JAVA

1. OOPs concepts in Java – 1

a . Write a program to create a class and implement a default, overloaded and copy Constructor.

<u>Sol.</u>

```
public class Car {
   // Instance variables
   private String model;
   private String color;

   // Default Constructor
   public Car() {
      this.model = "Unknown";
      this.color = "White";
      System.out.println("Default Constructor Called");
   }

   // Overloaded Constructor
   public Car(String model, String color) {
      this.model = model;
      this.color = color;
      System.out.println("Overloaded Constructor Called");
   }

   // Copy Constructor
   public Car(Car otherCar) {
      this.model = otherCar.model;
      this.color = otherCar.color;
      System.out.println("Copy Constructor Called");
   }
   // Display method
   public void displayInfo() {
      System.out.println("Car Model: " + model);
      System.out.println("Car Color: " + color);
      System.out.println();
   }
// Main method to test the constructors
    public static void main(String[] args) {
          // Using default constructor
        Car car1 = new Car();
        car1.displayInfo();
         // Using overloaded constructor
        Car car2 = new Car("Toyota", "Blue");
        car2.displayInfo();
         // Using copy constructor
        Car car3 = new Car(car2);
        car3.displayInfo();
    }
}
```

---

b. Write a Java program to create a class and implement the concepts of Method Overloading

<u>Sol.</u>

```
public class Calculator {
    // Method with two integer parameters
    public int add(int a, int b) {
        System.out.println("Method with two integer parameters");
        return a + b;
    }
    // Method with three integer parameters - overloaded based on
number of parameters
    public int add(int a, int b, int c) {
        System.out.println("Method with three integer parameters");
        return a + b + c;
    }
    // Method with two double parameters - overloaded based on
parameter type
    public double add(double a, double b) {
        System.out.println("Method with two double parameters");
        return a + b;
    }
    // Method with mixed parameter types - overloaded based on
parameter type
    public double add(int a, double b) {
        System.out.println("Method with integer and double
parameters");
        return a + b;
    }

    // Method with parameters in different order - overloaded based on
parameter order
    public double add(double a, int b) {
        System.out.println("Method with double and integer
parameters");
        return a + b;
    }

    public static void main(String[] args) {
        Calculator calc = new Calculator();

        // Testing different overloaded methods
        System.out.println("Result: " + calc.add(5, 10));
        System.out.println("Result: " + calc.add(5, 10, 15));
        System.out.println("Result: " + calc.add(5.5, 10.5));
        System.out.println("Result: " + calc.add(5, 10.5));
        System.out.println("Result: " + calc.add(5.5, 10));
    }
}
```

c . Write a Java program to create a class and implement the concepts of Static methods

Sol.

```
public class MathUtility {
    // Static variable that can be accessed directly using the class name
    public static final double PI = 3.14159;
```

```
// Static counter to track how many times methods have been called
private static int methodCallCount = 0;

// Instance variable (non-static)
private String utilityName;

// Constructor
public MathUtility(String name) {
    this.utilityName = name;
}

// Static method to calculate the square of a number
public static int square(int num) {
    methodCallCount++;
    return num * num;
}

// Static method to calculate the cube of a number
public static int cube(int num) {
    methodCallCount++;
    return num * num * num;
}

// Static method to calculate the area of a circle
public static double calculateCircleArea(double radius) {
    methodCallCount++;
    return PI * radius * radius;
}

// Static method to get the count of method calls
public static int getMethodCallCount() {
    return methodCallCount;
}

// Static method to reset the counter
public static void resetMethodCallCount() {
    methodCallCount = 0;
```

```
        System.out.println("Method call counter has been reset to 0");
    }

    // Regular instance method (non-static)
    public void displayInfo() {
        System.out.println("Utility Name: " + this.utilityName);
        System.out.println("Total method calls: " + methodCallCount);
    }

    public static void main(String[] args) {
        // Accessing static methods without creating an instance
        System.out.println("Square of 5: " + MathUtility.square(5));
        System.out.println("Cube of 3: " + MathUtility.cube(3));
        System.out.println("Area of circle with radius 7: " +
MathUtility.calculateCircleArea(7));
        System.out.println("Total method calls: " +
MathUtility.getMethodCallCount());

        // Creating instances of the class
        MathUtility util1 = new MathUtility("Standard Math Tools");
        MathUtility util2 = new MathUtility("Advanced Math Tools");

        // Calling static methods using instances (not recommended but
valid)
        System.out.println("Square of 4 using instance: " +
util1.square(4));

        // Calling instance methods
        util1.displayInfo();
        util2.displayInfo();

        // Resetting the static counter
        MathUtility.resetMethodCallCount();

        // Demonstrating that static variables are shared across all
instances
        System.out.println("Square of 6: " + MathUtility.square(6));
        util1.displayInfo();
        util2.displayInfo();
    }
}
```

2. OOPs concepts in Java – 2

a. Write a Java program to implement the concepts of Inheritance and Method overriding

