PROGRAM 1

Develop a JAVA program to add TWO matrices of suitable order N (The value of N should be read from command line arguments).

SOURCE CODE

```
import java.util.*;

public class MatrixAddition{

   public static void main(String args[]){



   int a[][]={{1,3,4},{2,4,3},{3,4,5}};

   int b[][]={{1,3,4},{2,4,3},{1,2,4}};


   int c[][]=new int[3][3];


   for(int i=0;i<3;i++){

   for(int j=0;j<3;j++){

   c[i][j]=a[i][j]+b[i][j];

   System.out.print(c[i][j]+" ");

   }

   System.out.println();

   }

   }}
```

OUTPUT

```
 2 6 8

4 8 6

 4 6 9
```

PROGRAM 2

Develop a stack class to hold a maximum of 10 integers with suitable methods. Develop a JAVA main method to illustrate Stack operations.

SOURCE CODE

```
import java.util.Scanner;
public class Stack {
        private static final int MAX_SIZE = 10;
        private int[] stackArray;
        private int top;

        public Stack() {
           stackArray = new int[MAX_SIZE];
           top = -1;
        }

        public void push(int value) {
           if (top < MAX_SIZE - 1) {
              stackArray[++top] = value;
              System.out.println("Pushed: " + value);
           } else {
              System.out.println("Stack Overflow! Cannot push " + value + ".");
           }
        }

        public int pop() {
           if (top >= 0) {
              int poppedValue = stackArray[top--];
              System.out.println("Popped: " + poppedValue);
              return poppedValue;
           } else {
              System.out.println("Stack Underflow! Cannot pop from an empty stack.");
              return -1; // Return a default value for simplicity
           }
        }

        public int peek() {
           if (top >= 0) {
              System.out.println("Peeked: " + stackArray[top]);
              return stackArray[top];
           } else {
            System.out.println("Stack is empty. Cannot peek.");
            return -1; // Return a default value for simplicity
        }
    }

    public void display() {
        if (top >= 0) {
            System.out.print("Stack Contents: ");
            for (int i = 0; i <= top; i++) {
                System.out.print(stackArray[i] + " ");
            }
            System.out.println();
        } else {
            System.out.println("Stack is empty.");
        }
    }

    public boolean isEmpty() {
        return top == -1;
    }

    public boolean isFull() {
        return top == MAX_SIZE - 1;
    }

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

        int choice;

        do {
            System.out.println("\nStack Menu:");
            System.out.println("1. Push");
            System.out.println("2. Pop");
            System.out.println("3. Peek");
            System.out.println("4. Display Stack Contents");
            System.out.println("5. Check if the stack is empty");
            System.out.println("6. Check if the stack is full");
            System.out.println("0. Exit");

            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();
```

```
                switch (choice) {
                    case 1:
                        System.out.print("Enter the value to push: ");
                        int valueToPush = scanner.nextInt();
                        stack.push(valueToPush);
                        break;
                    case 2:
                        stack.pop();
                        break;
                    case 3:
                        stack.peek();
                        break;
                    case 4:
                        stack.display();
                        break;
                    case 5:
                        System.out.println("Is the stack empty? " + stack.isEmpty());
                        break;
                    case 6:
                        System.out.println("Is the stack full? " + stack.isFull());
                        break;
                    case 0:
                        System.out.println("Exiting the program. Goodbye!");
                        break;
                    default:
                        System.out.println("Invalid choice. Please try again.");
                }
            } while (choice != 0);

            scanner.close();
        }
    }
```

