OOPS WITH JAVA LAB MANUAL - 2025 SCHEME

1BCS302

Prepared By:

Mr. Praveen Kumar K C
Assistant Professor
Dept. of ISE
CIT Gubbi

Approved By:

Dr. Thara D K
Professor & Head
Dept. of ISE
CIT Gubbi

JAVA PROGRAMMING LABORATORY

LAB MANUAL

General Instructions

This manual accompanies the Java Programming Laboratory record for Batch B1. Every lesson plan (LP) below includes the Aim, the complete Java source code, a sample console output, and the step-by-step execution procedure. Students must maintain a separate observation book with problem statements, algorithms and program write-ups, and use this manual as a reference while executing programs in the laboratory.

- LP.1 is executed purely with the JDK command-line tools (javac and java) in the Windows Command Prompt, without any IDE, so that students understand the raw compile-and-run cycle.
- LP.2 through LP.11 are executed using the Eclipse IDE, following the standard Java Project → Package → Class → Run workflow described in Part B below.
- LP.12 is a Micro Project to be completed in teams of 2–3 students; guidelines are provided instead of a single fixed program.
- LP.13 is a written/practical Test and therefore has no associated program.

Part A — Execution Procedure in Native Windows Environment (used for LP.1)

Use the following procedure to compile and run a plain Java program using only the JDK, without Eclipse or any other IDE.

1. Ensure the JDK is installed. Open Command Prompt and type: java -version and javac -version to confirm both commands are recognised. If not recognised, install the JDK and add its bin folder to the PATH environment variable.
2. Open Notepad (or any plain text editor). Type the Java program exactly as given, making sure the public class name matches the file name you will save.
3. Save the file with the exact name ClassName.java (e.g., HelloWorld.java) inside a working folder, e.g., C:\JavaLab\. In Notepad's Save dialog, set 'Save as type' to 'All Files' so it does not get saved as HelloWorld.java.txt.
4. Open Command Prompt (Win + R, type cmd, press Enter).
5. Navigate to the working folder using: cd C:\JavaLab
6. Compile the program: javac HelloWorld.java — this creates HelloWorld.class in the same folder if there are no errors.
7. Run the compiled program: java HelloWorld
8. Observe the output printed on the console. If the program uses Scanner for input, type the required values when prompted and press Enter.

Part B — Execution Procedure Using Eclipse IDE (used for LP.2 to LP.11)

Use the following general procedure for every program from LP.2 to LP.11. Program-specific notes (such as creating a package) are given under each lesson plan where needed.

1. Open Eclipse IDE and select (or create) a workspace, e.g., C:\JavaLab\workspace.
2. Go to File → New → Java Project. Enter a Project Name (e.g., JavaLabB1) and click Finish. If prompted to open the Java perspective, click Open.
3. In the Package Explorer, right-click on the src folder of the project → New → Class.
4. Enter the class Name exactly as given in the program (e.g., MatrixAddition). Tick 'public static void main(String[] args)' if you want Eclipse to generate the stub, then click Finish.
5. Delete any auto-generated stub content and type/paste the complete program given in this manual into the editor. Save using Ctrl+S (a red asterisk on the tab disappears once saved).
6. If the editor shows red-underline errors, check for missing semicolons, unmatched braces, or an incorrect class name vs. file name — Eclipse requires the public class name to match the file name.
7. Right-click anywhere in the editor (or on the class in Package Explorer) → Run As → Java Application.
8. The Console view opens at the bottom of Eclipse and shows the program's output. For programs that use Scanner to read input, click inside the Console view and type the input values, pressing Enter after each.
9. To run the program again after making changes, save the file and click the green Run button (or press Ctrl+F11).

LP.1 – Sample JAVA Programming

Aim: To write, compile and execute a simple Java program using the native JDK command-line tools (javac and java) in the Windows Command Prompt, and to understand the basic structure of a Java program (class, main method, statements).

