GROUP2.9 JAVA PROGRAMMING PROJECT

2362870

2363550

2362146

2363287

2363707

Project description

Our project is a quiz system. You can login and register, choose the topic and then choose to answer the quiz or view the Dashboard. After answering the quiz, the score, Dashboard and Leaderboard will be displayed. Questions will be randomly selected from varies difficulties of a specific topic, the number of extracted can be modified in the program, 1 point for each question. When using the program, run the" main" file and Login, readQuestion, and scoreManager classes will be called.

Algorithm

Users' data and login

To add, store, and interact with user information, our group created a Login class that includes the following functions:

```
public class Login {
```

1.regestrition converts the entered information into an array through the keyboard and stores the information in the array into a .csv file.

```
        // import
    String[] newUser = new String[3];
    newUser[0] = nickname;
    newUser[1] = name;
    newUser[2] = password;

    CSVWriter(newUser);
```

```
public void CSVWriter(String[] newUser) {   //Array type
```

2.login will convert all the information in the. csv file into a two-dimensional array and traverse the two-dimensional array for comparison with the information entered by the keyboard.

Question selecting and option shuffling

(1) Topic selection:

Read the names of .xml file which store the questions, select an optional topic, and then compare with the content entered by the keyboard to determine the selected topic, and import all questions of the topic into the array.

```
  public static Question[] SelectTopic() {
```

(2) Random selection:

According to the preset number of questions required for each difficulty, randomly select the corresponding number of questions in each difficulty.

```
  public static Question[] selectQuestions(Question[] allquestions, int EASY, int MEDIUM, int
HARD, int VERY_HARD) {
```

(3) Shuffle the options

```
public static void shuffleOptions(Option[] options) {
    for (int i = 0; i < options.length; i++) {

      int random = (int) (Math.random() * options.length);
```

```
            Option temp = options[random];
            options[random] = options[i];
            options[i] = temp;
        }
    }
```

Answering questions

Show the questions one by one and record the answers through the keyboard entry,

```
                // Present the questions
        for (int i = 0; i < selected.length; i++) {
            System.out.println(selected[i].getQuestionStatement());
            Option[] options = selected[i].getOptions();

            ReadQuestions.shuffleOptions(options); // Scramble options

            for (int j = 0; j < options.length; j++) {
                System.out.println((j + 1) + ": " + options[j].getAnswer());
            }
```

and determine whether the answer is right or wrong by comparing the answer entered by the keyboard with the preset answer.

Score calculating

The score is calculated by accumulation, and the score is displayed after all the questions are done

```
// answer
            int ans = -1;
            while (true) {
                System.out.println("Please print the answer (Print a number)");
                try {
                    ans = sc.nextInt();
                    if (ans >= 1 && ans <= options.length) { // Validity test
                        if (options[ans - 1].isCorrectAnswer()) {
                            totalpoints++;
                        }
                        break;
                    } else {
                        System.out.println("Invalid input");
                    }
                } catch (InputMismatchException e) {
```

```
            System.out.println("Invalid input");
            sc.next();
        }
    }
```

Save score

(1) Read all data in score.csv each time after finishing the attempt.

```
                    allData.add(fields);

        String user = fields[0];
        String topicName = fields[1];
        int score = Integer.parseInt(fields[2]);
        String type = fields[3];
```

(2) Store the read data which have been labeled "recent" into the recentScore array.

```
                    // recent 3 scores
        if (user.equals(username) && topicName.equals(topic) && type.equals("recent")) {
            recentScores.add(score);
        }
```

(3) Add the information about the attempt to the recentScore array.

(4) Check the number of elements in the array, if the number is greater than 3, then delete the first information.

```
    // If there are more than three records, delete them
    recentScores.add(currentScore);
    if (recentScores.size() > MAX_RECENT_SCORES) {
        recentScores = recentScores.subList(recentScores.size() - MAX_RECENT_SCORES,
recentScores.size());
    }
```

(5) Store the data which have been labeled "highest" in the read data into the relevant variable.

```
        // highest for each topic
        if (topicName.equals(topic) && type.equals("highest")) {
            topicHighestScore = score;
            highestUserForTopic = user;
        }
```

(6) Compare the data stored in the variable with the data about this attempt and do the update if is needed.

```
    // If it is the highest score, it will be updated (it will not be updated if it is the same score as
the highest score)
    if (currentScore > topicHighestScore) {
            topicHighestScore = currentScore;
            highestUserForTopic = username;
        }
```

