objective

To understand how Java handles binary data through byte streams and learn reading/writing files using `FileInputStream` and `FileOutputStream`.

Topics Covered

- What are byte streams?
- Overview of InputStream & OutputStream
- FileInputStream
- FileOutputStream
- Reading binary data byte by byte
- Writing bytes into files
- Closing streams & resource handling
- Exception handling for files

Detailed Explanation

Byte streams handle raw binary data such as images, audio, or any file type.
Students learn how to open a file, read bytes, write bytes, and manage file integrity.
Importance of closing streams using `finally` or try-with-resources is explained.

Code Example

```
FileInputStream fin = new FileInputStream("input.txt");

int i;

while((i = fin.read()) != -1) {

    System.out.print((char)i);

}

fin.close();
```

Hands-on Tasks

- **Read a file and print its content**
- **Copy a file (input to output)**
- **Count the number of bytes in a file**

Assignments (LeetCode)

1. **Find Pivot Index – https://leetcode.com/problems/find-pivot-index/**
2. **Running Sum of 1D Array – https://leetcode.com/problems/running-sum-of-1d-array/**

Outcome

Students understand binary file handling and efficient stream operations.

SESSION 17 — Character Streams (FileReader, FileWriter, BufferedReader)

Session Objective

To work with text files using character streams and buffered input/output for efficiency.

Topics Covered

- Reader & Writer classes
- FileReader
- FileWriter
- BufferedReader & readLine()
- Writing characters & strings
- Difference between byte and character streams

Detailed Explanation

**Character streams handle text data and automatically handle encoding.
Students learn how to use buffers to read lines efficiently and write content with FileWriter.**

Code Example

```
BufferedReader br = new BufferedReader(new FileReader("test.txt"));

String line;

while((line = br.readLine()) != null){

    System.out.println(line);

}

br.close();
```

Hands-on Tasks

- Read text file line by line
- Count words, lines, characters
- Append data to file

Assignments (LeetCode)

1. Valid Palindrome – https://leetcode.com/problems/valid-palindrome/
2. Reverse String – https://leetcode.com/problems/reverse-string/

Outcome

Students understand efficient text file reading/writing using character streams.

SESSION 18 — Object Serialization & Deserialization

Session Objective

To understand Java's serialization mechanism for storing and transferring objects.

Topics Covered

- **What is serialization?**
- **Serializable interface**
- **ObjectOutputStream**
- **ObjectInputStream**
- **serialVersionUID**
- **Transient keyword**
- **Reading/writing objects to files**

Detailed Explanation

Serialization converts objects into a byte stream.
Students learn when to serialize, how to maintain version control, and how transient fields protect sensitive data (passwords).

Code Example

```
ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("obj.dat"));

out.writeObject(new Student("Amit", 90));

out.close();
```

Hands-on Tasks

- **Serialize Student object**
- **Deserialize it and print data**
- **Use transient keyword**

Assignments (LeetCode)

1. **Balanced Binary Tree – https://leetcode.com/problems/balanced-binary-tree/**
2. **Maximum Depth of Binary Tree – https://leetcode.com/problems/maximum-depth-of-binary-tree/**

Outcome

Students understand object persistence and secure data handling using serialization.

\SESSION 19 — Multithreading Basics (Thread Class)

Session Objective

To introduce multithreading, thread lifecycle, and thread creation using the Thread class.

Topics Covered

- **Process vs Thread**
- **Multithreading need & use cases**
- **Thread lifecycle (NEW → RUNNABLE → RUNNING → BLOCKED → TERMINATED)**
- **Creating threads**
- **start() vs run()**
- **Thread methods (sleep, currentThread)**

Detailed Explanation

Students learn concurrent execution and how multiple threads work together.
Demonstrations differentiate between calling run() manually vs start().

Code Example

```
class MyThread extends Thread {

    public void run(){

        for(int i=1;i<=5;i++) System.out.println(i);

    }

}



new MyThread().start();
```

Hands-on Tasks

- **Create two threads printing sequences**

- Use sleep() for timed delays
- Display current thread name

Assignments (LeetCode)

1. Flood Fill – https://leetcode.com/problems/flood-fill/
2. Number of Islands – https://leetcode.com/problems/number-of-islands/

Outcome

Students can create, run, control, and manage threads effectively.

