8 Develop a Java program to demonstrate the concept of method overloading. The program should define a class that contains multiple methods with the same name but different parameter lists. These methods will perform similar operations (such as addition) on various types and numbers of inputs.

```
// Class to demonstrate method overloading
class Calculator {
    // Method to add two integers
    public int add(int a, int b) {
        return a + b;
    }

    // Method to add three integers
    public int add(int a, int b, int c) {
        return a + b + c;
    }

    // Method to add two double values
    public double add(double a, double b) {
        return a + b;
    }

    // Method to add an array of integers
    public int add(int[] numbers) {
        int sum = 0;
        for (int num : numbers) {
            sum += num;
        }
        return sum;
    }
}

public class MethodOverloadingDemo {
    public static void main(String[] args) {
        // Create an instance of the Calculator class
        Calculator calculator = new Calculator();

        // Call the add method with two integers
        System.out.println("Sum of 2 and 3: " + calculator.add(2, 3));

        // Call the add method with three integers
        System.out.println("Sum of 4, 5, and 6: " + calculator.add(4, 5, 6));

        // Call the add method with two double values
        System.out.println("Sum of 2.5 and 3.5: " + calculator.add(2.5, 3.5));

        // Call the add method with an array of integers
        int[] numbers = {1, 2, 3, 4, 5};
        System.out.println("Sum of array elements: " + calculator.add(numbers));
    }
}
```

User defined input

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

```java
class Calculator {
    // Method to add two integers
    public int add(int a, int b) {
        return a + b;
    }

    // Method to add three integers
    public int add(int a, int b, int c) {
        return a + b + c;
    }

    // Method to add two double values
    public double add(double a, double b) {
        return a + b;
    }

    // Method to add an array of integers
    public int add(int[] numbers) {
        int sum = 0;
        for (int num : numbers) {
            sum += num;
        }
        return sum;
    }
}

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

        // Input for adding two integers
        System.out.println("Enter two integers:");
        int int1 = scanner.nextInt();
        int int2 = scanner.nextInt();
        System.out.println("Sum of " + int1 + " and " + int2 + ": " + calculator.add(int1,
int2));

        // Input for adding three integers
        System.out.println("Enter three integers:");
        int int3 = scanner.nextInt();
        int int4 = scanner.nextInt();
        int int5 = scanner.nextInt();
        System.out.println("Sum of " + int3 + ", " + int4 + ", and " + int5 + ": " +
calculator.add(int3, int4, int5));

        // Input for adding two double values
        System.out.println("Enter two double values:");
        double double1 = scanner.nextDouble();
        double double2 = scanner.nextDouble();
        System.out.println("Sum of " + double1 + " and " + double2 + ": " +
calculator.add(double1, double2));
```

```
        // Input for adding an array of integers
        System.out.println("Enter the size of the array:");
        int size = scanner.nextInt();
        int[] numbers = new int[size];
        System.out.println("Enter " + size + " integers:");
        for (int i = 0; i < size; i++) {
            numbers[i] = scanner.nextInt();
        }
        System.out.println("Sum of array elements: " + calculator.add(numbers));

        scanner.close();
    }
}
```

9. Develop a Java program that demonstrates the concept of constructor overloading .The program should define a class with multiple constructors that have different parameter lists, allowing objects to be initialized in various ways.

```
// Class to demonstrate constructor overloading
class Person {
    private String name;
    private int age;
    private String address;

    // Default constructor
    public Person() {
        this.name = "Unknown";
        this.age = 0;
        this.address = "Not Provided";
    }

    // Constructor with name
    public Person(String name) {
        this.name = name;
        this.age = 0;
        this.address = "Not Provided";
    }

    // Constructor with name and age
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
        this.address = "Not Provided";
    }

    // Constructor with name, age, and address
    public Person(String name, int age, String address) {
        this.name = name;
        this.age = age;
        this.address = address;
    }

    // Method to display person information
    public void displayInfo() {
        System.out.println("Name: " + name);
```

```
        System.out.println("Age: " + age);
        System.out.println("Address: " + address);
        System.out.println();
    }
}

public class ConstructorOverloadingDemo {
    public static void main(String[] args) {
        // Create objects using different constructors
        Person person1 = new Person();
        Person person2 = new Person("Alice");
        Person person3 = new Person("Bob", 25);
        Person person4 = new Person("Charlie", 30, "123 Main St");

        // Display information for each person
        System.out.println("Person 1:");
        person1.displayInfo();

        System.out.println("Person 2:");
        person2.displayInfo();

        System.out.println("Person 3:");
        person3.displayInfo();

        System.out.println("Person 4:");
        person4.displayInfo();
    }
}
```

