Even Semester (2023)

BINUS UNIVERSITY

BINUS INTERNATIONAL

Object Oriented Programming Final Project

(Individual Work)

Student Information:

Surname: Nasima **Given Name:** Azza **Student ID:** 2602158166

Course Code : COMP6699001 **Course Name** : Object Oriented Programming

Class : L2BC **Lecturer:** Jude Joseph Lamug Martinez, MCS

Type of Assignment: Final Project Report

Submission Pattern

Due Date : 13 June 2023 **Submission Date** : 13 June 2023

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

BINUS International seriously regards all forms of plagiarism, cheating, and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity, and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept, and consent to BINUS International's terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student: Azza Nasima

Table of Contents

PROJECT DESCRIPTION

a. Purpose of Project

The purpose of my project is to fulfill the requirements of my Object-Oriented Programming (OOP) class by creating a final project that goes beyond the scope of what was taught in class. The goal is to assess our readiness to independently study Java programming. To achieve this, I decided to develop a Java quiz application with a graphical user interface (GUI) that aims to not only challenge users but also provide them with and entertaining experience. The key feature of the Java quiz application is its ability to generate a randomized set of questions from various categories. To add an element of urgency and excitement, each question is associated with a strict time limit of 7 seconds to challenge users to think quickly and make accurate decisions under pressure. Additionally, the application includes a scoring system that tracts user performance, providing motivation to improve.

b. Data Structures and Algorithms

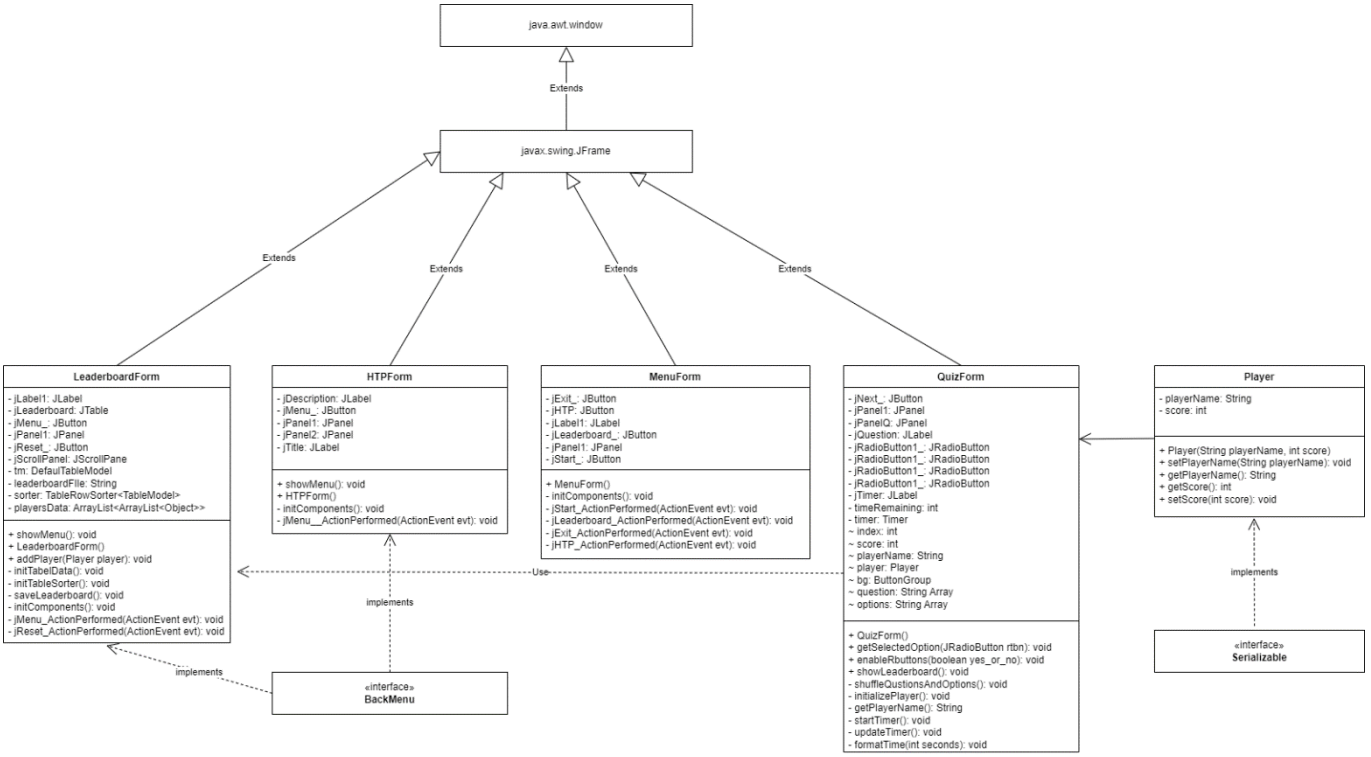
I developed the quiz application using several data structures and algorithms in order to create an interactive quiz application. These are some of the data structures and algorithms used in the "QuizForm" class, which is the main GUI from of the application. For data structures, there are the arrays "questions" and "options" which store the questions and corresponding answer options, including the correct answer for each question. These arrays are shuffled randomly at the beginning of the quiz to ensure a different order of questions in each session. This adds variety and prevent predictability. The algorithms in the "QuizForm" class handle shuffling questions, validating answers, managing the timer, and displaying the leaderboard. For example, shuffleQuestionsAndOptions() is used to shuffle the questions and options randomly. It iterates over the questions and swaps each question and its corresponding options with another randomly selected question. Additionally, the algorithm initiliazePlayer() initializes the player by obtaining the player's name and setting the initial score to 0. The player's name is obtained through an input dialog box when the application starts and a default name is used if no name is provided. After all the questions have been answered, the code checks if the index is equal to the length of the questions array. If true, it means all questions have been answered, and the final score is displayed.

