22CS203 - OBJECT-ORIENTED PROGRAMMING THROUGH JAVA

List of Lab Programs

Module 1

Unit – 1 Programs on OOP basics

1. Create a class Rectangle. The class has attributes length and width. It should have methods that calculate the perimeter and area of the rectangle. It should have read Attributes method to read length and width from user.

   Hint: Area of rectangle = length * width, Perimeter of rectangle = 2*(length+width).

Aim

To create a Rectangle class with attributes length and width. The class should have methods to calculate the perimeter and area of the rectangle and a method to read these attributes from the user.

Algorithm

Define the Class: Create a class named Rectangle.

Attributes: Declare private attributes length and width.

Constructor: Provide a default constructor to initialize attributes to default values.

Read Attributes: Implement a method to read length and width from the user.

Calculate Area: Implement a method to calculate and return the area of the rectangle.

Calculate Perimeter: Implement a method to calculate and return the perimeter of the rectangle.

Main Method: Create an instance of Rectangle, use the method to read attributes, and display the area and perimeter.

Source Code

```
import java.util.Scanner;


public class Rectangle {
    // Attributes
    private double length;
```

```
private double width;

// Default constructor
public Rectangle() {
    this.length = 0.0;
    this.width = 0.0;
}

// Method to read attributes from user
public void readAttributes() {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter length of the rectangle: ");
    this.length = scanner.nextDouble();

    System.out.print("Enter width of the rectangle: ");
    this.width = scanner.nextDouble();
}

// Method to calculate the area
public double calculateArea() {
    return this.length * this.width;
}

// Method to calculate the perimeter
public double calculatePerimeter() {
    return 2 * (this.length + this.width);
}
```

```
    public static void main(String[] args) {
        // Create an instance of Rectangle
        Rectangle rect = new Rectangle();

        // Read attributes
        rect.readAttributes();

        // Calculate and display area and perimeter
        System.out.println("Area of the rectangle: " + rect.calculateArea());
        System.out.println("Perimeter of the rectangle: " + rect.calculatePerimeter());
    }
}
```

Test Cases

Test Case 1:

Input: Length = 5.0, Width = 3.0

Expected Output:

Area: 15.0

Perimeter: 16.0

Test Case 2:

Input: Length = 7.5, Width = 4.2

Expected Output:

Area: 31.5

Perimeter: 23.4

2. Develop a Java application to generate Electricity bill. Create a class with the following members: Consumer no., consumer name, previous month reading, current month reading, type of EB connection (i.e domestic or commercial). Compute the bill amount using the following tariff.

   If the type of the EB connection is domestic, calculate the amount to be paid as follows:

   First 100 units – Rs. 1 per unit

   101-200 units – Rs. 2.50 per unit

   201 -500 units – Rs. 4 per unit

   > 501 units – Rs. 6 per unit

   If the type of the EB connection is commercial, calculate the amount to be paid as follows:

   First 100 units – Rs. 2 per unit

   101-200 units – Rs. 4.50 per unit

   201 -500 units – Rs. 6 per unit

   > 501 units – Rs. 7 per unit

Aim

To create a Java class that calculates the electricity bill based on consumer details and usage. The calculation should differ based on whether the connection is domestic or commercial.

Algorithm

Define the Class: Create a class named ElectricityBill.

Attributes: Declare private attributes for consumer number, consumer name, previous month reading, current month reading, and type of EB connection.

Constructor: Provide a constructor to initialize the attributes.

Calculate Bill: Implement a method to calculate the bill amount based on the type of EB connection and the usage.

Display Bill: Implement a method to display the consumer details and calculated bill.

Main Method: Create an instance of ElectricityBill, use the constructor to set attributes, and calculate and display the bill.

Source Code

```
import java.util.Scanner;


public class ElectricityBill {
   // Attributes
   private String consumerNo;
   private String consumerName;
   private double previousMonthReading;
   private double currentMonthReading;
   private String connectionType;


   // Constructor
   public ElectricityBill(String consumerNo, String consumerName, double previousMonthReading,
                          double currentMonthReading, String connectionType) {
      this.consumerNo = consumerNo;
      this.consumerName = consumerName;
      this.previousMonthReading = previousMonthReading;
      this.currentMonthReading = currentMonthReading;
      this.connectionType = connectionType;
   }


   // Method to calculate the bill amount
   public double calculateBill() {
      double unitsConsumed = currentMonthReading - previousMonthReading;
      double billAmount = 0.0;


      if (connectionType.equalsIgnoreCase("domestic")) {
         if (unitsConsumed <= 100) {
```

```
            billAmount = unitsConsumed * 1.0;
        } else if (unitsConsumed <= 200) {
            billAmount = 100 * 1.0 + (unitsConsumed - 100) * 2.5;
        } else if (unitsConsumed <= 500) {
            billAmount = 100 * 1.0 + 100 * 2.5 + (unitsConsumed - 200) * 4.0;
        } else {
            billAmount = 100 * 1.0 + 100 * 2.5 + 300 * 4.0 + (unitsConsumed - 500) *
6.0;
        }
    } else if (connectionType.equalsIgnoreCase("commercial")) {
        if (unitsConsumed <= 100) {
            billAmount = unitsConsumed * 2.0;
        } else if (unitsConsumed <= 200) {
            billAmount = 100 * 2.0 + (unitsConsumed - 100) * 4.5;
        } else if (unitsConsumed <= 500) {
            billAmount = 100 * 2.0 + 100 * 4.5 + (unitsConsumed - 200) * 6.0;
        } else {
            billAmount = 100 * 2.0 + 100 * 4.5 + 300 * 6.0 + (unitsConsumed - 500) *
7.0;
        }
    } else {
        System.out.println("Invalid connection type.");
    }

    return billAmount;
}

// Method to display bill details
public void displayBill() {
    double billAmount = calculateBill();
```

```
        System.out.println("Consumer No: " + consumerNo);
        System.out.println("Consumer Name: " + consumerName);
        System.out.println("Previous Month Reading: " + previousMonthReading);
        System.out.println("Current Month Reading: " + currentMonthReading);
        System.out.println("Connection Type: " + connectionType);
        System.out.println("Bill Amount: Rs. " + billAmount);
    }

    public static void main(String[] args) {
        // Input from user
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter Consumer Number: ");
        String consumerNo = scanner.nextLine();

        System.out.print("Enter Consumer Name: ");
        String consumerName = scanner.nextLine();

        System.out.print("Enter Previous Month Reading: ");
        double previousMonthReading = scanner.nextDouble();

        System.out.print("Enter Current Month Reading: ");
        double currentMonthReading = scanner.nextDouble();

        scanner.nextLine();  // Consume newline
        System.out.print("Enter Type of Connection (domestic/commercial): ");
        String connectionType = scanner.nextLine();

        // Create an instance of ElectricityBill
```

```
        ElectricityBill bill = new ElectricityBill(consumerNo, consumerName, previousMonthReading, currentMonthReading, connectionType);


        // Display the bill
        bill.displayBill();
    }
}
```

Test Cases

Test Case 1:

Input:

Consumer Number: C001

Consumer Name: John Doe

Previous Month Reading: 100

Current Month Reading: 250

Connection Type: domestic

Expected Output:

Area: 100 * 1.0 + 100 * 2.5 + 50 * 4.0 = Rs. 250 + 200 + 200 = Rs. 650

Test Case 2:

Input:

Consumer Number: C002

Consumer Name: Jane Smith

Previous Month Reading: 300

Current Month Reading: 600

Connection Type: commercial

Expected Output:

Area: 100 * 2.0 + 100 * 4.5 + 300 * 6.0 + 100 * 7.0 = Rs. 200 + 450 + 1800 + 700 = Rs. 3150

3. Write a JAVA program to search for an element in a given list of elements using binary search mechanism.

Aim

To implement a Java program that performs a binary search on a sorted list to find a specific element.

Algorithm

Define the Class: Create a class named BinarySearchExample.
Binary Search Method: Implement a method to perform binary search on the sorted list.
Main Method: Create a sorted list, prompt the user for the search element, and use the binary search method to find and display the element's position.

Source Code

```
import java.util.Scanner;

public class BinarySearchExample {
   // Method to perform binary search
   public static int binarySearch(int[] sortedArray, int target) {
      int left = 0;
      int right = sortedArray.length - 1;

      while (left <= right) {
         int mid = left + (right - left) / 2;

         // Check if the target is present at mid
         if (sortedArray[mid] == target) {
            return mid; // Target found
         }

         // If target is greater, ignore left half
         if (sortedArray[mid] < target) {
            left = mid + 1;
         }
         // If target is smaller, ignore right half
         else {
            right = mid - 1;
         }
      }

      // Target not found
      return -1;
   }
```

```
    public static void main(String[] args) {
        // Sample sorted list
        int[] sortedArray = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20};

        // Input from user
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the element to search: ");
        int target = scanner.nextInt();

        // Perform binary search
        int result = binarySearch(sortedArray, target);

        // Display result
        if (result != -1) {
            System.out.println("Element " + target + " found at index " + result + ".");
        } else {
            System.out.println("Element " + target + " not found in the list.");
        }
    }
}
```

Test Cases

Test Case 1:

Input: Element to search = 8
Expected Output: Element 8 found at index 3.
Test Case 2:

Input: Element to search = 15
Expected Output: Element 15 not found in the list.

4. Create a class Fact with attribute n and a method factorial. Develop a JAVA code to read n from user and find the factorial of a given number.

Aim

To create a Java class Fact that calculates the factorial of a number n and provides a method to read n from the user.

Algorithm

Define the Class: Create a class named Fact.
Attributes: Declare a private attribute n.
Constructor: Provide a constructor to initialize n.

Factorial Method: Implement a method to calculate the factorial of n.
Read Input and Display Result: In the main method, create an instance of Fact, read n from the user, compute the factorial, and display the result.

Source Code

```
import java.util.Scanner;

public class Fact {
    // Attribute
    private int n;

    // Constructor
    public Fact(int n) {
        this.n = n;
    }

    // Method to calculate factorial
    public long factorial() {
        long result = 1;
        for (int i = 1; i <= n; i++) {
            result *= i;
        }
        return result;
    }

    public static void main(String[] args) {
        // Input from user
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number to find its factorial: ");
        int number = scanner.nextInt();

        // Create an instance of Fact
        Fact fact = new Fact(number);

        // Calculate and display the factorial
        System.out.println("Factorial of " + number + " is: " + fact.factorial());
    }
}
```

Test Cases

Test Case 1:

Input: Number = 5
Expected Output: Factorial of 5 is: 120
Test Case 2:

Input: Number = 7
Expected Output: Factorial of 7 is: 5040

5. Write a program to display the employee details using Scanner class.

Aim

To create a Java program that collects and displays employee details such as employee ID, name, designation, and salary using the Scanner class.

Algorithm

Define the Class: Create a class named Employee.
Attributes: Declare private attributes for employee ID, name, designation, and salary.
Constructor: Provide a constructor to initialize these attributes.
Methods: Implement methods to set and get the details of the employee.
Main Method: Use the Scanner class to input employee details from the user, create an instance of the Employee class, and display the details.

Source Code

```
import java.util.Scanner;

public class Employee {
    // Attributes
    private int employeeId;
    private String name;
    private String designation;
    private double salary;

    // Constructor
    public Employee(int employeeId, String name, String designation, double salary) {
        this.employeeId = employeeId;
        this.name = name;
        this.designation = designation;
        this.salary = salary;
    }

    // Method to display employee details
    public void displayDetails() {
        System.out.println("Employee ID: " + employeeId);
        System.out.println("Name: " + name);
```

```
        System.out.println("Designation: " + designation);
        System.out.println("Salary: Rs. " + salary);
    }

    public static void main(String[] args) {
        // Input from user
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter Employee ID: ");
        int employeeId = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        System.out.print("Enter Name: ");
        String name = scanner.nextLine();

        System.out.print("Enter Designation: ");
        String designation = scanner.nextLine();

        System.out.print("Enter Salary: ");
        double salary = scanner.nextDouble();

        // Create an instance of Employee
        Employee employee = new Employee(employeeId, name, designation, salary);

        // Display employee details
        employee.displayDetails();
    }
}
```

Test Cases

Test Case 1:

Input:
Employee ID: 101
Name: Alice
Designation: Software Engineer
Salary: 75000

Employee ID: 101
Name: Alice
Designation: Software Engineer
Salary: Rs. 75000.0

Test Case 2:

Input:
Employee ID: 202
Name: Bob
Designation: Manager
Salary: 90000

Employee ID: 202
Name: Bob
Designation: Manager
Salary: Rs. 90000.0

6. Develop a java application to implement currency converter (Dollar to INR, EURO to INR, Yen to INR and vice versa), distance converter (meter to KM, miles to KM and vice versa) , time converter (hours to minutes, seconds and vice versa) using switch case.

Aim

To develop a Java application that allows conversion between various units of currency, distance, and time using the switch statement for selection.

Algorithm

Define the Class: Create a class named Converter.
Conversion Methods: Implement methods for each type of conversion (currency, distance, time).
User Input: Use the Scanner class to get user input for conversion type and values.
Switch Case: Implement a switch statement to handle different conversion options.
Display Results: Output the results of the conversions.

Source Code

```
import java.util.Scanner;

public class Converter {
    // Currency conversion rates
    private static final double DOLLAR_TO_INR = 82.0;
    private static final double EURO_TO_INR = 89.0;
    private static final double YEN_TO_INR = 0.56;
    private static final double INR_TO_DOLLAR = 1 / DOLLAR_TO_INR;
    private static final double INR_TO_EURO = 1 / EURO_TO_INR;
    private static final double INR_TO_YEN = 1 / YEN_TO_INR;

    // Conversion methods
```

```
public static void currencyConverter() {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Currency Converter");
    System.out.println("1. Dollar to INR");
    System.out.println("2. Euro to INR");
    System.out.println("3. Yen to INR");
    System.out.println("4. INR to Dollar");
    System.out.println("5. INR to Euro");
    System.out.println("6. INR to Yen");
    System.out.print("Select conversion type (1-6): ");
    int choice = scanner.nextInt();
    System.out.print("Enter amount: ");
    double amount = scanner.nextDouble();

    switch (choice) {
        case 1:
            System.out.println("Amount in INR: " + (amount * DOLLAR_TO_INR));
            break;
        case 2:
            System.out.println("Amount in INR: " + (amount * EURO_TO_INR));
            break;
        case 3:
            System.out.println("Amount in INR: " + (amount * YEN_TO_INR));
            break;
        case 4:
            System.out.println("Amount in Dollar: " + (amount * INR_TO_DOLLAR));
            break;
        case 5:
            System.out.println("Amount in Euro: " + (amount * INR_TO_EURO));
            break;
        case 6:
            System.out.println("Amount in Yen: " + (amount * INR_TO_YEN));
            break;
        default:
            System.out.println("Invalid choice.");
    }
}

public static void distanceConverter() {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Distance Converter");
    System.out.println("1. Meter to KM");
    System.out.println("2. Miles to KM");
    System.out.println("3. KM to Meter");
```

```
        System.out.println("4. KM to Miles");
        System.out.print("Select conversion type (1-4): ");
        int choice = scanner.nextInt();
        System.out.print("Enter distance: ");
        double distance = scanner.nextDouble();

        switch (choice) {
            case 1:
                System.out.println("Distance in KM: " + (distance / 1000));
                break;
            case 2:
                System.out.println("Distance in KM: " + (distance * 1.60934));
                break;
            case 3:
                System.out.println("Distance in Meter: " + (distance * 1000));
                break;
            case 4:
                System.out.println("Distance in Miles: " + (distance / 1.60934));
                break;
            default:
                System.out.println("Invalid choice.");
        }
    }

    public static void timeConverter() {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Time Converter");
        System.out.println("1. Hours to Minutes");
        System.out.println("2. Hours to Seconds");
        System.out.println("3. Minutes to Hours");
        System.out.println("4. Minutes to Seconds");
        System.out.println("5. Seconds to Hours");
        System.out.println("6. Seconds to Minutes");
        System.out.print("Select conversion type (1-6): ");
        int choice = scanner.nextInt();
        System.out.print("Enter time: ");
        double time = scanner.nextDouble();

        switch (choice) {
            case 1:
                System.out.println("Time in Minutes: " + (time * 60));
                break;
            case 2:
                System.out.println("Time in Seconds: " + (time * 3600));
```

```
            break;
        case 3:
            System.out.println("Time in Hours: " + (time / 60));
            break;
        case 4:
            System.out.println("Time in Seconds: " + (time * 60));
            break;
        case 5:
            System.out.println("Time in Hours: " + (time / 3600));
            break;
        case 6:
            System.out.println("Time in Minutes: " + (time / 60));
            break;
        default:
            System.out.println("Invalid choice.");
    }
  }

  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Select Conversion Type:");
    System.out.println("1. Currency Converter");
    System.out.println("2. Distance Converter");
    System.out.println("3. Time Converter");
    System.out.print("Enter your choice (1-3): ");
    int choice = scanner.nextInt();

    switch (choice) {
        case 1:
            currencyConverter();
            break;
        case 2:
            distanceConverter();
            break;
        case 3:
            timeConverter();
            break;
        default:
            System.out.println("Invalid choice.");
    }
  }
}
```

Test Cases

Currency Converter Test Cases:

Test Case 1:

Input: Convert 100 USD to INR
Expected Output: Amount in INR: 8200.0
Test Case 2:

Input: Convert 500 EUR to INR
Expected Output: Amount in INR: 44500.0
Distance Converter Test Cases:

Test Case 1:

Input: Convert 5000 meters to kilometers
Expected Output: Distance in KM: 5.0
Test Case 2:

Input: Convert 10 miles to kilometers
Expected Output: Distance in KM: 16.0934
Time Converter Test Cases:

Test Case 1:

Input: Convert 2 hours to minutes
Expected Output: Time in Minutes: 120.0
Test Case 2:

Input: Convert 180 seconds to minutes
Expected Output: Time in Minutes: 3.0

7. Implement a Java Program that reads a line of integers, and then displays each integer, and the sum of all the integers (use StringTokenizer class).