Program: HelloWorld.java

HelloWorld.java

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Welcome to Java Programming Laboratory");
        System.out.println("Hello, World!");

        int a = 10, b = 20;
        int sum = a + b;
        System.out.println("Sum of " + a + " and " + b + " is " + sum);
    }
}
```

Sample Output

Sample Output (Console)

```
Welcome to Java Programming Laboratory
Hello, World!
Sum of 10 and 20 is 30
```

Execution Procedure

- Type the program in Notepad and save it as HelloWorld.java in C:\JavaLab\.
- Open Command Prompt and change directory to C:\JavaLab\ using cd C:\JavaLab.
- Compile using: javac HelloWorld.java
- Run using: java HelloWorld
- Verify the output matches the sample output shown above.

LP.2 – Add two matrices of suitable order N

Aim: To develop a Java program that reads two square matrices of order N from the user and computes their sum, using two-dimensional arrays and nested loops.

Program: MatrixAddition.java

MatrixAddition.java

```
import java.util.Scanner;

public class MatrixAddition {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the order N of the square matrices: ");
        int n = sc.nextInt();

        int[][] a = new int[n][n];
        int[][] b = new int[n][n];
        int[][] c = new int[n][n];

        System.out.println("Enter elements of first matrix (" + n + " x " + n + "):");
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                a[i][j] = sc.nextInt();

        System.out.println("Enter elements of second matrix (" + n + " x " + n + "):");
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                b[i][j] = sc.nextInt();

        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                c[i][j] = a[i][j] + b[i][j];

        System.out.println("Sum of the two matrices:");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++)
                System.out.print(c[i][j] + "\t");
            System.out.println();
        }
        sc.close();
    }
}
```

Sample Output

Sample Output (Console)

```
Enter the order N of the square matrices: 2
Enter elements of first matrix (2 x 2):
1 2 3 4
Enter elements of second matrix (2 x 2):
5 6 7 8
Sum of the two matrices:
6     8
10    12
```

Execution Procedure

- Create a class MatrixAddition in Eclipse as described in Part B and paste the program.
- Run As → Java Application.
- In the Console, type the order N, press Enter, then type all elements of the first matrix (separated by spaces or Enter), then all elements of the second matrix.
- Observe the resulting sum matrix printed row-wise.

LP.3 – Stack class to hold a maximum of 10 integers

Aim: To develop a Stack class (array-based) that can hold a maximum of 10 integers, with push, pop and display methods, and a main method that demonstrates stack operations including overflow and underflow handling.

Program: StackDemo.java

StackDemo.java

```
class MyStack {
    private int[] stack;
    private int top;
    private final int MAX = 10;

    MyStack() {
        stack = new int[MAX];
        top = -1;
    }

    void push(int value) {
        if (top == MAX - 1) {
            System.out.println("Stack Overflow! Cannot push " + value);
            return;
        }
        stack[++top] = value;
        System.out.println(value + " pushed into stack");
    }

    int pop() {
        if (top == -1) {
            System.out.println("Stack Underflow! Stack is empty");
            return -1;
        }
        return stack[top--];
    }

    void display() {
        if (top == -1) {
            System.out.println("Stack is empty");
            return;
        }
        System.out.print("Stack elements (top to bottom): ");
        for (int i = top; i >= 0; i--)
            System.out.print(stack[i] + " ");
        System.out.println();
    }
}

public class StackDemo {
    public static void main(String[] args) {
        MyStack s = new MyStack();

        s.push(10);
        s.push(20);
        s.push(30);
        s.display();

        System.out.println("Popped element: " + s.pop());
```

```
        s.display();
    }
}
```

Sample Output

```
Sample Output (Console)
10 pushed into stack
20 pushed into stack
30 pushed into stack
Stack elements (top to bottom): 30 20 10
Popped element: 30
Stack elements (top to bottom): 20 10
```

Execution Procedure

- Create a single class file StackDemo.java in Eclipse containing both the MyStack class and the public StackDemo class (only one public class per file, matching the file name).
- Run As → Java Application and observe the push, display and pop operations in the Console.
- Optionally push more than 10 values to observe the Stack Overflow message, and pop from an empty stack to observe Stack Underflow.

LP.4 – Student class with USN, Name, Branch, Phone

Aim: To create a Java class Student with USN, Name, Branch and Phone as instance variables, create N Student objects, and print their details in a tabular format with suitable headings.

Program: StudentDemo.java

StudentDemo.java

