$$15.2 \div 0.2 = ?$$

COMPUTER SCIENCE – THE DOSSIER

MATH SPEED GAME

[1]

Name: Nahel Rifai Burneo

Candidate Number: D 000633 054

Teacher Name: Mr. Jason Silby

School Name: Markham College

School Number: 0633

Examination Session: Nov 2012

Date: Sep. 16, 2012

[1] http://www.ibo.org/communications/brand/downloads/files/jpg/world_school/TriWorldSchool2Colourlarge.jpg (Last Access 16/9/2012)

Content

A1 – Analysing the Problem

Change in Educational Environment

Markham College is divided into two different campuses, The Early Years (from Pre-Kindergarten to 1st grade) and Lower School (2nd to 5th grade) campus and the Higher School (from 6th grade upwards) campus. When students pass from 5th grade to 6th grade there is a drastic change in their educational environment. They face a bigger and more serious atmosphere with tougher teachers and harsher syllabuses.

Markham College, as most elementary schools in the world, takes mathematics as one of the most important areas of education for their pupils and recognizes that strong education of maths in younger years will have a substantial positive effect on the students' futures.

Improving the skills of the "Newcomers"

Maths teachers from 6th grade, like Sr. Rafael Salomon (a former "ex-markhamian" and one of the most able math teachers in school), is mainly concerned about the 6th grade student's fluidity and quickness with math problems. When interviewed he confessed that to improve his student's performance in this area he uses two different methods.

Homework

The first method lies in setting homework consisting of numerous short maths problems for the students to complete in a fast manner so they'll eventually improve their mathematic speed. Sr. Salomon tells them to take the time they took to complete all the questions and record it in header of the homework.

The first problem he faces is that he is not able to know the time the students took to complete the exercises as they could easily cheat and write any time. To try and avoid this the parents were advised to personally record the times so their children won't cheat and get better in the long run, unfortunately we must also consider that there are parents that don't have the time to do this and leave their children to do their homework alone. Moreover, he has to manually mark over 30 pieces of homework (around 30 students per math class); this takes at least 30 minutes of his time as he has to point out any errors so the students won't make the same mistake again. As the homework is set twice a week he spends around an hour marking the 6th grader's homework. In addition, he also has to manually create a different bank of 20 new questions two times a week plus the printing and handing out of sheets for the students to take home. Furthermore, for the first time in his life he has 7 different maths classes to teach every week, this means a lot of work to complete in such a short period of time. He needs a solution right away because he doesn't want to give up these pieces of practice homework for the 6th grade students as he strongly recognizes the advantages of constantly practicing quick math problems, especially in young pupils.

Following is a sample of the practice homework.

MARKHAM COLLEGE
MATHS DEPARTMENT

TIME TAKEN:
YEAR : P6

Exercise 1 : ADDITION

1. 27+31
2. 45+22
3. 234+17
4. 316+204
5. 50+911
6. 291+46
7. 299+197
8. 306+205

Exercise 2 : SUBTRACTION

1. 97−63
2. 69−41
3. 83−60
4. 87−5
5. 192−81
6. 214−10
7. 86−29
8. 52−37

Exercise 3 : MULTIPLICATION

1. 2.06×0.05
2. 47.55×3.2
3. 0.023×45.1
4. 23.01×9.6
5. 10×3.254
6. 67.82×1000
7. 13.52×54.2
8. 34.8×1.7

Exercise 4 : DIVISION

1. 180÷2.5
2. 3.45÷0.5
3. 43.2÷1.6
4. 0.032÷0.04
5. 0.0072÷0.8
6. 1000÷0.01
7. 82÷10000
8. 0.04565÷0.05

Exercise 4 : ORDERING NUMBERS
(*smallest first*)

1. 0.21 , 0.31 , 0.12 , 0.012
2. 0.67 , 0.0672 , 0.7 , 0.072
3. 0.1 , 0.09 , 0.089
4. 0.41 , 0.041 , 0.14
5. 0.5 , 0.407 , 0.047 , 0.470
6. 0.04 , 0.004 , 0.4
7. 1.57 , 1.568 , 1.5
8. 79.082 , 79.1 , 79.01

Exercise 5 : NEGATIVE NUMBERS

1. (-6)+(+2)
2. 9+(−8)
3. (+2)−(−4)
4. (−7)−(+1)
5. (+7)×(−5)
6. (−1)×(−3)
7. (−9)÷(+3)
8. (−12)÷(−3)