Aim

To develop a Java program that reads a line of integers, displays each integer, and calculates the sum of all integers using the StringTokenizer class.

Algorithm

Import Classes: Import necessary classes, including Scanner and StringTokenizer.
Read Input: Use Scanner to read a line of integers from the user.
Tokenize Input: Use StringTokenizer to tokenize the input string based on spaces.
Display and Sum: Iterate over tokens to display each integer and compute the sum.
Display Results: Output each integer and the total sum.

Source Code

```
import java.util.Scanner;
import java.util.StringTokenizer;

public class IntegerSum {
   public static void main(String[] args) {
      // Input from user
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter a line of integers separated by spaces: ");
      String line = scanner.nextLine();

      // Create a StringTokenizer to split the input line
      StringTokenizer tokenizer = new StringTokenizer(line);
      int sum = 0;

      // Display each integer and calculate the sum
      System.out.println("Entered integers are:");
      while (tokenizer.hasMoreTokens()) {
         String token = tokenizer.nextToken();
         int number = Integer.parseInt(token);
         System.out.println(number);
         sum += number;
      }

      // Display the sum of all integers
      System.out.println("Sum of all integers: " + sum);
   }
}
```

Test Cases

Test Case 1:

Input: 1 2 3 4 5
Expected Output:
Entered integers are:
1
2
3
4
5
Sum of all integers: 15

Test Case 2:

Input: 10 20 30
Expected Output:
Entered integers are:
10
20
30
Sum of all integers: 60

8. Implement a java program to print all tokens of a string on the bases of multiple separators (use StringTokenizer class).

Aim

To implement a Java program that prints all tokens of a string based on multiple separators using the StringTokenizer class.

Algorithm

Import Classes: Import necessary classes, including Scanner and StringTokenizer.
Read Input: Use Scanner to read a line of text from the user.
Specify Delimiters: Define a string containing all separator characters.
Tokenize Input: Use StringTokenizer to tokenize the input string based on the specified delimiters.
Display Tokens: Iterate over tokens and display each token.

Source Code

```
import java.util.Scanner;
import java.util.StringTokenizer;

public class MultiSeparatorTokenizer {
    public static void main(String[] args) {
        // Input from user
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a string with multiple separators: ");
        String input = scanner.nextLine();

        // Define multiple separators
        String delimiters = " ,;|:"; // Example: space, comma, semicolon, pipe, colon

        // Create a StringTokenizer to split the input based on the specified delimiters
        StringTokenizer tokenizer = new StringTokenizer(input, delimiters);
```

```
        // Display each token
        System.out.println("Tokens are:");
        while (tokenizer.hasMoreTokens()) {
            String token = tokenizer.nextToken();
            System.out.println(token);
        }
    }
}
```

Test Cases

Test Case 1:

Input: apple,orange;banana|grape:melon
Expected Output:

Tokens are:
apple
orange
banana
grape
melon

Test Case 2:

Input: cat dog;fish|bird:hamster
Expected Output:
Tokens are:
cat
dog
fish
bird
hamster

9. Develop a java application with Employee class with Emp_name, Emp_id, Address,Mail_id, Mobile_no as members. Inherit the classes, Programmer, Assistant Professor,Associate Professor and Professor from employee class. Add Basic Pay (BP) as the memberof all the inherited classes with 97% of BP as DA, 10 % of BP as HRA, 12% of BP as PF,0.1% of BP for staff club fund. Generate pay slips for the employees with their gross and netsalary.

Aim

To develop a Java application with an Employee class and its subclasses (Programmer, AssistantProfessor, AssociateProfessor, Professor). Each subclass includes salary calculation details to generate pay slips for employees.

Algorithm

Define the Employee Class: Create an Employee class with common attributes and methods.
Define Subclasses: Create subclasses for Programmer, AssistantProfessor, AssociateProfessor, and Professor inheriting from Employee.
Add Basic Pay (BP): Include BP as an attribute in each subclass.
Salary Calculation: Implement methods to calculate DA, HRA, PF, staff club fund, gross salary, and net salary.
Generate Pay Slips: Implement a method to generate pay slips displaying the salary details.
Main Method: Create instances of each subclass, set values, and display the pay slips.

Source Code

```
import java.util.Scanner;

class Employee {
    String empName;
    int empId;
    String address;
    String mailId;
    String mobileNo;

    // Constructor
    public Employee(String empName, int empId, String address, String mailId, String mobileNo) {
        this.empName = empName;
        this.empId = empId;
        this.address = address;
        this.mailId = mailId;
        this.mobileNo = mobileNo;
    }

    // Display common details
    public void displayDetails() {
        System.out.println("Employee Name: " + empName);
        System.out.println("Employee ID: " + empId);
        System.out.println("Address: " + address);
        System.out.println("Mail ID: " + mailId);
        System.out.println("Mobile No: " + mobileNo);
```

```
    }
}

class Programmer extends Employee {
    double basicPay;

    public Programmer(String empName, int empId, String address, String mailId,
String mobileNo, double basicPay) {
        super(empName, empId, address, mailId, mobileNo);
        this.basicPay = basicPay;
    }

    public void generatePaySlip() {
        double DA = 0.97 * basicPay;
        double HRA = 0.10 * basicPay;
        double PF = 0.12 * basicPay;
        double staffClubFund = 0.001 * basicPay;
        double grossSalary = basicPay + DA + HRA;
        double netSalary = grossSalary - PF - staffClubFund;

        displayDetails();
        System.out.println("Basic Pay: Rs. " + basicPay);
        System.out.println("DA: Rs. " + DA);
        System.out.println("HRA: Rs. " + HRA);
        System.out.println("PF: Rs. " + PF);
        System.out.println("Staff Club Fund: Rs. " + staffClubFund);
        System.out.println("Gross Salary: Rs. " + grossSalary);
        System.out.println("Net Salary: Rs. " + netSalary);
    }
}

class AssistantProfessor extends Employee {
    double basicPay;

    public AssistantProfessor(String empName, int empId, String address, String mailId,
String mobileNo, double basicPay) {
        super(empName, empId, address, mailId, mobileNo);
        this.basicPay = basicPay;
    }

    public void generatePaySlip() {
        double DA = 0.97 * basicPay;
        double HRA = 0.10 * basicPay;
        double PF = 0.12 * basicPay;
```

```
        double staffClubFund = 0.001 * basicPay;
        double grossSalary = basicPay + DA + HRA;
        double netSalary = grossSalary - PF - staffClubFund;

        displayDetails();
        System.out.println("Basic Pay: Rs. " + basicPay);
        System.out.println("DA: Rs. " + DA);
        System.out.println("HRA: Rs. " + HRA);
        System.out.println("PF: Rs. " + PF);
        System.out.println("Staff Club Fund: Rs. " + staffClubFund);
        System.out.println("Gross Salary: Rs. " + grossSalary);
        System.out.println("Net Salary: Rs. " + netSalary);
    }
}

class AssociateProfessor extends Employee {
    double basicPay;

    public AssociateProfessor(String empName, int empId, String address, String
mailId, String mobileNo, double basicPay) {
        super(empName, empId, address, mailId, mobileNo);
        this.basicPay = basicPay;
    }

    public void generatePaySlip() {
        double DA = 0.97 * basicPay;
        double HRA = 0.10 * basicPay;
        double PF = 0.12 * basicPay;
        double staffClubFund = 0.001 * basicPay;
        double grossSalary = basicPay + DA + HRA;
        double netSalary = grossSalary - PF - staffClubFund;

        displayDetails();
        System.out.println("Basic Pay: Rs. " + basicPay);
        System.out.println("DA: Rs. " + DA);
        System.out.println("HRA: Rs. " + HRA);
        System.out.println("PF: Rs. " + PF);
        System.out.println("Staff Club Fund: Rs. " + staffClubFund);
        System.out.println("Gross Salary: Rs. " + grossSalary);
        System.out.println("Net Salary: Rs. " + netSalary);
    }
}

class Professor extends Employee {
```

```
    double basicPay;

    public Professor(String empName, int empId, String address, String mailId, String mobileNo, double basicPay) {
        super(empName, empId, address, mailId, mobileNo);
        this.basicPay = basicPay;
    }

    public void generatePaySlip() {
        double DA = 0.97 * basicPay;
        double HRA = 0.10 * basicPay;
        double PF = 0.12 * basicPay;
        double staffClubFund = 0.001 * basicPay;
        double grossSalary = basicPay + DA + HRA;
        double netSalary = grossSalary - PF - staffClubFund;

        displayDetails();
        System.out.println("Basic Pay: Rs. " + basicPay);
        System.out.println("DA: Rs. " + DA);
        System.out.println("HRA: Rs. " + HRA);
        System.out.println("PF: Rs. " + PF);
        System.out.println("Staff Club Fund: Rs. " + staffClubFund);
        System.out.println("Gross Salary: Rs. " + grossSalary);
        System.out.println("Net Salary: Rs. " + netSalary);
    }
}

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

        // Example for creating a Programmer and generating a pay slip
        System.out.print("Enter Programmer Name: ");
        String name = scanner.nextLine();
        System.out.print("Enter Programmer ID: ");
        int id = scanner.nextInt();
        scanner.nextLine(); // consume newline
        System.out.print("Enter Programmer Address: ");
        String address = scanner.nextLine();
        System.out.print("Enter Programmer Mail ID: ");
        String mailId = scanner.nextLine();
        System.out.print("Enter Programmer Mobile No: ");
        String mobileNo = scanner.nextLine();
        System.out.print("Enter Programmer Basic Pay: ");
```

```
        double basicPay = scanner.nextDouble();

        Programmer programmer = new Programmer(name, id, address, mailId, mobileNo, basicPay);
        programmer.generatePaySlip();

        // You can repeat similar steps for Assistant Professor, Associate Professor, and Professor
    }
}
```

Test Cases

Test Case 1: Programmer

Input:
Name: John Doe
ID: 101
Address: 123 Elm St
Mail ID: john.doe@example.com
Mobile No: 1234567890
Basic Pay: 50000

Expected Output:
Employee Name: John Doe
Employee ID: 101
Address: 123 Elm St
Mail ID: john.doe@example.com
Mobile No: 1234567890
Basic Pay: Rs. 50000.0
DA: Rs. 48500.0
HRA: Rs. 5000.0
PF: Rs. 6000.0
Staff Club Fund: Rs. 50.0
Gross Salary: Rs. 103500.0
Net Salary: Rs. 97450.0

Test Case 2: Assistant Professor

Input:
Name: Jane Smith
ID: 102
Address: 456 Oak St
Mail ID: jane.smith@example.com

Mobile No: 0987654321
Basic Pay: 60000

Expected Output:
Employee Name: Jane Smith
Employee ID: 102
Address: 456 Oak St
Mail ID: jane.smith@example.com
Mobile No: 0987654321
Basic Pay: Rs. 60000.0
DA: Rs. 58200.0
HRA: Rs. 6000.0
PF: Rs. 7200.0
Staff Club Fund: Rs. 60.0
Gross Salary: Rs. 124200.0
Net Salary: Rs. 116940.0

Unit – 2

Inheritance, Interface & Packages

1. Write a Java program to create a vehicle class hierarchy. The base class should be Vehicle, with subclasses Truck, Car and Motorcycle. Each subclass should have properties such as make, model, year, and fuel type. Implement methods for calculating fuel efficiency, distance traveled, and maximum speed. Use the super keyword to initialize the superclass attributes and invoke superclass methods where appropriate. Apply the principles of inheritance and method overriding.

   Constraints:
   Fuel Efficiency:

   - ✓ Truck: 15 miles per gallon
   - ✓ Car: 30 miles per gallon
   - ✓ Motorcycle: 50 miles per gallon

   Distance Traveled:

   Calculated as fuelConsumed * calculateFuelEfficiency()

   Maximum Speed:
   - ✓ Truck: 120 mph
   - ✓ Car: 150 mph

- ✓ Motorcycle: 180 mph

Aim

To create a vehicle class hierarchy with a base class Vehicle and subclasses Truck, Car, and Motorcycle. Each subclass has properties such as make, model, year, and fuelType, and methods for calculating fuel efficiency, distance traveled, and maximum speed.

Algorithm

Define the Vehicle Class: Create a base class Vehicle with common attributes and methods.
Define Subclasses: Create subclasses Truck, Car, and Motorcycle inheriting from Vehicle.
Implement Methods: Override methods to calculate fuel efficiency, distance traveled, and maximum speed in each subclass.
Use the Super Keyword: Use super to initialize superclass attributes and invoke superclass methods where appropriate.
Main Method: Create instances of each subclass, set values, and display results.