Sol.

```
// Parent class
class Animal {
    // Common property
    String name;

    // Constructor
    public Animal(String name) {
        this.name = name;
    }

    // Common method that will be overridden
    public void makeSound() {
        System.out.println("Animal makes a sound");
    }

    // Method to display information
    public void displayInfo() {
        System.out.println("I am an animal named " + name);
    }
}

// Child class inheriting from Animal
class Dog extends Animal {
    // Additional property
    String breed;

    // Constructor
    public Dog(String name, String breed) {
        super(name);  // Call parent constructor
        this.breed = breed;
    }

    // Method overriding - changing the behavior of the parent method
    @Override
    public void makeSound() {
        System.out.println(name + " says: Woof! Woof!");
    }

    // Method overriding - enhancing the parent method
    @Override
    public void displayInfo() {
        super.displayInfo();  // Call the parent method
        System.out.println("I am a " + breed + " dog");
    }
}

// Another child class
class Cat extends Animal {
    // Constructor
    public Cat(String name) {
        super(name);
    }

    // Method overriding
    @Override
    public void makeSound() {
        System.out.println(name + " says: Meow! Meow!");
    }
}

// Main class to test our code
public class SimpleInheritanceDemo {
    public static void main(String[] args) {
        // Create an Animal object
        Animal myAnimal = new Animal("Generic Animal");
        myAnimal.displayInfo();
        myAnimal.makeSound();

        System.out.println("\n--------------\n");

        // Create a Dog object
        Dog myDog = new Dog("Buddy", "Golden Retriever");
        myDog.displayInfo();  // Calls overridden method
        myDog.makeSound();    // Calls overridden method

        System.out.println("\n--------------\n");

        // Create a Cat object
        Cat myCat = new Cat("Whiskers");
        myCat.displayInfo();  // Uses parent's method
        myCat.makeSound();    // Calls overridden method

        System.out.println("\n--------------\n");

        // Demonstrating polymorphism
        System.out.println("Polymorphism Example:");
        Animal animal1 = new Dog("Rex", "German Shepherd");
        Animal animal2 = new Cat("Felix");

        animal1.makeSound();  // Calls Dog's method
        animal2.makeSound();  // Calls Cat's method
    }
}
```

---

b. Write a Java program to implement the concepts of Abstract classes and methods

Sol.

```
// Abstract class
abstract class Shape {
    // Regular attribute
    protected String color;

    // Constructor
    public Shape(String color) {
        this.color = color;
    }

    // Regular method (non-abstract)
    public String getColor() {
        return color;
    }

    // Abstract method - must be implemented by concrete subclasses
    public abstract double calculateArea();

    // Abstract method
    public abstract double calculatePerimeter();

    // Regular method that uses abstract methods
    public void displayInfo() {
        System.out.println("Shape Color: " + color);
        System.out.println("Area: " + calculateArea());
        System.out.println("Perimeter: " + calculatePerimeter());
    }
}

// Concrete subclass of Shape
class Circle extends Shape {
    private double radius;

    public Circle(String color, double radius) {
        super(color);
        this.radius = radius;
    }

    // Implementation of abstract method
    @Override
    public double calculateArea() {
        return Math.PI * radius * radius;
    }

    // Implementation of abstract method
    @Override
    public double calculatePerimeter() {
        return 2 * Math.PI * radius;
    }

    // Additional method specific to Circle
    public double getRadius() {
        return radius;
    }
}

// Another concrete subclass of Shape
class Rectangle extends Shape {
    private double length;
    private double width;

    public Rectangle(String color, double length, double width) {
        super(color);
        this.length = length;
        this.width = width;
    }

    // Implementation of abstract method
    @Override
    public double calculateArea() {
        return length * width;
    }

    // Implementation of abstract method
    @Override
    public double calculatePerimeter() {
        return 2 * (length + width);
    }

    // Additional methods specific to Rectangle
    public double getLength() {
        return length;
    }

    public double getWidth() {
        return width;
    }
}

// Another concrete class that implements multiple abstract methods
class Triangle extends Shape {
    private double side1;
    private double side2;
    private double side3;

    public Triangle(String color, double side1, double side2, double
side3) {
        super(color);
        this.side1 = side1;
        this.side2 = side2;
        this.side3 = side3;
    }

    // Implementation of abstract method using Heron's formula
    @Override
    public double calculateArea() {
        double s = (side1 + side2 + side3) / 2;
        return Math.sqrt(s * (s - side1) * (s - side2) * (s - side3));
    }

    // Implementation of abstract method
    @Override
    public double calculatePerimeter() {
        return side1 + side2 + side3;
    }
}

// Main class to demonstrate abstract classes and methods
public class AbstractDemo {
    public static void main(String[] args) {
        // Cannot instantiate an abstract class
        // Shape shape = new Shape("Red"); // This would cause a
compilation error

        // Create concrete objects
        Circle circle = new Circle("Red", 5.0);
        Rectangle rectangle = new Rectangle("Blue", 4.0, 6.0);
        Triangle triangle = new Triangle("Green", 3.0, 4.0, 5.0);

        // Display information about each shape
        System.out.println("Circle Information:");
        circle.displayInfo();
        System.out.println("\nRectangle Information:");
        rectangle.displayInfo();
        System.out.println("\nTriangle Information:");
        triangle.displayInfo();

        // Polymorphism with abstract classes
        System.out.println("\nDemonstrating Polymorphism:");
        Shape[] shapes = new Shape[3];
        shapes[0] = circle;
        shapes[1] = rectangle;
        shapes[2] = triangle;

        for (Shape shape : shapes) {
            System.out.println("\nShape Details:");
            System.out.println("Color: " + shape.getColor());
            System.out.println("Area: " + shape.calculateArea());
            System.out.println("Perimeter: " +
shape.calculatePerimeter());
```

```
        }
    }
}
```

