Lab 8 - Assignment 4:- CSYE6200 - Harsh Sangani

Problem 1: -

Problem Description -

Here we developed a program that performs an analysis on a randomly generated n × m matrix. The matrix is filled with integers ranging from 0 to 9. The program should not only print the matrix but also determine and display the rows and columns with the highest sum of values. To enhance efficiency, implement the solution using two-dimensional arrays for matrix representation and ArrayList for indexing.

This problem aims to test your ability to manage and analyze matrix data efficiently using appropriate data structures while incorporating randomness in the generation process. Overall, the assignment provided a valuable opportunity to reinforce algorithmic thinking, data structure usage, and practical problem-solving skills.

Analysis -

- User Input:
  Accept user input for the dimensions of the matrix (n, m).
- Matrix Generation:
  Generate a random n × m matrix filled with integers between 0 and 9.
- Matrix Printing:
  Print the generated matrix for visual verification.
- Row and Column Sum Calculation:
  Calculate the sum of values for each row and each column in the matrix.
- Identify Max Sums:
  Identify the rows with the highest sum and the columns with the highest sum.
- Efficient Index Storage:
  Utilize a two-dimensional array for matrix representation.
  Use ArrayLists to efficiently store the indices of rows and columns with the highest sum.
- Output:
  Display the generated matrix.
  Print the rows with the highest sum.
  Print the columns with the highest sum.

Difficulties Encountered –

- Efficient Index Storage:

Ensuring efficient storage and retrieval of indices for rows and columns with the highest sum required careful consideration of data structures.
Ensuring that the chosen data structures do not compromise the overall efficiency of the program.

- Random Matrix Generation:

Generating a random matrix with a specified range of values while maintaining uniform distribution posed challenges. Ensuring that randomness is balanced and appropriate required attention.

Learnings –

- Efficient Data Structures:
  Learned to choose and implement data structures (two-dimensional arrays and ArrayLists) for optimized matrix analysis.
- Random Number Generation:
  Gained experience in generating random numbers within a specific range for matrix initialization.
- Algorithm Optimization:
  Focused on optimizing the algorithm to identify rows and columns with the highest sum efficiently.
- User Interaction:
  Improved user interaction by incorporating input prompts and meaningful output, enhancing the overall user experience.
- Debugging and Testing:
  Developed skills in debugging and testing to ensure the correctness of the algorithm and the robustness of the program.

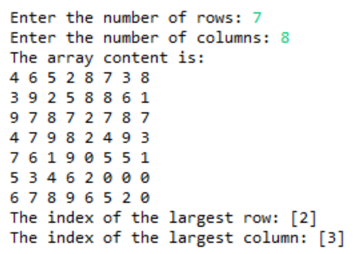
Source Code –

```
package edu.northeastern.csye6200;
import java.util.Random;
import java.util.Scanner;
import java.util.ArrayList;

public class LAB8P1 {
   public static void main(String[] args) {
       Scanner scanner = new Scanner(System.in);

       // Get the number of rows and columns from the user
       System.out.print("Enter the number of rows: ");
       int rows = scanner.nextInt();

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

       // Generate a matrix with random values
       int[][] matrix = generateMatrix(rows, columns);

       // Display the generated matrix
```

```
    System.out.println("The array content is:");
    printMatrix(matrix);

    // Find the indices of the largest row and column sums
    ArrayList<Integer> maxRowIndices = findLargestRow(matrix);
    ArrayList<Integer> maxColumnIndices = findLargestColumn(matrix);

    // Display the results
    System.out.println("The index of the largest row: " + maxRowIndices);
    System.out.println("The index of the largest column: " + maxColumnIndices);
}

// Method to generate a matrix with random values
public static int[][] generateMatrix(int rows, int columns) {
    Random random = new Random();
    int[][] matrix = new int[rows][columns];

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < columns; j++) {
            matrix[i][j] = random.nextInt(10);
        }
    }

    return matrix;
}

// Method to print the elements of a matrix
public static void printMatrix(int[][] matrix) {
    for (int[] row : matrix) {
        for (int value : row) {
            System.out.print(value + " ");
        }
        System.out.println();
    }
}

// Method to find the indices of the rows with the largest sum
public static ArrayList<Integer> findLargestRow(int[][] matrix) {
    int maxSum = Integer.MIN_VALUE;
    ArrayList<Integer> maxRowIndices = new ArrayList<>();

    for (int i = 0; i < matrix.length; i++) {
        int currentSum = sumRow(matrix[i]);
        if (currentSum > maxSum) {
            maxSum = currentSum;
            maxRowIndices.clear();
            maxRowIndices.add(i);
        } else if (currentSum == maxSum) {
            maxRowIndices.add(i);
        }
    }
```

```
        return maxRowIndices;
    }

    // Method to find the indices of the columns with the largest sum
    public static ArrayList<Integer> findLargestColumn(int[][] matrix) {
        int maxSum = Integer.MIN_VALUE;
        ArrayList<Integer> maxColumnIndices = new ArrayList<>();

        for (int j = 0; j < matrix[0].length; j++) {
            int currentSum = sumColumn(matrix, j);
            if (currentSum > maxSum) {
                maxSum = currentSum;
                maxColumnIndices.clear();
                maxColumnIndices.add(j);
            } else if (currentSum == maxSum) {
                maxColumnIndices.add(j);
            }
        }

        return maxColumnIndices;
    }

    // Method to calculate the sum of elements in a row
    public static int sumRow(int row[]) {
        int sum = 0;
        for (int value : row) {
            sum += value;
        }
        return sum;
    }

    // Method to calculate the sum of elements in a column
    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 –