Source Code

```
class Vehicle {
    String make;
    String model;
    int year;
    String fuelType;

    // Constructor
    public Vehicle(String make, String model, int year, String fuelType) {
        this.make = make;
        this.model = model;
        this.year = year;
        this.fuelType = fuelType;
    }

    // Calculate fuel efficiency (to be overridden by subclasses)
    public double calculateFuelEfficiency() {
        return 0;
    }

    // Calculate distance traveled
    public double calculateDistanceTraveled(double fuelConsumed) {
        return fuelConsumed * calculateFuelEfficiency();
    }

    // Calculate maximum speed (to be overridden by subclasses)
```

```
    public double calculateMaxSpeed() {
        return 0;
    }

    // Display vehicle details
    public void displayDetails() {
        System.out.println("Make: " + make);
        System.out.println("Model: " + model);
        System.out.println("Year: " + year);
        System.out.println("Fuel Type: " + fuelType);
    }
}

class Truck extends Vehicle {
    public Truck(String make, String model, int year, String fuelType) {
        super(make, model, year, fuelType);
    }

    @Override
    public double calculateFuelEfficiency() {
        return 15; // 15 miles per gallon
    }

    @Override
    public double calculateMaxSpeed() {
        return 120; // 120 mph
    }
}

class Car extends Vehicle {
    public Car(String make, String model, int year, String fuelType) {
        super(make, model, year, fuelType);
    }

    @Override
    public double calculateFuelEfficiency() {
        return 30; // 30 miles per gallon
    }

    @Override
    public double calculateMaxSpeed() {
        return 150; // 150 mph
    }
}
```

```
class Motorcycle extends Vehicle {
    public Motorcycle(String make, String model, int year, String fuelType) {
        super(make, model, year, fuelType);
    }

    @Override
    public double calculateFuelEfficiency() {
        return 50; // 50 miles per gallon
    }

    @Override
    public double calculateMaxSpeed() {
        return 180; // 180 mph
    }
}

public class VehicleTest {
    public static void main(String[] args) {
        Truck truck = new Truck("Ford", "F-150", 2021, "Diesel");
        Car car = new Car("Toyota", "Camry", 2022, "Petrol");
        Motorcycle motorcycle = new Motorcycle("Yamaha", "YZF-R3", 2023, "Petrol");

        System.out.println("Truck Details:");
        truck.displayDetails();
        System.out.println("Fuel Efficiency: " + truck.calculateFuelEfficiency() + " miles per gallon");
        System.out.println("Distance Traveled (for 10 gallons): " + truck.calculateDistanceTraveled(10) + " miles");
        System.out.println("Maximum Speed: " + truck.calculateMaxSpeed() + " mph\n");

        System.out.println("Car Details:");
        car.displayDetails();
        System.out.println("Fuel Efficiency: " + car.calculateFuelEfficiency() + " miles per gallon");
        System.out.println("Distance Traveled (for 10 gallons): " + car.calculateDistanceTraveled(10) + " miles");
        System.out.println("Maximum Speed: " + car.calculateMaxSpeed() + " mph\n");

        System.out.println("Motorcycle Details:");
        motorcycle.displayDetails();
        System.out.println("Fuel Efficiency: " + motorcycle.calculateFuelEfficiency() + " miles per gallon");
```

```
        System.out.println("Distance Traveled (for 10 gallons): " +
motorcycle.calculateDistanceTraveled(10) + " miles");
        System.out.println("Maximum Speed: " + motorcycle.calculateMaxSpeed() + "
mph");
    }
}
```

Test Cases

Test Case 1: Truck

Input: Truck details
Make: Ford
Model: F-150
Year: 2021
Fuel Type: Diesel

Expected Output:
Truck Details:
Make: Ford
Model: F-150
Year: 2021
Fuel Type: Diesel
Fuel Efficiency: 15 miles per gallon
Distance Traveled (for 10 gallons): 150 miles
Maximum Speed: 120 mph

Test Case 2: Car

Input: Car details
Make: Toyota
Model: Camry
Year: 2022
Fuel Type: Petrol

Expected Output:
Car Details:
Make: Toyota
Model: Camry
Year: 2022
Fuel Type: Petrol
Fuel Efficiency: 30 miles per gallon
Distance Traveled (for 10 gallons): 300 miles

Maximum Speed: 150 mph

2. Develop a JAVA program to create an abstract class Shape with abstract methods calculateArea() and calculatePerimeter(). Create subclasses Circle and Triangle that extend the Shape class and implement the respective methods to calculate the area and perimeter of each shape.

Aim

To create an abstract class Shape with abstract methods calculateArea() and calculatePerimeter(), and to implement subclasses Circle and Triangle that extend the Shape class and calculate the area and perimeter of each shape.

Algorithm

Define Abstract Class Shape: Create an abstract class Shape with abstract methods calculateArea() and calculatePerimeter().
Define Subclass Circle: Create a Circle class that extends Shape and implements the abstract methods.
Define Subclass Triangle: Create a Triangle class that extends Shape and implements the abstract methods.
Main Method: Create instances of Circle and Triangle, set values, and display the results.

Source Code

```
abstract class Shape {
    // Abstract methods
    abstract double calculateArea();
    abstract double calculatePerimeter();
}

class Circle extends Shape {
    double radius;

    // Constructor
    public Circle(double radius) {
        this.radius = radius;
    }

    @Override
    double calculateArea() {
        return Math.PI * radius * radius;
    }

    @Override
    double calculatePerimeter() {
```

```
        return 2 * Math.PI * radius;
    }
}

class Triangle extends Shape {
    double side1, side2, side3;

    // Constructor
    public Triangle(double side1, double side2, double side3) {
        this.side1 = side1;
        this.side2 = side2;
        this.side3 = side3;
    }

    @Override
    double calculateArea() {
        double s = (side1 + side2 + side3) / 2;
        return Math.sqrt(s * (s - side1) * (s - side2) * (s - side3));
    }

    @Override
    double calculatePerimeter() {
        return side1 + side2 + side3;
    }
}

public class ShapeTest {
    public static void main(String[] args) {
        Circle circle = new Circle(5);
        System.out.println("Circle:");
        System.out.println("Radius: " + circle.radius);
        System.out.println("Area: " + circle.calculateArea());
        System.out.println("Perimeter: " + circle.calculatePerimeter());

        Triangle triangle = new Triangle(3, 4, 5);
        System.out.println("\nTriangle:");
        System.out.println("Sides: " + triangle.side1 + ", " + triangle.side2 + ", " +
triangle.side3);
        System.out.println("Area: " + triangle.calculateArea());
        System.out.println("Perimeter: " + triangle.calculatePerimeter());
    }
}
```

Test Cases

Test Case 1: Circle

Input: Circle with radius 5
Expected Output:
Circle:
Radius: 5.0
Area: 78.53981633974483
Perimeter: 31.41592653589793

Test Case 2: Triangle

Input: Triangle with sides 3, 4, 5
Expected Output:
Triangle:
Sides: 3.0, 4.0, 5.0
Area: 6.0
Perimeter: 12.0

3. You are required to implement a banking system with a hierarchy of classes representing different types of bank accounts. The base class should be Bank, containing a list of accounts and methods for adding new accounts. Create an Account interface with methods for depositing, withdrawing, calculating interest, and viewing balances.

   Implement two types of accounts: SavingsAccount and CurrentAccount, both implementing the Account interface. Each account type should have its own unique methods and behaviors. Use the principles of interfaces, inheritance, and method overriding to create a comprehensive banking system.

   Constraints:
   Bank Class:
   Should maintain a list of accounts.
   Methods for adding accounts and viewing all accounts.

   Account Interface:

   - Methods: deposit(double amount), withdraw(double amount), calculateInterest(), viewBalance()

   SavingsAccount:

   - Interest rate: 3% per annum
   - Additional method: applyInterest()

CurrentAccount:

- Overdraft limit: $500
- Additional method: checkOverdraft()

Aim

To implement a banking system with a hierarchy of classes representing different types of bank accounts using principles of interfaces, inheritance, and method overriding.

Algorithm

Define the Account Interface: Create an interface Account with methods deposit(double amount), withdraw(double amount), calculateInterest(), and viewBalance().

Define the Bank Class: Create a Bank class to maintain a list of accounts and methods for adding accounts and viewing all accounts.

Define the SavingsAccount Class: Implement the Account interface and add an additional method applyInterest().

Define the CurrentAccount Class: Implement the Account interface and add an additional method checkOverdraft().

Main Method: Create instances of SavingsAccount and CurrentAccount, perform operations, and display results.

Source Code

```
import java.util.ArrayList;

import java.util.List;



// Account interface

interface Account {

    void deposit(double amount);

    void withdraw(double amount);

    void calculateInterest();

    double viewBalance();
```

```
}

// Bank class
class Bank {
    private List<Account> accounts = new ArrayList<>();

    // Method to add new account
    public void addAccount(Account account) {
        accounts.add(account);
    }

    // Method to view all accounts
    public void viewAllAccounts() {
        for (Account account : accounts) {
            System.out.println(account.toString());
        }
    }
}

// SavingsAccount class
class SavingsAccount implements Account {
    private double balance;
    private final double interestRate = 0.03;

    public SavingsAccount(double initialBalance) {
```

```
    this.balance = initialBalance;
}

@Override
public void deposit(double amount) {
    balance += amount;
    System.out.println("Deposited " + amount + ", new balance: " + balance);
}

@Override
public void withdraw(double amount) {
    if (amount <= balance) {
        balance -= amount;
        System.out.println("Withdrew " + amount + ", new balance: " + balance);
    } else {
        System.out.println("Insufficient balance for withdrawal!");
    }
}

@Override
public void calculateInterest() {
    double interest = balance * interestRate;
    balance += interest;
    System.out.println("Interest applied, new balance: " + balance);
}
```

```
    @Override
    public double viewBalance() {
        return balance;
    }

    public void applyInterest() {
        calculateInterest();
    }

    @Override
    public String toString() {
        return "SavingsAccount balance: " + balance;
    }
}

// CurrentAccount class
class CurrentAccount implements Account {
    private double balance;
    private final double overdraftLimit = 500;

    public CurrentAccount(double initialBalance) {
        this.balance = initialBalance;
    }
```

```
@Override
public void deposit(double amount) {
    balance += amount;
    System.out.println("Deposited " + amount + ", new balance: " + balance);
}

@Override
public void withdraw(double amount) {
    if (amount <= balance + overdraftLimit) {
        balance -= amount;
        System.out.println("Withdrew " + amount + ", new balance: " + balance);
    } else {
        System.out.println("Overdraft limit exceeded!");
    }
}

@Override
public void calculateInterest() {
    System.out.println("No interest calculation for current account.");
}

@Override
public double viewBalance() {
    return balance;
}
```

```
    public void checkOverdraft() {
        if (balance < 0) {
            System.out.println("Account is in overdraft by " + Math.abs(balance));
        } else {
            System.out.println("No overdraft, balance is " + balance);
        }
    }

    @Override
    public String toString() {
        return "CurrentAccount balance: " + balance;
    }
}

public class BankSystem {
    public static void main(String[] args) {
        Bank bank = new Bank();

        SavingsAccount savingsAccount = new SavingsAccount(1000);
        CurrentAccount currentAccount = new CurrentAccount(500);

        bank.addAccount(savingsAccount);
        bank.addAccount(currentAccount);
```

```
        System.out.println("Performing operations on Savings Account:");
        savingsAccount.deposit(200);
        savingsAccount.withdraw(150);
        savingsAccount.applyInterest();
        System.out.println("Final balance: " + savingsAccount.viewBalance() + "\n");

        System.out.println("Performing operations on Current Account:");
        currentAccount.deposit(300);
        currentAccount.withdraw(700);
        currentAccount.checkOverdraft();
        System.out.println("Final balance: " + currentAccount.viewBalance() + "\n");

        System.out.println("Viewing all accounts in the bank:");
        bank.viewAllAccounts();
    }
}
```

Test Cases

Test Case 1: Savings Account

Initial Balance: 1000

Operations:

Deposit: 200

Withdraw: 150

Apply Interest

Expected Output:

Deposited 200.0, new balance: 1200.0

Withdrew 150.0, new balance: 1050.0

Interest applied, new balance: 1081.5

Final balance: 1081.5

Test Case 2: Current Account

Initial Balance: 500

Operations:

Deposit: 300

Withdraw: 700

Check Overdraft

Expected Output:

Deposited 300.0, new balance: 800.0

Withdrew 700.0, new balance: 100.0

No overdraft, balance is 100.0

Final balance: 100.0

Test Case 3: Viewing All Accounts

Expected Output:

Viewing all accounts in the bank:

SavingsAccount balance: 1081.5

CurrentAccount balance: 100.0

4. Create a package ‘student.Fulltime.Btech‘ in your current working directory
   a. Create a default class student in the above package with the following attributes: Name, age, sex.
   b. Have methods for storing as well as displaying.

5. You are tasked with designing a Java application to manage a library system. The system involves different types of library items such as Books and DVDs, each with specific attributes and behaviors. Implement inheritance and interface implementation using the extends and implements keywords to model these entities.

   Requirements

   Library Item Interface:

   - Create an interface LibraryItem with the following methods:
     - void checkout() - to handle the checkout process for the item.
     - void returnItem() - to handle the return process for the item.

   Book Class:

   - Create a class Book that implements the LibraryItem interface.
   - Attributes:
     - String title (title of the book)
     - String author (author of the book)
   - Methods:
     - Implement the checkout() and returnItem() methods to specify how a book is checked out and returned.

   DVD Class:

   - Create a class DVD that also implements the LibraryItem interface.
   - Attributes:
     - String title (title of the DVD)
     - String director (director of the DVD)
   - Methods:
     - Implement the checkout() and returnItem() methods to specify how a DVD is checked out and returned.

   Aim

   To design a Java application to manage a library system using inheritance and interface implementation.

   Algorithm

   Define LibraryItem Interface: Create an interface LibraryItem with methods checkout() and returnItem().

Define Book Class: Create a Book class that implements the LibraryItem interface with attributes title and author.
Define DVD Class: Create a DVD class that implements the LibraryItem interface with attributes title and director.
Implement Methods: Implement the methods checkout() and returnItem() in both Book and DVD classes.
Main Method: Create instances of Book and DVD, perform operations, and display results.

Source Code

LibraryItem.java:

```
public interface LibraryItem {
    void checkout();
    void returnItem();
}
```

Book.java:

```
public class Book implements LibraryItem {
    private String title;
    private String author;
    private boolean isCheckedOut;

    // Constructor
    public Book(String title, String author) {
        this.title = title;
        this.author = author;
        this.isCheckedOut = false;
    }

    @Override
    public void checkout() {
        if (!isCheckedOut) {
            isCheckedOut = true;
            System.out.println("Book \"" + title + "\" by " + author + " has been checked
out.");
        } else {
            System.out.println("Book \"" + title + "\" by " + author + " is already checked
out.");
        }
    }

    @Override
    public void returnItem() {
        if (isCheckedOut) {
```

```
            isCheckedOut = false;
            System.out.println("Book \"" + title + "\" by " + author + " has been
returned.");
        } else {
            System.out.println("Book \"" + title + "\" by " + author + " was not checked
out.");
        }
    }

    @Override
    public String toString() {
        return "Book [Title: " + title + ", Author: " + author + "]";
    }

    public static void main(String[] args) {
        // Test Book class
        Book book = new Book("To Kill a Mockingbird", "Harper Lee");
        System.out.println(book);
        book.checkout();
        book.returnItem();
        book.returnItem();
    }
}
```

DVD.java:

```
public class DVD implements LibraryItem {
    private String title;
    private String director;
    private boolean isCheckedOut;

    // Constructor
    public DVD(String title, String director) {
        this.title = title;
        this.director = director;
        this.isCheckedOut = false;
    }

    @Override
    public void checkout() {
        if (!isCheckedOut) {
            isCheckedOut = true;
            System.out.println("DVD \"" + title + "\" directed by " + director + " has been
checked out.");
        } else {
```

```
            System.out.println("DVD \"" + title + "\" directed by " + director + " is already
checked out.");
        }
    }

    @Override
    public void returnItem() {
        if (isCheckedOut) {
            isCheckedOut = false;
            System.out.println("DVD \"" + title + "\" directed by " + director + " has been
returned.");
        } else {
            System.out.println("DVD \"" + title + "\" directed by " + director + " was not
checked out.");
        }
    }

    @Override
    public String toString() {
        return "DVD [Title: " + title + ", Director: " + director + "]";
    }

    public static void main(String[] args) {
        // Test DVD class
        DVD dvd = new DVD("Inception", "Christopher Nolan");
        System.out.println(dvd);
        dvd.checkout();
        dvd.returnItem();
        dvd.returnItem();
    }
}
```

LibraryTest.java:

```
public class LibraryTest {
    public static void main(String[] args) {
        // Create instances of Book and DVD
        Book book1 = new Book("1984", "George Orwell");
        DVD dvd1 = new DVD("The Matrix", "The Wachowskis");

        // Perform operations on Book
        System.out.println(book1);
        book1.checkout();
        book1.returnItem();
```

```
        // Perform operations on DVD
        System.out.println(dvd1);
        dvd1.checkout();
        dvd1.returnItem();
    }
}
```

Expected Output:

Book [Title: 1984, Author: George Orwell]
Book "1984" by George Orwell has been checked out.
Book "1984" by George Orwell has been returned.

DVD [Title: The Matrix, Director: The Wachowskis]
DVD "The Matrix" directed by The Wachowskis has been checked out.
DVD "The Matrix" directed by The Wachowskis has been returned.