(7) Update allData

```
// remove old data
        allData.removeIf(record -> record[0].equals(username) && record[1].equals(topic) &&
record[3].equals("recent")); // recent
        allData.removeIf(record -> record[1].equals(topic) && record[3].equals("highest")); //
highest
        // Write back to recent
        for (int score : recentScores) {
            allData.add(new String[] {username, topic, String.valueOf(score), "recent"});
        }
        // Write back to highest
        allData.add(new String[] {highestUserForTopic, topic, String.valueOf(topicHighestScore),
"highest"});
```

(8) Write back to score.csv

Dashboard

Read the score.csv file and display the data with the "recent" tag which also correspond to the user and the topic after each attempt. The Dashboard will display the last 3 attempts (involving the current attempt).

```
    public void displayRecentScores() {
```

Leaderboard

Read and display the data in the score.csv file with the label "highest" which correspond to the topic of this attempt.

```
    public void displayHighestScore() {
```

Tests

(1) Test on keyboard input

For all places where Scanner input is called, the input of empty strings, Chinese and all ASCII characters will not crash is ensured.

For all places where Scanner input is called, if something entered is not expected, it will be asked to re-enter.

```
  if (topics.contains(field)) {
      break; // If the input is valid, jump out of the loop
```

```
 } else {
    System.out.println("Invalid topic selected. Please select a valid topic.\n");
}
```

If the entered account does not exist, a new entry is required. If the account and the password does not match, a re-enter of the password is required.

```
if(!cardExists) {
          System.out.println("The username you entered does not exist, please try again\n");
          sc.nextLine();
          return null;
      }
```

```
                else {
                 System.out.println("The password is incorrect, please re-enter it");
              }
```

Ensure that username, name, and password are not empty. You cannot register a new account using the existing username.

```
                if (nickname.length() == 0 || name.length() == 0 || password.length() == 0) {
          System.out.println("Information cannot be empty\n");
          return; // jump out
      }
```

(2) Testing of dependent documents
If the users.csv file does not exist, a new users.csv file will be created.
If the score.csv file does not exist, a new score.csv file will be created.

```
      if (!file.exists()) {
          try {
              file.getParentFile().mkdirs();
              file.createNewFile();
          } catch (IOException e) {
              System.out.println("Error creating CSV file: " + e.getMessage());
          }
      }
```

(3) Test on reading questions
Questions from the questionBank will be read and determine whether they are valid or not. Questions that are not valid will not be selected and displayed to the user or counted in the number of questions.

```
// Determine whether the question is scrapped Scrap the question； getTopic() is empty，
getQuestionStatement() is empty， getOptions() length<=1， options[].isCorrectAnswer()true
number !=1
    for (int i = 0; i < allquestions.length; i++) {

      if (allquestions[i].getTopic() == null || allquestions[i].getQuestionStatement() == null)
{ // It is possible that this judgment can be commented out, because the ReadQuestion function
given by him cannot read null values
        canselect[i] = 1;
      }

      if (allquestions[i].getOptions().length <=1) {
        canselect[i] = 1;
      }

      int numofcorrect = 0;
      Option[] options = allquestions[i].getOptions();
      for (int j = 0; j < options.length; j++) {
        if (options[j].isCorrectAnswer()) {
          numofcorrect++;
        }
      }
      if (numofcorrect != 1) {
        canselect[i] = 1;
      }
    }
```

Determine whether there are enough questions for every difficulty in this topic. If the number of topics is insufficient, then throw new IllegalArgumentException("There are not enough questions. Please contact the teacher for assistance.").

```
    if (easycount < EASY || mediumcount < MEDIUM || hardcount < HARD || veryhardcount
< VERY_HARD) {
      throw new IllegalArgumentException("There are not enough questions. Please contact
the teacher for assistance.");
    }
```

Name	Student ID	Task(s)	Contribution(%)	Signature
Shuobai.Chen	2362870	User registration and authentication	20	陈烁百
Chengfeng.Zou	2363550	Topic selection	20	[illegible]
Yiduo.Xu	2362146	Quiz taking	20	许亦多
Qinzi Wang	2363287	Dashboard	20	王沁子
Ruihan.Xiong	2363707	Leaderboard	20	熊睿涵