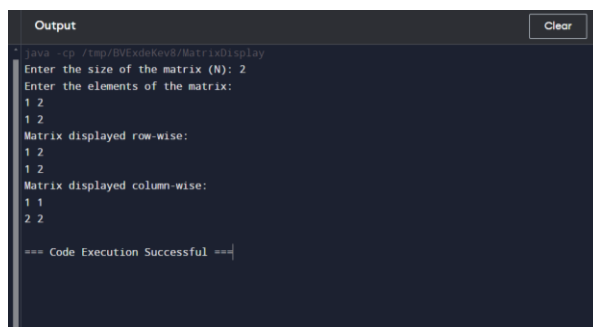


ASSIGNMENT-15

1. Write a program in Java to input an NxN matrix and display it row-wise and column-wise

```
import java.util.Scanner;
public class MatrixDisplay {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter the size of the matrix (N): ");
        int N = input.nextInt();
        int[][] matrix = new int[N][N];
        System.out.println("Enter the elements of the matrix:");
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                matrix[i][j] = input.nextInt();
            }
        }
        System.out.println("Matrix displayed row-wise:");
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                System.out.print(matrix[i][j] + " ");
            }
            System.out.println();
        }
        System.out.println("Matrix displayed column-wise:");
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                System.out.print(matrix[j][i] + " ");
            }
            System.out.println();
        }
        input.close();
    }
}
```



```
Output
java -cp /tmp/BVExdeKev8/MatrixDisplay
Enter the size of the matrix (N): 2
Enter the elements of the matrix:
1 2
1 2
Matrix displayed row-wise:
1 2
1 2
Matrix displayed column-wise:
1 1
2 2
=== Code Execution Successful ===
```

2. Write a java program which creates an interface IterF1 having 2 methods add () and sub (). Create a class which overloads the given methods for addition and subtraction of two numbers respectively.

```
// Interface IterF1
interface IterF1 {
```

```

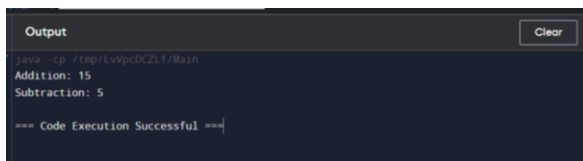
    void add(int a, int b);
    void sub(int a, int b);
}

// Class implementing the interface and overloading methods
class Calculation implements IterF1 {
    @Override
    public void add(int a, int b) {
        System.out.println("Addition: " + (a + b));
    }

    @Override
    public void sub(int a, int b) {
        System.out.println("Subtraction: " + (a - b));
    }
}

// Main class to test the interface and method overloading
public class Main {
    public static void main(String[] args) {
        Calculation calc = new Calculation();
        calc.add(10, 5);
        calc.sub(10, 5);
    }
}

```



```

Output
Addition: 15
Subtraction: 5
Code Execution Successful

```

3. Write a Java program to calculate the rate of interest of the employee's Provident Fund use the try and catch and finally block.

```

public class ProvidentFundCalculator {
    public static void main(String[] args) {
        try {
            double principal = 10000;
            double rate = 5; // 5% interest rate
            double time = 5; // 5 years

            double interest = (principal * rate * time) / 100;

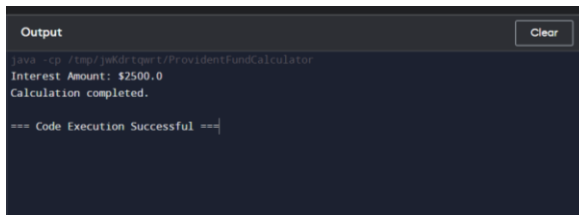
            System.out.println("Interest Amount: $" + interest);
        } catch (Exception e) {
            System.out.println("An error occurred: " + e.getMessage());
        } finally {
            System.out.println("Calculation completed.");
        }
    }
}

```

```

    }
}
}

```



```

Output
Clear
java -cp ./tmp/jwkdrtqert/ProvidentFundCalculator
Interest Amount: $2500.0
Calculation completed.

=== Code Execution Successful ===

```

- Write a program to create a class MyThread in this class a constructor, call the base class constructor, using super and starts 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

```

class MyThread extends Thread {
    MyThread() {
        super();
        start();
    }
    public void run() {
        System.out.println("Thread running...");
    }
    public static void main(String[] args) {
        MyThread thread = new MyThread();
        System.out.println("Main thread running concurrently with the child thread.");
    }
}

```



```

Output
Clear
java -cp ./tmp/188lp70bwdp/MyThread
Main thread running concurrently with the child thread.
Thread running...

=== Code Execution Successful ===

```

- Create a class Student with attributes roll no, name, age and course. Initialize values through parameterized constructor. If age of student is not in between 15 and 21 then generate user-defined exception "AgeNotWithinRangeException". If name contains numbers or special symbols raise exception "NameNotValidException". Define the two exception classes.

```

// Define the custom exception classes
class AgeNotWithinRangeException extends Exception {
    public AgeNotWithinRangeException(String message) {
        super(message);
    }
}
class NameNotValidException extends Exception {
    public NameNotValidException(String message) {

```

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

```
// Define the Student class
```

```
public class Student {
    private int rollNo;
    private String name;
    private int age;
    private String course;
    // Parameterized constructor
    public Student(int rollNo, String name, int age, String course) throws
    AgeNotWithinRangeException, NameNotValidException {
        if (age < 15 || age > 21) {
            throw new AgeNotWithinRangeException("Age must be between 15 and 21");
        }
        if (!name.matches("[a-zA-Z ]+")) {
            throw new NameNotValidException("Name must contain only letters and spaces");
        }
        this.rollNo = rollNo;
        this.name = name;
        this.age = age;
        this.course = course;
    }
    // Getters and setters
    public int getRollNo() {
        return rollNo;
    }
    public void setRollNo(int rollNo) {
        this.rollNo = rollNo;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) throws NameNotValidException {
        if (!name.matches("[a-zA-Z ]+")) {
            throw new NameNotValidException("Name must contain only letters and spaces");
        }
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) throws AgeNotWithinRangeException {
```

```

        if (age < 15 || age > 21) {
            throw new AgeNotWithinRangeException("Age must be between 15 and 21");
        }
        this.age = age;
    }

    public String getCourse() {
        return course;
    }

    public void setCourse(String course) {
        this.course = course;
    }

    // Main method
    public static void main(String[] args) {
        try {
            Student student = new Student(1, "John Doe", 20, "Computer Science");
            System.out.println("Student created successfully!");
            System.out.println("Roll No: " + student.getRollNo());
            System.out.println("Name: " + student.getName());
            System.out.println("Age: " + student.getAge());
            System.out.println("Course: " + student.getCourse());
        } catch (AgeNotWithinRangeException e) {
            System.out.println("Error: " + e.getMessage());
        } catch (NameNotValidException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}

```

```

Student created successfully!
Roll No: 1
Name: John Doe
Age: 20
Course: Computer Science

=== Code Execution Successful ===

```