```
class Student {
    String usn;
    String name;
    String branch;
    String phone;

    Student(String usn, String name, String branch, String phone) {
        this.usn = usn;
        this.name = name;
        this.branch = branch;
        this.phone = phone;
    }
}

public class StudentDemo {
    public static void main(String[] args) {
        int n = 3;
        Student[] students = new Student[n];

        students[0] = new Student("1AB21CS001", "Arjun Rao", "CSE", "9900011111");
        students[1] = new Student("1AB21IS002", "Bhavya S",  "ISE", "9900022222");
        students[2] = new Student("1AB21EC003", "Chetan K",  "ECE", "9900033333");

        System.out.printf("%-15s%-15s%-8s%-12s%n", "USN", "Name", "Branch", "Phone");
        System.out.println("-----------------------------------------------------------");
        for (Student s : students) {
            System.out.printf("%-15s%-15s%-8s%-12s%n", s.usn, s.name, s.branch, s.phone);
        }
    }
}
```

Sample Output

Sample Output (Console)

```
USN            Name           Branch  Phone
-----------------------------------------------------------
1AB21CS001     Arjun Rao      CSE     9900011111
1AB21IS002     Bhavya S       ISE     9900022222
1AB21EC003     Chetan K       ECE     9900033333
```

Execution Procedure

- Create StudentDemo.java in Eclipse with the Student class and the public StudentDemo class in the same file.
- Run As → Java Application.
- Observe the tabular output with column headings USN, Name, Branch and Phone.

- To take N as user input instead of a fixed value of 3, add a Scanner, read N, and create the Student objects inside a loop.

LP.5 – Shape class with Circle, Triangle, Square subclasses (Polymorphism)

Aim: To develop a Java program with a base class Shape and three subclasses — Circle, Triangle and Square - each overriding draw() and erase() methods, and to demonstrate runtime polymorphism using a Shape reference array.

Program: ShapeDemo.java

ShapeDemo.java

```
abstract class Shape {
    abstract void draw();
    abstract void erase();
}

class Circle extends Shape {
    void draw()  { System.out.println("Drawing a Circle"); }
    void erase() { System.out.println("Erasing a Circle"); }
}

class Triangle extends Shape {
    void draw()  { System.out.println("Drawing a Triangle"); }
    void erase() { System.out.println("Erasing a Triangle"); }
}

class Square extends Shape {
    void draw()  { System.out.println("Drawing a Square"); }
    void erase() { System.out.println("Erasing a Square"); }
}

public class ShapeDemo {
    public static void main(String[] args) {
        Shape[] shapes = new Shape[3];
        shapes[0] = new Circle();
        shapes[1] = new Triangle();
        shapes[2] = new Square();

        for (Shape s : shapes) {
            s.draw();
            s.erase();
            System.out.println();
        }
    }
}
```

Sample Output

Sample Output (Console)

```
Drawing a Circle
Erasing a Circle

Drawing a Triangle
Erasing a Triangle

Drawing a Square
Erasing a Square
```

Execution Procedure

- Create ShapeDemo.java in Eclipse containing the abstract Shape class, the three subclasses, and the public ShapeDemo class.
- Run As → Java Application.
- Observe that although the array is declared of type Shape, calling s.draw() invokes the overridden method of the actual object (Circle, Triangle or Square) at runtime — this is polymorphism (dynamic method dispatch).

LP.6 – Resizable interface implemented by Rectangle

Aim: To create an interface Resizable with methods resizeWidth(int width) and resizeHeight(int height), and a class Rectangle that implements this interface to demonstrate how an interface allows an object to be resized.

Program: ResizableDemo.java

ResizableDemo.java