---

c. Write a Java program to implement the concept of interfaces

Sol.

```
// Basic interface definition
interface Animal {
    // Abstract methods that must be implemented
    void makeSound();
    void move();
}

// Second interface for demonstration
interface Pet {
    void play();
    void feed();
}

// Class implementing a single interface
class Dog implements Animal {
    private String name;

    public Dog(String name) {
        this.name = name;
    }

    // Implementing required methods from Animal interface
    @Override
    public void makeSound() {
        System.out.println(name + " says: Woof! Woof!");
    }

    @Override
    public void move() {
        System.out.println(name + " runs on four legs");
    }
}

// Class implementing multiple interfaces
class Cat implements Animal, Pet {
    private String name;

    public Cat(String name) {
        this.name = name;
    }

    // Implementing Animal interface methods
    @Override
    public void makeSound() {
        System.out.println(name + " says: Meow! Meow!");
    }

    @Override
    public void move() {
        System.out.println(name + " walks gracefully");
    }

    // Implementing Pet interface methods
    @Override
    public void play() {
        System.out.println(name + " plays with a ball of yarn");
    }

    @Override
    public void feed() {
        System.out.println(name + " eats cat food");
    }
}

// Main class to demonstrate interfaces
public class SimpleInterfaceDemo {
    public static void main(String[] args) {
        // Create a Dog object
        Dog dog = new Dog("Buddy");
        dog.makeSound();
        dog.move();

        System.out.println("\n-----------------\n");

        // Create a Cat object
        Cat cat = new Cat("Whiskers");
        cat.makeSound();
        cat.move();
        cat.play();
        cat.feed();

        System.out.println("\n-----------------\n");

        // Using interface as a type (polymorphism)
        System.out.println("Using Animal interface as a type:");
        Animal animal1 = new Dog("Rex");
        Animal animal2 = new Cat("Felix");

        animal1.makeSound();  // Calls Dog's method
        animal2.makeSound();  // Calls Cat's method

        // Using Pet interface
        System.out.println("\nUsing Pet interface as a type:");
        Pet pet = new Cat("Mittens");
        pet.play();
        pet.feed();
    }
}
```

---

3. Exceptions

a. Write a Java program to raise built-in exceptions and raise them as per the requirements

Sol.

```
public class SimpleExceptionDemo {
    public static void main(String[] args) {
        SimpleExceptionDemo demo = new SimpleExceptionDemo();

        // Example 1: ArithmeticException
        System.out.println("Example 1: Division by zero");
        try {
            int result = demo.divide(10, 0);
            System.out.println("Result: " + result); // This won't execute
        } catch (ArithmeticException e) {
            System.out.println("Error: " + e.getMessage());
        }

        // Example 2: ArrayIndexOutOfBoundsException
        System.out.println("\nExample 2: Array index out of bounds");
        try {
            int value = demo.getArrayElement(new int[]{1, 2, 3}, 5);
            System.out.println("Value: " + value); // This won't execute
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Error: Index out of range");
        }

        // Example 3: NullPointerException
        System.out.println("\nExample 3: Null pointer");
        try {
            int length = demo.getStringLength(null);
            System.out.println("Length: " + length); // This won't execute
        } catch (NullPointerException e) {
            System.out.println("Error: String cannot be null");
        }

        // Example 4: IllegalArgumentException
        System.out.println("\nExample 4: Illegal argument");
        try {
            demo.verifyAge(-5);
        } catch (IllegalArgumentException e) {
            System.out.println("Error: " + e.getMessage());
        }

        // Example 5: Multiple catch blocks and finally
        System.out.println("\nExample 5: Multiple exceptions");
        try {
            String numberStr = "abc";
            int number = Integer.parseInt(numberStr); // Will throw
NumberFormatException
            int result = 100 / number;              // Won't get executed
        } catch (NumberFormatException e) {
            System.out.println("Error: Cannot convert to number");
        } catch (ArithmeticException e) {
            System.out.println("Error: Cannot divide by zero");
        } finally {
            System.out.println("This always executes, with or without
exceptions");
        }
    }

    // Method that throws ArithmeticException
    public int divide(int a, int b) {
        return a / b; // This will throw ArithmeticException if b is 0
    }

    // Method that throws ArrayIndexOutOfBoundsException
    public int getArrayElement(int[] array, int index) {
        return array[index]; // Throws exception if index is invalid
    }

    // Method that throws NullPointerException
    public int getStringLength(String text) {
        return text.length(); // Throws exception if text is null
    }
    // Method that explicitly throws IllegalArgumentException
    public void verifyAge(int age) {
        if (age < 0) {
            throw new IllegalArgumentException("Age cannot be
negative");
        }
        System.out.println("Age verified: " + age);
    }
}
```

---

b. Write a Java program to define user defined exceptions and raise them as per the requirements