10. Write a java program to demonstrate the usage of static variables and static methods in Java, highlighting their shared behaviour across instances and their accessibility without creating an object instance.

```
// Class to demonstrate static variables and methods
class Counter {
    // Static variable to count the number of objects created
    private static int count = 0;

    // Constructor to increment the static count variable
    public Counter() {
        count++;
    }

    // Static method to access the count variable
    public static int getCount() {
        return count;
    }
}

public class StaticDemo {
    public static void main(String[] args) {
        // Access static method without creating an object
        System.out.println("Initial count: " + Counter.getCount());

        // Create objects of the Counter class
```

```
        Counter c1 = new Counter();
        Counter c2 = new Counter();
        Counter c3 = new Counter();

        // Access the static method to get the updated count
        System.out.println("Count after creating 3 objects: " + Counter.getCount());

        // Static variable is shared, so the count remains consistent across instances
        System.out.println("Accessing count through another instance:");
        System.out.println("Count from c1: " + Counter.getCount());
        System.out.println("Count from c2: " + Counter.getCount());
        System.out.println("Count from c3: " + Counter.getCount());
    }
}
```

11.Write a Java program that takes an array of integers as input and finds the minimum number in the array. The program should display the minimum number found.

```
import java.util.Scanner;

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

        // Input the size of the array
        System.out.print("Enter the size of the array: ");
        int size = scanner.nextInt();

        // Initialize the array
        int[] numbers = new int[size];

        // Input the elements of the array
        System.out.println("Enter " + size + " integers:");
        for (int i = 0; i < size; i++) {
            numbers[i] = scanner.nextInt();
        }

        // Find the minimum number
        int min = numbers[0];
        for (int i = 1; i < size; i++) {
            if (numbers[i] < min) {
                min = numbers[i];
            }
        }

        // Display the minimum number
        System.out.println("The minimum number in the array is: " + min);

        scanner.close();
    }
}
```

12. Write a Java program that takes an array of integers as input and sorts the array in both ascending and descending order. The program should first display the sorted array in ascending order, followed by the sorted array in descending order.

```
import java.util.Scanner;

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

        // Input the size of the array
        System.out.print("Enter the size of the array: ");
        int size = scanner.nextInt();

        // Initialize the array
        int[] numbers = new int[size];

        // Input the elements of the array
        System.out.println("Enter " + size + " integers:");
        for (int i = 0; i < size; i++) {
            numbers[i] = scanner.nextInt();
        }

        // Sort the array in ascending order
        sortArray(numbers, true);
        System.out.println("Array in ascending order:");
        printArray(numbers);

        // Sort the array in descending order
        sortArray(numbers, false);
        System.out.println("Array in descending order:");
        printArray(numbers);

        scanner.close();
    }

    // Method to sort the array
    public static void sortArray(int[] array, boolean ascending) {
        for (int i = 0; i < array.length - 1; i++) {
            for (int j = 0; j < array.length - 1 - i; j++) {
                if ((ascending && array[j] > array[j + 1]) || (!ascending && array[j] <
array[j + 1])) {
                    // Swap elements
                    int temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                }
            }
        }
    }

    // Method to print the array
    public static void printArray(int[] array) {
        for (int num : array) {
```

```
            System.out.print(num + " ");
        }
        System.out.println();
    }
}
```