6. You are given an interface AdvancedArithmetic which contains a method signature int divisor_sum(int n). You need to write a class called MyCalculator which implements the interface.
   divisorSum function just takes an integer as input and return the sum of all its divisors. For example divisors of 6 are 1, 2, 3 and 6, so divisor_sum should return 12. The value of n will be at most 1000.

   Sample Input
   6

   Sample Output

   I implemented: AdvancedArithmetic
   12

Aim

To implement a class MyCalculator that calculates the sum of all divisors of a given integer n by implementing the AdvancedArithmetic interface.

Algorithm

Define AdvancedArithmetic Interface: Create an interface AdvancedArithmetic with a method int divisor_sum(int n).
Implement MyCalculator Class: Implement the AdvancedArithmetic interface in the MyCalculator class.
Implement divisor_sum Method: In the MyCalculator class, implement the divisor_sum method to calculate and return the sum of all divisors of n.

Main Method: Create an instance of MyCalculator, invoke the divisor_sum method, and display the result.

Source Code

AdvancedArithmetic.java:

```
public interface AdvancedArithmetic {
   int divisor_sum(int n);
}
```

MyCalculator.java:

```
public class MyCalculator implements AdvancedArithmetic {

   @Override
   public int divisor_sum(int n) {
      int sum = 0;
      for (int i = 1; i <= n; i++) {
         if (n % i == 0) {
            sum += i;
         }
      }
      return sum;
   }

   public static void main(String[] args) {
      MyCalculator myCalculator = new MyCalculator();
      int n = 6;  // Sample Input
      System.out.println("I implemented: AdvancedArithmetic");
      System.out.println(myCalculator.divisor_sum(n));  // Sample Output
   }
}
```

Expected Output

```
I implemented: AdvancedArithmetic
12
```

7. Implement a Java program for the following
   a) Creation of simple package.
   b) Accessing a package.

Aim

To create a simple package in Java and access it from another Java file.

Algorithm

Create Package Directory: Create a directory structure for the package.
Define a Class in the Package: Create a class within the package.
Compile the Package: Compile the class to generate the package.
Access the Package: Create another class in a different file that accesses the class from the package.

Steps

Create Directory Structure:
mypackage/

Define a Class in the Package:
Create a file named MyClass.java inside the mypackage/ directory.

Source Code

mypackage/MyClass.java:

```
package mypackage;

public class MyClass {
    public void displayMessage() {
        System.out.println("Hello from MyClass in mypackage!");
    }
}
```

Compile the Package:

Navigate to the parent directory of mypackage and compile the class:

```
javac mypackage/MyClass.java
```

Access the Package:

Create another file named PackageTest.java in the parent directory of mypackage.
PackageTest.java:

```
import mypackage.MyClass;

public class PackageTest {
    public static void main(String[] args) {
        MyClass myClass = new MyClass();
        myClass.displayMessage();
    }
}
```

Compile and Run the Program:

Compile PackageTest.java
javac PackageTest.java
java PackageTest

To compile and execute the code, ensure the directory structure is correct and run the following commands:
javac mypackage/MyClass.java
javac PackageTest.java
java PackageTest

Expected Output
Hello from MyClass in mypackage!

Exception Handling

1. a). Implement a Java program to create a method that takes an integer as a parameter and throws an exception if the number is odd.

   Aim
   To implement a Java program that creates a method taking an integer as a parameter and throws an exception if the number is odd.

   Algorithm
   Define a Custom Exception: Create a custom exception class that extends Exception.
   Create a Method to Check the Number: Define a method that takes an integer as a parameter and throws the custom exception if the number is odd.
   Test the Method: Implement a main method to test the functionality

   Source Code
   OddNumberException.java:

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

   OddNumberChecker.java:

```
public class OddNumberChecker {
    // Method to check if the number is odd and throw an exception
```

```
    public void checkNumber(int number) throws OddNumberException {
        if (number % 2 != 0) {
            throw new OddNumberException("The number " + number + " is odd.");
        } else {
            System.out.println("The number " + number + " is even.");
        }
    }

    public static void main(String[] args) {
        OddNumberChecker checker = new OddNumberChecker();

        int testNumber = 3; // You can change this number to test different cases

        try {
            checker.checkNumber(testNumber);
        } catch (OddNumberException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Expected Output

If the test number is odd (e.g., 3):
The number 3 is odd.

If the test number is even (e.g., 4):
The number 4 is even.

b). Implement a Java program that reads a list of integers from the user and throws an exception if any numbers are duplicates.

Aim

To implement a Java program that reads a list of integers from the user and throws an exception if any numbers are duplicates.

Algorithm

Define a Custom Exception: Create a custom exception class that extends Exception.

Create a Method to Check for Duplicates: Define a method that takes a list of integers as a parameter and throws the custom exception if any duplicates are found.

Read User Input: Read a list of integers from the user.

Test the Method: Implement a main method to test the functionality.

Source Code

DuplicateNumberException.java:

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

DuplicateNumberChecker.java:

```
import java.util.HashSet;
import java.util.Scanner;
import java.util.Set;

public class DuplicateNumberChecker {
    // Method to check for duplicates
    public void checkForDuplicates(int[] numbers) throws DuplicateNumberException {
        Set<Integer> numberSet = new HashSet<>();
        for (int number : numbers) {
            if (!numberSet.add(number)) {
                throw new DuplicateNumberException("Duplicate number found: " + number);
            }
        }
        System.out.println("No duplicates found.");
    }

    public static void main(String[] args) {
        DuplicateNumberChecker checker = new DuplicateNumberChecker();
```

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

        System.out.println("Enter the number of integers:");
        int n = scanner.nextInt();

        int[] numbers = new int[n];
        System.out.println("Enter the integers:");

        for (int i = 0; i < n; i++) {
            numbers[i] = scanner.nextInt();
        }

        try {
            checker.checkForDuplicates(numbers);
        } catch (DuplicateNumberException e) {
            System.out.println(e.getMessage());
        }

        scanner.close();
    }
}
```

Expected Output

If the input list has duplicates:

Enter the number of integers:

5

Enter the integers:

1 2 3 2 4

Duplicate number found: 2

If the input list has no duplicates:

Enter the number of integers:

5

Enter the integers:

1 2 3 4 5

No duplicates found.

2. a.) Implement a Java program to create a method that takes a string as input and throws an exception if the string does not contain vowels.

Aim

To implement a Java program that creates a method taking a string as input and throws an exception if the string does not contain vowels.

Algorithm

Define a Custom Exception: Create a custom exception class that extends Exception.
Create a Method to Check for Vowels: Define a method that takes a string as input and throws the custom exception if the string does not contain vowels.
Test the Method: Implement a main method to test the functionality.

Source Code

NoVowelException.java:

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

VowelChecker.java:

```
import java.util.Scanner;

public class VowelChecker {

    // Method to check if the string contains vowels
    public void checkForVowels(String input) throws NoVowelException {
        if (!input.matches(".*[AEIOUaeiou].*")) {
            throw new NoVowelException("The string does not contain any vowels: " + input);
        } else {
            System.out.println("The string contains vowels.");
        }
    }
```

```
    public static void main(String[] args) {
        VowelChecker checker = new VowelChecker();
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter a string:");
        String input = scanner.nextLine();

        try {
            checker.checkForVowels(input);
        } catch (NoVowelException e) {
            System.out.println(e.getMessage());
        }

        scanner.close();
    }
}
```

Expected Output

If the input string does not contain vowels:
Enter a string:
bcdfg
The string does not contain any vowels: bcdfg

If the input string contains vowels:
Enter a string:
hello
The string contains vowels.

b). Create a class which accepts a number as choice and returns the Month of the year. If the entered number is greater than 12 or less than 1 throw InvalidChoiceException. If the entered option is not a number throw NotANumberException.

Aim

To implement a Java program that returns the month of the year based on the user's choice, handling invalid choices and non-numeric inputs using custom exceptions.

Algorithm

Define Custom Exceptions: Create custom exception classes for invalid choices and non-numeric inputs.

Create a Class to Handle the Choice: Define a method that takes a user's choice and returns the corresponding month or throws the appropriate exception.

Test the Method: Implement a main method to test the functionality.

Source Code

InvalidChoiceException.java:

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

NotANumberException.java:

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

MonthChooser.java:

```
import java.util.Scanner;

public class MonthChooser {

    // Method to get the month based on the choice
    public String getMonth(int choice) throws InvalidChoiceException {
        String[] months = {
            "January", "February", "March", "April", "May", "June",
            "July", "August", "September", "October", "November", "December"
        };

        if (choice < 1 || choice > 12) {
```

```
            throw new InvalidChoiceException("Invalid choice: " + choice + ". Please enter a number between 1 and 12.");
        }

        return months[choice - 1];
    }

    public static void main(String[] args) {
        MonthChooser chooser = new MonthChooser();
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter a number to get the corresponding month:");

        try {
            String input = scanner.nextLine();
            int choice;

            try {
                choice = Integer.parseInt(input);
            } catch (NumberFormatException e) {
                throw new NotANumberException("The entered option is not a number: " + input);
            }

            String month = chooser.getMonth(choice);
            System.out.println("The corresponding month is: " + month);

        } catch (InvalidChoiceException | NotANumberException e) {
            System.out.println(e.getMessage());
        }
```

```
        scanner.close();
    }
}
```

Expected Output

If the input is a valid month number (e.g., 5):

Enter a number to get the corresponding month:

5

The corresponding month is: May

If the input is an invalid month number (e.g., 13):

Enter a number to get the corresponding month:

13

Invalid choice: 13. Please enter a number between 1 and 12.

If the input is not a number (e.g., "abc"):

Enter a number to get the corresponding month:

abc

The entered option is not a number: abc

3. Create an Animal class which have four methods (eat, sleep, swim, walk). Extend a class Fish that overrides walk method. Extend class Dolphin from Fish that overrides the walk method. If a walk method is called form Dolphin class, generate an exception (as fish can't walk).

Aim

To implement a Java program that demonstrates inheritance and method overriding with custom exceptions for invalid actions.

Algorithm

Define a Custom Exception: Create a custom exception class that extends Exception.
Create the Animal Class: Define the Animal class with the four methods (eat, sleep, swim, and walk).
Create the Fish Class: Extend the Animal class and override the walk method.
Create the Dolphin Class: Extend the Fish class and override the walk method to throw the custom exception.

Test the Methods: Implement a main method to test the functionality

Source Code

CannotWalkException.java:

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

Animal.java:

```
public class Animal {
    public void eat() {
        System.out.println("The animal is eating.");
    }

    public void sleep() {
        System.out.println("The animal is sleeping.");
    }

    public void swim() {
        System.out.println("The animal is swimming.");
    }

    public void walk() {
        System.out.println("The animal is walking.");
    }
}
```

Fish.java:

```
public class Fish extends Animal {
    @Override
    public void walk() {
        System.out.println("Fish don't walk, they swim.");
    }
}
```

Dolphin.java:

```
public class Dolphin extends Fish {
    @Override
    public void walk() throws CannotWalkException {
        throw new CannotWalkException("Dolphins cannot walk, they swim.");
```

```
    }
}
```

AnimalTest.java:

```
public class AnimalTest {
    public static void main(String[] args) {
        Animal animal = new Animal();
        Fish fish = new Fish();
        Dolphin dolphin = new Dolphin();

        // Testing Animal methods
        animal.eat();
        animal.sleep();
        animal.swim();
        animal.walk();

        // Testing Fish methods
        fish.eat();
        fish.sleep();
        fish.swim();
        fish.walk();

        // Testing Dolphin methods
        dolphin.eat();
        dolphin.sleep();
        dolphin.swim();
        try {
            dolphin.walk();
        } catch (CannotWalkException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Expected Output

```
The animal is eating.
The animal is sleeping.
The animal is swimming.
The animal is walking.
The animal is eating.
The animal is sleeping.
The animal is swimming.
```

Fish don't walk, they swim.
The animal is eating.
The animal is sleeping.
The animal is swimming.
Dolphins cannot walk, they swim.

4. Implement a java program in which you have two Rides (Car, Boat) and two locations (Land, Water). Ask from the user about the ride and the location. If user selects ride a car and location water, then throw an exception. Similarly, if user selects ride a boat and location water, then also throw an exception.

Aim

To implement a Java program that asks the user to select a ride and a location, and throws exceptions for invalid ride-location combinations.

Algorithm

Define Custom Exceptions: Create custom exception classes for invalid ride-location combinations.
Create the Ride and Location Handling Class: Define methods to handle the user input and check for valid combinations.
Test the Program: Implement a main method to test the functionality.

Source Code

InvalidRideLocationException.java:

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

RideLocationHandler.java:

```
import java.util.Scanner;

public class RideLocationHandler {

   public void selectRideAndLocation(String ride, String location) throws InvalidRideLocationException {
      if (ride.equalsIgnoreCase("Car") && location.equalsIgnoreCase("Water")) {
         throw new InvalidRideLocationException("You cannot ride a car on water.");
      } else if (ride.equalsIgnoreCase("Boat") && location.equalsIgnoreCase("Land")) {
         throw new InvalidRideLocationException("You cannot ride a boat on land.");
      } else {
         System.out.println("You can ride the " + ride + " on " + location + ".");
```

```
        }
    }

    public static void main(String[] args) {
        RideLocationHandler handler = new RideLocationHandler();
        Scanner scanner = new Scanner(System.in);

        System.out.println("Select your ride (Car/Boat):");
        String ride = scanner.nextLine();

        System.out.println("Select your location (Land/Water):");
        String location = scanner.nextLine();

        try {
            handler.selectRideAndLocation(ride, location);
        } catch (InvalidRideLocationException e) {
            System.out.println(e.getMessage());
        }

        scanner.close();
    }
}
```

Expected Output

For a valid ride-location combination:

```
Select your ride (Car/Boat):
Car
Select your location (Land/Water):
Land
You can ride the Car on Land.
```

For an invalid ride-location combination:

```
Select your ride (Car/Boat):
Car
Select your location (Land/Water):
Water
You cannot ride a car on water.

Select your ride (Car/Boat):
Boat
Select your location (Land/Water):
```

Land
You cannot ride a boat on land.

5. There are N number of bottles and M number of glasses that will be filled from these bottles. Each bottle can fill up 5 glasses. You have to Implement JAVA code that will ask user to input N and M, and check if the bottles are enough to fill M glasses. If bottles are not enough throw an exception informing the user. (Total number of glasses that can be made from bottles = 5*number of bottles).

Aim

To implement a Java program that asks the user to input the number of bottles and glasses, checks if the bottles are enough to fill the glasses, and throws an exception if they are not.

Algorithm

Define a custom exception for insufficient bottles.
Create a class to handle the logic for filling glasses with bottles.
Implement the main method to get user input and perform the checks.

Source Code

InsufficientBottlesException.java:

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

BottleGlassHandler.java:

```
import java.util.Scanner;

public class BottleGlassHandler {

   public void checkBottles(int numberOfBottles, int numberOfGlasses) throws
InsufficientBottlesException {
      int totalGlasses = numberOfBottles * 5;
      if (totalGlasses < numberOfGlasses) {
         throw new InsufficientBottlesException("Not enough bottles to fill " +
numberOfGlasses + " glasses.");
      } else {
         System.out.println("Sufficient bottles to fill " + numberOfGlasses + " glasses.");
      }
   }

   public static void main(String[] args) {
```

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

        System.out.println("Enter the number of bottles:");
        int numberOfBottles = scanner.nextInt();

        System.out.println("Enter the number of glasses:");
        int numberOfGlasses = scanner.nextInt();

        BottleGlassHandler handler = new BottleGlassHandler();

        try {
            handler.checkBottles(numberOfBottles, numberOfGlasses);
        } catch (InsufficientBottlesException e) {
            System.out.println(e.getMessage());
        }

        scanner.close();
    }
}
```

Expected Output

For a sufficient number of bottles:

```
Enter the number of bottles:
10
Enter the number of glasses:
50
Sufficient bottles to fill 50 glasses.
```

For an insufficient number of bottles:

```
Enter the number of bottles:
5
Enter the number of glasses:
30
Not enough bottles to fill 30 glasses.
```

6. Create a three-level hierarchy of exceptions. Now create a base-class A with a method that throws an exception at the base of your hierarchy. Inherit B from A and override the method so it throws an exception at level two of your hierarchy. Repeat by inheriting class C from B. In main (), create a C and up cast it to A, then call the method.