```
interface Resizable {
    void resizeWidth(int width);
    void resizeHeight(int height);
}

class Rectangle implements Resizable {
    int width;
    int height;

    Rectangle(int width, int height) {
        this.width = width;
        this.height = height;
    }

    public void resizeWidth(int width) {
        this.width = width;
        System.out.println("Width resized to: " + width);
    }

    public void resizeHeight(int height) {
        this.height = height;
        System.out.println("Height resized to: " + height);
    }

    void display() {
        System.out.println("Rectangle -> Width: " + width + ", Height: " + height);
    }
}

public class ResizableDemo {
    public static void main(String[] args) {
        Rectangle r = new Rectangle(10, 5);
        System.out.println("Before resizing:");
        r.display();

        r.resizeWidth(20);
        r.resizeHeight(15);

        System.out.println("After resizing:");
        r.display();
    }
}
```

Sample Output

Sample Output (Console)

```
Before resizing:
Rectangle -> Width: 10, Height: 5
Width resized to: 20
```

```
Height resized to: 15
After resizing:
Rectangle -> Width: 20, Height: 15
```

Execution Procedure

- Create ResizableDemo.java in Eclipse containing the Resizable interface, the Rectangle class and the public ResizableDemo class.
- Run As → Java Application and observe the width/height values before and after resizing.

LP.7 – Create and use a package named mypack

Aim: To create a user-defined package named mypack containing a class with reusable methods, and to import and use that package from another class, demonstrating Java's package mechanism for code organisation and reuse.

Program: mypack/MyPackClass.java

mypack/MyPackClass.java

```
package mypack;

public class MyPackClass {

    public void display() {
        System.out.println("This method is from package mypack");
    }

    public int square(int n) {
        return n * n;
    }
}
```

Program: PackageDemo.java

PackageDemo.java

```
import mypack.MyPackClass;

public class PackageDemo {
    public static void main(String[] args) {
        MyPackClass obj = new MyPackClass();
        obj.display();
        System.out.println("Square of 5 is: " + obj.square(5));
    }
}
```

Sample Output

Sample Output (Console)

```
This method is from package mypack
Square of 5 is: 25
```

Execution Procedure

- In Eclipse, right-click src → New → Package, and name it mypack. Click Finish.
- Right-click the mypack package → New → Class, name it MyPackClass, and paste the mypack/MyPackClass.java code (Eclipse fills in the package mypack; line automatically).
- Right-click src (the default package, not mypack) → New → Class, name it PackageDemo, and paste the PackageDemo.java code, which imports mypack.MyPackClass.
- Right-click PackageDemo.java → Run As → Java Application.
- Observe that methods of MyPackClass, defined inside package mypack, are successfully used from the default package after importing it.

LP.8 – Custom exception for DivisionByZero (try, catch, throw, finally)

Aim: To create a user-defined exception class DivisionByZeroException and use try, catch, throw and finally blocks to detect and handle a division-by-zero condition gracefully.

Program: CustomExceptionDemo.java

CustomExceptionDemo.java

```
class DivisionByZeroException extends Exception {
    DivisionByZeroException(String message) {
        super(message);
    }
}

public class CustomExceptionDemo {

    static int divide(int a, int b) throws DivisionByZeroException {
        if (b == 0) {
            throw new DivisionByZeroException("Cannot divide " + a + " by zero");
        }
        return a / b;
    }

    public static void main(String[] args) {
        int[] numerators   = {10, 20};
        int[] denominators = {2, 0};

        for (int i = 0; i < numerators.length; i++) {
            try {
                int result = divide(numerators[i], denominators[i]);
                System.out.println("Result: " + result);
            } catch (DivisionByZeroException e) {
                System.out.println("Custom Exception Caught: " + e.getMessage());
            } finally {
                System.out.println("Finally block executed for iteration " + (i + 1));
                System.out.println();
            }
        }
    }
}
```

Sample Output

Sample Output (Console)