```
Enter the number of rows: 7
Enter the number of columns: 8
The array content is:
4 6 5 2 8 7 3 8
3 9 2 5 8 8 6 1
9 7 8 7 2 7 8 7
4 7 9 8 2 4 9 3
7 6 1 9 0 5 5 1
5 3 4 6 2 0 0 0
6 7 8 9 6 5 2 0
The index of the largest row: [2]
The index of the largest column: [3]
```

Problem 2: -

Problem Description –

Here we designed a Java program that visualizes the distribution of percentages for different components contributing to an overall grade using a bar chart. The program should utilize the JavaFX Rectangle class to represent each component's percentage, with distinctive colors for clarity. The distribution includes project (35% - blue), exams (30% - green), assignments (30% - red), and attendance (5% - orange).

This assignment provided hands-on experience in using JavaFX for graphical representation and emphasized the importance of thoughtful design for effective data visualization. It allowed for practical application of programming skills in a visual context.

Analysis –

- Initialization**:**
  Use a data structure (e.g., a HashMap) to store component names and their corresponding percentages. This allows for easy scalability if more components are added in the future.
- JavaFX Bar Chart:
  Dynamically create rectangles based on the stored percentages, ensuring flexibility for changes in component percentages.
  Use a loop to iterate through the components, calculate the width of each rectangle based on its percentage, and position them appropriately on the chart.
- Color Palette:
  Consider using a color palette or a color-generating algorithm to automatically assign visually appealing and distinguishable colors to each component. This ensures consistency and easy modification of colors in the future.
- Visualization:
  Implement a modular approach for creating the bar chart to enhance code readability and maintainability.
  Include labels or legends dynamically based on the components and their assigned colors.
- User Interaction:
  If implementing user interaction features, use JavaFX functionalities for zooming, panning, or tooltips. Leverage JavaFX's event-driven model for a more seamless user experience.
- Error Handling:
  Incorporate error handling mechanisms for scenarios where the sum of percentages exceeds 100% or falls below it, providing informative messages to the user.

Difficulties Encountered –

- Dynamic Components:

Ensuring that the program can adapt to changes in the number of components without extensive code modifications. The use of data structures helps in a more flexible and scalable solution.

- Color Assignment:

Utilizing a color palette or algorithm reduces the manual selection of colors, ensuring better visual appeal and avoiding conflicts between component colors.

Learnings –

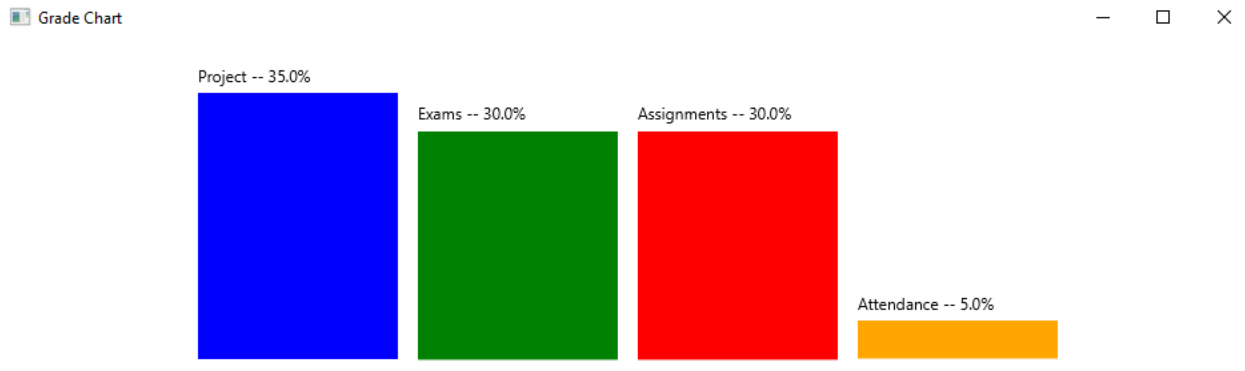
- Scalable Solutions:
  Improved understanding of creating scalable solutions that can accommodate changes in the problem requirements without significant code rewrites.
- Modular Design:
  Recognized the benefits of modular design for creating maintainable and readable code.
- Automatic Color Assignment:
  Learned how to automate the color assignment process, enhancing the program's adaptability and reducing the risk of visual inconsistency.
- User Interaction Best Practices:
  If applicable, gained insights into implementing user interaction features using JavaFX in a way that aligns with best practices for a smooth user experience.
- Error Handling Importance:
  Understood the importance of implementing error handling to ensure the program's robustness and provide meaningful feedback to users in case of unexpected scenarios.

Source Code –