13. Write a Java program to perform matrix multiplication. The program should take two matrices (2D arrays) as input, multiply them, and display the resulting matrix. Assume that the number of columns in the first matrix is equal to the number of rows in the second matrix.

```
import java.util.Scanner;

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

        // Input dimensions for the first matrix
        System.out.print("Enter the number of rows in the first matrix: ");
        int rows1 = scanner.nextInt();
        System.out.print("Enter the number of columns in the first matrix: ");
        int cols1 = scanner.nextInt();

        // Input dimensions for the second matrix
        System.out.print("Enter the number of rows in the second matrix: ");
        int rows2 = scanner.nextInt();
        System.out.print("Enter the number of columns in the second matrix: ");
        int cols2 = scanner.nextInt();

        // Check if multiplication is possible
        if (cols1 != rows2) {
            System.out.println("Matrix multiplication is not possible. The number of
columns in the first matrix must equal the number of rows in the second matrix.");
            return;
        }

        // Initialize the matrices
        int[][] matrix1 = new int[rows1][cols1];
        int[][] matrix2 = new int[rows2][cols2];
        int[][] result = new int[rows1][cols2];

        // Input elements of the first matrix
        System.out.println("Enter elements of the first matrix:");
        for (int i = 0; i < rows1; i++) {
            for (int j = 0; j < cols1; j++) {
                matrix1[i][j] = scanner.nextInt();
            }
        }

        // Input elements of the second matrix
        System.out.println("Enter elements of the second matrix:");
        for (int i = 0; i < rows2; i++) {
            for (int j = 0; j < cols2; j++) {
                matrix2[i][j] = scanner.nextInt();
            }
```

```
        }

        // Perform matrix multiplication
        for (int i = 0; i < rows1; i++) {
            for (int j = 0; j < cols2; j++) {
                for (int k = 0; k < cols1; k++) {
                    result[i][j] += matrix1[i][k] * matrix2[k][j];
                }
            }
        }

        // Display the resulting matrix
        System.out.println("Resulting matrix after multiplication:");
        for (int i = 0; i < rows1; i++) {
            for (int j = 0; j < cols2; j++) {
                System.out.print(result[i][j] + " ");
            }
            System.out.println();
        }

        scanner.close();
    }
}
```

14. To demonstrate the basic operations of the ArrayList class in Java, including adding, accessing, updating, removing elements, and iterating through the list, while understanding how an ArrayList dynamically manages its size and provides an easy-to-use alternative to arrays for managing collections of objects.

```
import java.util.ArrayList;

public class ArrayListOperations {
    public static void main(String[] args) {
        // Create an ArrayList of strings
        ArrayList<String> list = new ArrayList<>();

        // Add elements to the ArrayList
        list.add("Apple");
        list.add("Banana");
        list.add("Cherry");
        System.out.println("After adding elements: " + list);

        // Access elements in the ArrayList
        System.out.println("Element at index 1: " + list.get(1));

        // Update an element in the ArrayList
        list.set(1, "Blueberry");
        System.out.println("After updating index 1: " + list);

        // Remove an element from the ArrayList
        list.remove(0); // Removes the element at index 0
        System.out.println("After removing index 0: " + list);

        // Iterate through the ArrayList using a for-each loop
        System.out.println("Iterating through the ArrayList:");
```

```
        for (String item : list) {
            System.out.println(item);
        }

        // Check the size of the ArrayList
        System.out.println("Size of the ArrayList: " + list.size());

        // Clear all elements from the ArrayList
        list.clear();
        System.out.println("After clearing, ArrayList: " + list);
    }
}
```

15. Write a Java program to demonstrate the concept of wrapper classes in Java, including the process of autoboxing and boxing.

```
public class WrapperClassDemo {
    public static void main(String[] args) {
        // Autoboxing: Converting a primitive type to its wrapper class
        int primitiveInt = 10;
        Integer wrappedInt = primitiveInt; // Autoboxing
        System.out.println("Primitive int: " + primitiveInt);
        System.out.println("Autoboxed Integer: " + wrappedInt);

        // Manual boxing: Using a wrapper class constructor (deprecated since Java 9)
        Integer manualBoxedInt = Integer.valueOf(20);
        System.out.println("Manually boxed Integer: " + manualBoxedInt);

        // Unboxing: Converting a wrapper class back to a primitive type
        int unboxedInt = wrappedInt; // Unboxing
        System.out.println("Unboxed int: " + unboxedInt);

        // Wrapper classes in collections
        System.out.println("\nWrapper classes in collections:");
        java.util.ArrayList<Double> doubleList = new java.util.ArrayList<>();
        doubleList.add(3.14); // Autoboxing of double to Double
        doubleList.add(1.618);
        doubleList.add(2.718);

        // Accessing and unboxing elements
        double firstElement = doubleList.get(0); // Unboxing Double to double
        System.out.println("List of Doubles: " + doubleList);
        System.out.println("First element (unboxed): " + firstElement);

        // Parsing string to wrapper objects
        System.out.println("\nParsing strings to wrapper objects:");
        String numberStr = "42";
        Integer parsedInt = Integer.parseInt(numberStr);
        System.out.println("Parsed Integer from string: " + parsedInt);

        String decimalStr = "3.14";
        Double parsedDouble = Double.parseDouble(decimalStr);
        System.out.println("Parsed Double from string: " + parsedDouble);
    }
```

```
}
```