Aim

To implement a Java program with a three-level hierarchy of exceptions and a class hierarchy where each class throws exceptions from different levels of the exception hierarchy.

Algorithm

Define a Three-Level Hierarchy of Exceptions: Create three custom exception classes.
Create a Class Hierarchy: Implement classes A, B, and C, with each overriding a method to throw an exception from the respective level of the hierarchy.
Test the Program: In the main method, upcast an instance of C to A and call the method to observe the exception.

Source Code

LevelOneException.java:

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

LevelTwoException.java:

```
public class LevelTwoException extends LevelOneException {
    public LevelTwoException(String message) {
        super(message);
    }
}
```

LevelThreeException.java:

```
public class LevelThreeException extends LevelTwoException {
    public LevelThreeException(String message) {
        super(message);
    }
}
```

A.java:

```
public class A {
    public void method() throws LevelOneException {
        throw new LevelOneException("Exception from Level One");
    }
}
```

B.java:

```
public class B extends A {
    @Override
    public void method() throws LevelTwoException {
        throw new LevelTwoException("Exception from Level Two");
    }
}
```

C.java:

```
public class C extends B {
    @Override
    public void method() throws LevelThreeException {
        throw new LevelThreeException("Exception from Level Three");
    }
}
```

ExceptionTest.java:

```
public class ExceptionTest {
    public static void main(String[] args) {
        A a = new C();

        try {
            a.method();
        } catch (LevelOneException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Expected Output

Exception from Level Three

7. Implement a java program using multiple catch blocks. Create a class CatchExercise inside the try block declare an array a[] and initialize with value a[5] =30/5; . In each catch block show Arithmetic exception and ArrayIndexOutOfBoundsException.

Aim

To implement a Java program that demonstrates the use of multiple catch blocks by handling ArithmeticException and ArrayIndexOutOfBoundsException.

Algorithm

Create a Class: Create a class named CatchExercise.

Implement Try-Catch Blocks: Inside the try block, declare and initialize an array. Handle ArithmeticException and ArrayIndexOutOfBoundsException in separate catch blocks.

Test the Program: Implement the main method to test the functionality.

Source Code

CatchExercise.java:

```
public class CatchExercise {
   public static void main(String[] args) {
      try {
         int[] a = new int[5];
         a[5] = 30 / 5; // This will cause ArrayIndexOutOfBoundsException
         System.out.println("Value at a[5]: " + a[5]);
      } catch (ArithmeticException e) {
         System.out.println("Arithmetic Exception: " + e.getMessage());
      } catch (ArrayIndexOutOfBoundsException e) {
         System.out.println("Array Index Out Of Bounds Exception: " + e.getMessage());
      }
   }
}
```

Expected Output

Array Index Out Of Bounds Exception: Index 5 out of bounds for length 5

8. Implement a java program that count how many prime numbers between minimum and maximum values provided by user. If minimum value is greater than or equal to maximum value, the program should throw a InvalidRange exception and handle it to display a message to the user on the following format: Invalid range: minimum is greater than or equal to maximum. (For example, if the user provided 10 as maximum and 20 as minimum, the message should be: Invalid range: 20 is greater than or equal to 10).

Aim

To implement a Java program that counts the number of prime numbers within a specified range and handles invalid ranges using custom exceptions.

Algorithm

Define a Custom Exception: Create an exception class for invalid ranges.
Implement Prime Number Check Logic: Create a method to check if a number is prime.
Implement the Main Class: Create a class to handle user input, validate the range, and count the prime numbers.
Handle the Exception: Throw and catch the custom exception when the range is invalid.

Source Code

InvalidRangeException.java:

```
public class InvalidRangeException extends Exception {
   public InvalidRangeException(String message) {
```

```
        super(message);
    }
}
```

PrimeCounter.java:

```
import java.util.Scanner;

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

        System.out.println("Enter the minimum value:");
        int min = scanner.nextInt();

        System.out.println("Enter the maximum value:");
        int max = scanner.nextInt();

        try {
            int primeCount = countPrimesInRange(min, max);
            System.out.println("Number of prime numbers between " + min + " and " + max +
": " + primeCount);
        } catch (InvalidRangeException e) {
            System.out.println(e.getMessage());
        }

        scanner.close();
    }

    public static int countPrimesInRange(int min, int max) throws InvalidRangeException {
        if (min >= max) {
            throw new InvalidRangeException("Invalid range: " + min + " is greater than or
equal to " + max);
        }

        int count = 0;
        for (int i = min; i <= max; i++) {
            if (isPrime(i)) {
                count++;
            }
        }
        return count;
    }
```

```
    public static boolean isPrime(int num) {
        if (num <= 1) {
            return false;
        }
        for (int i = 2; i <= Math.sqrt(num); i++) {
            if (num % i == 0) {
                return false;
            }
        }
        return true;
    }
}
```

Expected Output

For a valid range:
Enter the minimum value:
10
Enter the maximum value:
20
Number of prime numbers between 10 and 20: 4

For an invalid range:
Enter the minimum value:
20
Enter the maximum value:
10
Invalid range: 20 is greater than or equal to 10

Module – 2

Multi-Threading

1. Develop a multi-function utility application in Java that consists of the following threads:

   Thread 1: Prints numbers from 1 to 10 as part of a counting utility.

   Thread 2: Prints the alphabet from A to J to simulate a letter generation function.

   Thread 3: Prints the first 10 prime numbers, simulating a prime number generator tool.

Ensure that each thread runs concurrently and independently. Include appropriate user interface elements (such as console outputs) to display the tasks performed by each thread.

Aim

To implement a Java program with three concurrent threads, each performing different tasks: counting numbers, printing letters, and generating prime numbers.

Algorithm

Define Runnable Classes: Create separate classes for each task that implements the Runnable interface.

Implement the Run Method: Define the tasks in the run method of each class.

Create and Start Threads: In the main method, create and start each thread.

Display Outputs: Print outputs to the console to display the tasks performed by each thread.

Source Code

CountNumbers.java:

```
public class CountNumbers implements Runnable {
    @Override
    public void run() {
        for (int i = 1; i <= 10; i++) {
            System.out.println("Number: " + i);
            try {
                Thread.sleep(500); // Sleep for 500 milliseconds
            } catch (InterruptedException e) {
                System.out.println(e.getMessage());
            }
        }
    }
}
```

PrintLetters.java:

```
public class PrintLetters implements Runnable {
```

```
    @Override
    public void run() {
        for (char c = 'A'; c <= 'J'; c++) {
            System.out.println("Letter: " + c);
            try {
                Thread.sleep(500); // Sleep for 500 milliseconds
            } catch (InterruptedException e) {
                System.out.println(e.getMessage());
            }
        }
    }
}
```

GeneratePrimes.java:

```
public class GeneratePrimes implements Runnable {
    @Override
    public void run() {
        int count = 0;
        int number = 2;
        while (count < 10) {
            if (isPrime(number)) {
                System.out.println("Prime: " + number);
                count++;
            }
            number++;
            try {
                Thread.sleep(500); // Sleep for 500 milliseconds
            } catch (InterruptedException e) {
                System.out.println(e.getMessage());
            }
```

```
        }
    }

    private boolean isPrime(int num) {
        if (num <= 1) {
            return false;
        }
        for (int i = 2; i <= Math.sqrt(num); i++) {
            if (num % i == 0) {
                return false;
            }
        }
        return true;
    }
}
```

MultiFunctionUtility.java:

```
public class MultiFunctionUtility {
    public static void main(String[] args) {
        Thread countNumbersThread = new Thread(new CountNumbers());
        Thread printLettersThread = new Thread(new PrintLetters());
        Thread generatePrimesThread = new Thread(new GeneratePrimes());

        countNumbersThread.start();
        printLettersThread.start();
        generatePrimesThread.start();
    }
}
```

Compilation and Execution

javac CountNumbers.java PrintLetters.java GeneratePrimes.java MultiFunctionUtility.java

java MultiFunctionUtility

Expected Output

Number: 1

Letter: A

Prime: 2

Number: 2

Letter: B

Prime: 3

Number: 3

Letter: C

Prime: 5

Number: 4

Letter: D

Prime: 7

Number: 5

Letter: E

Prime: 11

Number: 6

Letter: F

Prime: 13

Number: 7

Letter: G

Prime: 17

Number: 8

Letter: H

Prime: 19

Number: 9

Letter: I

Prime: 23

Number: 10

Letter: J

Prime: 29

2. Create a Java application that simulates different states of a thread:

Begin with creating a new thread (New state).

Set the thread to runnable (Runnable state) and start the thread to transition it to the running state (Running state).

Use Thread.sleep() and synchronized blocks to simulate a waiting or blocked state.

Terminate the thread and showcase its termination (Terminated state).

Provide detailed console logs to visualize and explain the transitions between different states of the thread life cycle.

Aim

To implement a Java program that simulates and logs the various states of a thread: New, Runnable, Running, Blocked/Waiting, and Terminated.

Algorithm

Create a Thread Class: Implement a class that extends Thread and overrides the run method.

Simulate Thread States: Use Thread.sleep() and synchronized blocks to simulate waiting or blocked states.

Log Thread Transitions: Add print statements to log the state transitions.

Manage Thread Lifecycle: Create and manage the thread lifecycle in the main method.

Source Code

CustomThread.java:

```
public class CustomThread extends Thread {
    @Override
    public void run() {
        System.out.println("Thread is now in RUNNING state.");
```

```
        synchronized(this) {
            try {
                System.out.println("Thread is entering WAITING state.");
                wait(2000); // Simulate waiting state
            } catch (InterruptedException e) {
                System.out.println(e.getMessage());
            }
        }
        System.out.println("Thread is now in RUNNING state again.");
        try {
            System.out.println("Thread is entering TIMED_WAITING state.");
            Thread.sleep(2000); // Simulate timed waiting state
        } catch (InterruptedException e) {
            System.out.println(e.getMessage());
        }
        System.out.println("Thread is now TERMINATED.");
    }
}
```

ThreadStateSimulation.java:

```
public class ThreadStateSimulation {
    public static void main(String[] args) {
        System.out.println("Creating a new thread (NEW state).");
        CustomThread customThread = new CustomThread();

        System.out.println("Setting the thread to RUNNABLE state.");
        customThread.start(); // Thread transitions to RUNNABLE state

        try {
            // Main thread waits for the custom thread to complete
```

```
            customThread.join();
        } catch (InterruptedException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Compilation and Execution

javac CustomThread.java ThreadStateSimulation.java

java ThreadStateSimulation

Expected Output

Creating a new thread (NEW state).

Setting the thread to RUNNABLE state.

Thread is now in RUNNING state.

Thread is entering WAITING state.

Thread is now in RUNNING state again.

Thread is entering TIMED_WAITING state.

Thread is now TERMINATED.

3. Develop a task executor application where threads are created by extending the Thread class. Each thread should perform a specific task, such as printing a message indicating its execution.

Aim

To implement a Java application with multiple threads created by extending the Thread class, where each thread performs a specific task such as printing a message.

Algorithm

Create Multiple Thread Classes: Extend the Thread class to create different thread classes, each with its own task.
Override the Run Method: Define the specific task in the run method of each thread class.
Manage Thread Execution: Create and start the threads in the main method.
Display Outputs: Print messages to indicate the execution of each thread.

Source Code

TaskThread1.java:

```
public class TaskThread1 extends Thread {
    @Override
    public void run() {
        System.out.println("TaskThread1 is executing its task.");
    }
}
```

TaskThread2.java:

```
public class TaskThread2 extends Thread {
    @Override
    public void run() {
        System.out.println("TaskThread2 is executing its task.");
    }
}
```

TaskThread3.java:

```
public class TaskThread3 extends Thread {
    @Override
    public void run() {
        System.out.println("TaskThread3 is executing its task.");
    }
}
```

TaskExecutorApplication.java:

```
public class TaskExecutorApplication {
    public static void main(String[] args) {
        TaskThread1 thread1 = new TaskThread1();
        TaskThread2 thread2 = new TaskThread2();
        TaskThread3 thread3 = new TaskThread3();

        thread1.start();
        thread2.start();
        thread3.start();

        try {
            thread1.join();
            thread2.join();
            thread3.join();
        } catch (InterruptedException e) {
            System.out.println(e.getMessage());
        }
```

```
        System.out.println("All tasks have been executed.");
    }
}
```

Compilation and Execution

javac TaskThread1.java TaskThread2.java TaskThread3.java TaskExecutorApplication.java
java TaskExecutorApplication

Expected Output

TaskThread1 is executing its task.
TaskThread2 is executing its task.
TaskThread3 is executing its task.
All tasks have been executed.

4. Develop another version of the task executor application where threads are created by implementing the Runnable interface. Each thread should also perform a specific task such as printing a message indicating its execution.

   Compare and discuss the two approaches, highlighting the differences in implementation and use-case scenarios.

   Aim

   To implement a Java application with multiple threads created by implementing the Runnable interface, where each thread performs a specific task such as printing a message.

   Algorithm

   Create Runnable Classes: Implement the Runnable interface to create different task classes.

   Override the Run Method: Define the specific task in the run method of each Runnable class.

   Manage Thread Execution: Create and start the threads in the main method.

   Display Outputs: Print messages to indicate the execution of each thread.

   Source Code

   TaskRunnable1.java:

```
public class TaskRunnable1 implements Runnable {
    @Override
    public void run() {
```

```
        System.out.println("TaskRunnable1 is executing its task.");
    }
}
```

TaskRunnable2.java:

```
public class TaskRunnable2 implements Runnable {
    @Override
    public void run() {
        System.out.println("TaskRunnable2 is executing its task.");
    }
}
```

TaskRunnable3.java:

```
public class TaskRunnable3 implements Runnable {
    @Override
    public void run() {
        System.out.println("TaskRunnable3 is executing its task.");
    }
}
```

TaskExecutorRunnableApplication.java:

```
public class TaskExecutorRunnableApplication {
    public static void main(String[] args) {
        Runnable task1 = new TaskRunnable1();
        Runnable task2 = new TaskRunnable2();
        Runnable task3 = new TaskRunnable3();

        Thread thread1 = new Thread(task1);
        Thread thread2 = new Thread(task2);
        Thread thread3 = new Thread(task3);
```

```
        thread1.start();
        thread2.start();
        thread3.start();

        try {
            thread1.join();
            thread2.join();
            thread3.join();
        } catch (InterruptedException e) {
            System.out.println(e.getMessage());
        }

        System.out.println("All tasks have been executed.");
    }
}
```

Compilation and Execution

javac TaskRunnable1.java TaskRunnable2.java TaskRunnable3.java TaskExecutorRunnableApplication.java

java TaskExecutorRunnableApplication

Expected Output

TaskRunnable1 is executing its task.

TaskRunnable2 is executing its task.

TaskRunnable3 is executing its task.

All tasks have been executed.

Implementation

Extending the Thread Class

```
class TaskThread extends Thread {
    @Override
    public void run() {
        System.out.println("TaskThread is executing its task.");
    }
}

public class TaskExecutorThread {
    public static void main(String[] args) {
        TaskThread thread1 = new TaskThread();
        thread1.start();
    }
}

class TaskRunnable implements Runnable {
    @Override
    public void run() {
        System.out.println("TaskRunnable is executing its task.");
    }
}

public class TaskExecutorRunnable {
    public static void main(String[] args) {
        Runnable task = new TaskRunnable();
        Thread thread1 = new Thread(task);
        thread1.start();
    }
}
```

5. Implement a Java application that uses multiple threads to increment a shared counter:

   Implement the application without any synchronization to showcase the race condition and potential incorrect final value of the counter.

   Modify the application by adding synchronization using the synchronized keyword, ensuring that the final value of the counter is accurate.

   Include detailed console outputs and comments to explain the race condition and how synchronization resolves it.

   Aim

   To implement a Java application that uses multiple threads to increment a shared counter, showcasing the race condition without synchronization and resolving it with the synchronized keyword.

