Vikas Meneni **Lab 8 – Assignment 4** **NUID:002309537**

1.Problem 1

Problem Description:

The task involves solving a matrix-related problem, which is essential in computer science for understanding two-dimensional data structures. The program generates a matrix with dimensions filled with random integers between 0 and 9. It then calculates the sum of each row and column to identify the row and column with the highest sums. This problem matters because it demonstrates how to process and analyze data stored in a matrix format, which is widely used in fields like image processing, machine learning, and database management. By solving this, we learn efficient ways to manipulate and traverse two-dimensional arrays, which can be applied to more complex problems involving multi-dimensional data. From this problem, I can transfer the learning to similar challenges such as finding specific patterns in data, optimizing search algorithms in matrices, and even analyzing real-world tabular data. It builds a foundation for tackling problems requiring nested loops, data aggregation, and comparison operations.

Analysis:

The program is designed to generate a two-dimensional matrix, populate it with random values, and identify the row and column with the largest sums. The solution leverages fundamental programming concepts, including arrays, loops, and conditional logic.

Design and Solution

1.Matrix Representation:
A two-dimensional array is used to represent the n×m matrix. This structure allows efficient storage and traversal of rows and columns.

2.Matrix Population:
The matrix is filled with random integers between 0 and 9 using the Random class. Nested loops iterate through each cell to populate the matrix and simultaneously print its content.

3.Finding the Largest Row and Column Sums:

Row Sums: A single loop iterates through each row, calculating its sum by traversing its columns. The maximum sum and its index are updated during this process.

Column Sums: Similarly, another loop iterates through each column, summing values from all rows. The largest column sum and its index are tracked.

A key challenge was ensuring correct traversal of the matrix without index out-of-bounds errors while accurately calculating row and column sums. Managing indices for tracking the maximum sums added complexity, requiring precise logic. Handling edge cases, such as matrices with a single row or column or ties in maximum sums, also required additional checks to ensure correctness.

Optimizing the sum calculation by using a single traversal to compute both row and column sums is an alternative approach that could reduce computational overhead. For larger matrices, parallel processing offers another option to improve performance by dividing the workload into smaller tasks. Additionally, leveraging dynamic structures like ArrayList provides more flexibility for handling varying matrix sizes. Incorporating error handling for input validation and random value generation is another strategy to enhance the program's reliability and robustness.

Source code:

```
package edu.northeastern.csye6200;


import java.util.ArrayList;

import java.util.Scanner;


public class LAB8P1 {

  public static void main(String[] args) {

    Scanner scanner = new Scanner(System.in);



    System.out.print("Enter the number of rows: ");

    int rows = scanner.nextInt();
```

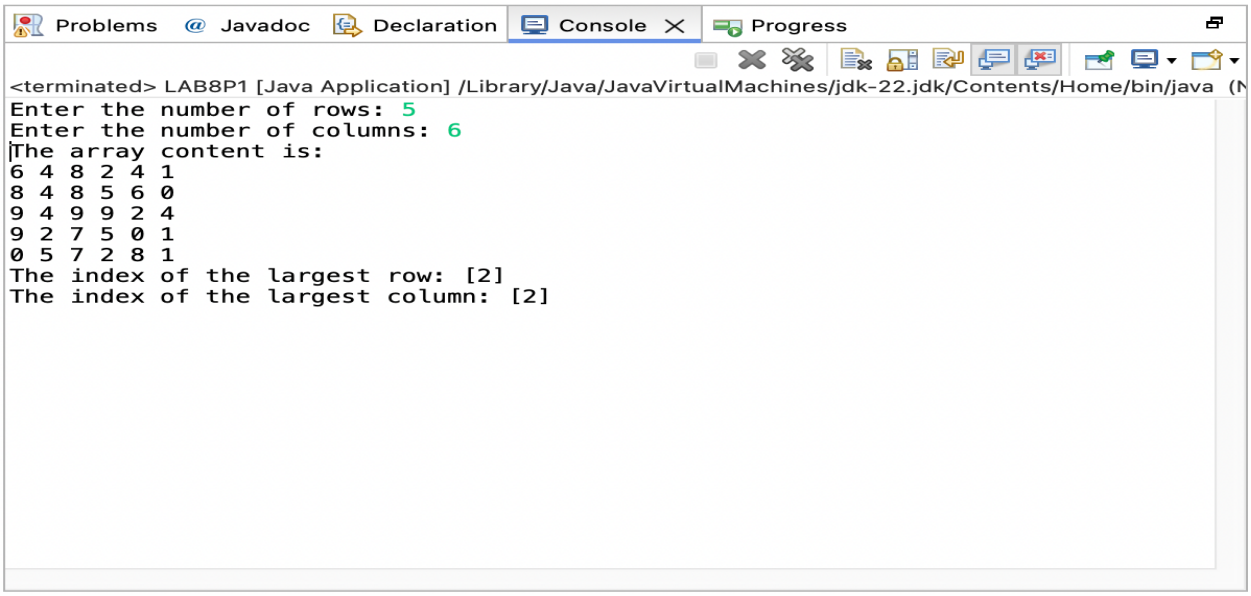
```
System.out.print("Enter the number of columns: ");
int cols = scanner.nextInt();


int[][] matrix = new int[rows][cols];


System.out.println("The array content is:");
for (int i = 0; i < rows; i++) {
   for (int j = 0; j < cols; j++) {
      matrix[i][j] = (int) (Math.random() * 10);
      System.out.print(matrix[i][j] + " ");
   }
   System.out.println();
}


int maxRowSum = Integer.MIN_VALUE;
ArrayList<Integer> largestRowIndices = new ArrayList<>();
for (int i = 0; i < rows; i++) {
   int rowSum = sumRow(matrix[i]);
   if (rowSum > maxRowSum) {
      maxRowSum = rowSum;
      largestRowIndices.clear();
      largestRowIndices.add(i);
```

```
      } else if (rowSum == maxRowSum) {
         largestRowIndices.add(i);
      }
   }


   int maxColSum = Integer.MIN_VALUE;
   ArrayList<Integer> largestColIndices = new ArrayList<>();
   for (int j = 0; j < cols; j++) {
      int colSum = sumColumn(matrix, j);
      if (colSum > maxColSum) {
         maxColSum = colSum;
         largestColIndices.clear();
         largestColIndices.add(j);
      } else if (colSum == maxColSum) {
         largestColIndices.add(j);
      }
   }


   System.out.println("The index of the largest row: " + largestRowIndices);
   System.out.println("The index of the largest column: " + largestColIndices);
}


public static int sumRow(int row[]) {
```

```
        int sum = 0;
        for (int i = 0; i < row.length; i++) {
            sum += row[i];
        }
        return sum;
    }


    public static int sumColumn(int matrix[][], int column) {
        int sum = 0;
        for (int i = 0; i < matrix.length; i++) {
            sum += matrix[i][column];
        }
        return sum;
    }
}
```