c. Expected Outcomes

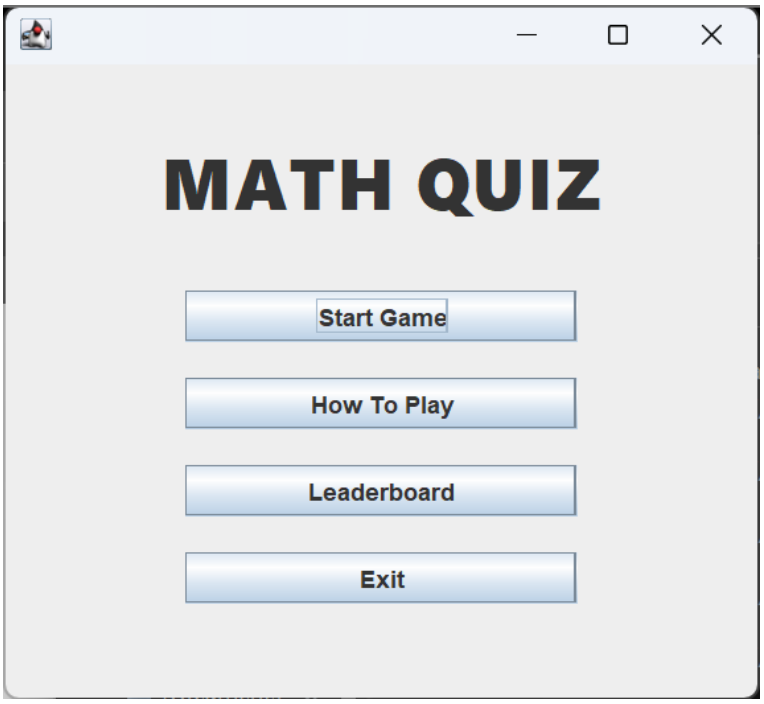
Firstly, the randomized shuffling of questions and answer options at the beginning of each session ensures that users will encounter a unique order of questions, enhancing the overall user experience. This randomization prevents predictability and repetition, making the quiz more engaging and challenging for players. Second, the graphical user interface (GUI) elements, including radio buttons or the multiple-choice buttons and a timer, facilitate a user-friendly experience while encouraging quick thinking a time management skill. Users must answer questions promptly, enhancing their ability to think quickly and make decisions under pressure. Overall, the implementation of the quiz application is expected to provide an interactive and enjoyable platform for users to test their knowledge and enhance their learning experience.