<u>Sol.</u>

```
public class SimpleExceptionDemo {
    public static void main(String[] args) {
        Calculator calc = new Calculator();

        try {
            // This will work fine
            System.out.println("10 / 5 = " + calc.divide(10, 5));

            // This will throw DivideByZeroException
            System.out.println("10 / 0 = " + calc.divide(10, 0));
        } catch (DivideByZeroException e) {
            System.out.println("Error: " + e.getMessage());
        }

        try {
            // This will throw NegativeNumberException
            calc.squareRoot(-4);
        } catch (NegativeNumberException e) {
            System.out.println("Error: " + e.getMessage());
            System.out.println("Invalid value: " + e.getNumber());
        }
    }
```

```
}

// Custom exception for division by zero
class DivideByZeroException extends Exception {
    public DivideByZeroException() {
        super("Cannot divide by zero");
    }
}

// Custom exception for negative numbers
class NegativeNumberException extends Exception {
    private double number;

    public NegativeNumberException(double number) {
        super("Cannot calculate square root of a negative number");
        this.number = number;
    }

    public double getNumber() {
        return number;
    }
}

// Calculator class that uses custom exceptions
class Calculator {
    public double divide(double a, double b) throws
DivideByZeroException {
        if (b == 0) {
            throw new DivideByZeroException();
        }
        return a / b;
    }

    public double squareRoot(double a) throws
NegativeNumberException {
        if (a < 0) {
            throw new NegativeNumberException(a);
        }
        return Math.sqrt(a);
    }
}
```

---

4. Multithreading

a. Write a java application to demonstrate 5 bouncing balls of different colors using threads.

Sol.

```
import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

public class BouncingBalls extends JFrame {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            BouncingBalls app = new BouncingBalls();
            app.setVisible(true);
        });
    }

    private static final int WIDTH = 600;
    private static final int HEIGHT = 400;
    private BallPanel ballPanel;

    public BouncingBalls() {
        setTitle("Bouncing Balls with Threads");
        setSize(WIDTH, HEIGHT);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);

        ballPanel = new BallPanel();
        add(ballPanel);
```

```
        // Create and start 5 ball threads with different colors
        Color[] colors = {Color.RED, Color.BLUE, Color.GREEN,
Color.ORANGE, Color.MAGENTA};
        for (int i = 0; i < 5; i++) {
            Ball ball = new Ball(ballPanel, colors[i]);
            Thread ballThread = new Thread(ball);
            ballThread.start();
        }
    }
}

class BallPanel extends JPanel {
    private List<BallInfo> balls = new ArrayList<>();

    public BallPanel() {
        setBackground(Color.BLACK);
    }

    public synchronized void addBall(BallInfo ball) {
        balls.add(ball);
    }

    public synchronized void updateBall(BallInfo ball) {
        repaint();
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);

        synchronized (this) {
            for (BallInfo ball : balls) {
                g.setColor(ball.color);
                g.fillOval(ball.x, ball.y, ball.size, ball.size);
            }
        }
    }
}

class BallInfo {
    int x, y;           // Position
    int xSpeed, ySpeed; // Velocity
    int size;
    Color color;

    public BallInfo(int x, int y, int xSpeed, int ySpeed, int size, Color
color) {
        this.x = x;
        this.y = y;
        this.xSpeed = xSpeed;
        this.ySpeed = ySpeed;
        this.size = size;
        this.color = color;
    }
}

class Ball implements Runnable {
    private BallPanel panel;
    private BallInfo ball;
    private Random random = new Random();

    public Ball(BallPanel panel, Color color) {
        this.panel = panel;

        // Random initial position
        int size = random.nextInt(20) + 30; // Size between 30-50
        int x = random.nextInt(panel.getWidth() - size);
        int y = random.nextInt(panel.getHeight() - size);

        // Random velocity
        int xSpeed = random.nextInt(5) + 2; // 2-7 pixels per step
        int ySpeed = random.nextInt(5) + 2; // 2-7 pixels per step
        ball = new BallInfo(x, y, xSpeed, ySpeed, size, color);
        panel.addBall(ball);
    }

    @Override
    public void run() {
        try {
            // Allow time for the panel to be fully initialized
            Thread.sleep(100);

            // Animation loop
            while (true) {
                // Move the ball
                moveBall();

                // Update display
                panel.updateBall(ball);

                // Control animation speed
                Thread.sleep(20);
            }
        } catch (InterruptedException e) {
            // Thread interrupted, exit gracefully
        }
    }

    private void moveBall() {
        // Get panel dimensions (may change if window is resized)
        int panelWidth = panel.getWidth();
        int panelHeight = panel.getHeight();

        // Update position
        ball.x += ball.xSpeed;
        ball.y += ball.ySpeed;

        // Check for collisions with walls
        // Right or left wall
        if (ball.x <= 0 || ball.x + ball.size >= panelWidth) {
            ball.xSpeed = -ball.xSpeed; // Reverse x direction

            // Ensure the ball stays within bounds
            if (ball.x <= 0) {
                ball.x = 0;
            } else {
                ball.x = panelWidth - ball.size;
            }
        }

        // Bottom or top wall
        if (ball.y <= 0 || ball.y + ball.size >= panelHeight) {
            ball.ySpeed = -ball.ySpeed; // Reverse y direction

            // Ensure the ball stays within bounds
            if (ball.y <= 0) {
                ball.y = 0;
            } else {
                ball.y = panelHeight - ball.size;
            }
        }
    }
}
```

---

5. JDBC

a. Write a JDBC program that displays the data of a given table in a GUI Table.