SESSION 20 — Runnable Interface & Thread Management

Session Objective

To implement threads using Runnable interface and understand why Runnable is preferred over Thread class.

Topics Covered

- Runnable interface
- Implementing run()
- Passing Runnable to Thread constructor
- Thread states
- yield, join methods
- Thread priorities

Detailed Explanation

**Runnable is preferred because Java supports single inheritance.
By separating job (Runnable) and executor (Thread), programs become modular.**

Code Example

```
class Task implements Runnable {

    public void run(){

        System.out.println("Task running");

    }

}
```

```
Thread t = new Thread(new Task());

t.start();
```

Hands-on Tasks

- Create 3 Runnable tasks
- Demonstrate join()
- Demonstrate priorities

Assignments (LeetCode)

1. Rotting Oranges – https://leetcode.com/problems/rotting-oranges/
2. Max Area of Island – https://leetcode.com/problems/max-area-of-island/

Outcome

Students understand industry-standard thread implementation using Runnable.

SESSION 21 — Thread Synchronization

Session Objective

To understand how to prevent race conditions in multithreading using synchronization techniques.

Topics Covered

- Race condition problem
- Critical section
- synchronized keyword
  - synchronized method
  - synchronized block
- Locks & object monitors
- Thread-safe operations

- Real-world examples (Bank withdrawal, Ticket booking)

Detailed Explanation

When two or more threads access shared data simultaneously, inconsistent results may occur.
Synchronization ensures only one thread enters the critical section at a time.
Students learn how Java uses intrinsic locks to manage thread access.

Code Example

```
class Counter {
    int count = 0;
    synchronized void increment(){
        count++;
    }
}
```

Hands-on Tasks

- Demonstrate race condition with non-synchronized code
- Fix it using synchronized keyword
- Simulate bank withdrawal with shared balance

Assignments (LeetCode)

1. House Robber – https://leetcode.com/problems/house-robber/
2. Min Cost Climbing Stairs – https://leetcode.com/problems/min-cost-climbing-stairs/

Outcome

Students understand safe shared-resource handling in multi-threaded programs.

SESSION 22 — Inter-Thread Communication

Session Objective

To understand controlled communication between threads using wait(), notify(), notifyAll().

Topics Covered

- Producer-consumer problem
- Thread communication need
- wait() method
- notify() method
- notifyAll()
- Synchronized + wait/notify combined
- Blocking conditions and thread states

Detailed Explanation

wait() pauses the current thread until another thread notifies it.
notify() wakes one waiting thread; notifyAll() wakes all.
Students learn how to design coordinated thread systems.

Code Example

```
synchronized void produce() throws Exception {
    wait();
}

synchronized void consume() {
    notify();
}
```

Hands-on Tasks

- Implement classic producer-consumer
- Odd-even printing using two threads
- Thread coordination simulation

Assignments (LeetCode)

1. Binary Tree Level Order Traversal – https://leetcode.com/problems/binary-tree-level-order-traversal/
2. Symmetric Tree – https://leetcode.com/problems/symmetric-tree/

Outcome

Students manage multi-thread workflows using proper communication methods.

SESSION 23 — Functional Interfaces (Java 8)

Session Objective

To introduce functional programming concepts and Java's predefined functional interfaces.

Topics Covered

- Introduction to functional programming
- Functional interface concept (SAM)
- @FunctionalInterface annotation
- Predefined functional interfaces:
  - Predicate
  - Function
  - Consumer
  - Supplier
- Use cases and real-world application

Detailed Explanation

A functional interface contains exactly one abstract method, enabling Lambda Expressions.
Students learn how functional interfaces simplify code and improve readability.