```
package edu.northeastern.csye6200;

import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.layout.HBox;
import javafx.scene.layout.StackPane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class LAB8P2 extends Application {

  // Constants representing components, their percentages, and corresponding colors
  private static final String[] COMPONENTS = {"Project", "Exams", "Assignments", "Attendance"};
  private static final double[] PERCENTAGES = {35, 30, 30, 5};
  private static final Color[] COLORS = {Color.BLUE, Color.GREEN, Color.RED, Color.ORANGE};

  @Override
  public void start(Stage primaryStage) throws Exception {
    // Create an HBox to hold the components
    HBox hBox = new HBox(15);
    hBox.setAlignment(Pos.BOTTOM_CENTER);

    // Calculate the maximum percentage to normalize the heights of the rectangles
    double max = getMax(PERCENTAGES);
    double height = 200; // Fixed height for all rectangles
    double width = 150;  // Fixed width for all rectangles

    // Create a StackPane to hold the HBox
    StackPane pane = new StackPane();
    pane.setPadding(new javafx.geometry.Insets(20, 15, 5, 15));

    // Create rectangles and texts for each component
    for (int i = 0; i < COMPONENTS.length; i++) {
      Rectangle rectangle = new Rectangle(0, 0, width, height * PERCENTAGES[i] / max);
      rectangle.setFill(COLORS[i]);

      Text text = new Text(0, 0, COMPONENTS[i] + " -- " + PERCENTAGES[i] + "%");

      // Add each VBox (text and rectangle) to the HBox
      hBox.getChildren().add(getVBox(text, rectangle));
    }

    // Add the HBox to the StackPane
    pane.getChildren().add(hBox);

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane);
    primaryStage.setTitle("Grade Chart");
    primaryStage.setScene(scene);
    primaryStage.show();
  }