16. Write a Java program to convert an integer to a double using implicit type conversion.

```
import java.util.Scanner;

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

        // Prompt the user for an integer value
        System.out.print("Enter an integer value: ");
        int integerValue = scanner.nextInt();

        // Implicit type conversion (widening)
        double doubleValue = integerValue;

        // Display the results
        System.out.println("Integer value: " + integerValue);
        System.out.println("Converted double value: " + doubleValue);

        scanner.close();
    }
}
```

17. Write a Java program to convert a float to an integer using explicit casting

```
import java.util.Scanner;

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

        // Prompt the user for a float value
        System.out.print("Enter a float value: ");
        float floatValue = scanner.nextFloat();

        // Explicit type conversion (narrowing)
        int intValue = (int) floatValue;

        // Display the results
        System.out.println("Float value: " + floatValue);
        System.out.println("Converted integer value (after casting): " + intValue);

        scanner.close();
    }
}
```

18. Write a Java program to find the maximum occurring character in string

```
import java.util.HashMap;

public class MaxOccurringCharacter {
    public static void main(String[] args) {
        String input = "programming"; // Example input string
```

```
        // Use a HashMap to count occurrences of each character
        HashMap<Character, Integer> charCountMap = new HashMap<>();

        // Traverse the string and populate the HashMap
        for (char ch : input.toCharArray()) {
            charCountMap.put(ch, charCountMap.getOrDefault(ch, 0) + 1);
        }

        // Find the character with the maximum occurrence
        char maxChar = '\0';
        int maxCount = 0;

        for (char ch : charCountMap.keySet()) {
            if (charCountMap.get(ch) > maxCount) {
                maxChar = ch;
                maxCount = charCountMap.get(ch);
            }
        }

        // Display the result
        System.out.println("String: " + input);
        System.out.println("Maximum occurring character: " + maxChar);
        System.out.println("Occurrences: " + maxCount);
    }
}
```

User defined input

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

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

        // Prompt the user to enter a string
        System.out.print("Enter a string: ");
        String input = scanner.nextLine();

        // Use a HashMap to count occurrences of each character
        HashMap<Character, Integer> charCountMap = new HashMap<>();

        // Traverse the string and populate the HashMap
        for (char ch : input.toCharArray()) {
            charCountMap.put(ch, charCountMap.getOrDefault(ch, 0) + 1);
        }

        // Find the character with the maximum occurrence
        char maxChar = '\0';
        int maxCount = 0;

        for (char ch : charCountMap.keySet()) {
```

```
            if (charCountMap.get(ch) > maxCount) {
                maxChar = ch;
                maxCount = charCountMap.get(ch);
            }
        }

        // Display the result
        System.out.println("String: " + input);
        System.out.println("Maximum occurring character: " + maxChar);
        System.out.println("Occurrences: " + maxCount);

        scanner.close();
    }
}
```

19. Write a Java program to demonstrate the use of StringBuffer. Create a program that performs the following operations: i. Append a string to an existing string. ii. Insert a substring at a specified index. iii. Reverse the string. iv. Display the length and capacity of the StringBuffer.

```
public class StringBufferDemo {
    public static void main(String[] args) {
        // Create a StringBuffer with an initial string
        StringBuffer sb = new StringBuffer("Hello");

        // i. Append a string to the existing string
        sb.append(" World");
        System.out.println("After append: " + sb);

        // ii. Insert a substring at a specified index
        sb.insert(6, "Java ");
        System.out.println("After insert: " + sb);

        // iii. Reverse the string
        sb.reverse();
        System.out.println("After reverse: " + sb);

        // iv. Display the length and capacity of the StringBuffer
        System.out.println("Length of StringBuffer: " + sb.length());
        System.out.println("Capacity of StringBuffer: " + sb.capacity());
    }
}
```

User defined input

```
import java.util.Scanner;

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

        // Take user input for the initial string
        System.out.print("Enter the initial string: ");
        String userInput = scanner.nextLine();
```

```
        // Create a StringBuffer with the user-defined string
        StringBuffer sb = new StringBuffer(userInput);