Code Example

```
Predicate<Integer> p = n -> n % 2 == 0;
System.out.println(p.test(10)); // true
```

Hands-on Tasks

- Use Predicate for filtering numbers
- Transform list using Function
- Print student list using Consumer

Assignments (LeetCode)

1. **Filter Restaurants by Conditions –** https://leetcode.com/problems/filter-restaurants-by-vegan-friendly-price-and-distance/
2. **Sum of Unique Elements –** https://leetcode.com/problems/sum-of-unique-elements/

Outcome

Students understand Java's functional model and how to use functional interfaces in real programs.

SESSION 24 — Lambda Expressions

Session Objective

To teach lambda expressions and how they provide concise syntax for functional interfaces.

Topics Covered

- **Lambda expression syntax**
- **Why lambdas? Benefits**
- **Lambdas with Predicate, Function, Consumer**
- **Using lambdas with Collections**
- **Capturing variables**
- **Comparator using Lambda**

Detailed Explanation

Lambda expressions allow writing inline implementations of functional interfaces, making the code shorter and more expressive.
Students learn how lambdas work behind the scenes and how Java uses invokedynamic for lambda implementation.

Code Example

```
List<Integer> list = Arrays.asList(1,4,2,3);
list.sort((a,b) -> a - b);
```

Hands-on Tasks

- Sort array using lambda
- Filter even/odd numbers
- Create inline Runnable using lambda

Assignments (LeetCode)

1. Sort Array by Increasing Frequency – https://leetcode.com/problems/sort-array-by-increasing-frequency/
2. Largest Number – https://leetcode.com/problems/largest-number/

Outcome

Students write compact, professional-grade Java code using modern syntax.

SESSION 25 — Method References

Session Objective

To understand method references as a shorthand alternative to lambda expressions.

Topics Covered

- Types of method references
  - Static method reference
  - Instance method reference
  - Constructor reference
- Double colon (::) operator
- Method reference vs lambda
- When to prefer method references

Detailed Explanation

Method references refer to an existing method rather than creating a new lambda.
Students learn how they improve code readability and reduce boilerplate.

Code Example

```
class Util {
    static void printMsg(String s){ System.out.println(s); }
}

Consumer<String> c = Util::printMsg;
c.accept("Hello");
```

Hands-on Tasks

- **Sort using method reference**
- **Use constructor reference to create objects**
- **Replace lambdas with method references**

Assignments (LeetCode)

1. **Sort the People – https://leetcode.com/problems/sort-the-people/**
2. **Kth Largest Element in an Array – https://leetcode.com/problems/kth-largest-element-in-an-array/**

Outcome

Students master concise coding using method references and understand their use cases.

SESSION 26 — Stream API Basics (map, filter, reduce)

Session Objective

To introduce the Java Stream API and teach how map, filter, and reduce operations help process data efficiently in a functional style.

Topics Covered

- **What is Stream? (NOT data structure)**
- **Creating streams (List, Arrays, Stream.of)**
- **Intermediate operations:**
  - **filter()**
  - **map()**

    - sorted()
- Terminal operations:
    - collect()
    - reduce()
    - forEach()
- Lazy evaluation & pipeline execution

Detailed Explanation

Streams allow processing collections in a declarative, functional manner.
Students learn how filter removes unwanted elements, map transforms elements, and reduce combines them to produce a single output.
Performance benefits and immutability in streams are explained.

Code Example

```
List<Integer> nums = Arrays.asList(1,2,3,4,5);

int sum = nums.stream()
              .filter(n -> n % 2 == 1)
              .map(n -> n * n)
              .reduce(0, Integer::sum);

System.out.println(sum);
```

Hands-on Tasks

- Filter even numbers
- Map elements to their squares
- Reduce values to calculate sum or product

Assignments (LeetCode)

1. Unique Number of Occurrences – https://leetcode.com/problems/unique-number-of-occurrences/
2. Find All Numbers Disappeared – https://leetcode.com/problems/find-all-numbers-disappeared-in-an-array/

Outcome

Students can build Stream pipelines and understand functional-style processing.

SESSION 27— Stream API Advanced (Collectors, grouping, sorting)

Session Objective

To master advanced Stream operations like grouping, partitioning, custom sorting, and collecting results into maps or lists.