OUTPUT

```
Stack Menu:
1. Push
2. Pop
3. Peek
4. Display Stack Contents
5. Check if the stack is empty
6. Check if the stack is full
0. Exit
Enter your choice: 4
Stack is empty.
Stack Menu:
1. Push
2. Pop
3. Peek
4. Display Stack Contents
5. Check if the stack is empty
6. Check if the stack is full
0. Exit
Enter your choice: 5
Is the stack empty? true

Stack Menu:
1. Push
2. Pop
3. Peek
4. Display Stack Contents
5. Check if the stack is empty
6. Check if the stack is full
0. Exit
Enter your choice: 6
Is the stack full? false

Stack Menu:
1. Push
2. Pop
3. Peek
4. Display Stack Contents
5. Check if the stack is empty
6. Check if the stack is full
0. Exit
Enter your choice: 1
Enter the value to push: 10
Pushed: 10

Stack Menu:
1. Push
2. Pop
3. Peek
4. Display Stack Contents
5. Check if the stack is empty
6. Check if the stack is full
0. Exit
Enter your choice: 1
Enter the value to push: 20
Pushed: 20

Stack Menu:
1. Push
2. Pop
3. Peek
4. Display Stack Contents
5. Check if the stack is empty
6. Check if the stack is full
0. Exit
Enter your choice: 4
Stack Contents: 10 20

Stack Menu:
1. Push
2. Pop
3. Peek
4. Display Stack Contents
5. Check if the stack is empty
6. Check if the stack is full
0. Exit
Enter your choice: 3
Peeked: 20
```

PROGRAM 3

A class called Employee, which models an employee with an ID, name and salary, is designed as shown in the following class diagram. The method raise Salary (percent) increases the salary by the given percentage. Develop the Employee class and suitable main method for demonstration.

SOURCE CODE

```
public class Employee {
   private int id;
   private String name;
   private double salary;

   public Employee(int id, String name, double salary) {
      this.id = id;
      this.name = name;
      this.salary = salary;
   }

   public void raiseSalary(double percent) {
      if (percent > 0) {
         double raiseAmount = salary * (percent / 100);
         salary += raiseAmount;
         System.out.println(name + "'s salary raised by " + percent + "%. New salary: $" +
salary);
      } else {
         System.out.println("Invalid percentage. Salary remains unchanged.");
      }
   }

   public String toString() {
      return "Employee ID: " + id + ", Name: " + name + ", Salary: $" + salary;
   }

   public static void main(String[] args) {
      // Creating an Employee object
      Employee employee = new Employee(1, "John Doe", 50000.0);

      // Displaying employee details
      System.out.println("Initial Employee Details:");
      System.out.println(employee);

      // Raising salary by 10%
```

```
        employee.raiseSalary(10);

        // Displaying updated employee details
        System.out.println("\nEmployee Details after Salary Raise:");
        System.out.println(employee);
    }
}
```

OUTPUT

```
Initial Employee Details:
Employee ID: 1, Name: John Doe, Salary: $50000.0
John Doe's salary raised by 10.0%. New salary: $55000.0

Employee Details after Salary Raise:
Employee ID: 1, Name: John Doe, Salary: $55000.0
```

PROGRAM 4

A class called MyPoint, which models a 2D point with x and y coordinates, is designed as follows:

- Two instance variables x (int) and y (int).
- A default (or "no-arg") constructor that construct a point at the default location of (0, 0).
- A overloaded constructor that constructs a point with the given x and y coordinates.
- A method setXY() to set both x and y.
- A method getXY() which returns the x and y in a 2-element int array.
- A toString() method that returns a string description of the instance in the format "(x, y)".
- A method called distance(int x, int y) that returns the distance from this point to another point at the given (x, y) coordinates
- An overloaded distance(MyPoint another) that returns the distance from this point to the given MyPoint instance (called another)
- Another overloaded distance() method that returns the distance from this point to the origin (0,0) Develop the code for the class MyPoint. Also develop a JAVA program (called TestMyPoint) to test all the methods defined in the class.