        // i. Append a string
        System.out.print("Enter the string to append: ");
        String toAppend = scanner.nextLine();
        sb.append(toAppend);
        System.out.println("After append: " + sb);

        // ii. Insert a substring
        System.out.print("Enter the substring to insert: ");
        String toInsert = scanner.nextLine();
        System.out.print("Enter the index to insert at: ");
        int insertIndex = scanner.nextInt();
        scanner.nextLine(); // Consume the leftover newline
        if (insertIndex >= 0 && insertIndex <= sb.length()) {
            sb.insert(insertIndex, toInsert);
            System.out.println("After insert: " + sb);
        } else {
            System.out.println("Invalid index. No insertion performed.");
        }

        // iii. Reverse the string
        sb.reverse();
        System.out.println("After reverse: " + sb);

        // iv. Display the length and capacity of the StringBuffer
        System.out.println("Length of StringBuffer: " + sb.length());
        System.out.println("Capacity of StringBuffer: " + sb.capacity());

        scanner.close();
    }
}
```

20. Create a base class Vehicle that has a constructor to initialize its name and a method display(). Then, create a derived class Car that adds an additional property for the number of doors. Use the super() keyword in the Car class to call the constructor of the Vehicle class. Demonstrate the use of super() by creating an object of the Car class and displaying its details.

```
// Base class
class Vehicle {
    String name;

    // Constructor of the Vehicle class
    Vehicle(String name) {
        this.name = name;
    }

    // Method to display the vehicle details
    void display() {
        System.out.println("Vehicle Name: " + name);
    }
}
```

```java
// Derived class
class Car extends Vehicle {
    int numberOfDoors;

    // Constructor of the Car class
    Car(String name, int numberOfDoors) {
        super(name); // Calling the constructor of the Vehicle class
        this.numberOfDoors = numberOfDoors;
    }

    // Overriding the display method to include Car details
    @Override
    void display() {
        super.display(); // Call the display method of the base class
        System.out.println("Number of Doors: " + numberOfDoors);
    }
}

// Main class to demonstrate functionality
public class Main {
    public static void main(String[] args) {
        // Create an object of the Car class
        Car myCar = new Car("Toyota Corolla", 4);

        // Display the details of the Car
        myCar.display();
    }
}
```

User defined input

```java
import java.util.Scanner;

// Base class
class Vehicle {
    String name;

    // Constructor of the Vehicle class
    Vehicle(String name) {
        this.name = name;
    }

    // Method to display the vehicle details
    void display() {
        System.out.println("Vehicle Name: " + name);
    }
}

// Derived class
class Car extends Vehicle {
    int numberOfDoors;

    // Constructor of the Car class
    Car(String name, int numberOfDoors) {
```

```
        super(name); // Calling the constructor of the Vehicle class
        this.numberOfDoors = numberOfDoors;
    }

    // Overriding the display method to include Car details
    @Override
    void display() {
        super.display(); // Call the display method of the base class
        System.out.println("Number of Doors: " + numberOfDoors);
    }
}

// Main class to demonstrate functionality
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Get user input for the vehicle name
        System.out.print("Enter the name of the vehicle: ");
        String vehicleName = scanner.nextLine();

        // Get user input for the number of doors
        System.out.print("Enter the number of doors: ");
        int doors = scanner.nextInt();

        // Create an object of the Car class
        Car myCar = new Car(vehicleName, doors);

        // Display the details of the Car
        myCar.display();

        scanner.close();
    }
}
```

21. Create an abstract class Shape with an abstract method area() to calculate the area of different shapes. Then, create subclasses Circle and Rectangle that extend Shape and implement the area() method for their respective shapes. Write a Java program to: Calculate the area of a circle when given its radius. Calculate the area of a rectangle when given its length and width.