Topics Covered

- **Collectors.toList(), toSet(), toMap()**
- **groupingBy()**
- **partitioningBy()**
- **Counting, averaging, summing with collectors**
- **Sorting with comparator in Stream**
- **Chaining multiple stream operations**

Detailed Explanation

Advanced collectors allow complex transformation of collections.
Students learn to group employees by department, partition numbers as even/odd, and perform advanced sorting.
Real-world data processing is demonstrated.

Code Example

```
Map<String, List<Student>> grp =
    students.stream()
            .collect(Collectors.groupingBy(s -> s.branch));
```

Hands-on Tasks

- **Group employees by department**

- **Find highest marks using stream**
- **Partition numbers into odd/even lists**

Assignments (LeetCode)

1. **Top K Frequent Elements – https://leetcode.com/problems/top-k-frequent-elements/**
2. **Group Anagrams – https://leetcode.com/problems/group-anagrams/**

Outcome

Students perform complex data transformations using powerful Stream API tools.

SESSION 28— Optional Class

Session Objective

To understand how `Optional` prevents NullPointerException and promotes safer code.

Topics Covered

- **What is Optional?**
- **empty(), of(), ofNullable()**
- **get(), isPresent(), ifPresent()**
- **orElse(), orElseGet(), orElseThrow()**
- **Using Optional in real-world scenarios**

Detailed Explanation

Optional is a container object that may or may not contain a value.
Students learn how to replace null checks with Optional for cleaner and safer code.

Code Example

```
Optional<String> name = Optional.ofNullable(null);
System.out.println(name.orElse("No Name"));
```

Hands-on Tasks

- Use Optional to avoid null checks
- Process optional values using ifPresent
- Convert methods to return Optional

Assignments (LeetCode)

1. Missing Number – https://leetcode.com/problems/missing-number/
2. Contains Duplicate II – https://leetcode.com/problems/contains-duplicate-ii/

Outcome

Students learn null-safe coding practices using Optional.

SESSION 29 — Annotations (Built-in + Custom)

Session Objective

To introduce Java annotations and teach how to define custom annotations used in frameworks like Spring.

Topics Covered

- What is annotation?
- Built-in annotations:
  - @Override
  - @Deprecated
  - @SuppressWarnings
- Meta-annotations:
  - @Target
  - @Retention
  - @Inherited
- Creating custom annotations

- Real use cases

Detailed Explanation

Annotations add metadata to Java programs. They do not change code behavior but help frameworks and compilers.
Students create custom annotations and apply them to classes and methods.

Code Example

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
@interface Test {}

@Test
void display(){
    System.out.println("Testing...");
}
```

Hands-on Tasks

- **Create custom annotation @Info**
- **Process annotation using reflection**
- **Use @Override, @Deprecated**

Assignments (LeetCode)

1. **Find the Difference – https://leetcode.com/problems/find-the-difference/**
2. **Majority Element – https://leetcode.com/problems/majority-element/**

Outcome

Students understand annotation basics required for Spring Framework.

SESSION 30 — Inner Classes & Static Nested Classes

Session Objective

To understand nested class types, encapsulation using inner classes, and practical use cases.

Topics Covered

- Member Inner Class
- Static Nested Class
- Local Inner Class
- Anonymous Inner Class (intro)
- Accessing outer class members
- Why inner classes are useful?

Detailed Explanation

Inner classes are used when one class logically belongs to another.
Static nested classes do not require the outer class instance.
Students learn encapsulation, scoping, and event-based programming.

Code Example

```
class Outer {
    static class Inner {
        void show(){ System.out.println("Inside Inner"); }
    }
}
Outer.Inner obj = new Outer.Inner();
obj.show();
```

Hands-on Tasks

- Create static nested class for utility methods
- Use member inner class for encapsulation
- Explore real uses (e.g., Map.Entry)

Assignments (LeetCode)

1. Binary Tree Paths – https://leetcode.com/problems/binary-tree-paths/
2. Path Sum – https://leetcode.com/problems/path-sum/

Outcome

Students can structure complex classes using inner and nested classes.

SESSION 31 — Anonymous Inner Classes

Session Objective

To teach how to create classes without names for short-lived, one-time use cases such as event handling, comparators, and threading.

Topics Covered

- **What is an anonymous inner class?**
- **Syntax and rules**
- **Anonymous class implementing an interface**
- **Anonymous class extending a class**
- **Use cases:**
  - **Runnable implementation**
  - **Comparator implementation**
  - **UI/event handling (conceptual)**

Detailed Explanation

Anonymous inner classes are created when you need an immediate, single-use implementation without writing a separate class file.
They provide concise syntax but should be used carefully for readability.

Code Example