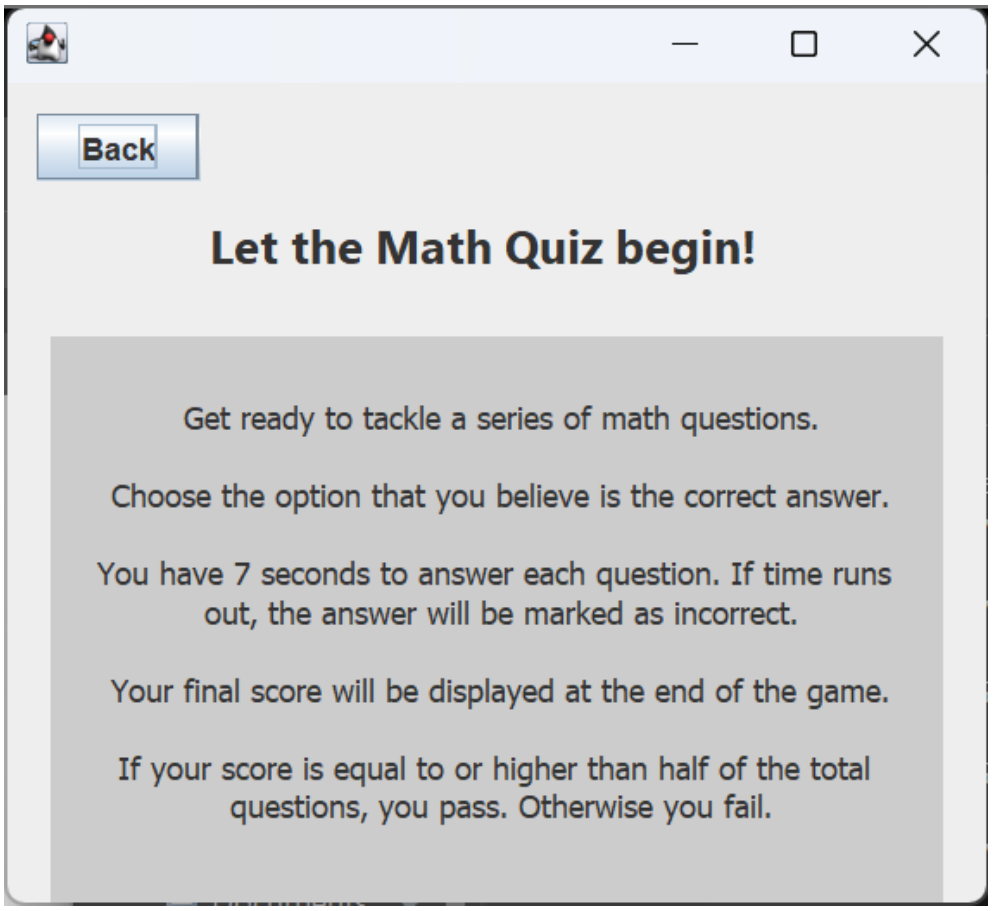
SOLUTION DESIGN

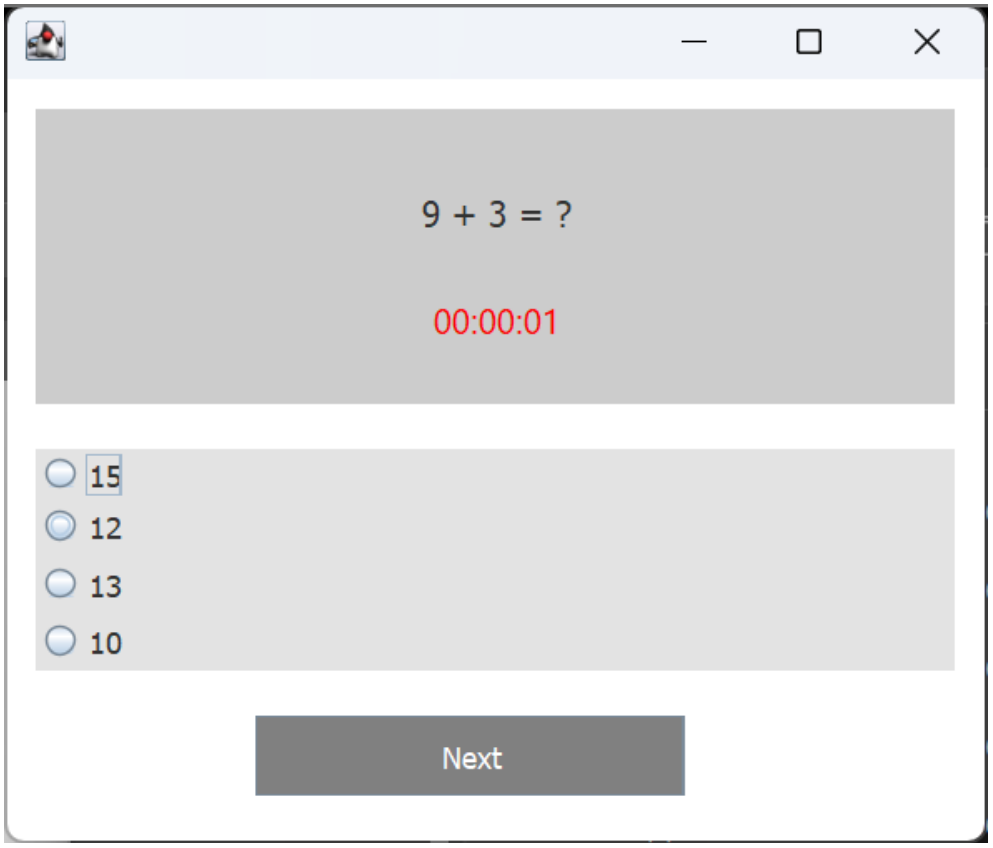
a. UML Class Diagram

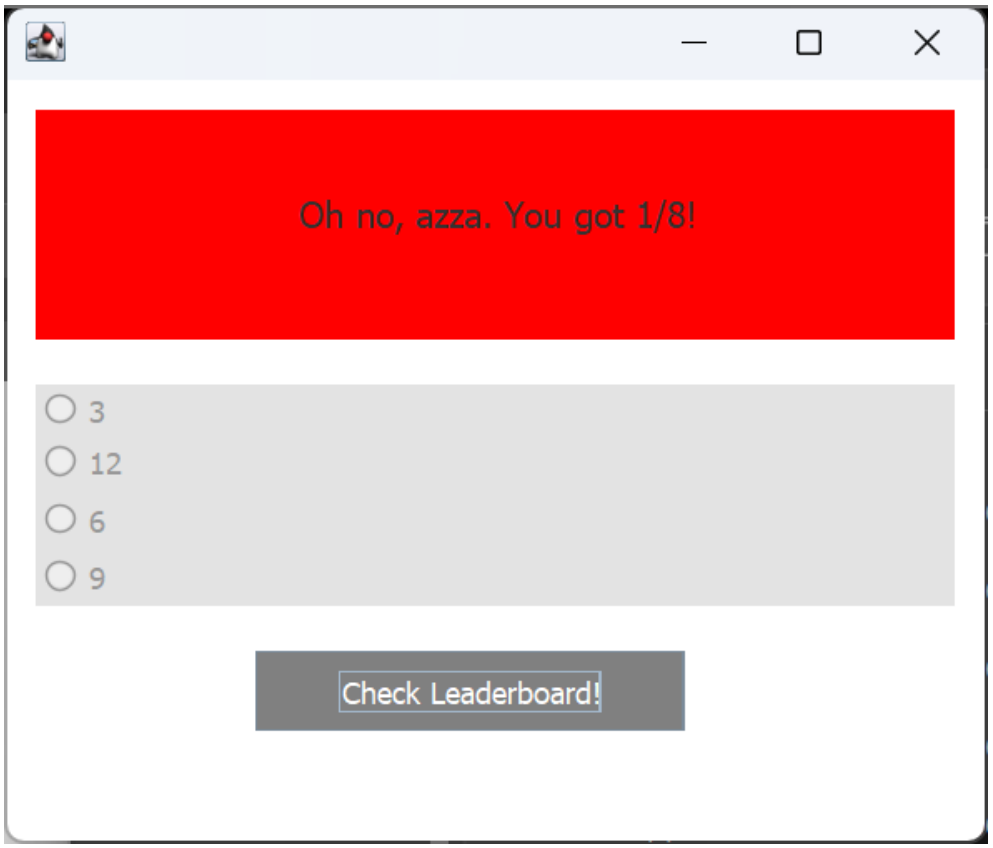


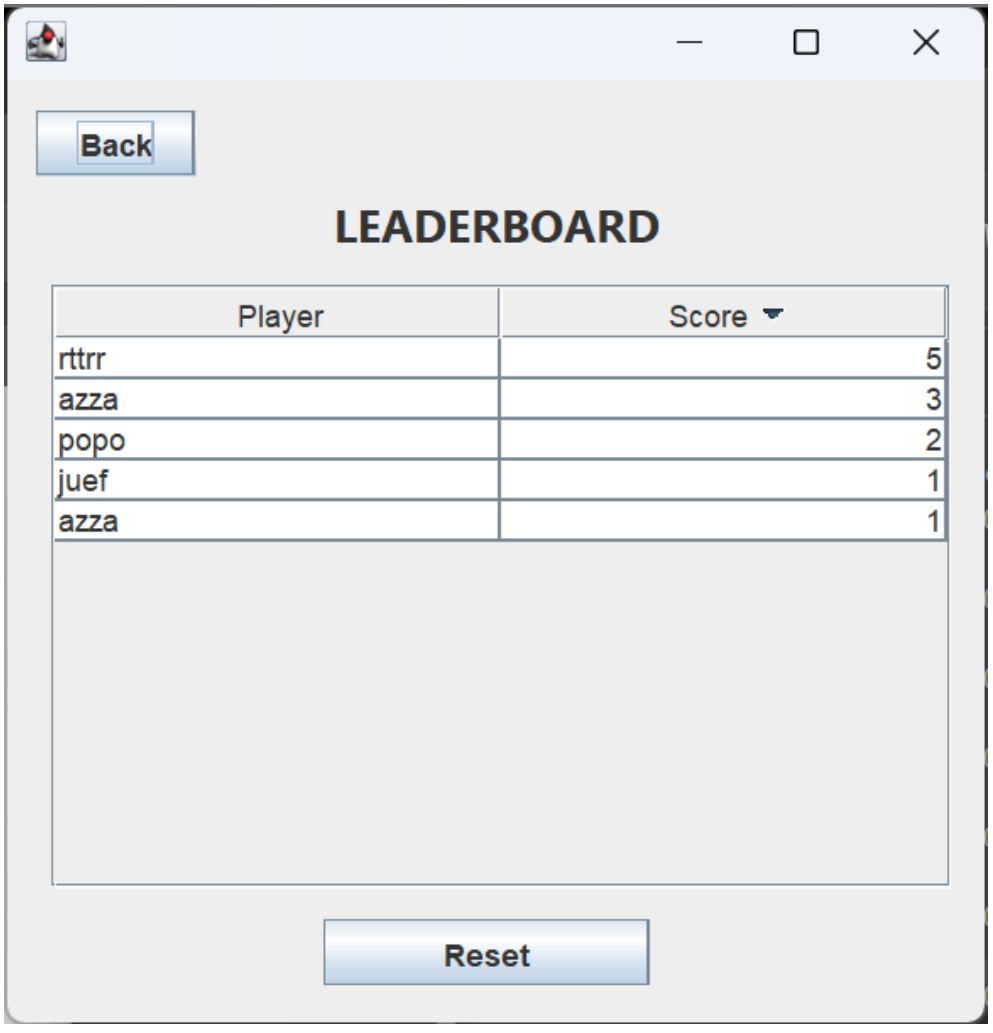
b. Screenshots




Back
Let the Math Quiz begin!
Get ready to tackle a series of math questions.
Choose the option that you believe is the correct answer.
You have 7 seconds to answer each question. If time runs out, the answer will be marked as incorrect.
Your final score will be displayed at the end of the game.
If your score is equal to or higher than half of the total questions, you pass. Otherwise you fail.


9 + 3 = ?
00:00:01
15
12
13
10
Next


Oh no, azza. You got 1/8!
3
12
6
9
Check Leaderboard!