```
Result: 5
Finally block executed for iteration 1

Custom Exception Caught: Cannot divide 20 by zero
Finally block executed for iteration 2
```

Execution Procedure

- Create CustomExceptionDemo.java in Eclipse containing the DivisionByZeroException class and the public CustomExceptionDemo class.
- Run As → Java Application.

- Observe that for the first pair (10, 2) the division succeeds and finally still executes; for the second pair (20, 0) the custom exception is thrown, caught, and finally still executes — showing finally always runs.

LP.9 – MyThread class demonstrating concurrent execution

Aim: To create a class MyThread that extends Thread, with a constructor that calls the base class constructor using super(), and a run() method. To start the thread using start() and observe that the main thread and the child thread execute concurrently.

Program: ThreadDemo.java

ThreadDemo.java

```
class MyThread extends Thread {

    MyThread(String name) {
        super(name);   // calls Thread(String name) constructor
    }

    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println(getName() + " -> count: " + i);
            try {
                Thread.sleep(200);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

public class ThreadDemo {
    public static void main(String[] args) {
        MyThread t = new MyThread("Child-Thread");
        t.start();   // starts the child thread; run() executes concurrently

        for (int i = 1; i <= 5; i++) {
            System.out.println("Main-Thread -> count: " + i);
            try {
                Thread.sleep(200);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Sample Output

Sample Output (Console)

```
Main-Thread -> count: 1
Child-Thread -> count: 1
Main-Thread -> count: 2
Child-Thread -> count: 2
Main-Thread -> count: 3
Child-Thread -> count: 3
Main-Thread -> count: 4
Child-Thread -> count: 4
Main-Thread -> count: 5
Child-Thread -> count: 5
```

```
Note: The exact interleaving of "Main-Thread" and "Child-Thread"
lines may vary between runs, since both threads are scheduled
concurrently by the JVM/OS.
```

Execution Procedure

- Create ThreadDemo.java in Eclipse containing the MyThread class (extending Thread) and the public ThreadDemo class.
- Run As → Java Application.
- Observe that lines from Main-Thread and Child-Thread appear interleaved in the Console, confirming that both threads run concurrently once start() is called.
- Run the program a few times and note that the exact order of interleaving may differ between runs — this is expected with concurrent threads.

LP.10 – Enumeration Department (CSE, ISE, ECE, MECH) using valueOf()

Aim: To develop a Java program using an enumeration Department containing the constants CSE, ISE, ECE and MECH; to accept a department name as input from the user and display the selected department using the Enum.valueOf() method.

Program: EnumDemo.java

EnumDemo.java

```
import java.util.Scanner;

enum Department {
    CSE, ISE, ECE, MECH
}

public class EnumDemo {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter department name (CSE/ISE/ECE/MECH): ");
        String input = sc.next().toUpperCase();

        try {
            Department dept = Department.valueOf(input);
            System.out.println("Selected Department: " + dept);
        } catch (IllegalArgumentException e) {
            System.out.println("Invalid department name entered! Please enter one of: CSE,
ISE, ECE, MECH");
        }

        sc.close();
    }
}
```

Sample Output

Sample Output (Console)

```
Enter department name (CSE/ISE/ECE/MECH): ise
Selected Department: ISE
```

Execution Procedure

- Create EnumDemo.java in Eclipse containing the enum Department and the public EnumDemo class.
- Run As → Java Application.
- In the Console, type a department name such as CSE, ISE, ECE or MECH (any case) and press Enter.
- Run again with an invalid value (e.g., 'CIVIL') to observe the IllegalArgumentException being caught and the friendly message being printed.

LP.11 – Wrapper classes: marks as strings, autoboxing/unboxing, total and average

Aim: To develop a Java program that reads the marks of three subjects as Strings, converts them into wrapper (Integer) objects, calculates the total and average marks using autoboxing and unboxing, and displays the result.