```
Runnable r = new Runnable() {
    public void run() {
        System.out.println("Running...");
    }
};
new Thread(r).start();
```

Hands-on Tasks

- **Create anonymous Runnable**
- **Write anonymous Comparator for sorting**

- Use anonymous class to override a method instantly

Assignments (LeetCode)

1. Minimum Time Visiting All Points – https://leetcode.com/problems/minimum-time-visiting-all-points/
2. Repeated Substring Pattern – https://leetcode.com/problems/repeated-substring-pattern/

Outcome

Students understand how to build quick, lightweight implementations using anonymous classes.

SESSION 32 — Java Records

Session Objective

To learn modern Java records, which provide concise, immutable data carriers ideal for DTOs & API responses.

Topics Covered

- What is a record?
- Syntax & auto-generated constructor
- Auto-generated methods: equals, hashCode, toString
- Compact constructors
- Validation inside records
- Record immutability
- Using records in collections & APIs

Detailed Explanation

Records reduce boilerplate code dramatically by auto-generating common methods.
Students learn how immutability improves program safety, especially in multi-threaded software and Spring Boot APIs.

Code Example

```java
public record Student(String name, int marks) {}
```

```
Student s = new Student("Amit", 85);
System.out.println(s.name());
```

Hands-on Tasks

- Create Student, Product, and Employee records
- Implement compact constructors
- Use records in Collections

Assignments (LeetCode)

1. Remove Duplicates from Sorted Array – https://leetcode.com/problems/remove-duplicates-from-sorted-array/
2. Majority Element II – https://leetcode.com/problems/majority-element-ii/

Outcome

Students understand concise data modeling with modern Java record features.

SESSION 33— Sealed Classes

Session Objective

To learn how sealed classes restrict which classes can extend them, enabling controlled inheritance hierarchies.

Topics Covered

- sealed keyword
- permits clause
- Inheritance rules (sealed, non-sealed, final)
- When to use sealed classes
- Comparison with enum/abstract class
- Pattern matching with sealed classes

Detailed Explanation

Sealed classes give developers full control over which classes can extend a base class.
Useful in domain-driven design, compiler design, geometry modeling, etc.

Code Example

```
public sealed class Shape permits Circle, Square {}

final class Circle extends Shape {}
non-sealed class Square extends Shape {}
```

Hands-on Tasks

- Create sealed Shape hierarchy
- Use instanceof pattern matching
- Implement sealed class for Payment types

Assignments (LeetCode)

1. Set Mismatch – https://leetcode.com/problems/set-mismatch/
2. Sort Array By Parity – https://leetcode.com/problems/sort-array-by-parity/

Outcome

Students learn controlled inheritance for secure and predictable architecture.

SESSION 34 — Modern Switch Expressions (Java 14+)

Session Objective

To master modern switch syntax using arrow labels, yield, and pattern matching.

Topics Covered

- Traditional switch vs new switch
- Arrow (→) switch syntax
- Multiple labels
- yield keyword
- switch as an expression returning value

- Rewriting complex if-else logic

Detailed Explanation

Modern switch reduces boilerplate code and prevents fall-through issues.
Students learn returning values, cleaner syntax, and real project examples.

Code Example