Exercise 6 : PERCENTAGES
(*convert into percentages*)

1. $\frac{3}{4}$
2. $\frac{2}{5}$
3. $\frac{1}{2}$
4. $\frac{7}{100}$
5. $\frac{17}{50}$
6. $\frac{27}{40}$

Exercise 7 : PERCENTAGES
(*convert into decimals*)

1. 75%
2. 12%
3. 100%
4. 0.4%
5. 14.8%
6. 64 ½ %

Generating the Questions

The mathematical problems are manually made by Sr. Salomon. He writes them down in a gridline paper and then copies them into Microsoft Word when he finishes. He doesn't just write any math problem up; he has to follow certain standards imposed by the P6 Math's syllabus. This is why when he generates the questions he must remember the syllabus and follow certain parameters. If he forgets or is not sure about a question he must check the syllabus to see what mathematic knowledge is expected from the p6 students.

Following is the parts of the syllabus that Sr. Salomon uses in order to make the math problems.

The first part of the syllabus and the base of all of it is the "Four Operations" section.

Number		
Unit 1: The Four Operations		
Presumed Knowledge	**All (Meets)**	**Most (Meets +)**
• Add and subtract numbers including decimals by a written method • Multiply by two and three digit numbers by a written method • Divide whole numbers using a written method • Find equivalent fractions, simplify fractions and convert between improper fractions and mixed numbers • Find fractions of an amount • Add and subtract fractions • Solve worded problems involving fractions • Convert decimals to fractions using tenths and hundredths	• Multiply decimals • Multiply and divide fractions and mixed numbers • Use equivalent fractions (with tenths and hundredths) to convert fractions to decimals • Divide with decimal results • Perform calculations in the correct order (including use of brackets)	• Solve ratio and proportion problems by unitary method • Divide by decimals • Estimate calculations using rounding

Adding and Subtracting Questions

As we can clearly see here there are no parameters for problems about addition and subtraction, but Sr. Salomon states that generally they don't exceed the 3 digits or decimal places (eg: 431+223 0.431-0.223).

Multiplication Questions

Moreover, the students must be able to multiply two and three digits numbers; this is another parameter Sr. Salomon must follow, he must not create problems that multiply numbers with more than 3 digits or 3 decimal planes (eg: 431*223 or 0.431*0.223).

Division Questions

Also, the students need to know how to divide whole numbers and divide with decimal results. Sr. Salomon rarely makes a division that results with many decimal places. He states that he doesn't have a specific parameter of the size of the division that he puts on the homework but he makes sure that the solution to the division is either a whole number or a number with a maximum of 2 decimal places so the students don't struggle too much and take lots of time to do the divisions.

Discussion

As it has been said Sr. Salomon does everything manually, he tries to remember the exact parameters which the questions of each section must follow and if he forgets he must go over the syllabus again or ask other teachers from the math department. He states that if these questions and their solutions were generated manually following all the parameters he would save a lot of time.

Generating the score

In order to give a mark over 100 Sr. Salomon uses the accuracy of the(a) amount of correct answers and the time taken to complete them to calculate a final score.

a) Amount of Correct answers

First of all he goes all over the homework and records the amount of correct answers over the amount of problems at the top of the homework (eg: 51/60) and finally at the side he records the percentage of correct answers (eg : : 51/60 = 85%).

b) Time taken

Sr. Salomon then multiplies the amount of questions in the homework by 25 (e.g. 60 * 25 = 1500) this number, in seconds, is the maximum time the students should take to complete the homework (25 seconds per question).

After he has done the calculation he checks the time taken (in seconds) for each student and divides it by the maximum time. For example if the student has taken 250 seconds and the maximum time is 600 seconds he has a bonus score of 2.4. If the student has taken 600 seconds or more he has no bonus score.

The bonus score is multiplied by the percentage of correct answer (eg: 85% * 2.4 = 204) and this number is divided by two. This is the final score over 100. If the score is bigger than 100 it is still recorded as 100%.

See the following page to see an example of a student's homework and Sr. Salomon's calculations in order to get the final score percentage.