Sol.

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.sql.*;
import java.util.Vector;
```

```
public class SimpleJDBCViewer extends JFrame {
    private JTextField urlField, userField, passwordField, tableField;
    private JButton viewButton;
    private JTable dataTable;

    public SimpleJDBCViewer() {
        setTitle("JDBC Table Viewer");
        setSize(600, 500);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Input panel
        JPanel inputPanel = new JPanel(new GridLayout(5, 2, 5, 5));
        inputPanel.add(new JLabel("Database URL:"));
        urlField = new JTextField("jdbc:mysql://localhost:3306/mydb");
        inputPanel.add(urlField);

        inputPanel.add(new JLabel("Username:"));
        userField = new JTextField("root");
        inputPanel.add(userField);

        inputPanel.add(new JLabel("Password:"));
        passwordField = new JTextField();
        inputPanel.add(passwordField);

        inputPanel.add(new JLabel("Table Name:"));
        tableField = new JTextField("users");
        inputPanel.add(tableField);

        viewButton = new JButton("View Table Data");
        inputPanel.add(new JLabel(""));
        inputPanel.add(viewButton);

        // Table for displaying data
        dataTable = new JTable();
        JScrollPane scrollPane = new JScrollPane(dataTable);

        // Layout
        setLayout(new BorderLayout());
        add(inputPanel, BorderLayout.NORTH);
        add(scrollPane, BorderLayout.CENTER);

        // Button action
        viewButton.addActionListener(e -> loadTableData());
    }

    private void loadTableData() {
        try {
            // Get connection details
            String url = urlField.getText();
            String user = userField.getText();
            String password = passwordField.getText();
            String table = tableField.getText();

            // Connect to database
            Connection conn = DriverManager.getConnection(url, user,
password);

            // Execute query
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM " +
table);

            // Get column names
            ResultSetMetaData metaData = rs.getMetaData();
            int columnCount = metaData.getColumnCount();
            Vector<String> columnNames = new Vector<>();
            for (int i = 1; i <= columnCount; i++) {
                columnNames.add(metaData.getColumnName(i));
            }

            // Get data rows
            Vector<Vector<Object>> data = new Vector<>();
            while (rs.next()) {
                Vector<Object> row = new Vector<>();
                for (int i = 1; i <= columnCount; i++) {
                    row.add(rs.getObject(i));
                }
                data.add(row);
            }

            // Update table with data
            dataTable.setModel(new DefaultTableModel(data,
columnNames));

            // Clean up
            rs.close();
            stmt.close();
            conn.close();

        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(this,
                "Database error: " + ex.getMessage(),
                "Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            new SimpleJDBCViewer().setVisible(true);
        });
    }
}
```

---

b. Write a JDBC program to Show the details of a specified product from a given table selected using Combobox.

Sol.

```
import javax.swing.*;
import java.awt.*;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class ProductDetailsViewer extends JFrame {
    private JComboBox<String> productComboBox;
    private JTextArea detailsTextArea;
    private Connection connection;
    private String tableName = "products"; // Default table name

    public ProductDetailsViewer() {
        setTitle("Product Details Viewer");
        setSize(500, 400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);

        // Create UI components
        JPanel mainPanel = new JPanel(new BorderLayout(10, 10));
        mainPanel.setBorder(BorderFactory.createEmptyBorder(10, 10,
10, 10));

        JPanel topPanel = new JPanel(new
FlowLayout(FlowLayout.LEFT));
        topPanel.add(new JLabel("Select Product:"));

        productComboBox = new JComboBox<>();
        productComboBox.setPreferredSize(new Dimension(300, 25));
        topPanel.add(productComboBox);

        detailsTextArea = new JTextArea();
        detailsTextArea.setEditable(false);
        JScrollPane scrollPane = new JScrollPane(detailsTextArea);

        mainPanel.add(topPanel, BorderLayout.NORTH);
        mainPanel.add(scrollPane, BorderLayout.CENTER);

        add(mainPanel);
        // Add action listener to the combobox
        productComboBox.addActionListener(e ->
displayProductDetails());

        // Connect to database and populate combobox
        connectToDatabase();
    }

    private void connectToDatabase() {
        try {
            // Database connection parameters
            String url = "jdbc:mysql://localhost:3306/inventory";
            String username = "root";
            String password = "";

            // Establish connection
            connection = DriverManager.getConnection(url, username,
password);

            // Populate the combo box with product names
            populateProductComboBox();

        } catch (SQLException e) {
            JOptionPane.showMessageDialog(this,
                "Database connection error: " + e.getMessage(),
                "Connection Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    private void populateProductComboBox() {
        try {
            // Query to get product IDs and names
            String query = "SELECT product_id, product_name FROM "
+ tableName;
            Statement stmt = connection.createStatement();
            ResultSet rs = stmt.executeQuery(query);
            // Clear existing items
            productComboBox.removeAllItems();