Back
LEADERBOARD
Player
Score
rttrr 5
azza 3
popo 2
juef 1
azza 1
Reset

IMPLEMENTATION

a. Functions

- shuffleQuestionsAndOptions() : shuffles the questions and options randomly.

```
private void shuffleQuestionsAndOptions() {
    Random rnd = new Random();
    for (int i = questions.length - 1; i > 0; i--) {
        int index1 = rnd.nextInt( bound: i + 1);

        // shuffle questions
        String tempQuestion = questions[index1];
        questions[index1] = questions[i];
        questions[i] = tempQuestion;

        // shuffle options
        String[] tempOptions = options[index1];
        options[index1] = options[i];
        options[i] = tempOptions;
    }
}
```

- initializePlayer(): initializes the player by getting the player name and score.

```
private void initializePlayer(){
    // initialize the player by getting the player name and score
    playerName = getPlayerName();
    player = new Player(playerName,score);
}
```

- getPlayerName(): prompts the user to enter their name and returns the name as a string.

```
private String getPlayerName(){
    // prompt the user to enter their name and return the name as a string
    // if no name is entered, the name will be set as "Anonymous"
    String name = JOptionPane.showInputDialog( parentComponent: null, message: "Enter your name: ");
    if (name == null || name.trim().isEmpty()){
        name = "Anonymous";
    }
    return name;
}
```

- getSelectedOption(JRadioButton rbtn): gets the selected value from the radio button, checks if it's the correct answer, and updates the score.

```
public void getSelectedOption(JRadioButton rbtn){
    // get the selected value from the radiobutton
    // if the selected answer is correct, increment the score
    // disable the radiobuttons after selection is made

    System.out.println(rbtn.getText());
    System.out.println(options[index][4]);

    if(rbtn.getText().equals(options[index][4])){
      score++;
      player.setScore(score);
    }

    enableRbuttons( yes_or_no: false);
    index++;
}
```

- enableRbutton(boolean_yes_or_no): enables or disbales the radio buttons based on the input parameter.

```
public void enableRbuttons(boolean yes_or_no){
    // enable/disable radiobuttons based on the input parameter
    jRadioButton1_.setEnabled(yes_or_no);
    jRadioButton2_.setEnabled(yes_or_no);
    jRadioButton3_.setEnabled(yes_or_no);
    jRadioButton4_.setEnabled(yes_or_no);
}
```

- updateTimer(): updates the timer label every second and stops the timer when it runs out.

```
private void updateTimer() {
    // update the timer label every second
    // when the timer runs out, stop the timer, disable the radiobuttons, and move to the next question
    timeRemaining--;

    if (timeRemaining >= 0) {
        jTimer.setText(formatTime(timeRemaining));
    } else {
        jTimer.setText("Time's up!");
        timer.stop();
        jRadioButton1_.setEnabled(false);
        jRadioButton2_.setEnabled(false);
        jRadioButton3_.setEnabled(false);
        jRadioButton4_.setEnabled(false);
        index++;
    }
}
```

- formatTime(int seconds): formats the time in hh:mm:ss format and return it as a string.

```
private String formatTime(int seconds) {
    // format the time in hh:mm:ss format and return as a string
    int hours = seconds / 3600;
    int minutes = (seconds % 3600) / 60;
    int remainingSeconds = seconds % 60;

    return String.format("%02d:%02d:%02d", hours, minutes, remainingSeconds);
}
```

- showLeaderboard(): creates a new leaderboard form, adds a player to the leaderboard, and sets the leaderboard form visible.

```
public void showLeaderboard() {
    // create a new leaderboard form
    // add a player to the leaderboard
    // ser the leaderborad form visible
    LeaderboardForm leaderboardForm = new LeaderboardForm();
    leaderboardForm.addPlayer(player); // Assuming "player" is the Player object from QuizForm
    leaderboardForm.setVisible(true); // Set the leaderboard form visible
}
```

- jStart_ActionPerformed: it creates an instance of QuizForm and displays it, while closing the current frame.

```
private void jStart_ActionPerformed(java.awt.event.ActionEvent evt)
    // to start the game
    QuizForm qf = new QuizForm();
    qf.show();

    dispose(); // close current frame
}//GEN-LAST:event_jStart_ActionPerformed
```

- jLeaderboard_ActionPerformed: it creates an instance of LeaderboardForm and displays it, while closing the current frame.

```
private void jLeaderboard_ActionPerformed(java.awt.event.ActionEvent evt)
    // to show the LeaderboardForm
    LeaderboardForm lf = new LeaderboardForm();
    lf.show(); // display
    dispose(); // close current frame
}//GEN-LAST:event_jLeaderboard_ActionPerformed
```