```
// Abstract class Shape
abstract class Shape {
    // Abstract method to calculate area
    abstract double area();
}

// Subclass Circle
class Circle extends Shape {
    private double radius;

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

```
    // Implement the area method
    @Override
    double area() {
        return Math.PI * radius * radius;
    }
}

// Subclass Rectangle
class Rectangle extends Shape {
    private double length, width;

    // Constructor to initialize length and width
    public Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    // Implement the area method
    @Override
    double area() {
        return length * width;
    }
}

// Main class
public class Main {
    public static void main(String[] args) {
        // Calculate the area of a Circle
        double radius = 5.0;
        Shape circle = new Circle(radius);
        System.out.println("Area of Circle: " + circle.area());

        // Calculate the area of a Rectangle
        double length = 7.0, width = 4.0;
        Shape rectangle = new Rectangle(length, width);
        System.out.println("Area of Rectangle: " + rectangle.area());
    }
}
```

User Defined Input

```
import java.util.Scanner;

// Abstract class Shape
abstract class Shape {
    // Abstract method to calculate area
    abstract double area();
}

// Subclass Circle
class Circle extends Shape {
    private double radius;
```

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

    // Implement the area method
    @Override
    double area() {
        return Math.PI * radius * radius;
    }
}

// Subclass Rectangle
class Rectangle extends Shape {
    private double length, width;

    // Constructor to initialize length and width
    public Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    // Implement the area method
    @Override
    double area() {
        return length * width;
    }
}

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

        // Input for Circle
        System.out.print("Enter the radius of the circle: ");
        double radius = scanner.nextDouble();
        Shape circle = new Circle(radius);
        System.out.println("Area of Circle: " + circle.area());

        // Input for Rectangle
        System.out.print("Enter the length of the rectangle: ");
        double length = scanner.nextDouble();
        System.out.print("Enter the width of the rectangle: ");
        double width = scanner.nextDouble();
        Shape rectangle = new Rectangle(length, width);
        System.out.println("Area of Rectangle: " + rectangle.area());

        scanner.close();
    }
}
```

22.Create a Java program that demonstrates the use of an interface. Define an interface Shape with two methods: area(): Calculates and returns the area of the shape.

perimeter(): Calculates and returns the perimeter of the shape.

Implement the Shape interface in two classes:

Circle: Accepts a radius and calculates the area and perimeter

Rectangle: Accepts length and width and calculates the area and perimeter.

In the main method, create instances of Circle and Rectangle, call their area() and perimeter() methods, and print the results.

```
// Define the Shape interface
interface Shape {
    // Abstract methods to calculate area and perimeter
    double area();
    double perimeter();
}

// Implement the Circle class that implements the Shape interface
class Circle implements Shape {
    private double radius;

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

    // Implement the area method for Circle
    @Override
    public double area() {
        return Math.PI * radius * radius;
    }

    // Implement the perimeter method for Circle
    @Override
    public double perimeter() {
        return 2 * Math.PI * radius;
    }
}

// Implement the Rectangle class that implements the Shape interface
class Rectangle implements Shape {
    private double length;
    private double width;

    // Constructor to initialize length and width
    public Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    // Implement the area method for Rectangle
    @Override
    public double area() {
        return length * width;
    }
```

```
    // Implement the perimeter method for Rectangle
    @Override
    public double perimeter() {
        return 2 * (length + width);
    }
}

// Main class
public class Main {
    public static void main(String[] args) {
        // Create an instance of Circle and Rectangle
        Shape circle = new Circle(5.0);  // Radius of 5 units
        Shape rectangle = new Rectangle(7.0, 4.0);  // Length 7 and Width 4

        // Display the area and perimeter of the Circle
        System.out.println("Circle: ");
        System.out.println("Area: " + circle.area());
        System.out.println("Perimeter: " + circle.perimeter());

        // Display the area and perimeter of the Rectangle
        System.out.println("\nRectangle: ");
        System.out.println("Area: " + rectangle.area());
        System.out.println("Perimeter: " + rectangle.perimeter());
    }
}
```

User Defined Input

```
import java.util.Scanner;

// Define the Shape interface
interface Shape {
    // Abstract methods to calculate area and perimeter
    double area();
    double perimeter();
}

// Implement the Circle class that implements the Shape interface
class Circle implements Shape {
    private double radius;

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

    // Implement the area method for Circle
    @Override
    public double area() {
        return Math.PI * radius * radius;
    }

    // Implement the perimeter method for Circle
```

```
    @Override
    public double perimeter() {
        return 2 * Math.PI * radius;
    }
}

// Implement the Rectangle class that implements the Shape interface
class Rectangle implements Shape {
    private double length;
    private double width;

    // Constructor to initialize length and width
    public Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    // Implement the area method for Rectangle
    @Override
    public double area() {
        return length * width;
    }

    // Implement the perimeter method for Rectangle
    @Override
    public double perimeter() {
        return 2 * (length + width);
    }
}