SOURCE CODE

```
public class MyPoint {

   private int x;

   private int y;

 public MyPoint() {

      this.x = 0;

      this.y = 0;

   }

      public MyPoint(int x, int y) {

      this.x = x;

      this.y = y;

   }
 public void setXY(int x, int y) {
     this.x = x;
     this.y = y;
 }
public int[] getXY() {
     int[] coordinates = {x, y};
     return coordinates;
 }
public String toString() {
     return "(" + x + ", " + y + ")";
 }

 public double distance(int x, int y) {
     int xDiff = this.x - x;
     int yDiff = this.y - y;
     return Math.sqrt(xDiff * xDiff + yDiff * yDiff);
 }
 public double distance(MyPoint another) {
     int xDiff = this.x - another.x;
     int yDiff = this.y - another.y;
     return Math.sqrt(xDiff * xDiff + yDiff * yDiff);
 }

 public double distance() {
     return Math.sqrt(x * x + y * y);
 }
```

```
}

//And here's the code for the `TestMyPoint` program to test all the methods defined in the
`MyPoint` class: java


public class TestMyPoint {

        public static void main(String[] args) {

          MyPoint point1 = new MyPoint();

          MyPoint point2 = new MyPoint(3, 4);


          System.out.println("point1: " + point1.toString());

          System.out.println("point2: " + point2.toString());


          point1.setXY(5, 6);

          System.out.println("point1 after setXY: " + point1.toString());


          int[] coordinates = point2.getXY();

          System.out.println("point2 coordinates: (" + coordinates[0] + ", " +
      coordinates[1] + ")");


          double distance1 = point1.distance(2, 3);

          System.out.println("Distance from point1 to (2, 3): " + distance1);


          double distance2 = point1.distance(point2);

          System.out.println("Distance from point1 to point2: " + distance2);


          double distance3 = point1.distance();

          System.out.println("Distance from point1 to origin: " + distance3);
        }
}
```

OUTPUT

```
point1: (0, 0)

point2: (3, 4)

point1 after setXY: (5, 6)

point2 coordinates: (3, 4)

Distance from point1 to (2, 3): 4.242640687119285

Distance from point1 to point2: 2.8284271247461903

Distance from point1 to origin: 7.810249675906654
```

PROGRAM 5

Develop a JAVA program to create a class named shape. Create three sub classes namely: circle, triangle and square, each class has two member functions named draw () and erase (). Demonstrate polymorphism concepts by developing suitable methods, defining member data and main program.

SOURCE CODE

```
class Shape {

    protected String name;



    public Shape(String name) {

        this.name = name;

    }



    public void draw() {

        System.out.println("Drawing a " + name);

    }



    public void erase() {

        System.out.println("Erasing a " + name);

    }

}



class Circle extends Shape {

    private double radius;



    public Circle(String name, double radius) {

        super(name);

        this.radius = radius;
    }

    public void draw() {
        System.out.println("Drawing a circle with radius " + radius);
    }

    public void erase() {
        System.out.println("Erasing a circle with radius " + radius);
    }
}

class Triangle extends Shape {
    private double base;
    private double height;

    public Triangle(String name, double base, double height) {
        super(name);
        this.base = base;
        this.height = height;
    }
    public void draw() {
        System.out.println("Drawing a triangle with base " + base + " and height " +  height);
    }

    public void erase() {
        System.out.println("Erasing a triangle with base " + base + " and height " + height);
    }
}

class Square extends Shape {
    private double side;

    public Square(String name, double side) {
        super(name);
        this.side = side;
    }


    public void draw() {
        System.out.println("Drawing a square with side length " + side);
    }


    public void erase() {
        System.out.println("Erasing a square with side length " + side);
    }
}

public class ShapeDemo {
    public static void main(String[] args) {
        Shape[] shapes = new Shape[3];
        shapes[0] = new Circle("Circle", 5.0);

        shapes[1] = new Triangle("Triangle", 4.0, 6.0);

        shapes[2] = new Square("Square", 3.0);


        for (Shape shape : shapes) {

            shape.draw();

            shape.erase();

            System.out.println();

        }

    }

    }
```

OUTPUT

```
Drawing a circle with radius 5.0

Erasing a circle with radius 5.0


Drawing a triangle with base 4.0 and height 6.0

Erasing a triangle with base 4.0 and height 6.0


Drawing a square with side length 3.0

Erasing a square with side length 3.0
```

PROGRAM 6

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.

SOURCE CODE

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

class Circle extends Shape {
    private double radius;

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


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


    double calculatePerimeter() {
        return 2 * Math.PI * radius;
    }
}

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

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


    double calculateArea() {
        // Using Heron's formula to calculate the area of a triangle
        double s = (side1 + side2 + side3) / 2;
        return Math.sqrt(s * (s - side1) * (s - side2) * (s - side3));
    }
    double calculatePerimeter() {
        return side1 + side2 + side3;
    }
}

public class ShapeDemo {
    public static void main(String[] args) {
        // Creating Circle and Triangle objects
        Circle circle = new Circle(5.0);
        Triangle triangle = new Triangle(3.0, 4.0, 5.0);