   Algorithm

   Create a Counter Class: This class will have a method to increment the counter.

   Implement Threads Without Synchronization: Create threads to increment the counter and demonstrate the race condition.

   Implement Threads With Synchronization: Modify the counter increment method to use the synchronized keyword and demonstrate how it resolves the race condition.

   Print Console Outputs: Include detailed console outputs to explain the race condition and the effect of synchronization.

   Source Code

   Counter Class

```
class Counter {
    private int count = 0;

    // Method to increment the counter without synchronization
    public void increment() {
        count++;
    }
```

```
    // Synchronized method to increment the counter
    public synchronized void synchronizedIncrement() {
        count++;
    }

    public int getCount() {
        return count;
    }
}
```

Threads Without Synchronization

```
public class RaceConditionExample implements Runnable {
    private Counter counter;

    public RaceConditionExample(Counter counter) {
        this.counter = counter;
    }

    @Override
    public void run() {
        for (int i = 0; i < 1000; i++) {
            counter.increment();
        }
    }
```

```
    public static void main(String[] args) {
        Counter counter = new Counter();
        Thread t1 = new Thread(new RaceConditionExample(counter));
        Thread t2 = new Thread(new RaceConditionExample(counter));

        t1.start();
        t2.start();

        try {
            t1.join();
            t2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Final counter value without synchronization: " + counter.getCount());
    }
}
```

Threads With Synchronization

```
public class SynchronizedExample implements Runnable {
    private Counter counter;

    public SynchronizedExample(Counter counter) {
        this.counter = counter;
```

```
}

@Override
public void run() {
    for (int i = 0; i < 1000; i++) {
        counter.synchronizedIncrement();
    }
}

public static void main(String[] args) {
    Counter counter = new Counter();
    Thread t1 = new Thread(new SynchronizedExample(counter));
    Thread t2 = new Thread(new SynchronizedExample(counter));

    t1.start();
    t2.start();

    try {
        t1.join();
        t2.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    System.out.println("Final counter value with synchronization: " + counter.getCount());
}
```

}

Compilation and Execution

javac Counter.java RaceConditionExample.java SynchronizedExample.java

java RaceConditionExample

java SynchronizedExample

Expected Output

RaceConditionExample Output:

Final counter value without synchronization: [less than or equal to 2000, likely less due to race condition]

SynchronizedExample Output:

Final counter value with synchronization: 2000

6. Create a priority-based task scheduler application in Java:

   Develop multiple threads with different priorities (e.g., MIN_PRIORITY, NORM_PRIORITY, MAX_PRIORITY).

   Run the threads and observe their execution order and completion time.

   Aim

   To create a priority-based task scheduler application in Java that uses multiple threads with different priorities and observes their execution order and completion time.

   Algorithm

   Create a Task Class: This class will implement the Runnable interface and define the task each thread will perform.

   Create Multiple Threads: Instantiate multiple threads with different priorities.

   Run the Threads: Start the threads and observe the execution order and completion time.

Print Console Outputs: Include console outputs to track thread execution and priority.

Source Code

Task Class

```
class Task implements Runnable {
    private String taskName;

    public Task(String taskName) {
        this.taskName = taskName;
    }

    @Override
    public void run() {
        System.out.println("Executing task: " + taskName + " with priority: " + Thread.currentThread().getPriority());
        try {
            // Simulating task execution time
            Thread.sleep(100);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Completed task: " + taskName + " with priority: " + Thread.currentThread().getPriority());
    }
}
```

Create and Run Threads with Different Priorities

```
public class PriorityTaskScheduler {
    public static void main(String[] args) {
        Task task1 = new Task("Task 1");
        Task task2 = new Task("Task 2");
        Task task3 = new Task("Task 3");

        Thread thread1 = new Thread(task1);
        Thread thread2 = new Thread(task2);
        Thread thread3 = new Thread(task3);

        // Setting thread priorities
        thread1.setPriority(Thread.MIN_PRIORITY);
        thread2.setPriority(Thread.NORM_PRIORITY);
        thread3.setPriority(Thread.MAX_PRIORITY);

        // Start the threads
        thread1.start();
        thread2.start();
        thread3.start();

        // Join the threads to ensure main thread waits for their completion
        try {
            thread1.join();
            thread2.join();
            thread3.join();
        } catch (InterruptedException e) {
```

```
            e.printStackTrace();
        }

        System.out.println("All tasks completed.");
    }
}
```

Compilation and Execution

javac Task.java PriorityTaskScheduler.java

java PriorityTaskScheduler

Expected Output

Executing task: Task 3 with priority: 10

Executing task: Task 2 with priority: 5

Executing task: Task 1 with priority: 1

Completed task: Task 3 with priority: 10

Completed task: Task 2 with priority: 5

Completed task: Task 1 with priority: 1

All tasks completed.

7. Develop a producer-consumer simulation in Java:

   Create a SharedResource class that holds shared data.

   Implement a Producer thread that generates data and puts it in the SharedResource.

   Implement a Consumer thread that retrieves and processes the data from the SharedResource.

Utilize wait(), notify(), and notifyAll() methods to manage synchronization and communication between the producer and consumer threads. Provide detailed comments and console outputs to explain how data flow and synchronization are managed.

Aim

To develop a producer-consumer simulation in Java using synchronization methods wait(), notify(), and notifyAll() for managing the communication between producer and consumer threads.

Algorithm

Create a SharedResource Class: This class will hold the shared data and provide synchronized methods for data insertion and retrieval.

Implement Producer Class: This thread will generate data and put it in the SharedResource.

Implement Consumer Class: This thread will retrieve and process data from the SharedResource.

Use Synchronization Methods: Utilize wait(), notify(), and notifyAll() to manage thread communication and synchronization.

Source Code

SharedResource Class

```
import java.util.LinkedList;

import java.util.Queue;


class SharedResource {

   private Queue<Integer> queue = new LinkedList<>();

   private int capacity;


   public SharedResource(int capacity) {

      this.capacity = capacity;

   }
```

```
    public synchronized void produce(int value) throws InterruptedException {
        while (queue.size() == capacity) {
            wait();
        }
        queue.add(value);
        System.out.println("Produced: " + value);
        notifyAll();
    }

    public synchronized int consume() throws InterruptedException {
        while (queue.isEmpty()) {
            wait();
        }
        int value = queue.poll();
        System.out.println("Consumed: " + value);
        notifyAll();
        return value;
    }
}
```

Producer Class

```
class Producer implements Runnable {
    private SharedResource resource;

    public Producer(SharedResource resource) {
```

```
        this.resource = resource;
    }

    @Override
    public void run() {
        int value = 0;
        while (true) {
            try {
                resource.produce(value++);
                Thread.sleep(100); // Simulate time taken to produce the item
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Consumer Class

```
class Consumer implements Runnable {
    private SharedResource resource;

    public Consumer(SharedResource resource) {
        this.resource = resource;
    }

    @Override
```

```
    public void run() {
        while (true) {
            try {
                resource.consume();
                Thread.sleep(150); // Simulate time taken to consume the item
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Main Class

```
public class ProducerConsumerSimulation {
    public static void main(String[] args) {
        SharedResource resource = new SharedResource(5);

        Thread producerThread = new Thread(new Producer(resource));
        Thread consumerThread = new Thread(new Consumer(resource));

        producerThread.start();
        consumerThread.start();
    }
}
```

Compilation and Execution

javac SharedResource.java Producer.java Consumer.java ProducerConsumerSimulation.java

java ProducerConsumerSimulation

Expected Output

Produced: 0

Produced: 1

Produced: 2

Produced: 3

Produced: 4

Consumed: 0

Produced: 5

Consumed: 1

Produced: 6

Consumed: 2

Produced: 7

...

8. Develop a Java application that simulates a simple data processing pipeline with:

   DataBuffer Class: A buffer to hold a single piece of data shared between the producer and consumer.

   DataProducer Thread: A thread that reads data (simulates reading data from a file or sensor) and places it into the shared buffer.

   DataProcessor Thread: A thread that processes the data from the shared buffer (simulates data processing such as transformation or analysis).

   Synchronization: Utilize basic synchronization mechanisms to ensure thread-safe operations.

   Aim

To develop a Java application that simulates a simple data processing pipeline with a shared DataBuffer class, a DataProducer thread, and a DataProcessor thread. Basic synchronization mechanisms will be used to ensure thread-safe operations.

Algorithm

Create a DataBuffer Class: This class will hold a single piece of shared data between the producer and consumer.

Implement DataProducer Class: This thread will read data (simulated) and place it into the shared buffer.

Implement DataProcessor Class: This thread will process the data from the shared buffer.

Use Synchronization: Utilize basic synchronization mechanisms (wait(), notify()) to ensure thread-safe operations.

Source Code

DataBuffer Class

```
class DataBuffer {
    private int data;
    private boolean isEmpty = true;

    public synchronized void produce(int newData) throws InterruptedException {
        while (!isEmpty) {
            wait();
        }
        data = newData;
        isEmpty = false;
        System.out.println("Produced: " + data);
        notifyAll();
    }
```

```
    public synchronized int consume() throws InterruptedException {
        while (isEmpty) {
            wait();
        }
        isEmpty = true;
        System.out.println("Consumed: " + data);
        notifyAll();
        return data;
    }
}
```

DataProducer Class

```
class DataProducer implements Runnable {
    private DataBuffer buffer;

    public DataProducer(DataBuffer buffer) {
        this.buffer = buffer;
    }

    @Override
    public void run() {
        int value = 0;
        while (true) {
            try {
                buffer.produce(value++);
```

```
                Thread.sleep(100); // Simulate time taken to produce the item
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Main Class

```
public class DataProcessingPipeline {
    public static void main(String[] args) {
        DataBuffer buffer = new DataBuffer();

        Thread producerThread = new Thread(new DataProducer(buffer));
        Thread processorThread = new Thread(new DataProcessor(buffer));

        producerThread.start();
        processorThread.start();
    }
}
```

Compilation and Execution

```
javac DataBuffer.java DataProducer.java DataProcessor.java DataProcessingPipeline.java
java DataProcessingPipeline
```

Expected Output

Produced: 0

Consumed: 0

Processed: 0

Produced: 1

Consumed: 1

Processed: 2

Produced: 2

Consumed: 2

Processed: 4

...

Java Collection Framework

1. Create a Java program that manages a music playlist using an ArrayList. Each song should have a title, artist, and duration (in seconds). Implement the following functionalities:
   - Add a new song to the playlist.
   - Remove a song from the playlist by title.
   - Display the total duration of the playlist.
   - List all songs.
   - Shuffle the playlist (randomize the order of songs).

   Write the necessary classes and methods to support these operations.

Aim

To develop a Java program that manages a music playlist using an ArrayList. The program should be able to add new songs, remove songs by title, display the total duration of the playlist, list all songs, and shuffle the playlist.

Algorithm

Create a Song Class: This class will represent a song with attributes title, artist, and duration.

Create a Playlist Class: This class will manage the list of songs using an ArrayList and implement the required functionalities.

Implement Methods: Add methods to add songs, remove songs by title, display the total duration, list all songs, and shuffle the playlist.

Main Class: Create a main class to demonstrate the functionalities.

Source Code

Song Class

```
class Song {
    private String title;
    private String artist;
    private int duration; // Duration in seconds

    public Song(String title, String artist, int duration) {
        this.title = title;
        this.artist = artist;
        this.duration = duration;
    }

    public String getTitle() {
        return title;
    }

    public String getArtist() {
        return artist;
    }

    public int getDuration() {
```

```
        return duration;
    }

    @Override
    public String toString() {
        return "Title: " + title + ", Artist: " + artist + ", Duration: " + duration + " seconds";
    }
}
```

Playlist Class

```
import java.util.ArrayList;
import java.util.Collections;

class Playlist {
    private ArrayList<Song> songs;

    public Playlist() {
        this.songs = new ArrayList<>();
    }

    public void addSong(Song song) {
        songs.add(song);
        System.out.println("Added: " + song);
    }

    public void removeSong(String title) {
```

```
        songs.removeIf(song -> song.getTitle().equalsIgnoreCase(title));
        System.out.println("Removed song with title: " + title);
    }

    public int getTotalDuration() {
        int totalDuration = 0;
        for (Song song : songs) {
            totalDuration += song.getDuration();
        }
        return totalDuration;
    }

    public void listAllSongs() {
        System.out.println("Playlist:");
        for (Song song : songs) {
            System.out.println(song);
        }
    }

    public void shuffle() {
        Collections.shuffle(songs);
        System.out.println("Shuffled playlist");
    }
}
```

Main Class

```
import java.util.Scanner;

public class MusicPlaylistManager {
    public static void main(String[] args) {
        Playlist playlist = new Playlist();
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("1. Add a new song");
            System.out.println("2. Remove a song by title");
            System.out.println("3. Display total duration of the playlist");
            System.out.println("4. List all songs");
            System.out.println("5. Shuffle the playlist");
            System.out.println("6. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume the newline character

            switch (choice) {
                case 1:
                    System.out.print("Enter title: ");
                    String title = scanner.nextLine();
                    System.out.print("Enter artist: ");
                    String artist = scanner.nextLine();
                    System.out.print("Enter duration (in seconds): ");
                    int duration = scanner.nextInt();
```

```
                scanner.nextLine(); // Consume the newline character
                Song song = new Song(title, artist, duration);
                playlist.addSong(song);
                break;
            case 2:
                System.out.print("Enter title of the song to remove: ");
                String removeTitle = scanner.nextLine();
                playlist.removeSong(removeTitle);
                break;
            case 3:
                int totalDuration = playlist.getTotalDuration();
                System.out.println("Total duration of the playlist: " + totalDuration + " seconds");
                break;
            case 4:
                playlist.listAllSongs();
                break;
            case 5:
                playlist.shuffle();
                break;
            case 6:
                System.out.println("Exiting...");
                scanner.close();
                System.exit(0);
            default:
                System.out.println("Invalid choice. Please try again.");
```

```
            }
        }
    }
}
```

Compilation and Execution

javac Song.java Playlist.java MusicPlaylistManager.java

java MusicPlaylistManager

Expected Output

1. Add a new song

2. Remove a song by title

3. Display total duration of the playlist

4. List all songs

5. Shuffle the playlist

6. Exit

Enter your choice: 1

Enter title: Song1

Enter artist: Artist1

Enter duration (in seconds): 300

Added: Title: Song1, Artist: Artist1, Duration: 300 seconds

...

2. Implement a program that manages a shopping list using a LinkedList in Java. Each node in the list should store an item name and its quantity. The program should support the following operations:

- Add an item and its quantity to the end of the list.
- Remove an item from the list by specifying its name.
- Update the quantity of an existing item.
- Display the entire shopping list.

Ensure the program handles edge cases like empty lists and non-existent items gracefully.

Aim

To implement a Java program that manages a shopping list using a LinkedList. Each node in the list should store an item name and its quantity. The program should support adding items, removing items, updating quantities, and displaying the entire list.

Algorithm

Create a ShoppingItem Class: This class will represent each item with attributes for name and quantity.

Create a ShoppingList Class: This class will manage the list of shopping items using a LinkedList and implement the required functionalities.

Implement Methods: Add methods to add items, remove items, update quantities, and display the list.

Main Class: Create a main class to demonstrate the functionalities.

Source Code

ShoppingItem Class

```
class ShoppingItem {
    private String name;
    private int quantity;

    public ShoppingItem(String name, int quantity) {
        this.name = name;
        this.quantity = quantity;
    }
```

```
    public String getName() {
        return name;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    @Override
    public String toString() {
        return "Item: " + name + ", Quantity: " + quantity;
    }
}
```

ShoppingList Class

