Program 1

Aim:

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

Code:

```
package ProgramMatrix;

import java.util.Scanner;

public class MatrixAddition {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter the order of the matrices:");
        int n = input.nextInt();

        int[][] matrix1 = new int[n][n];
        int[][] matrix2 = new int[n][n];
        int[][] sum = new int[n][n];

        System.out.println("Enter elements of first matrix:");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                matrix1[i][j] = input.nextInt();
            }
        }

        System.out.println("Enter elements of second matrix:");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                matrix2[i][j] = input.nextInt();
            }
        }

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                sum[i][j] = matrix1[i][j] + matrix2[i][j];
            }
        }

        System.out.println("Sum of the matrices is:");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                System.out.print(sum[i][j] + " ");
            }
            System.out.println();
        }

        input.close();
    }
}
```

Output:

```
Enter the order of the matrices:
2
Enter elements of first matrix:
1
2
2
1
Enter elements of second matrix:
1
2
1
2
Sum of the matrices is:
2 4
3 3
```

Description:

This program will first prompt you to enter the order of the matrices, and then it will prompt you to enter the elements of the matrices.
This is a Java program that uses Object-Oriented Programming (OOP) concepts to add two matrices. The program reads the order of the matrices (N) from the command line arguments.This program defines a Matrix class with a constructor that takes an integer N as an argument and initializes a 2D array of size N x N.

The fillMatrix method fills the matrix with random integers. The addMatrices method takes two Matrix objects as arguments and returns a new Matrix object that is the result of adding the two input matrices. The main method creates two Matrix objects, fills them with random integers, adds them together, and prints the result. The order of the matrices (N) is read from the command line arguments.

Program 2

Aim:

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

Code:

```
public class Stack {
    private int maxSize;
    private int top;
    private int[] stackArray;

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

    public void push(int value) {
        if (top < maxSize - 1) {
            stackArray[++top] = value;
            System.out.println("Pushed " + value + " to the stack");
        } else {
            System.out.println("Stack is full. Can't push " + value);
        }
    }

    public void pop() {
        if (top >= 0) {
            System.out.println("Popped " + stackArray[top--] + " from the stack");
        } else {
            System.out.println("Stack is empty. Can't pop");
        }
    }

    public static void main(String[] args) {
        Stack stack = new Stack(10);
        stack.push(5);
        stack.push(10);
        stack.push(15);
        stack.pop();
        stack.pop();
        stack.pop();
    }
}
```

Output:

```
Pushed 0 to the stack
Pushed 1 to the stack
Pushed 2 to the stack
Pushed 3 to the stack
Pushed 4 to the stack
Pushed 5 to the stack
Pushed 6 to the stack
Pushed 7 to the stack
Pushed 8 to the stack
Pushed 9 to the stack
Stack is full. Cannot push 10
Top element is 9
Popped 9 from the stack
Popped 8 from the stack
Popped 7 from the stack
Popped 6 from the stack
Popped 5 from the stack
Popped 4 from the stack
Popped 3 from the stack
Popped 2 from the stack
Popped 1 from the stack
Popped 0 from the stack
Stack is empty. Cannot pop
```

Description:

This is a simple Java program that uses Object-Oriented Programming (OOP) concepts to create a stack class for holding a maximum of 10 integers. The program also includes a main method to illustrate stack operations.

This program will try to push 11 integers to the stack, which exceeds its maximum size, and then try to pop 11 integers from the stack, which is more than the number of integers in the stack.

This is to illustrate how the stack handles overflow and underflow situations. The peek method is used to check the top element of the stack.

This output shows the results of pushing integers from 0 to 10 onto the stack, peeking at the top element, and then popping all elements from the stack.

When trying to push the 11th element (integer 10), the program outputs “Stack is full. Cannot push 10” because the stack has a maximum size of 10. Similarly, when trying to pop an element from an empty stack, the program outputs “Stack is empty. Cannot pop”. This demonstrates how the stack handles overflow and underflow situations. The peek operation shows the top element of the stack without removing it. In this case, the top element is 9, which is the last element that was successfully pushed onto the stack.

Program 3

Aim:

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.