Screenshots of sample runs:

Screenshot :-



2.Problem 2

Problem Description:

The task is to create a bar chart that visually represents the distribution of a grade based on different components: project, exams, assignments, and attendance. Each component has a fixed percentage value: project (35%), exams (30%), assignments (30%), and attendance (5%). These percentages must be displayed in a bar chart format using JavaFX's Rectangle class, with each component represented by a bar of corresponding color: blue for project, green for exams, red for assignments, and orange for attendance. The program must calculate the appropriate widths of the bars based on the percentages and display them in a structured layout without using SceneBuilder. This problem is significant because it demonstrates how to translate data into a visual format, helping in understanding how JavaFX handles UI elements like shapes and layout. It also encourages the use of basic graphical programming techniques in Java, which is applicable in building interactive and visually rich applications.

Analysis:

Bar Chart Representation:
The bar chart is constructed using JavaFX's Rectangle class to visually represent each component. Each rectangle's width is determined by the percentage value it represents, scaled to fit the available window size.

Width Calculation:
The width of each rectangle is calculated by multiplying the percentage of the component by a scaling factor that adjusts the bar's size based on the window's total width. This ensures that the bars are proportional to the values they represent.

Bar Coloring:
Each component is associated with a specific color using JavaFX's color utilities. The Rectangle elements are colored as follows: blue for the project, green for exams, red for assignments, and orange for attendance. These colors are assigned during the creation of the Rectangle objects.

Layout and Arrangement:
The bars are arranged vertically in a structured layout using JavaFX's VBox or similar layout manager to ensure that each bar is displayed clearly and proportionally. No Scene Builder is used, so the entire layout is manually coded to maintain flexibility and control over the design

Difficulties Encountered:
One of the main challenges is ensuring that the bars are scaled correctly according to their percentages. Calculating the width of each bar accurately without SceneBuilder can be tedious, especially when considering the layout and alignment of the bars. Additionally, ensuring proper color assignment for each bar and managing any potential layout issues with spacing and alignment can be tricky without automatic visual aids.

Better Solutions:
A more flexible solution would involve creating a dynamic method that adjusts the layout based on the window size or available space, making the program adaptable to different screen resolutions. Another approach could be using custom JavaFX components like HBox or GridPane to further refine the positioning and scaling of the bars. Moreover, adding labels or tooltips to each bar would improve the clarity of the chart. Finally, incorporating input validation for percentage values would make the program more robust and user-friendly.