- jExit_ActionPerformed: it displays a confirmation dialog and exits the application if the user confirms.

```
private void jExit_ActionPerformed(java.awt.event.ActionEvent evt)
    // to quit the game
    int quit = JOptionPane.showConfirmDialog( parentComponent: this,
    if(quit == JOptionPane.YES_OPTION){
        System.exit( status: 0);
    }
}//GEN-LAST:event_jExit_ActionPerformed
```

- jHTP_ActionPerformed: it creates an instance of HTPForm and displays it, while closing the current frame.

```
private void jHTP_ActionPerformed(java.awt.event.ActionEvent evt)
    // to show the How To Play Form
    HTPForm hf = new HTPForm();
    hf.show();
    dispose();
}//GEN-LAST:event_jHTP_ActionPerformed
```

- initTableData(): initializes the table data by reading it from a file. It connects the DefaultTableModel with the jLeaderboard table, reads the player data from the leaderboardFile, and adds the data to the DefaultTableModel.

```
private void initTableData(){
    // connect the DefaultTableModel with the jleaderboard table
    tm = (DefaultTableModel) jLeaderboard.getModel();
    // create an ArrayList to store the player data
    playersData = new ArrayList<>();

    try{
        // read the player data from the leaderboardFile
        FileInputStream fs = new FileInputStream(leaderboardFile);
        ObjectInputStream os = new ObjectInputStream(fs);
        playersData = (ArrayList<ArrayList<Object>>) os.readObject();

        os.close();
        fs.close();
    }
    catch(IOException | ClassNotFoundException e){}

    // add the data from playersData to the DefaultTableModel
    for (ArrayList<Object> rowData : playersData) {
        tm.addRow(rowData.toArray());
    }
}
```

- showMenu(): the function is implemented from the BackMenu() interface. It creates an instance of the MenuForm() class to display the menu.

```
public interface BackMenu {
    2 usages  2 implementations
    void showMenu();
}
```

- initTableSorter(): initializes the table sorter to allow sorting based on the column.

```
private void initTableSorter(){
    // create a TableRowSroter using the DefaulTableModel
    sorter = new TableRowSorter<>(tm);
    // set the sorter for the jleaderboard table
    jLeaderboard.setRowSorter(sorter);

    // create an ArrayList to specify the sorting order
    ArrayList<SortKey> keys = new ArrayList<>();
    // add a Sortkey for the second column (score) in descending order
    keys.add(new SortKey( column: 1, SortOrder.DESCENDING));

    // set the sort keys for the sorter
    sorter.setSortKeys(keys);
}
```

- saveLeaderboard(): saves the leaderboard data to a file. It writes the playerData to the leaderboardFile.

```
private void saveLeaderboard(){
    try{
        // write the playerData to the leaderboardFile
        FileOutputStream fs = new FileOutputStream(leaderboardFile);
        ObjectOutputStream os = new ObjectOutputStream(fs);

        os.writeObject(playersData);

        os.close();
        fs.close();
    }catch(IOException e){}

}
```

- Player(String playerName, int score): constructor method that initializes a Player object with the given playerName and score.

```
public Player(String playerName, int score){
    this.playerName = playerName;
    this.score = score;
}
```

- setPlayerName(String playerName): setter method to set the playerName of the Player object.
- getPlayerName(): getter method to retrieve the playerName of the Player object.

```
public void setPlayerName(String playerName) {
    this.playerName = playerName;
}

1 usage
public String getPlayerName(){
    return playerName;
}
```

- getScore(): getter method to retrieve the score of the Player object.
- setScore(int score): setter method to set the score of the Player object.

```
public int getScore() {
    return score;
}

1 usage
public void setScore(int score){
    this.score = score;
}
```