            // Store product IDs for later use
            List<String> productIds = new ArrayList<>();

            // Add items to combo box
            while (rs.next()) {
                String productId = rs.getString("product_id");
                String productName = rs.getString("product_name");
                productComboBox.addItem(productId + " - " +
productName);
                productIds.add(productId);
            }

            // Close resources
            rs.close();
            stmt.close();

            // Select first item if available
            if (productComboBox.getItemCount() > 0) {
                productComboBox.setSelectedIndex(0);
            }

        } catch (SQLException e) {
            JOptionPane.showMessageDialog(this,
                "Error loading products: " + e.getMessage(),
                "Data Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    private void displayProductDetails() {
        if (productComboBox.getSelectedItem() == null) {
            return;
        }

        try {
            // Get the selected product ID
            String selectedItem = (String)
productComboBox.getSelectedItem();
            String productId = selectedItem.split(" - ")[0];

            // Query for product details
            String query = "SELECT * FROM " + tableName + "
WHERE product_id = ?";
            PreparedStatement pstmt =
connection.prepareStatement(query);
            pstmt.setString(1, productId);

            ResultSet rs = pstmt.executeQuery();

            // Display details in text area
            if (rs.next()) {
                ResultSetMetaData metaData = rs.getMetaData();
                int columnCount = metaData.getColumnCount();

                StringBuilder details = new StringBuilder();
                details.append("PRODUCT DETAILS:\n");
                details.append("===============\n\n");

                for (int i = 1; i <= columnCount; i++) {
                    String columnName = metaData.getColumnName(i);
                    Object value = rs.getObject(i);
                    details.append(columnName).append(":
").append(value).append("\n");
                }

                detailsTextArea.setText(details.toString());
            } else {
                detailsTextArea.setText("No details found for the selected
product.");
            }

            // Close resources
            rs.close();
            pstmt.close();

        } catch (SQLException e) {
            JOptionPane.showMessageDialog(this,
                "Error retrieving product details: " + e.getMessage(),
                "Data Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    private void closeConnection() {
        try {
            if (connection != null && !connection.isClosed()) {
                connection.close();
            }
        } catch (SQLException e) {
            System.err.println("Error closing connection: " +
e.getMessage());
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            ProductDetailsViewer viewer = new ProductDetailsViewer();
            viewer.setVisible(true);
        });
    }
}
```

---

c. Write a GUI application to Navigate forward and reverse result set data.

Sol.

```
import javax.swing.*;
import java.awt.*;
import java.sql.*;
```

```
public class SimpleNavigator extends JFrame {
    private Connection conn;
    private ResultSet rs;
    private JTextField[] fields;
    private JButton firstBtn, prevBtn, nextBtn, lastBtn;
    private JLabel statusLabel;
    private int currentRow = 0;
    private int totalRows = 0;

    public SimpleNavigator() {
        setTitle("Record Navigator");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Create UI
        JPanel dataPanel = new JPanel(new GridLayout(0, 2, 5, 5));
        JPanel navPanel = new JPanel();

        // Navigation buttons
        firstBtn = new JButton("<<");
        prevBtn = new JButton("<");
        nextBtn = new JButton(">");
        lastBtn = new JButton(">>");
        statusLabel = new JLabel("Record 0 of 0");

        navPanel.add(firstBtn);
        navPanel.add(prevBtn);
        navPanel.add(statusLabel);
        navPanel.add(nextBtn);
        navPanel.add(lastBtn);

        // Add to frame
        setLayout(new BorderLayout(10, 10));
        add(new JScrollPane(dataPanel), BorderLayout.CENTER);
        add(navPanel, BorderLayout.SOUTH);
        // Add listeners
        firstBtn.addActionListener(e -> moveToFirst());
        prevBtn.addActionListener(e -> moveToPrevious());
        nextBtn.addActionListener(e -> moveToNext());
        lastBtn.addActionListener(e -> moveToLast());

        // Connect to database
        try {
            // Change these to match your database
            String url = "jdbc:mysql://localhost:3306/testdb";
            String user = "root";
            String password = "";

            conn = DriverManager.getConnection(url, user, password);
            Statement stmt = conn.createStatement(
                ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_READ_ONLY
            );

            // Execute query - change table name if needed
            rs = stmt.executeQuery("SELECT * FROM employees");

            // Count rows
            rs.last();
            totalRows = rs.getRow();
            rs.beforeFirst();

            // Get metadata
            ResultSetMetaData metaData = rs.getMetaData();
            int columnCount = metaData.getColumnCount();

            // Create fields
            fields = new JTextField[columnCount];
            for (int i = 0; i < columnCount; i++) {
                dataPanel.add(new JLabel(metaData.getColumnName(i+1)
+ ":"));
                fields[i] = new JTextField(15);
                fields[i].setEditable(false);
                dataPanel.add(fields[i]);
            }

            // Show first record
            if (rs.next()) {
                currentRow = 1;
                displayRecord();
            }

            updateButtons();