Source Code:

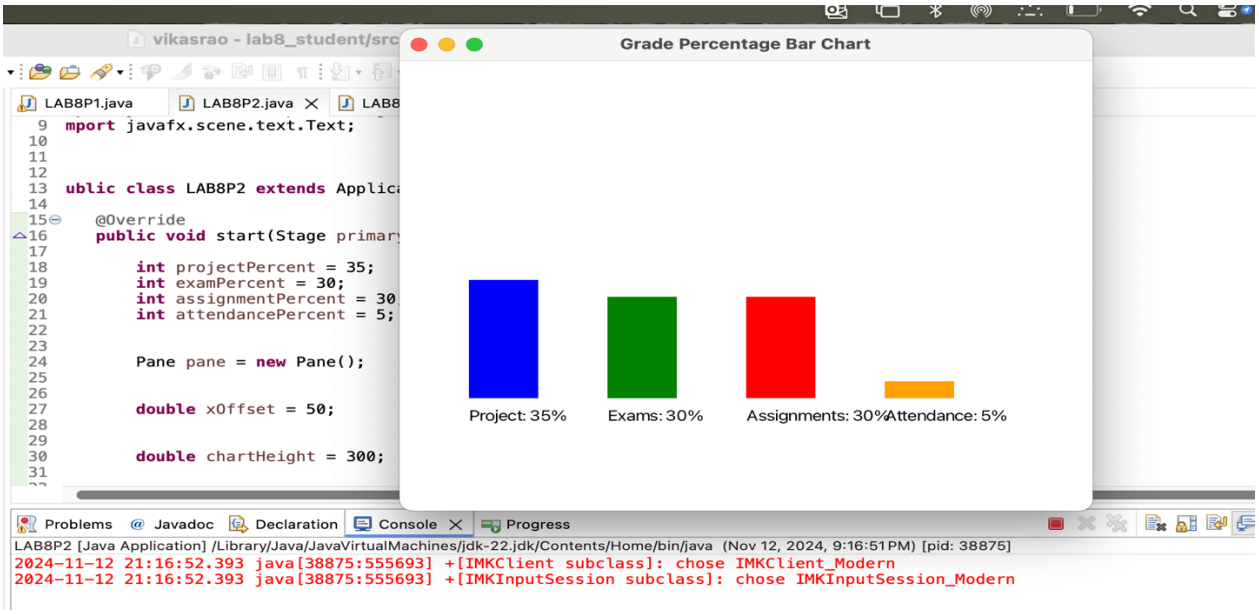
```
package edu.northeastern.csye6200;
```

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Text;



public class LAB8P2 extends Application{

        @Override
        public void start(Stage primaryStage) {

    int projectPercent = 35;
    int examPercent = 30;
    int assignmentPercent = 30;
    int attendancePercent = 5;



    Pane pane = new Pane();



    double xOffset = 50;
```

```
double chartHeight = 300;



Rectangle projectBar = new Rectangle(xOffset, chartHeight - (projectPercent * 3), 50, projectPercent * 3);

projectBar.setFill(Color.BLUE);

Text projectLabel = new Text(xOffset, chartHeight + 20, "Project: 35%");


Rectangle examBar = new Rectangle(xOffset + 100, chartHeight - (examPercent * 3), 50, examPercent * 3);

examBar.setFill(Color.GREEN);

Text examLabel = new Text(xOffset + 100, chartHeight + 20, "Exams: 30%");


Rectangle assignmentBar = new Rectangle(xOffset + 200, chartHeight - (assignmentPercent * 3), 50, assignmentPercent * 3);

assignmentBar.setFill(Color.RED);

Text assignmentLabel = new Text(xOffset + 200, chartHeight + 20, "Assignments: 30%");


Rectangle attendanceBar = new Rectangle(xOffset + 300, chartHeight - (attendancePercent * 3), 50, attendancePercent * 3);

attendanceBar.setFill(Color.ORANGE);

Text attendanceLabel = new Text(xOffset + 300, chartHeight + 20, "Attendance: 5%");



pane.getChildren().addAll(projectBar, projectLabel, examBar, examLabel, assignmentBar, assignmentLabel, attendanceBar, attendanceLabel);
```

```
        Scene scene = new Scene(pane, 500, 400);
        primaryStage.setTitle("Grade Percentage Bar Chart");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
        public static void main(String[] args) {
            launch(args);
        }

}
```

Screenshots of sample runs:



3.Problem 3:

Source Code:

```
package edu.northeastern.csye6200;

import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;

import java.util.Random;


public class LAB8P3 extends Application {

        @Override
        public void start(Stage primaryStage) {

    GridPane gridPane = new GridPane();
    gridPane.setAlignment(Pos.CENTER);
    gridPane.setHgap(5);
    gridPane.setVgap(5);
```

```
    Random random = new Random();



    for (int row = 0; row < 10; row++) {
        for (int col = 0; col < 10; col++) {

            int randomNumber = random.nextInt(2);



            TextField textField = new TextField(String.valueOf(randomNumber));
            textField.setPrefWidth(40);
            textField.setAlignment(Pos.CENTER);
            textField.setEditable(false);



            gridPane.add(textField, col, row);
        }
    }



    Scene scene = new Scene(gridPane, 400, 400);
    primaryStage.setTitle("10x10 Random Matrix");
    primaryStage.setScene(scene);
    primaryStage.show();
}
        public static void main(String[] args) {
```

```
        launch(args);
    }
}
```

Screen shots of Sample Runs:-