- addPlayer(Player player): adds a player to the leaderboard by updating the table and saving the data. It extracts the player's name and score.

```
public void addPlayer(Player player) {
    // get the player name and score
    String playerName = player.getPlayerName();
    int score = player.getScore();

    // create a new row of data for the player
    ArrayList<Object> rowData = new ArrayList<>();
    rowData.add(playerName);
    rowData.add(score);

    // add the row to the DefaultTableModel and playersData
    tm.addRow(rowData.toArray());
    playersData.add(rowData);

    // sort the table based on the sort keys
    sorter.sort();
    // save the updated leaderboard data
    saveLeaderboard();
    // make the LeaderboardForm visible
    this.setVisible(true);
}
```

b. Modules

- java.awt.Color: used for manipulating colors.
- java.awt.event.ActionEvent: used for handling action events.
- java.awt.event.ActionListener: used as an ActionListener interface for handling action events.
- java.util.Random: used for generating random numbers.
- javax.swing.ButtonGroup: used to group radio buttons together.
- javax.swing.JOptionPane: used for displaying input dialogs and message dialogs.
- javax.swing.JRadioButton: used for creating radio buttons.
- javax.swing.Timer: used for implementing timers.

```
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Random;
import javax.swing.ButtonGroup;
import javax.swing.JOptionPane;
import javax.swing.JRadioButton;
import javax.swing.Timer;
```

- java.io.FileInputStream: used for reading binary data from a file.
- java.io.FileOutputStream: used for writing binary data to a file.
- java.io.IOException: thrown when an input/output operation encounters an error or unexpected condition.
- java.io.ObjectInputStream: used for deserializing objects from an input stream
- java.io.ObjectOutputStream: used for serializing objects to an output stream.
- java.util.ArrayList: it provides methods to add, remove, and access elements in the list.
- javax.swing.RowSorter.SortKey: represents a sort key used for sorting rows in a table.
- javax.swing.SortOrder: represents the sort order used for sorting rows in a table.
- javax.swing.table.DefaultTableModel: an implementation of the TableModel interface that provides a default implementation for managing tabular data.
- javax.swing.table.TableModel: defines the methods for managing tabular data.
- javax.swing.table.TableRowSorter: provides sorting and filtering functionality for a table. It allows sorting rows based on the values in one or more columns.

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import javax.swing.RowSorter.SortKey;
import javax.swing.SortOrder;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;
import javax.swing.table.TableRowSorter;
```

c. GUI

- JFrame: provides the basic framework for the GUI and holds other components.

```
public class MenuForm extends javax.swing.JFrame {
```

- JLabel: used to display text.
- JButton: represents a clickable button.
- JPanel: used to organize and group other components together within the JFrame.

```
    private javax.swing.JLabel jLabel1;
    5 usages
    private javax.swing.JButton jLeaderboard_;
    5 usages
    private javax.swing.JPanel jPanel1;
```

- JRadioButton: a button that can be selected or deselected.

```
    private javax.swing.JRadioButton jRadioButton1_;
    14 usages
    private javax.swing.JRadioButton jRadioButton2_;
    14 usages
    private javax.swing.JRadioButton jRadioButton3_;
    12 usages
    private javax.swing.JRadioButton jRadioButton4_;
```

- JOptionPane: used to display message dialogs or input dialogs to interact with the user.
  In the code below, it is used to prompt for the player's name and shoe message dialogs for score and leaderboard.

```
    String name = JOptionPane.showInputDialog( parentComponent: null,  message: "Enter your name: ");
    if (name == null || name.trim().isEmpty()){
        name = "Anonymous";
    }
    return name;
```

REFLECTION

Developing the Java quiz application as my final project for the Object-Oriented Programming class was both challenging and rewarding. Throughout the project, I had the opportunity to apply the concepts learned in class to create a functional and interactive application. One of the main challenges was designing the class structure and ensuring proper inheritance and polymorphism. Another significant aspect of the project was implementing the graphical user interface (GUI). This required me to understand how to create interactive elements such as buttons, text fields, and tables. It was essential to strike a balance between aesthetics and usability.

Overall, this project was a valuable learning experience that deepened my understanding of creating an application using java. This experience allowed me to sharpen my problem-solving skills, improve my code design abilities, and gain confidence in developing real-world applications. While I recognize that there is always room for improvement, I am proud of the final outcome of my Java quiz application and the valuable knowledge and skills I have gained during the process.

Link to demo video: https://www.youtube.com/watch?v=Y7Wb0Mc6eEQ

RESOURCES

https://1bestcsharp.blogspot.com/2021/08/java-quiz-app-source-code.html
https://youtu.be/5P8lCgteYKQ