        } catch (SQLException e) {
            JOptionPane.showMessageDialog(this, "Database error: " +
e.getMessage());
        }
}

private void displayRecord() {
        try {
            for (int i = 0; i < fields.length; i++) {
                fields[i].setText(rs.getString(i+1));
            }
            statusLabel.setText("Record " + currentRow + " of " +
totalRows);
        } catch (SQLException e) {
            JOptionPane.showMessageDialog(this, "Error: " +
e.getMessage());
        }
}

private void moveToFirst() {
        try {
            if (rs.first()) {
                currentRow = 1;
                displayRecord();
                updateButtons();
            }
        } catch (SQLException e) {
            JOptionPane.showMessageDialog(this, "Error: " +
e.getMessage());
        }
    }

    private void moveToPrevious() {
        try {
            if (rs.previous()) {
                currentRow--;
                displayRecord();
                updateButtons();
            }
        } catch (SQLException e) {
            JOptionPane.showMessageDialog(this, "Error: " +
e.getMessage());
        }
    }

    private void moveToNext() {
        try {
            if (rs.next()) {
                currentRow++;
                displayRecord();
                updateButtons();
            }
        } catch (SQLException e) {
            JOptionPane.showMessageDialog(this, "Error: " +
e.getMessage());
        }
    }

    private void moveToLast() {
        try {
            if (rs.last()) {
                currentRow = totalRows;
                displayRecord();
                updateButtons();
            }
        } catch (SQLException e) {
            JOptionPane.showMessageDialog(this, "Error: " +
e.getMessage());
        }
    }

    private void updateButtons() {
        firstBtn.setEnabled(currentRow > 1);
        prevBtn.setEnabled(currentRow > 1);
        nextBtn.setEnabled(currentRow < totalRows);
        lastBtn.setEnabled(currentRow < totalRows);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            new SimpleNavigator().setVisible(true);
        });
    }
}
```

---

6. Swing

a. Create a swing application that randomly changes color on button click.

Sol.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Random;

public class RandomColorChanger extends JFrame {
  private JPanel colorPanel;
  private JButton changeColorButton;
private Random random;

public RandomColorChanger() {
   // Set up the frame
   setTitle("Random Color Changer");
   setSize(400, 300);
   setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   setLocationRelativeTo(null);

   // Create components
   colorPanel = new JPanel();
   colorPanel.setBackground(Color.WHITE);

   changeColorButton = new JButton("Change Color");
   random = new Random();

   // Add action listener to button
   changeColorButton.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent e) {
         changeRandomColor();
      }
   });

   // Set up layout
   setLayout(new BorderLayout());
   add(colorPanel, BorderLayout.CENTER);
   add(changeColorButton, BorderLayout.SOUTH);
}

private void changeRandomColor() {
   // Generate random RGB values
   int r = random.nextInt(256);
   int g = random.nextInt(256);
   int b = random.nextInt(256);

   // Create new color and apply it
```

```
        Color newColor = new Color(r, g, b);
        colorPanel.setBackground(newColor);

        // Display RGB values as text
        colorPanel.removeAll();
        JLabel rgbLabel = new JLabel("RGB: " + r + ", " + g + ", " + b);
        rgbLabel.setForeground(getContrastColor(newColor));
        rgbLabel.setFont(new Font("Arial", Font.BOLD, 16));
        colorPanel.add(rgbLabel);

        // Refresh panel
        colorPanel.revalidate();
        colorPanel.repaint();
    }

    // Calculate a contrasting color for text visibility
    private Color getContrastColor(Color bg) {
        // Calculate brightness using perceived brightness formula
        double brightness = (bg.getRed() * 0.299 +
                             bg.getGreen() * 0.587 +
                             bg.getBlue() * 0.114);

        // Return black for light colors, white for dark colors
        return brightness > 130 ? Color.BLACK : Color.WHITE;
    }

    public static void main(String[] args) {
        // Create and show the application on the EDT
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                RandomColorChanger app = new RandomColorChanger();
                app.setVisible(true);
            }
        });
    }
}
```

b. Create a Swing application to demonstrate use of TextArea using scrollpane to show contest of text file in textarea selected using file chooser.

Sol.

```
import javax.swing.*;
import java.awt.*;
import java.io.*;

public class SimpleTextViewer extends JFrame {
   private JTextArea textArea;

   public SimpleTextViewer() {
      // Basic frame setup
      setTitle("Simple Text Viewer");
      setSize(600, 400);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      // Create text area inside scroll pane
      textArea = new JTextArea();
      textArea.setEditable(false);
      JScrollPane scrollPane = new JScrollPane(textArea);

      // Create open button
      JButton openButton = new JButton("Open File");
      openButton.addActionListener(e -> openFile());

      // Add components to frame
      add(scrollPane, BorderLayout.CENTER);
      add(openButton, BorderLayout.NORTH);

      setLocationRelativeTo(null);
   }

   private void openFile() {
      JFileChooser chooser = new JFileChooser();
```

```
        int result = chooser.showOpenDialog(this);

        if (result == JFileChooser.APPROVE_OPTION) {
            try {
                // Read and display file content
                File file = chooser.getSelectedFile();
                BufferedReader reader = new BufferedReader(new
FileReader(file));
                textArea.setText("");
                String line;
                while ((line = reader.readLine()) != null) {
                    textArea.append(line + "\n");
                }
                reader.close();
            } catch (IOException ex) {
                JOptionPane.showMessageDialog(this, "Error reading
file");
            }
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new
SimpleTextViewer().setVisible(true));
    }
}
```

---

c. Create a Swing application to demonstrate use of scrollpane to change its color selected using colour chooser.