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

        // Input for Circle
        System.out.print("Enter the radius of the circle: ");
        double radius = scanner.nextDouble();
        Shape circle = new Circle(radius);
        System.out.println("Circle:");
        System.out.println("Area: " + circle.area());
        System.out.println("Perimeter: " + circle.perimeter());

        // Input for Rectangle
        System.out.print("\nEnter the length of the rectangle: ");
        double length = scanner.nextDouble();
        System.out.print("Enter the width of the rectangle: ");
        double width = scanner.nextDouble();
        Shape rectangle = new Rectangle(length, width);
        System.out.println("Rectangle:");
        System.out.println("Area: " + rectangle.area());
        System.out.println("Perimeter: " + rectangle.perimeter());

        scanner.close();
```

```
    }
}
```

23.Create a Java program that demonstrates multiple inheritance using interfaces. Define two interfaces, Vehicle and Engine, with the following methods:

Vehicle:

start(): Indicates the vehicle is starting.

stop(): Indicates the vehicle is stopping.

Engine:

engineType(): Returns the type of the engine. Create a class Car that implements both interfaces and provides implementations for their methods.

The start() method should print a message indicating the car is starting. The stop() method should print a message indicating the car is stopping. The engineType() method should return the type of engine used . In the main method, create an instance of Car, call its methods, and print the results.

```
// Define the Vehicle interface
interface Vehicle {
    // Method to indicate the vehicle is starting
    void start();

    // Method to indicate the vehicle is stopping
    void stop();
}

// Define the Engine interface
interface Engine {
    // Method to return the type of engine
    String engineType();
}

// Implementing both interfaces in the Car class
class Car implements Vehicle, Engine {
    private String typeOfEngine;

    // Constructor to initialize the engine type
    public Car(String typeOfEngine) {
        this.typeOfEngine = typeOfEngine;
    }

    // Implementing the start method from the Vehicle interface
    @Override
    public void start() {
        System.out.println("The car is starting.");
    }

    // Implementing the stop method from the Vehicle interface
    @Override
    public void stop() {
        System.out.println("The car is stopping.");
    }
```

```
    // Implementing the engineType method from the Engine interface
    @Override
    public String engineType() {
        return typeOfEngine;
    }
}

// Main class
public class Main {
    public static void main(String[] args) {
        // Create an instance of Car with a specified engine type
        Car myCar = new Car("V8");

        // Call methods from both interfaces
        myCar.start();
        myCar.stop();
        System.out.println("Engine Type: " + myCar.engineType());
    }
}
```

24. Write a code demonstrating the use of multiple catch blocks with one try.

```
public class MultipleCatchDemo {
    public static void main(String[] args) {
        try {
            // Example of risky code
            int[] numbers = {10, 20, 30};
            System.out.println("Accessing element: " + numbers[3]); //
ArrayIndexOutOfBoundsException

            int result = 10 / 0; // ArithmeticException
            System.out.println("Result: " + result);

        } catch (ArrayIndexOutOfBoundsException e) {
            // Handles exception for accessing invalid array index
            System.out.println("Error: Attempted to access an invalid array index.");
        } catch (ArithmeticException e) {
            // Handles exception for division by zero
            System.out.println("Error: Division by zero is not allowed.");
        } catch (Exception e) {
            // Handles any other exceptions
            System.out.println("Error: An unexpected exception occurred: " +
e.getMessage());
        }

        System.out.println("Program execution continues...");
    }
}
```

25.Develop a program in Java that demonstrates multithreading by extending the Thread class

```
// Define a custom thread class by extending the Thread class
class CustomThread extends Thread {
```

```
    private String threadName;

    // Constructor to initialize the thread name
    public CustomThread(String threadName) {
        this.threadName = threadName;
    }

    // Override the run() method to define the task for the thread
    @Override
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println(threadName + " is executing iteration " + i);
            try {
                // Simulate some delay in execution
                Thread.sleep(500); // Sleep for 500 milliseconds
            } catch (InterruptedException e) {
                System.out.println(threadName + " was interrupted.");
            }
        }
        System.out.println(threadName + " has completed execution.");
    }
}

// Main class
public class MultithreadingDemo {
    public static void main(String[] args) {
        // Create instances of the CustomThread class
        CustomThread thread1 = new CustomThread("Thread-1");
        CustomThread thread2 = new CustomThread("Thread-2");

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

        // Main thread continues its execution
        System.out.println("Main thread is running...");
    }
}
```