  // Method to find the maximum value in an array
  private double getMax(double[] values) {
    double max = values[0];
    for (double value : values) {
      if (value > max)
        max = value;
    }
    return max;
  }

  // Method to create a VBox containing text and a rectangle
  private VBox getVBox(Text text, Rectangle rectangle) {
    VBox vBox = new VBox(5);
    vBox.setAlignment(Pos.BOTTOM_LEFT);
    vBox.getChildren().addAll(text, rectangle);
    return vBox;
  }
```

```
   // Main method to launch the JavaFX application
   public static void main(String[] args) {
      launch(args);
   }
}
```

Screenshots of sample runs –



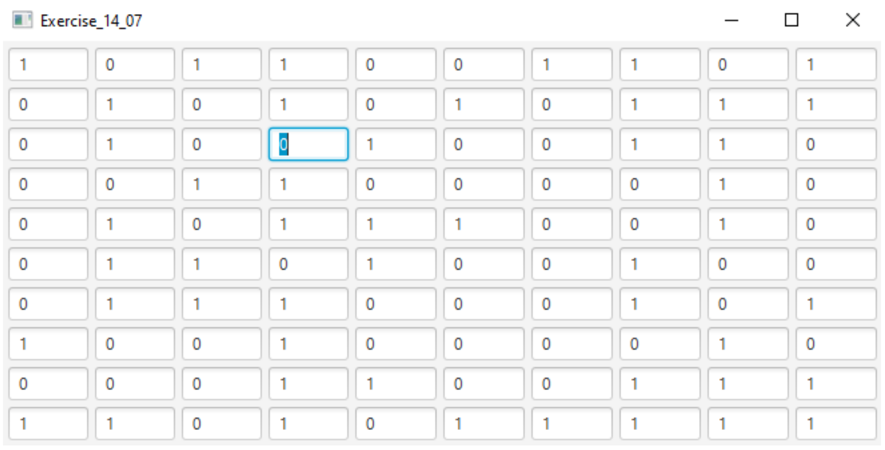
Problem 3: -

Source Code –

```
package edu.northeastern.csye6200;

import javafx.application.Application;
import javafx.geometry.HPos;
import javafx.geometry.Insets;
import javafx.geometry.VPos;
import javafx.scene.Scene;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;

public class LAB8P3 extends Application {

   @Override
   public void start(Stage primaryStage) {
      // Create a GridPane and set its properties
      GridPane pane = new GridPane();
      pane.setPadding(new Insets(5, 5, 5, 5));
      pane.setHgap(5);
      pane.setVgap(5);

      // Populate the GridPane with TextFields containing random 0 or 1
      for (int i = 0; i < 10; i++) {
         for (int j = 0; j < 10; j++) {
            TextField text = new TextField();
            text.setMaxWidth(Double.MAX_VALUE); // Allow the TextField to expand horizontally
            text.setText(String.valueOf((int) (Math.random() * 2))); // Set random 0 or 1
```

```
                pane.add(text, i, j); // Add the TextField to the specified row and column
                GridPane.setHalignment(text, HPos.CENTER); // Center the text horizontally
                GridPane.setValignment(text, VPos.CENTER); // Center the text vertically
            }
        }

        // Create a scene and place it in the stage
        Scene scene = new Scene(pane);
        primaryStage.setTitle("Exercise_14_07"); // Set the stage title
        primaryStage.setScene(scene); // Place the scene in the stage
        primaryStage.show(); // Display the stage

    }

    // Main method to launch the JavaFX application
    public static void main(String[] args) {
        launch(args);
    }
}
```

Screenshots of sample runs –



Exercise_14_07

```
1  0  1  1  0  0  1  1  0  1
0  1  0  1  0  1  0  1  1  1
0  1  0  0  1  0  0  1  1  0
0  0  1  1  0  0  0  0  1  0
0  1  0  1  1  1  0  0  1  0
0  1  1  0  1  0  0  1  0  0
0  1  1  1  0  0  0  1  0  1
1  0  0  1  0  0  0  0  1  0
0  0  0  1  1  0  0  1  1  1
1  1  0  1  0  1  1  1  1  1
```