```
String dayType = switch(day) {
    case 1,2,3,4,5 -> "Weekday";
    case 6,7 -> "Weekend";
    default -> "Invalid";
};
```

Hands-on Tasks

- Create switch-based calculator
- Check month names using switch
- Replace if-else chain using modern switch

Assignments (LeetCode)

1. Evaluate Reverse Polish Notation – https://leetcode.com/problems/evaluate-reverse-polish-notation/
2. Basic Calculator II – https://leetcode.com/problems/basic-calculator-ii/

Outcome

Students can implement concise and powerful logic with modern switch expressions.

SESSION 35 — Collection Framework Overview

Session Objective

To introduce the architecture of Java Collections and understand core interfaces of the framework.

Topics Covered

- Need for Collections
- Limitations of arrays
- Collection hierarchy
- Iterable → Collection → List / Set / Queue
- Map hierarchy
- Generic Collections
- Iterator basics

Detailed Explanation

Java Collections Framework is a unified architecture for storing and manipulating groups of objects.
Students learn the purpose of each interface and how generics ensure type safety.

Code Example

```
Collection<String> list = new ArrayList<>();
list.add("A");
list.add("B");

Iterator<String> it = list.iterator();
while(it.hasNext()) System.out.println(it.next());
```

Hands-on Tasks

- Create List, Set, Queue examples
- Iterate using Iterator & forEach
- Convert array to list

Assignments (LeetCode)

1. Ransom Note – https://leetcode.com/problems/ransom-note/
2. Intersection of Two Arrays II – https://leetcode.com/problems/intersection-of-two-arrays-ii/

Outcome

Students understand the full structure and purpose of the Java Collections Framework.

SESSION 36 — List Interface (ArrayList, LinkedList, Vector, Stack)

Session Objective

To understand List-based data structures, their internal working, performance differences, and real-time usage.

Topics Covered

- List Interface overview
- ArrayList
  - Dynamic resizing
  - Random access
  - Performance (add/get)
- LinkedList
  - Node-based storage
  - Fast insertion/deletion
- Vector (legacy)
- Stack (LIFO)
  - push(), pop(), peek()

Detailed Explanation

**ArrayList is faster for searching; LinkedList is faster for insert/delete.
Students learn use-cases like playlist management, undo/redo using Stack, browser history, etc.**

Code Example

```
List<Integer> list = new ArrayList<>();
list.add(10); list.add(20);

Stack<String> st = new Stack<>();
st.push("A");
System.out.println(st.pop());
```

Hands-on Tasks

- Create an Employee list with CRUD operations
- Implement stack operations
- Convert LinkedList to ArrayList

Assignments (LeetCode)

1. **Design Linked List – https://leetcode.com/problems/design-linked-list/**
2. **Min Stack – https://leetcode.com/problems/min-stack/**

Outcome

Students master List-based structures and use them in real applications.

SESSION 37 — Set Interface (HashSet, LinkedHashSet, TreeSet)

Session Objective

To understand set-based collections used for storing unique elements and performing fast lookups.

Topics Covered

- **HashSet (no duplicates, no order)**
- **LinkedHashSet (insertion order)**
- **TreeSet (sorted order)**
- **Use cases:**
  - **Removing duplicates**
  - **Fast membership check**
  - **Sorted unique data**

Detailed Explanation

Sets rely on hashing (HashSet) or balanced trees (TreeSet).
Students understand why duplicates aren't allowed and how hashing works internally.

Code Example

```
Set<String> set = new HashSet<>();
set.add("A");
set.add("A"); // Duplicate ignored
```

Hands-on Tasks

- **Create unique word extractor**
- **Sort numbers using TreeSet**
- **Compare HashSet vs LinkedHashSet**

Assignments (LeetCode)

1. Happy Number – https://leetcode.com/problems/happy-number/
2. Longest Consecutive Sequence – https://leetcode.com/problems/longest-consecutive-sequence/

Outcome

Students understand uniqueness constraints, hashing, and sorted sets.

SESSION 38 (Page 38) — Map Interface (HashMap, LinkedHashMap, TreeMap, Hashtable)

Session Objective

To master key–value data structures used extensively in real-world Java applications.

Topics Covered

- Map Interface
- HashMap (unordered, fast lookup)
- LinkedHashMap (insertion order)
- TreeMap (sorted keys)
- Hashtable (synchronized legacy map)
- Collision handling basics
- Practical use cases:
    - Dictionary
    - Frequency counter
    - Caching

Detailed Explanation

Students understand hashing, how Map stores entries, and difference between load factor, buckets, and collisions.
Real-life uses like user sessions, API caching, and counting frequencies are explored.

Code Example