Program: WrapperDemo.java

WrapperDemo.java

```
import java.util.Scanner;

public class WrapperDemo {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] marksStr = new String[3];

        System.out.println("Enter marks of 3 subjects (as strings, separated by Enter):");
        for (int i = 0; i < 3; i++) {
            marksStr[i] = sc.next();
        }

        Integer[] marks = new Integer[3];
        int total = 0;

        for (int i = 0; i < 3; i++) {
            marks[i] = Integer.parseInt(marksStr[i]);  // String -> int -> autoboxed to
Integer
            total += marks[i];                          // Integer unboxed to int for addition
        }

        double average = total / 3.0;

        System.out.println("Marks entered : " + marks[0] + ", " + marks[1] + ", " + marks[2]);
        System.out.println("Total marks   : " + total);
        System.out.println("Average marks : " + average);

        sc.close();
    }
}
```

Sample Output

Sample Output (Console)

```
Enter marks of 3 subjects (as strings, separated by Enter):
78
85
91
Marks entered : 78, 85, 91
Total marks   : 254
Average marks : 84.66666666666667
```

Execution Procedure

- Create WrapperDemo.java in Eclipse containing the public WrapperDemo class.
- Run As → Java Application.

- In the Console, type each subject's marks (as text) one at a time, pressing Enter after each value.
- Observe how the String values are parsed into int, autoboxed into Integer objects for storage in the array, and unboxed back to int during the total calculation.

LP.12 – Micro Project

Aim: A team of two to three students shall design and develop a small Java project and submit a brief report, based on the concepts covered in LP.1 to LP.11 (classes & objects, inheritance, polymorphism, interfaces, packages, exception handling, multithreading, enumerations, and wrapper classes/autoboxing).

Guidelines

- Form a team of 2–3 students. The course instructor will assign or approve the specific problem statement for each team.
- The project should use at least 4–5 of the core concepts covered in LP.2 to LP.11 (e.g., classes, inheritance/polymorphism, an interface, exception handling, and collections/enums/threads where applicable).
- Develop the project in Eclipse as a proper Java Project with meaningful package names, class names and comments.
- Prepare a brief report (3–5 pages) containing: Problem Statement, Objectives, Class Diagram/Design, Source Code listing, Sample Output (screenshots), and a short Conclusion.
- Each team member should be able to explain every module of the project during evaluation, since marks are individual even though development is in a team.

Sample Project Ideas

- Student Result Management System – classes for Student and Subject, inheritance for different student categories, exception handling for invalid marks, and file/array-based storage.
- Library Management System – classes for Book and Member, interfaces for Borrowable items, custom exceptions for BookNotAvailable, and enums for book categories.
- Bank Account Simulator – base class Account with SavingsAccount and CurrentAccount subclasses (polymorphism), custom exception for InsufficientFunds, and a thread simulating periodic interest calculation.
- Employee Payroll System – classes for Employee and Department (enum), interface Payable implemented by different employee types, and wrapper classes used for salary computations.

Execution / Evaluation Procedure

1. Each team imports/opens its Java Project in Eclipse in the laboratory during the evaluation slot.
2. The team runs the project (Run As → Java Application) and demonstrates the working modules live to the instructor.
3. The team submits the printed/soft-copy report along with the project source code (exported as a zip or shared via the course repository).

LP.13 – Test – 1

This session is a written/practical internal Test covering the concepts and programs of LP.1 to LP.12. There is no fixed program associated with this lesson plan; students will be asked to write, trace, debug and/or execute a program (or answer questions) drawn from any of the topics covered so far, under examination conditions.

Guidelines for Students

- Revise all programs from LP.1 to LP.11 along with their underlying concepts (arrays, classes & objects, inheritance, polymorphism, interfaces, packages, exception handling, threads, enumerations, wrapper classes).
- Be prepared to compile and execute a program independently, either using the native Windows Command Prompt (javac/java) or Eclipse, as instructed by the examiner.
- Carry the completed laboratory observation book / record for reference as permitted by the course instructor.