```
import java.util.LinkedList;

class ShoppingList {
    private LinkedList<ShoppingItem> items;

    public ShoppingList() {
```

```
    this.items = new LinkedList<>();
}

public void addItem(String name, int quantity) {
    ShoppingItem item = new ShoppingItem(name, quantity);
    items.add(item);
    System.out.println("Added: " + item);
}

public void removeItem(String name) {
    ShoppingItem itemToRemove = null;
    for (ShoppingItem item : items) {
        if (item.getName().equalsIgnoreCase(name)) {
            itemToRemove = item;
            break;
        }
    }
    if (itemToRemove != null) {
        items.remove(itemToRemove);
        System.out.println("Removed: " + itemToRemove);
    } else {
        System.out.println("Item not found: " + name);
    }
}

public void updateQuantity(String name, int newQuantity) {
```

```
        for (ShoppingItem item : items) {
            if (item.getName().equalsIgnoreCase(name)) {
                item.setQuantity(newQuantity);
                System.out.println("Updated: " + item);
                return;
            }
        }
        System.out.println("Item not found: " + name);
    }

    public void displayList() {
        if (items.isEmpty()) {
            System.out.println("The shopping list is empty.");
        } else {
            System.out.println("Shopping List:");
            for (ShoppingItem item : items) {
                System.out.println(item);
            }
        }
    }
}
```

Main Class

```
import java.util.Scanner;

public class ShoppingListManager {
```

```
public static void main(String[] args) {
    ShoppingList shoppingList = new ShoppingList();
    Scanner scanner = new Scanner(System.in);

    while (true) {
        System.out.println("1. Add an item");
        System.out.println("2. Remove an item by name");
        System.out.println("3. Update the quantity of an item");
        System.out.println("4. Display the shopping list");
        System.out.println("5. Exit");
        System.out.print("Enter your choice: ");
        int choice = scanner.nextInt();
        scanner.nextLine(); // Consume the newline character

        switch (choice) {
            case 1:
                System.out.print("Enter item name: ");
                String addName = scanner.nextLine();
                System.out.print("Enter quantity: ");
                int addQuantity = scanner.nextInt();
                scanner.nextLine(); // Consume the newline character
                shoppingList.addItem(addName, addQuantity);
                break;
            case 2:
                System.out.print("Enter item name to remove: ");
                String removeName = scanner.nextLine();
```

```
                shoppingList.removeItem(removeName);
                break;
            case 3:
                System.out.print("Enter item name to update: ");
                String updateName = scanner.nextLine();
                System.out.print("Enter new quantity: ");
                int newQuantity = scanner.nextInt();
                scanner.nextLine(); // Consume the newline character
                shoppingList.updateQuantity(updateName, newQuantity);
                break;
            case 4:
                shoppingList.displayList();
                break;
            case 5:
                System.out.println("Exiting...");
                scanner.close();
                System.exit(0);
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    }
}
```

Compilation and Execution

javac ShoppingItem.java ShoppingList.java ShoppingListManager.java

java ShoppingListManager

Expected Output

1. Add an item

2. Remove an item by name

3. Update the quantity of an item

4. Display the shopping list

5. Exit

Enter your choice: 1

Enter item name: Apples

Enter quantity: 5

Added: Item: Apples, Quantity: 5

...

3. Create a ticket reservation system for a concert using Java's Vector. Each ticket request must store: Fan Name, Number of Tickets

   The system should allow organizers to:

   - Add a ticket request to the queue.
   - Process requests in the order they were received (first-come, first-served).
   - Display the current queue of requests.

   Aim
   To develop a ticket reservation system using Java's Vector that handles fan ticket requests by maintaining a queue and processing requests in order.

   Algorithm
   Create a TicketRequest Class: This class will represent each ticket request with attributes for the fan's name and the number of tickets.
   Create a TicketReservationSystem Class: This class will use a Vector to store and manage the ticket requests.
   Implement Methods: Add methods to add requests, process requests, and display the queue.

Main Class: Create a main class to demonstrate the functionalities.

Source Code

TicketRequest Class

```
class TicketRequest {
    private String fanName;
    private int numberOfTickets;

    public TicketRequest(String fanName, int numberOfTickets) {
        this.fanName = fanName;
        this.numberOfTickets = numberOfTickets;
    }

    public String getFanName() {
        return fanName;
    }

    public int getNumberOfTickets() {
        return numberOfTickets;
    }

    @Override
    public String toString() {
        return "Fan: " + fanName + ", Tickets: " + numberOfTickets;
    }
}
```

TicketReservationSystem Class

```
import java.util.Vector;

class TicketReservationSystem {
    private Vector<TicketRequest> requestQueue;

    public TicketReservationSystem() {
        this.requestQueue = new Vector<>();
    }

    public void addRequest(String fanName, int numberOfTickets) {
        TicketRequest request = new TicketRequest(fanName, numberOfTickets);
        requestQueue.add(request);
        System.out.println("Added: " + request);
    }

    public void processRequest() {
```

```
        if (requestQueue.isEmpty()) {
            System.out.println("No requests to process.");
        } else {
            TicketRequest request = requestQueue.remove(0);
            System.out.println("Processed: " + request);
        }
    }

    public void displayQueue() {
        if (requestQueue.isEmpty()) {
            System.out.println("The queue is empty.");
        } else {
            System.out.println("Current Queue:");
            for (TicketRequest request : requestQueue) {
                System.out.println(request);
            }
        }
    }
}
```

Main Class

```
import java.util.Scanner;

public class TicketReservationManager {
    public static void main(String[] args) {
        TicketReservationSystem reservationSystem = new TicketReservationSystem();
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("1. Add a ticket request");
            System.out.println("2. Process a ticket request");
            System.out.println("3. Display current queue");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch (choice) {
                case 1:
                    System.out.print("Enter fan name: ");
                    String fanName = scanner.nextLine();
                    System.out.print("Enter number of tickets: ");
                    int numberOfTickets = scanner.nextInt();
                    scanner.nextLine(); // Consume newline
```

```
                reservationSystem.addRequest(fanName, numberOfTickets);
                break;
            case 2:
                reservationSystem.processRequest();
                break;
            case 3:
                reservationSystem.displayQueue();
                break;
            case 4:
                System.out.println("Exiting...");
                scanner.close();
                System.exit(0);
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    }
  }
}
```

Compilation and Execution

javac TicketRequest.java TicketReservationSystem.java TicketReservationManager.java
java TicketReservationManager

Expected Output

1. Add a ticket request
2. Process a ticket request
3. Display current queue
4. Exit
Enter your choice: 1
Enter fan name: Alice
Enter number of tickets: 2
Added: Fan: Alice, Tickets: 2
...

4. Develop a Java program to find the intersection of two sets of student names. The program should:
   - Take two sets of student names as input.
   - Find and display the names that are common to both sets.

   Aim

   To create a Java program that finds and displays the intersection of two sets of student names.

   Algorithm

Create Two Sets: Use HashSet to store the names of students in each set.
Input Names: Prompt the user to input names for both sets.
Compute Intersection: Use the retainAll() method to find the intersection of the two sets.
Display Results: Print the names that are common to both sets.

Source Code

```
import java.util.HashSet;
import java.util.Scanner;
import java.util.Set;

public class StudentSetIntersection {

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

        // Input for first set
        System.out.println("Enter the number of students in the first set:");
        int size1 = scanner.nextInt();
        scanner.nextLine();  // Consume newline

        Set<String> set1 = new HashSet<>();
        System.out.println("Enter the names of students in the first set:");
        for (int i = 0; i < size1; i++) {
            set1.add(scanner.nextLine());
        }

        // Input for second set
        System.out.println("Enter the number of students in the second set:");
        int size2 = scanner.nextInt();
        scanner.nextLine();  // Consume newline

        Set<String> set2 = new HashSet<>();
        System.out.println("Enter the names of students in the second set:");
        for (int i = 0; i < size2; i++) {
            set2.add(scanner.nextLine());
        }

        // Find intersection
        Set<String> intersection = new HashSet<>(set1);
        intersection.retainAll(set2);

        // Display result
        if (intersection.isEmpty()) {
            System.out.println("No common names between the two sets.");
```

```
        } else {
            System.out.println("Common names between the two sets:");
            for (String name : intersection) {
                System.out.println(name);
            }
        }

        scanner.close();
    }
}
```

Compilation and Execution

```
javac StudentSetIntersection.java
java StudentSetIntersection
```

Sample Output

```
Enter the number of students in the first set:
3
Enter the names of students in the first set:
Alice
Bob
Charlie
Enter the number of students in the second set:
4
Enter the names of students in the second set:
Bob
David
Alice
Eve
Common names between the two sets:
Alice
Bob
```

5. Create a Java program to count the frequency of each word in a given text string using a HashMap. The program should:
   - Take a string of text as input.
   - Use a HashMap to store each unique word along with its frequency.
   - Display each word and its corresponding frequency at the end.

   Aim

   To develop a Java program that counts and displays the frequency of each word in a given text string using HashMap.

Algorithm

Input Text: Read a string of text from the user.
Split Text into Words: Use a method to split the text into individual words.
Count Frequencies: Use a HashMap to store each word and its frequency.
Display Results: Print each word along with its frequency.

Source Code

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class WordFrequencyCounter {

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

        // Input text
        System.out.println("Enter a text string:");
        String text = scanner.nextLine();

        // Split text into words
        String[] words = text.split("\\s+");

        // Create a HashMap to store word frequencies
        Map<String, Integer> wordFrequencyMap = new HashMap<>();

        // Count frequencies
        for (String word : words) {
            // Convert to lowercase to ensure case-insensitive comparison
            word = word.toLowerCase();
            if (wordFrequencyMap.containsKey(word)) {
                // Increment frequency count
                wordFrequencyMap.put(word, wordFrequencyMap.get(word) + 1);
            } else {
                // Add new word with count 1
                wordFrequencyMap.put(word, 1);
            }
        }

        // Display word frequencies
        System.out.println("Word frequencies:");
        for (Map.Entry<String, Integer> entry : wordFrequencyMap.entrySet()) {
            System.out.println(entry.getKey() + ": " + entry.getValue());
        }
```

```
        scanner.close();
    }
}
```

Sample Output

Enter a text string:
Hello world! Welcome to the world of Java programming. Java is fun.
Word frequencies:
hello: 1
world!: 1
welcome: 1
to: 1
the: 1
world: 2
of: 1
java: 2
programming.: 1
is: 1
fun.: 1

6. Develop a Java program to implement a student grading system using a HashMap. The program should allow you to:
   - Add a student's name and their grade.
   - Remove a student by their name.
   - Update the grade of an existing student.
   - Retrieve the grade of a student by their name.
   - Display all students and their grades.

Aim

To create a Java program that manages student names and their grades using a HashMap. The program will support adding, removing, updating, retrieving, and displaying student grades.

Algorithm

Create a HashMap: Use HashMap<String, Integer> to store student names and their corresponding grades.
Add Student: Allow the user to add a new student with their grade.
Remove Student: Allow the user to remove a student by their name.
Update Grade: Allow the user to update the grade of an existing student.
Retrieve Grade: Allow the user to retrieve and display the grade of a specific student.
Display All: Display all students and their grades.

Source Code

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class StudentGradingSystem {
    private static Map<String, Integer> studentGrades = new HashMap<>();

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String choice;

        do {
            System.out.println("Student Grading System");
            System.out.println("1. Add a student and grade");
            System.out.println("2. Remove a student");
            System.out.println("3. Update a student's grade");
            System.out.println("4. Retrieve a student's grade");
            System.out.println("5. Display all students and their grades");
            System.out.println("6. Exit");
            System.out.print("Enter your choice: ");
            choice = scanner.nextLine();

            switch (choice) {
                case "1":
                    addStudent(scanner);
                    break;
                case "2":
                    removeStudent(scanner);
                    break;
                case "3":
                    updateGrade(scanner);
                    break;
                case "4":
                    retrieveGrade(scanner);
                    break;
                case "5":
                    displayAllStudents();
                    break;
                case "6":
                    System.out.println("Exiting...");
                    break;
                default:
                    System.out.println("Invalid choice. Please try again.");
                    break;
```

```
        }
    } while (!choice.equals("6"));

    scanner.close();
}

private static void addStudent(Scanner scanner) {
    System.out.print("Enter student's name: ");
    String name = scanner.nextLine();
    System.out.print("Enter grade: ");
    int grade = Integer.parseInt(scanner.nextLine());
    studentGrades.put(name, grade);
    System.out.println("Student added/updated successfully.");
}

private static void removeStudent(Scanner scanner) {
    System.out.print("Enter student's name to remove: ");
    String name = scanner.nextLine();
    if (studentGrades.containsKey(name)) {
        studentGrades.remove(name);
        System.out.println("Student removed successfully.");
    } else {
        System.out.println("Student not found.");
    }
}

private static void updateGrade(Scanner scanner) {
    System.out.print("Enter student's name to update: ");
    String name = scanner.nextLine();
    if (studentGrades.containsKey(name)) {
        System.out.print("Enter new grade: ");
        int newGrade = Integer.parseInt(scanner.nextLine());
        studentGrades.put(name, newGrade);
        System.out.println("Grade updated successfully.");
    } else {
        System.out.println("Student not found.");
    }
}

private static void retrieveGrade(Scanner scanner) {
    System.out.print("Enter student's name to retrieve grade: ");
    String name = scanner.nextLine();
    if (studentGrades.containsKey(name)) {
        System.out.println(name + "'s grade: " + studentGrades.get(name));
```

```
        } else {
            System.out.println("Student not found.");
        }
    }

    private static void displayAllStudents() {
        if (studentGrades.isEmpty()) {
            System.out.println("No students in the system.");
        } else {
            System.out.println("All students and their grades:");
            for (Map.Entry<String, Integer> entry : studentGrades.entrySet()) {
                System.out.println("Name: " + entry.getKey() + ", Grade: " + entry.getValue());
            }
        }
    }
}
```

Sample Output

Student Grading System
1. Add a student and grade
2. Remove a student
3. Update a student's grade
4. Retrieve a student's grade
5. Display all students and their grades
6. Exit
Enter your choice: 1
Enter student's name: John
Enter grade: 85
Student added/updated successfully.
Enter your choice: 5
All students and their grades:
Name: John, Grade: 85

Unit – 2

Java Swings

1. Create a Java program that handles various mouse events, including when the mouse enters, exits, clicks, presses, releases, drags, and moves within the client area. Demonstrate how each event can be used to perform specific actions, such as changing the background color, displaying coordinates, or drawing shapes, to enhance user interaction in a simple graphical application.

   Aim

To create a Java program that handles various mouse events to enhance user interaction in a graphical application.

Algorithm

Create a JFrame: Set up the main window using Swing.
Create a JPanel: This panel will listen for mouse events and perform actions accordingly.
Implement MouseListener and MouseMotionListener: Override the necessary methods to handle different mouse events.
Add Actions: Define specific actions for each mouse event (e.g., change background color, display coordinates, draw shapes).
Add JPanel to JFrame: Integrate the panel into the main window and make the window visible.

Source Code

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

public class MouseEventDemo extends JFrame {
    private JPanel panel;
    private JLabel statusLabel;

    public MouseEventDemo() {
        setTitle("Mouse Event Demo");
        setSize(600, 400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        panel = new CustomPanel();
        panel.setBackground(Color.WHITE);

        statusLabel = new JLabel("Mouse Event Status");

        add(panel, BorderLayout.CENTER);
        add(statusLabel, BorderLayout.SOUTH);

        panel.addMouseListener(new CustomMouseListener());
        panel.addMouseMotionListener(new CustomMouseMotionListener());
    }

    private class CustomPanel extends JPanel {
        private int x = -1, y = -1;

        @Override
        protected void paintComponent(Graphics g) {
```

```
            super.paintComponent(g);
            if (x >= 0 && y >= 0) {
                g.setColor(Color.RED);
                g.fillOval(x - 10, y - 10, 20, 20);
            }
        }
    }

    private class CustomMouseListener implements MouseListener {
        @Override
        public void mouseClicked(MouseEvent e) {
            statusLabel.setText("Mouse Clicked at (" + e.getX() + ", " + e.getY() + ")");
            panel.repaint();
        }

        @Override
        public void mousePressed(MouseEvent e) {
            statusLabel.setText("Mouse Pressed at (" + e.getX() + ", " + e.getY() + ")");
        }

        @Override
        public void mouseReleased(MouseEvent e) {
            statusLabel.setText("Mouse Released at (" + e.getX() + ", " + e.getY() + ")");
        }

        @Override
        public void mouseEntered(MouseEvent e) {
            statusLabel.setText("Mouse Entered");
            panel.setBackground(Color.LIGHT_GRAY);
        }

        @Override
        public void mouseExited(MouseEvent e) {
            statusLabel.setText("Mouse Exited");
            panel.setBackground(Color.WHITE);
        }
    }

    private class CustomMouseMotionListener implements MouseMotionListener {
        @Override
        public void mouseDragged(MouseEvent e) {
            statusLabel.setText("Mouse Dragged at (" + e.getX() + ", " + e.getY() + ")");
            panel.repaint();
        }
```

```
        @Override
        public void mouseMoved(MouseEvent e) {
            statusLabel.setText("Mouse Moved at (" + e.getX() + ", " + e.getY() + ")");
        }
    }

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

Compilation and Execution

javac MouseEventDemo.java
java MouseEventDemo

2. Implement a java program using swings to design a multiple choice question having three options (use radio button), display the message using dialog box “Your answer is wrong” if the user selects wrong option otherwise display, “Your answer is correct."

Aim

To create a Swing-based Java application that presents a multiple-choice question with three options and provides feedback on the user's selection.

Algorithm

Create a JFrame: Set up the main window using Swing.
Add JLabel for Question: Display the question using a JLabel.
Add JRadioButtons for Options: Create three JRadioButtons for the options.
Group the JRadioButtons: Use a ButtonGroup to ensure only one option can be selected at a time.
Add JButton for Submission: Add a button to submit the answer.
Handle Button Click Event: Implement the ActionListener for the button to check the selected answer and display a message using JOptionPane.
Add Components to JFrame: Add all components to the JFrame and set its properties.

Source Code

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