```
Map<String, Integer> freq = new HashMap<>();
freq.put("apple", 2);
freq.put("banana", 1);
```

Hands-on Tasks

- Word-frequency counter
- Student mark record (TreeMap)
- LinkedHashMap-based LRU demo

Assignments (LeetCode)

1. Two Sum – https://leetcode.com/problems/two-sum/
2. Word Pattern – https://leetcode.com/problems/word-pattern/

Outcome

Students confidently use Maps for fast and efficient data lookups.

SESSION 39 — Comparable, Comparator & Sorting

Session Objective

To perform custom sorting on objects using Comparable and Comparator.

Topics Covered

- Natural ordering
- Comparable interface
- compareTo() implementation
- Custom sorting using Comparator
- Lambda-based comparators
- Sorting collections of objects

Detailed Explanation

Comparable defines natural order (e.g., sort Students by roll).
Comparator defines custom orders (sort by marks, name, city).
Students learn real placement-style sorting.

Code Example

```
class Student implements Comparable<Student> {
    int marks;
    public int compareTo(Student s){
        return this.marks - s.marks;
    }
}
```

Hands-on Tasks

- Sort Product list by price
- Sort Students by name then marks
- Use Comparator.comparing()

Assignments (LeetCode)

1. Merge Intervals – https://leetcode.com/problems/merge-intervals/
2. Sort Characters by Frequency – https://leetcode.com/problems/sort-characters-by-frequency/

Outcome

Students can implement both natural and custom ordering in professional applications.

SESSION 40 — Spring Core & Dependency Injection (Practical)

Session Objective

To introduce Spring Framework basics and implement Dependency Injection (DI) using annotations.

Topics Covered

- What is Spring Framework?
- Features of Spring
- Inversion of Control (IoC)
- Dependency Injection (Constructor + Setter)
- Beans & Bean Lifecycle
- ApplicationContext
- @Component, @Autowired

Detailed Explanation

Students learn how Spring creates and manages objects, injecting dependencies automatically.
They understand bean scopes and how IoC reduces boilerplate code.

Practical Work (No LeetCode)

- Create a Spring project
- Create a service class + component class
- Apply @Autowired to inject dependency
- Print outputs from beans

Outcome

Students build their first Spring DI-based mini application.

SESSION 41— Spring Framework Advanced (AOP + Bean Scopes) — Practical

Session Objective

To understand Aspect-Oriented Programming for cross-cutting concerns and advanced bean scopes.

Topics Covered

- What is AOP?
- Aspect, JoinPoint, Advice
- @Before, @After, @Around
- Logging aspect example
- Bean scopes:
  - singleton
  - prototype
  - request
  - session

Detailed Explanation

AOP handles logging, security, transactions without touching business code.
Students implement logging on method calls using @Aspect.

Practical Work

- Create @Aspect with @Before advice
- Add logging to service methods
- Test bean scopes in Spring

Outcome

Students implement AOP professionally for logging and monitoring.

SESSION 42 — Spring Boot, REST API & Mini Project (Practical)

Session Objective

To build a complete CRUD-based REST API using Spring Boot.

Topics Covered

- Spring Boot structure
- Starters, auto configuration
- Creating controllers
- @RestController, @RequestMapping
- GET, POST, PUT, DELETE
- @RequestBody, @PathVariable
- ResponseEntity
- Exception handling
- Running & testing APIs (Postman)

Detailed Explanation

Students create a complete backend service using layered architecture (Controller → Service → Repository).
Focus is on real-time API development.

Practical Work (Mini Project)

Project: Student Management REST API

- Add Student
- Get Student
- Update Student

- **Delete Student**
- **Custom exceptions**
- **JSON request & response**
- **Build runnable JAR**

Outcome

Students complete the course with a real-world REST API project using Spring Boot.