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 int getId() {
return id;
}

public String getName() {
return name;
}

public double getSalary() {
return salary;
}

public void raiseSalary(double percent) {

salary += salary * percent / 100.0;
}

public String toString() {
return "Employee [id=" + id + ", name=" + name + ", salary=" + salary + "]";
}

public static void main(String[] args) {
    Employee emp = new Employee(1, "Manisha Mehta", 50000.0);

    System.out.println("Before raise:");
    System.out.println(emp);

    emp.raiseSalary(10);

    System.out.println("After 10% raise:");
    System.out.println(emp);
        }
  }
```

Output:

```
Before raise:
Employee [id=1, name=Manisha Mehta, salary=50000.0]
After 10% raise:
Employee [id=1, name=Manisha Mehta, salary=55000.0]
```

Description:

This is a simple Java program that uses Object-Oriented Programming (OOP) concepts to create an Employee class. The program also includes a main method to demonstrate the functionality of the Employee class.

This program creates an Employee object with an id of 1, a name of “Manisha Mehta”, and a salary of 50000.0. It then prints the employee’s details, raises the employee’s salary by 10%, and prints the employee’s details again to show the increased salary. The toString method provides a string representation of the Employee object, which includes the id, name, and salary. After the salary raise, you can see that the salary has increased from 50000.0 to 55000.0.

This output shows the details of the Employee object before and after a 10% salary raise. The toString method provides a string representation of the Employee object, which includes the id, name, and salary. After the salary raise, you can see that the salary has increased from 50000.0 to 55000.0.

Program 4

Aim:

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.

Code:

```
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() {
return new int[]{x, y};
 }
public String toString() {
return "(" + x + ", " + y + ")";
}
public double distance(int x, int y) {
return Math.sqrt(Math.pow(this.x - x, 2) + Math.pow(this.y - y, 2));
}
public double distance(MyPoint another) {
return distance(another.x, another.y);
 }
public double distance() {
return distance(0, 0);
}
}
public class TestMyPoint {
public static void main(String[] args) {
MyPoint point = new MyPoint();
System.out.println(point);
point.setXY(1, 1);
System.out.println(point);
System.out.println("Distance from origin: " + point.distance());
MyPoint anotherPoint = new MyPoint(2, 2);
System.out.println(anotherPoint);
System.out.println("Distance from another point: " + point.distance(anotherPoint));
}
 }
```

Output:

```
(0, 0)
(1, 1)
Distance from origin: 1.4142135623730951
(2, 2)
Distance from another point: 1.4142135623730951
```

Description:

In the TestMyPoint class, a MyPoint object is created and its methods are tested. The output of the program will be the results of these method calls.

Program 5

Aim:

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.

Code:

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



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



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

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

```
 }
public class Main {
   public static void main(String[] args) {
      Shape s;
     s = new Circle();
// This line creates a new Circle object and assigns it to the ‘s’ variable of type Shape.
This demonstrates polymorphism, where a more general type (Shape) can refer to a more
specific type (Circle).

     s.draw();
// Invokes the draw() method on the s object.
      s.erase();

     s = new Triangle();
     s.draw();
     s.erase();

    s = new Square();
   s.draw();
   s.erase();
}
 }
```

Output:

```
Drawing Circle
Erasing Circle
Drawing Triangle
Erasing Triangle
Drawing Square
Erasing Square
```

Description:

This a simple Java program that uses Object-Oriented Programming (OOP) concepts to create a Shape class and three subclasses: Circle, Triangle, and Square. The program demonstrates the concept of polymorphism.

This program defines an abstract Shape class with two abstract methods: draw and erase. The Circle, Triangle, and Square classes extend the Shape class and provide implementations for the draw and erase methods. The main method in the Main class demonstrates polymorphism by declaring a Shape reference s and using it to refer to Circle, Triangle, and Square objects.

Program 6

Aim:

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.

Code:

```
abstract class Shape {
abstract double calculateArea();
abstract double calculatePerimeter();
}
class Circle extends Shape {
double radius;
Circle(double radius) {
this.radius = radius;
 }
double calculateArea() {
return Math.PI * Math.pow(radius, 2);
}
double calculatePerimeter() {
 return 2 * Math.PI * radius;
}
}
class Triangle extends Shape {
double base;
double height;
double side1;
 double side2;
Triangle(double base, double height, double side1, double side2) {
this.base = base;
this.height = height;
this.side1 = side1;
this.side2 = side2;
 }
double calculateArea() {
return 0.5 * base * height;
}
double calculatePerimeter() {
return base + side1 + side2;
}
  }
public class Main {
public static void main(String[] args) {
Circle circle = new Circle(5);
System.out.println("Circle Area: " + circle.calculateArea());
 System.out.println("Circle Perimeter: " + circle.calculatePerimeter());
Triangle triangle = new Triangle(5, 10, 5, 5);
System.out.println("Triangle Area: " + triangle.calculateArea());
System.out.println("Triangle Perimeter: " + triangle.calculatePerimeter());
}
  }
```

Output:

```
Circle Area: 78.53981633974483
Circle Perimeter: 31.41592653589793
Triangle Area: 25.0
Triangle Perimeter: 15.0
```

Description:

This is a Java program that uses Object-Oriented Programming (OOP) concepts to create an abstractShape class with abstract methods calculateArea() and calculatePerimeter(). The program also creates Circle and Triangle subclasses that extend the Shape class and implement the respective methods to calculate the area and perimeter of each shape.

This program defines an abstract Shape class with two abstract methods: calculateArea and calculatePerimeter. The Circle and Triangle classes extend the Shape class and provide implementations for the calculateArea and calculatePerimeter methods. The main method in the Main class demonstrates the functionality of the Circle and Triangle classes.

Program 7

Aim:

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.

Code:

```
interface Resizable {
void resizeWidth(int width);
void resizeHeight(int height);
}
class Rectangle implements Resizable {
int width;
int height;
Rectangle(int width, int height) {
this.width = width;
this.height = height;
}
public void resizeWidth(int width) {
this.width = width;
System.out.println("Rectangle width resized to: " + width);
}
public void resizeHeight(int height) {
this.height = height;
System.out.println("Rectangle height resized to: " + height);
 }
public static void main(String[] args) {
Rectangle rectangle = new Rectangle(10, 20);
rectangle.resizeWidth(15);
rectangle.resizeHeight(25);
}
 }
```

Output:

```
Rectangle width resized to: 15
Rectangle height resized to: 25
```

Description:

This is a Java program that uses Object-Oriented Programming (OOP) concepts to create a Resizable interface with resizeWidth(int width) and resizeHeight(int height) methods. The program also creates a Rectangle class that implements the Resizable interface and implements the resize methods.

This program defines a Resizable interface with two methods: resizeWidth and resizeHeight. The Rectangle class implements the Resizable interface and provides implementations for the resizeWidth and resizeHeight methods. The main method in the Rectangle class demonstrates the functionality of the Rectangle class.

Program 8

Aim:

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.

Code:

```
public class OuterClass {
void display() {
System.out.println("This is the display method of the outer class.");
}
class InnerClass {
void display() {
System.out.println("This is the display method of the inner class.");
}
}
public static void main(String[] args) {
OuterClass outer = new OuterClass();
OuterClass.InnerClass inner = outer.new InnerClass();
outer.display();
inner.display();
}
}
```

Output:

```
This is the display method of the outer class.
This is the display method of the inner class.
```

Description:

This is a Java program that uses Object-Oriented Programming (OOP) concepts to create an outer class with a display method. The program also creates an inner class within the outer class with its own display method. The main method in the outer class demonstrates the functionality of both display methods.

This program defines an OuterClass with a display method that prints a message. The InnerClass is defined within the OuterClass and has its own display method that prints a different message. The main method in the OuterClass creates an instance of the OuterClass and an instance of the InnerClass, and then calls the display method on each instance.

Program 9

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

Code:

```
class DivisionByZeroException extends Exception {
DivisionByZeroException(String message) {
super(message);
}
}
public class Main {
static void divide(int a, int b) throws DivisionByZeroException {
if (b == 0) {
throw new DivisionByZeroException("Division by zero is not allowed.");
} else {
System.out.println("The result is " + (a / b));
}
}public static void main(String[] args) {
try {
divide(10, 0);
} catch (DivisionByZeroException e) {
System.out.println("Caught exception: " + e.getMessage());
} finally {
System.out.println("End of program.");
}
}
 }
```

Output:
```
Caught exception: Division by zero is not allowed.
End of program.
```

Description:
This program defines a DivisionByZeroException class that extends the Exception class. The divide method in the Main class throws a DivisionByZeroException if the divisor is zero. The main method in the Main class demonstrates the use of try, catch, throw, and finally to handle the custom exception.

Program 10

Aim:

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

Code:

```
// MyClass.java
package mypack;
public class MyClass {
public void display() {
System.out.println("This is MyClass of mypack.");
}
}
```

Then your workspace (not inside mypack), create a file named Main.java with the following code:

```
// Main.java
import mypack.MyClass;
public class Main {
public static void main(String[] args) {
MyClass obj = new MyClass();
obj.display();
}
}
```

Output:

```
This is MyClass of mypack.
```

Description:

This is a Java code that creates a package named mypack with a class MyClass, and another class Main that imports and uses MyClass.

The Java program consists of two parts:

1. MyClass.java: This is a part of the package mypack. It contains a single class MyClass with a method display() that prints a message to the console.
2. Main.java: This is the main part of the program. It imports MyClass from the mypack package. In the main method, it creates an instance of MyClass and calls the display() method.

   When you run the Main class, it creates an object of MyClass and calls the display method, which prints “This is MyClass of mypack.” to the console.

This program demonstrates the use of packages in Java. Packages are used to group related classes and interfaces together. By organizing classes into packages, you can avoid conflicts in class names. A package provides a namespace for the classes it contains. Classes in the same package can access each other's package-access members. In this program, MyClass is defined in the mypack package, and it's used in the Main class in the default package. The Main class can use MyClass because it imports mypack.MyClass.

The program also demonstrates the use of the import statement in Java. The import statement is used to bring certain classes, or entire packages, into visibility. Once imported, a class can be referred to directly, without the need to use its fully qualified name.

Finally, this program shows how to define methods in a class and how to create objects of a class and call their methods. The display method is defined in MyClass, and it's called on an object of MyClass in the Main class.

Program 11

Aim:

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).

Code:

```
public class MyRunnable implements Runnable {
private String name;
public MyRunnable(String name) {
this.name = name;
}

public void run() {
for (int i = 0; i < 5; i++) {
System.out.println(name + " is running: " + i);
try {
Thread.sleep(500);
} catch (InterruptedException e) {
System.out.println(name + " was interrupted.");
}
}
}
public static void main(String[] args) {
Thread thread1 = new Thread(new MyRunnable("Thread 1"));
Thread thread2 = new Thread(new MyRunnable("Thread 2"));
thread1.start();
thread2.start();
}
}
```

Output:

```
Thread 1 is running: 0
Thread 2 is running: 0
Thread 1 is running: 1
Thread 2 is running: 1
Thread 1 is running: 2
Thread 2 is running: 2
Thread 1 is running: 3
Thread 2 is running: 3
Thread 1 is running: 4
Thread 2 is running: 4
```

Description:

Above Java program that illustrates the creation of threads using the Runnable interface. The start() method is used to start each newly created thread. Inside the run() method, there is a sleep() call to suspend the thread for 500 milliseconds.

In this program, MyRunnable is a class that implements the Runnable interface. The run() method is overridden to define the behavior of the thread. The Thread.sleep(500) call causes the current thread to suspend execution for 500 milliseconds.
The main() method creates two threads, thread1 and thread2, each with an instance of MyRunnable as its target. The start() method is called on each thread to begin their execution. The threads will run concurrently, each printing a message and then sleeping for 500 milliseconds in a loop

Note: that the actual output may vary each time you run the program because thread scheduling can be influenced by many factors and is not deterministic.

Program 12

Aim:

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.

Code:

```
class MyThread extends Thread {
MyThread() {
super("my extending thread");
System.out.println("my thread created " + this);
start();
 }
public void run() {
try {
for (int i = 0; i < 10; i++) {
System.out.println("Printing the count " + i);
Thread.sleep(1000);
}
} catch (InterruptedException e) {
System.out.println("my thread interrupted");
}
System.out.println("my thread run is over");
}
 }
public class ExtendingExample {
public static void main(String[] args) {
MyThread myThread = new MyThread();
try {
while (myThread.isAlive()) {
System.out.println("Main thread will be alive till the child thread is live");
Thread.sleep(1500);
}
} catch (InterruptedException e) {
System.out.println("Main thread interrupted");
}
System.out.println("Main thread's run is over");
}
 }
```

Output:

```
my thread created Thread[my extending thread,5,main]
Printing the count 0
Main thread will be alive till the child thread is live
Printing the count 1
Printing the count 2
Main thread will be alive till the child thread is live
Printing the count 3
Printing the count 4
Main thread will be alive till the child thread is live
Printing the count 5
Printing the count 6
Main thread will be alive till the child thread is live
Printing the count 7
Printing the count 8
Printing the count 9
my thread run is over
Main thread will be alive till the child thread is live
Main thread's run is over
```

Description:

This Java program that creates a class MyThread which extends the Thread class. The constructor of MyThread calls the base class constructor using super and starts the thread. The run method of the class is executed after the thread starts. Both the main thread and the child thread created by MyThread are executed concurrently.

In this program, MyThread is a class that extends the Thread class. The constructor of MyThread calls the constructor of the Thread class using super and starts the thread by calling start(). The run() method is overridden to define the behavior of the thread. The main() method in the ExtendingExample class creates an instance of MyThread and waits for it to finish.

Please note that the actual output may vary each time you run the program because thread scheduling can be influenced by many factors and is not deterministic.