        // Calculating and displaying area and perimeter
        System.out.println("Circle Area: " + circle.calculateArea());
        System.out.println("Circle Perimeter: " + circle.calculatePerimeter());

        System.out.println("\nTriangle Area: " + triangle.calculateArea());
        System.out.println("Triangle Perimeter: " + triangle.calculatePerimeter());
    }
}
```

<u>OUTPUT</u>

```
Circle Area: 78.53981633974483
Circle Perimeter: 31.41592653589793

Triangle Area: 6.0
Triangle Perimeter: 12.0
```

PROGRAM 7

Develop a JAVA program to create an interface Resizable with methods resizeWidth(int width) and resizeHeight(int height) that allow an object to be resized. Create a class Rectangle that implements the Resizable interface and implements the resize methods

SOURCE CODE

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

//Rectangle class implementing Resizable interface
class Rectangle implements Resizable {
 private int width;
 private int height;

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

 // Implementation of Resizable interface

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

 // Additional methods for Rectangle class
 public int getWidth() {
     return width;
 }

 public int getHeight() {
     return height;
 }

 public void displayInfo() {
     System.out.println("Rectangle: Width = " + width + ", Height = " + height);
 }
}

//Main class to test the implementation
public class ResizeDemo {
 public static void main(String[] args) {
     // Creating a Rectangle object
     Rectangle rectangle = new Rectangle(10, 5);
```

```
    // Displaying the original information
    System.out.println("Original Rectangle Info:");
    rectangle.displayInfo();

    // Resizing the rectangle
    rectangle.resizeWidth(15);
    rectangle.resizeHeight(8);

    // Displaying the updated information
    System.out.println("\nUpdated Rectangle Info:");
    rectangle.displayInfo();
 }
}
```

OUTPUT

```
Original Rectangle Info:
Rectangle: Width = 10, Height = 5
Resized width to: 15
Resized height to: 8

Updated Rectangle Info:
Rectangle: Width = 15, Height = 8
```

PROGRAM 8

Develop a JAVA program to create an outer class with a function display. Create another class inside the outer class named inner with a function called display and call the two functions in the main class.

SOURCE CODE

```
class Outer {
   void display() {
      System.out.println("Outer class display method");
   }

   class Inner {
      void display() {
         System.out.println("Inner class display method");
      }
   }
}

public class OuterInnerDemo {
   public static void main(String[] args) {
      // Create an instance of the Outer class
      Outer outer = new Outer();

      // Call the display method of the Outer class
      outer.display();

      // Create an instance of the Inner class (nested inside Outer)
      Outer.Inner inner = outer.new Inner();

      // Call the display method of the Inner class
      inner.display();
   }
}
```

OUTPUT

```
Outer class display method

Inner class display method
```

PROGRAM 9

Develop a JAVA program to raise a custom exception (user defined exception) for DivisionByZero using try, catch, throw and finally.

SOURCE CODE

```
//Custom exception class

class DivisionByZeroException extends Exception {

 public DivisionByZeroException(String message) {

     super(message);

 }

}


public class CustomExceptionDemo {

 // Method to perform division and throw custom exception if denominator is zero

 static double divide(int numerator, int denominator) throws DivisionByZeroException {

     if (denominator == 0) {

         throw new DivisionByZeroException("Cannot divide by zero!");

     }

     return (double) numerator / denominator;

 }


 public static void main(String[] args) {

     int numerator = 10;

     int denominator = 0;


     try {

         double result = divide(numerator, denominator);
```

```
            System.out.println("Result of division: " + result);
        } catch (DivisionByZeroException e) {
            System.out.println("Exception caught: " + e.getMessage());
        } finally {
            System.out.println("Finally block executed");
        }
    }
}
```

OUTPUT

```
Exception caught: Cannot divide by zero!
Finally block executed
```

PROGRAM 10

Develop a JAVA program to create a package named mypack and import & implement it in a suitable class