<u>Sol.</u>

```
import javax.swing.*;
import java.awt.*;

public class SimpleScrollPaneColorDemo extends JFrame {
    private JScrollPane scrollPane;
    public SimpleScrollPaneColorDemo() {
        // Basic setup
        setTitle("ScrollPane Color Demo");
        setSize(500, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Create text area with sample content
        JTextArea textArea = new JTextArea();
        textArea.setOpaque(false);
        for (int i = 1; i <= 30; i++) {
            textArea.append("Line " + i + ": Sample text for scrolling.\n");
        }

        // Create scroll pane
        scrollPane = new JScrollPane(textArea);
        scrollPane.getViewport().setBackground(Color.LIGHT_GRAY);
// Default color

        // Create color button
        JButton colorButton = new JButton("Choose Color");
        colorButton.addActionListener(e -> {
            Color newColor = JColorChooser.showDialog(this, "Select
Color",
                    scrollPane.getViewport().getBackground());
            if (newColor != null) {
                scrollPane.getViewport().setBackground(newColor);
                repaint();
            }
        });

        // Add components
        add(scrollPane, BorderLayout.CENTER);
        add(colorButton, BorderLayout.SOUTH);

        setLocationRelativeTo(null);
    }
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new
SimpleScrollPaneColorDemo().setVisible(true));
    }
}
```

7. Layouts: Write a Java program for the following layouts:

a. Flow Layout

Sol.

```
import javax.swing.*;
import java.awt.*;

public class FlowLayoutDemo extends JFrame {

    public FlowLayoutDemo() {
        // Set up the frame
        setTitle("Flow Layout Demonstration");
        setSize(400, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);

        // Create a panel with FlowLayout
        JPanel panel = new JPanel();

        // Set FlowLayout with centered alignment and 10-pixel
horizontal and vertical gaps
        panel.setLayout(new FlowLayout(FlowLayout.CENTER, 10,
10));

        // Add several buttons to demonstrate the layout
        panel.add(new JButton("Button 1"));
        panel.add(new JButton("Button 2"));
        panel.add(new JButton("Button 3"));
        panel.add(new JButton("Long Button 4"));
        panel.add(new JButton("Button 5"));
        panel.add(new JButton("Button 6"));
        panel.add(new JButton("Button 7"));
        panel.add(new JButton("Button 8"));

        // Add panel to the frame
        add(panel);
    }

    public static void main(String[] args) {
        // Launch the application
        SwingUtilities.invokeLater(() -> {
            FlowLayoutDemo demo = new FlowLayoutDemo();
            demo.setVisible(true);
        });
    }
}
```

---

b. Grid Layout

<u>Sol.</u>

```
import javax.swing.*;
import java.awt.*;

public class GridLayoutDemo extends JFrame {

    public GridLayoutDemo() {
        // Set up the frame
        setTitle("Grid Layout Demonstration");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);

        // Create a panel with GridLayout (3 rows, 2 columns)
        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(3, 2, 10, 10));
        // Add buttons to demonstrate the layout
        panel.add(new JButton("Button 1"));
        panel.add(new JButton("Button 2"));
        panel.add(new JButton("Button 3"));
        panel.add(new JButton("Button 4"));
        panel.add(new JButton("Button 5"));
        panel.add(new JButton("Button 6"));

        // Add panel to the frame
        add(panel);
    }

    public static void main(String[] args) {
        // Launch the application
        SwingUtilities.invokeLater(() -> {
            GridLayoutDemo demo = new GridLayoutDemo();
            demo.setVisible(true);
        });
    }
}
```

---

c. Border Layout

Sol.

```
import javax.swing.*;
import java.awt.*;

public class BorderLayoutDemo extends JFrame {

    public BorderLayoutDemo() {
        // Set up the frame
        setTitle("Border Layout Demonstration");
        setSize(500, 400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
```

```
}

        // BorderLayout is the default layout for JFrame's content pane
        // but we'll set it explicitly for clarity
        setLayout(new BorderLayout(10, 10)); // 10-pixel horizontal and
vertical gaps

        // Create and add components to the five regions
        add(createColoredPanel("NORTH", Color.RED),
BorderLayout.NORTH);
        add(createColoredPanel("SOUTH", Color.BLUE),
BorderLayout.SOUTH);
        add(createColoredPanel("EAST", Color.GREEN),
BorderLayout.EAST);
        add(createColoredPanel("WEST", Color.YELLOW),
BorderLayout.WEST);
        add(createColoredPanel("CENTER", Color.WHITE),
BorderLayout.CENTER);
    }

    // Helper method to create a colored panel with a label
    private JPanel createColoredPanel(String text, Color color) {
        JPanel panel = new JPanel();
        panel.setBackground(color);
        panel.add(new JLabel(text));
        return panel;
    }

    public static void main(String[] args) {
        // Launch the application
        SwingUtilities.invokeLater(() -> {
            BorderLayoutDemo demo = new BorderLayoutDemo();
            demo.setVisible(true);
        });
    }
}
```