```
import java.awt.event.ActionListener;

public class MultipleChoiceQuestion extends JFrame {
    public MultipleChoiceQuestion() {
        setTitle("Multiple Choice Question");
        setSize(400, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);

        // Create components
        JLabel questionLabel = new JLabel("What is the capital of France?");
        JRadioButton option1 = new JRadioButton("Berlin");
        JRadioButton option2 = new JRadioButton("Paris");
        JRadioButton option3 = new JRadioButton("Madrid");

        // Group the radio buttons
        ButtonGroup optionsGroup = new ButtonGroup();
        optionsGroup.add(option1);
        optionsGroup.add(option2);
        optionsGroup.add(option3);

        JButton submitButton = new JButton("Submit");

        // Set layout and add components
        setLayout(new GridLayout(5, 1));
        add(questionLabel);
        add(option1);
        add(option2);
        add(option3);
        add(submitButton);

        // Add action listener to the submit button
        submitButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (option2.isSelected()) {
                    JOptionPane.showMessageDialog(null, "Your answer is correct.");
                } else {
                    JOptionPane.showMessageDialog(null, "Your answer is wrong.");
                }
            }
        });
    }
```

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

Compilation and Execution

javac MultipleChoiceQuestion.java
java MultipleChoiceQuestion

3. Develop a Java program that handles various key events, including when a key is pressed, released, and typed within a client area. Demonstrate how each event can be used to perform specific actions, such as updating a display with the pressed key to enhance user interaction.

Aim

To create a Swing-based Java application that captures key events and performs actions such as updating a display with the pressed key to enhance user interaction.

Algorithm

Create a JFrame: Set up the main window using Swing.
Add a JLabel for Display: Display the key event information using a JLabel.
Implement KeyListener: Create a class that implements the KeyListener interface to handle key events.
Override KeyListener Methods: Override keyPressed, keyReleased, and keyTyped methods to handle key events.
Add Components to JFrame: Add the JLabel to the JFrame and set its properties.
Add KeyListener to Component: Add the KeyListener to the component that should capture key events.

Source Code

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

public class KeyEventDemo extends JFrame implements KeyListener {
    private JLabel keyLabel;

    public KeyEventDemo() {
        setTitle("Key Event Demo");
        setSize(400, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
```

```
        keyLabel = new JLabel("Press any key...", SwingConstants.CENTER);
        keyLabel.setFont(new Font("Serif", Font.PLAIN, 24));

        // Add components to JFrame
        setLayout(new BorderLayout());
        add(keyLabel, BorderLayout.CENTER);

        // Add KeyListener to JFrame
        addKeyListener(this);
    }

    @Override
    public void keyPressed(KeyEvent e) {
        keyLabel.setText("Key Pressed: " + KeyEvent.getKeyText(e.getKeyCode()));
    }

    @Override
    public void keyReleased(KeyEvent e) {
        keyLabel.setText("Key Released: " + KeyEvent.getKeyText(e.getKeyCode()));
    }

    @Override
    public void keyTyped(KeyEvent e) {
        keyLabel.setText("Key Typed: " + e.getKeyChar());
    }

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

Compilation and Execution

```
javac KeyEventDemo.java
java KeyEventDemo
```

4. Design a Java application that uses Swing components to create a simple calculator with buttons for digits (0-9), arithmetic operations (+, -, *, /), and an equals (=) button. Utilize event listeners to handle button clicks and display the result of arithmetic operations in a text field.

Aim

To design a Java application that utilizes Swing components to create a simple calculator with buttons for digits (0-9), arithmetic operations (+, -, *, /), and an equals (=) button. The application will handle button clicks using event listeners and display the result of arithmetic operations in a text field.

Algorithm

Create the Main JFrame:

Set up the main window with a title, size, and default close operation.
Add a JTextField for Display:

Create a JTextField to display numbers and results.
Set the text field to be non-editable and aligned to the right.
Create Buttons for Digits and Operations:

Create buttons for digits (0-9) and arithmetic operations (+, -, *, /, =).
Assign an ActionListener to each button.
Handle Button Clicks:

Implement the actionPerformed method to handle button clicks.
Append digits to the current input when digit buttons are clicked.
Store the first operand and the operator when an operator button is clicked.
Calculate the result when the equals button is clicked.
Clear the input when the clear button is clicked.
Update the Display:

Update the JTextField based on the button clicked and the current state of the input.

Source Code

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class SimpleCalculator extends JFrame implements ActionListener {
    private JTextField display;
    private StringBuilder currentInput;
    private double firstOperand;
    private String operator;

    public SimpleCalculator() {
        setTitle("Simple Calculator");
        setSize(400, 500);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
    setLocationRelativeTo(null);

    currentInput = new StringBuilder();

    display = new JTextField();
    display.setFont(new Font("Arial", Font.PLAIN, 24));
    display.setEditable(false);
    display.setHorizontalAlignment(JTextField.RIGHT);

    JPanel panel = new JPanel();
    panel.setLayout(new GridLayout(4, 4, 10, 10));

    String[] buttons = {
            "7", "8", "9", "/",
            "4", "5", "6", "*",
            "1", "2", "3", "-",
            "0", "C", "=", "+"
    };

    for (String text : buttons) {
        JButton button = new JButton(text);
        button.setFont(new Font("Arial", Font.PLAIN, 24));
        button.addActionListener(this);
        panel.add(button);
    }

    setLayout(new BorderLayout());
    add(display, BorderLayout.NORTH);
    add(panel, BorderLayout.CENTER);
}

@Override
public void actionPerformed(ActionEvent e) {
    String command = e.getActionCommand();

    if (command.matches("\\d")) { // If digit is pressed
        currentInput.append(command);
        display.setText(currentInput.toString());
    } else if (command.matches("[+\\-*/]")) { // If operator is pressed
        firstOperand = Double.parseDouble(currentInput.toString());
        operator = command;
        currentInput.setLength(0);
    } else if (command.equals("=")) { // If equals is pressed
        double secondOperand = Double.parseDouble(currentInput.toString());
```

```
            double result = 0;

            switch (operator) {
                case "+":
                    result = firstOperand + secondOperand;
                    break;
                case "-":
                    result = firstOperand - secondOperand;
                    break;
                case "*":
                    result = firstOperand * secondOperand;
                    break;
                case "/":
                    if (secondOperand != 0) {
                        result = firstOperand / secondOperand;
                    } else {
                        JOptionPane.showMessageDialog(this, "Cannot divide by zero");
                        currentInput.setLength(0);
                        display.setText("");
                        return;
                    }
                    break;
            }
            display.setText(String.valueOf(result));
            currentInput.setLength(0);
        } else if (command.equals("C")) { // If clear is pressed
            currentInput.setLength(0);
            display.setText("");
        }
    }

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

Compilation and Execution

```
javac SimpleCalculator.java
java SimpleCalculator
```

5. Develop a Java application that includes two JButton components labeled "Enable" and "Disable". Implement functionality so that clicking the "Enable" button enables a third JButton labeled "Action" and clicking the "Disable" button disables the "Action" button.

Aim

To develop a Java application that includes two JButton components labeled "Enable" and "Disable". Implement functionality so that clicking the "Enable" button enables a third JButton labeled "Action" and clicking the "Disable" button disables the "Action" button.

Algorithm

Create the Main JFrame:

Set up the main window with a title, size, and default close operation.
Add Buttons:

Create three buttons: "Enable", "Disable", and "Action".
Set the initial state of the "Action" button to disabled.
Implement Action Listeners:

Attach action listeners to the "Enable" and "Disable" buttons.
In the "Enable" button listener, enable the "Action" button.
In the "Disable" button listener, disable the "Action" button.
Add Components to the JFrame:

Arrange the buttons in the JFrame using an appropriate layout manager.

Source Code

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class ButtonControl extends JFrame implements ActionListener {
    private JButton enableButton;
    private JButton disableButton;
    private JButton actionButton;

    public ButtonControl() {
        setTitle("Button Control Example");
        setSize(300, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
        setLayout(new FlowLayout());

        enableButton = new JButton("Enable");
```

```
        disableButton = new JButton("Disable");
        actionButton = new JButton("Action");

        // Initially disable the Action button
        actionButton.setEnabled(false);

        // Add action listeners
        enableButton.addActionListener(this);
        disableButton.addActionListener(this);

        // Add buttons to the frame
        add(enableButton);
        add(disableButton);
        add(actionButton);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == enableButton) {
            actionButton.setEnabled(true);
        } else if (e.getSource() == disableButton) {
            actionButton.setEnabled(false);
        }
    }

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

Compilation and Execution

javac ButtonControl.java
java ButtonControl

6. Implement a Java application for selecting a gender using JRadioButton (Male and Female). Implement functionality so that selecting a radio button updates a JLabel to display the selected gender.

Aim

To implement a Java application for selecting a gender using JRadioButton (Male and Female). Implement functionality so that selecting a radio button updates a JLabel to display the selected gender.

Algorithm

Create the Main JFrame:

Set up the main window with a title, size, and default close operation.
Add Radio Buttons and Button Group:

Create two JRadioButton components for "Male" and "Female".
Add these radio buttons to a ButtonGroup to ensure only one can be selected at a time.
Add a JLabel:

Create a JLabel to display the selected gender.
Implement Action Listeners:

Attach action listeners to the radio buttons.
In the action listener, update the JLabel to display the selected gender.
Add Components to the JFrame:

Arrange the radio buttons and the label in the JFrame using an appropriate layout manager.

Source Code

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class GenderSelector extends JFrame implements ActionListener {
    private JRadioButton maleButton;
    private JRadioButton femaleButton;
    private JLabel genderLabel;

    public GenderSelector() {
        setTitle("Gender Selector");
        setSize(300, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
        setLayout(new FlowLayout());

        maleButton = new JRadioButton("Male");
        femaleButton = new JRadioButton("Female");
```

```
        // Add radio buttons to a button group
        ButtonGroup genderGroup = new ButtonGroup();
        genderGroup.add(maleButton);
        genderGroup.add(femaleButton);

        // Add action listeners
        maleButton.addActionListener(this);
        femaleButton.addActionListener(this);

        // Label to display selected gender
        genderLabel = new JLabel("Selected Gender: None");

        // Add components to the frame
        add(maleButton);
        add(femaleButton);
        add(genderLabel);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == maleButton) {
            genderLabel.setText("Selected Gender: Male");
        } else if (e.getSource() == femaleButton) {
            genderLabel.setText("Selected Gender: Female");
        }
    }

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

Compilation and Execution

javac GenderSelector.java

java GenderSelector

7. Create a Java program that displays a list of cities (Delhi, Hyderabad, Kolkata. Mumbai) using ‘JList’. Implement functionality so that selecting a city from the list updates a ‘JLabel’ with additional information about the city, such as population and country.

   Aim

To create a Java program that displays a list of cities using JList and implements functionality to update a JLabel with additional information about the selected city.

Algorithm

Create the Main JFrame:

Set up the main window with a title, size, and default close operation.
Create the JList:

Initialize a JList with an array of city names (Delhi, Hyderabad, Kolkata, Mumbai).
Add a JLabel:

Create a JLabel to display additional information about the selected city.
Implement List Selection Listener:

Attach a ListSelectionListener to the JList.
In the listener, update the JLabel with information about the selected city.
Add Components to the JFrame:

Arrange the JList and the JLabel in the JFrame using an appropriate layout manager.

Source Code

```
import javax.swing.*;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import java.awt.*;

public class CityInfo extends JFrame {
   private JList<String> cityList;
   private JLabel infoLabel;

   public CityInfo() {
      setTitle("City Information");
      setSize(400, 300);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setLocationRelativeTo(null);
      setLayout(new BorderLayout());

      // List of cities
      String[] cities = {"Delhi", "Hyderabad", "Kolkata", "Mumbai"};
      cityList = new JList<>(cities);
      cityList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

      // Label to display city information
```

```
        infoLabel = new JLabel("Select a city to see the information", SwingConstants.CENTER);

        // Add components to the frame
        add(new JScrollPane(cityList), BorderLayout.CENTER);
        add(infoLabel, BorderLayout.SOUTH);

        // Add list selection listener
        cityList.addListSelectionListener(new ListSelectionListener() {
            @Override
            public void valueChanged(ListSelectionEvent e) {
                if (!e.getValueIsAdjusting()) {
                    String selectedCity = cityList.getSelectedValue();
                    if (selectedCity != null) {
                        updateCityInfo(selectedCity);
                    }
                }
            }
        });
    }

    private void updateCityInfo(String city) {
        switch (city) {
            case "Delhi":
                infoLabel.setText("Delhi - Population: 19 million, Country: India");
                break;
            case "Hyderabad":
                infoLabel.setText("Hyderabad - Population: 9.7 million, Country: India");
                break;
            case "Kolkata":
                infoLabel.setText("Kolkata - Population: 14.8 million, Country: India");
                break;
            case "Mumbai":
                infoLabel.setText("Mumbai - Population: 20 million, Country: India");
                break;
            default:
                infoLabel.setText("Information not available");
        }
    }

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

```
        });
    }
}
```

Compilation and Execution
javac CityInfo.java
java CityInfo

8. Develop a Java program that displays a 'JComboBox' with a list of colors (e.g., Red, Green, Blue). Implement functionality to print the selected color to the console when a user selects an item from the dropdown.

Aim
To create a Java program that displays a JComboBox with a list of colors and prints the selected color to the console when a user selects an item from the dropdown.

Algorithm
Create the Main JFrame:

Set up the main window with a title, size, and default close operation.
Create the JComboBox:

Initialize a JComboBox with an array of color names (Red, Green, Blue).
Add an ActionListener:

Attach an ActionListener to the JComboBox.
In the listener, print the selected color to the console.
Add Components to the JFrame:

Add the JComboBox to the JFrame using an appropriate layout manager.

Source Code

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class ColorSelector extends JFrame {
    private JComboBox<String> colorComboBox;

    public ColorSelector() {
        setTitle("Color Selector");
        setSize(300, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
```

```
        setLayout(new BorderLayout());

        // List of colors
        String[] colors = {"Red", "Green", "Blue"};
        colorComboBox = new JComboBox<>(colors);

        // Add components to the frame
        add(colorComboBox, BorderLayout.CENTER);

        // Add action listener to JComboBox
        colorComboBox.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String selectedColor = (String) colorComboBox.getSelectedItem();
                System.out.println("Selected Color: " + selectedColor);
            }
        });
    }

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

Compilation and Execution

```
javac ColorSelector.java
java ColorSelector
```