SOURCE CODE

```
public class MyPackageClass {

   public void displayMessage() {

       System.out.println("Hello from MyPackageClass in mypack package!");

   }



   // New utility method

   public static int addNumbers(int a, int b) {

       return a + b;

   }

  }

//Main program outside the mypack folder

import mypack.MyPackageClass;

//import mypack.*;

public class PackageDemo {

public static void main(String[] args) {

  // Creating an instance of MyPackageClass from the mypack package

  MyPackageClass myPackageObject = new MyPackageClass();


  // Calling the displayMessage method from MyPackageClass

  myPackageObject.displayMessage();


  // Using the utility method addNumbers from MyPackageClass
  int result = MyPackageClass.addNumbers(5, 3);

  System.out.println("Result of adding numbers: " + result);

}

}
```

OUTPUT

```
Hello from MyPackageClass in mypack package!

Result of adding numbers: 8
```

PROGRAM 11

Write a program to illustrate creation of threads using runnable class. (start method start each of the newly created thread. Inside the run method there is sleep() for suspend the thread for 500 milliseconds).

SOURCE CODE

```
class MyRunnable implements Runnable {
   private volatile boolean running = true;


      public void run() {
       while (running) {
          try {
             // Suppress deprecation warning for Thread.sleep()
             Thread.sleep(500);
             System.out.println("Thread ID: " + Thread.currentThread().getId() + " is
running.");
          } catch (InterruptedException e) {
             System.out.println("Thread interrupted.");
          }
       }
   }

   public void stopThread() {
      running = false;
   }
}


public class RunnableThreadExample {
   public static void main(String[] args) {
      // Create five instances of MyRunnable
      MyRunnable myRunnable1 = new MyRunnable();
      MyRunnable myRunnable2 = new MyRunnable();
      MyRunnable myRunnable3 = new MyRunnable();
      MyRunnable myRunnable4 = new MyRunnable();
      MyRunnable myRunnable5 = new MyRunnable();

      // Create five threads and associate them with MyRunnable instances
      Thread thread1 = new Thread(myRunnable1);
      Thread thread2 = new Thread(myRunnable2);
      Thread thread3 = new Thread(myRunnable3);
```

```
        Thread thread4 = new Thread(myRunnable4);
        Thread thread5 = new Thread(myRunnable5);

        // Start the threads
        thread1.start();
        thread2.start();
        thread3.start();
        thread4.start();
        thread5.start();

        // Sleep for a while to allow the threads to run
        try {
            Thread.sleep(500);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        // Stop the threads gracefully
        myRunnable1.stopThread();
        myRunnable2.stopThread();
        myRunnable3.stopThread();
        myRunnable4.stopThread();
        myRunnable5.stopThread();
    }
}
```

OUTPUT

```
Thread ID: 17 is running.

Thread ID: 15 is running.

Thread ID: 14 is running.

Thread ID: 16 is running.

Thread ID: 18 is running.
```

PROGRAM 12

Develop a program to create a class MyThread in this class a constructor, call the base class constructor, using super and start the thread. The run method of the class starts after this. It can be observed that both main thread and created child thread are executed concurrently.

SOURCE CODE

```
class MyThread extends Thread {

    // Constructor calling base class constructor using super

    public MyThread(String name) {

        super(name);

        start(); // Start the thread in the constructor

    }

  // The run method that will be executed when the thread starts

      public void run() {

        for (int i = 1; i <= 5; i++) {

            System.out.println(Thread.currentThread().getName() + " Count: " + i);

            try {

                Thread.sleep(500); // Sleep for 500 milliseconds

            } catch (InterruptedException e) {

                System.out.println(Thread.currentThread().getName() + " Thread interrupted.");

            }

        }

    }

}
```

```
public class ThreadConcurrentExample {
    public static void main(String[] args) {
        // Create an instance of MyThread
        MyThread myThread = new MyThread("Child Thread");

        // Main thread
        for (int i = 1; i <= 5; i++) {
            System.out.println(Thread.currentThread().getName() + " Thread Count: " + i);
            try {
                Thread.sleep(500); // Sleep for 500 milliseconds
            } catch (InterruptedException e) {
                System.out.println(Thread.currentThread().getName() + " Thread interrupted.");
            }
        }
    }
}
```

OUTPUT

```
main Thread Count: 1
Child Thread Count: 1
main Thread Count: 2
Child Thread Count: 2
```

```
main Thread Count: 3

Child Thread Count: 3

main Thread Count: 4

Child Thread Count: 4

main Thread Count: 5

Child Thread Count: 5
```