MARKHAM COLLEGE Aurelio Ascenzo 80.4%
MATHS DEPARTMENT

TIME TAKEN: 10:25
YEAR : P6

Exercise 1 : ADDITION

1. 27+31 = 58
2. 45+22 = 67
3. 234+17 = 251
4. 316+204 = 520
5. 50+911 = 961
6. 291+46 = 337
7. 299+197 = 496
8. 306+205 = 511

Exercise 2 : SUBTRACTION

1. 97−63 = 34
2. 69−41 = 28
3. 83−60 = 24 x = 23
4. 87−5 = 82
5. 192−81 = 113 x = 111
6. 214−10 = 204
7. 86−29 = 56 x = 57
8. 52−37 = 15

Exercise 3 : MULTIPLICATION

1. 2.06×0.05 = 0.103
2. 47.55×3.2 = 152.16
3. 0.023×45.1 = 1.04
4. 23.01×9.6 = 120.8 x = 220.9
5. 10×3.254 = 32.54
6. 67.82×1000 = 67820
7. 13.52×54.2 = 724.3 x = 732.8
8. 34.8×1.7 = 54.7 x = 59.16

Exercise 4 : DIVISION

1. 180÷2.5 = 72
2. 3.45÷0.5 = 6.9
3. 43.2÷1.6 = 27
4. 0.032÷0.04 = 1.8 x 0.8
5. 0.0072÷0.8 = 9.03 x 0.009
6. 1000÷0.01 = 100000
7. 82÷10000 = 0.0082 x 0.0008
8. 0.04565÷0.05 = 0.313 x 0.913

Exercise 4 : ORDERING NUMBERS

(*smallest first*)

1. 0.21 , 0.31 , 0.12 , 0.012
   = 0.012, 0.12, 0.21, 0.31
2. 0.67 , 0.0672 , 0.7 , 0.072
   = 0.0672, 0.072, 0.67, 0.7
3. 0.1 , 0.09 , 0.089
   = 0.089, 0.09, 0.1
4. 0.41 , 0.041 , 0.14
   = 0.041, 0.14, 0.41
5. 0.5 , 0.407 , 0.047 , 0.470
   = 0.047, 0.407, 0.470, 0.5
6. 0.04 , 0.004 , 0.4
   = 0.004, 0.04, 0.4
7. 1.57 , 1.568 , 1.5
   = 1.5, 1.568, 1.57
8. 79.082 , 79.1 , 79.01
   = 79.01, 79.082, 79.1

Exercise 5 : NEGATIVE NUMBERS

1. (-6)+(+2) = −4
2. 9+(−8) = 1
3. (+2)−(−4) = −2 x 6
4. (−7)−(+1) = −6 x −8
5. (+7)×(−5) = −30
6. (−1)×(−3) = −3 x 3
7. (−9)÷(+3) = 3 x −3
8. (−12)÷(−3) = 4

Exercise 6 : PERCENTAGES

(*convert into percentages*)

1. $\frac{3}{4}$ = 75 %
2. $\frac{2}{5}$ = 40 %
3. $\frac{1}{2}$ = 50 %
4. $\frac{7}{100}$ 7 %
5. $\frac{17}{50}$ 34 %
6. $\frac{27}{40}$ 40.5 %

Exercise 7 : PERCENTAGES

(*convert into decimals*)

1. 75% = 3/4 x 0.75
2. 12% = 3/25 x 0.2
3. 100% = 1/1 x 1.0
4. 0.4% = 2/5 x 0.04
5. 14.8% = 14.8/100 x 0.148
6. 64 ½ % = 64.5/100 x 0.645

Example Student #1- Amount of Questions in Homework: **60**

Amount of correct answers: 40

Percentage of Correct answers: 67%

Maximum time (60* 25 seconds) = 1500 seconds (25 minutes)

Actual Time taken for Student #1: 625 seconds

Bonus score (1500/ 625) = 2.4

Final Score (67% * 2.4 = 160.8) → 160.8/2 = **80.4% is the final score of Student #1**

Discussion

Sr. Salomon doesn't struggle at all with finding the final score of each student as he can do the calculations mentally. He uses a gridline paper and a pen to calculate everything and he states that in order to mark a whole class' homework he takes around 10 minutes.

As stated there are two major problems. The first one is the waste of time in generating the questions manually by taking into account the syllabus' parameters for each question and also the marking of them to get the final score. The second one is that the students can cheat on the time recorded in order to get a bigger bonus score.

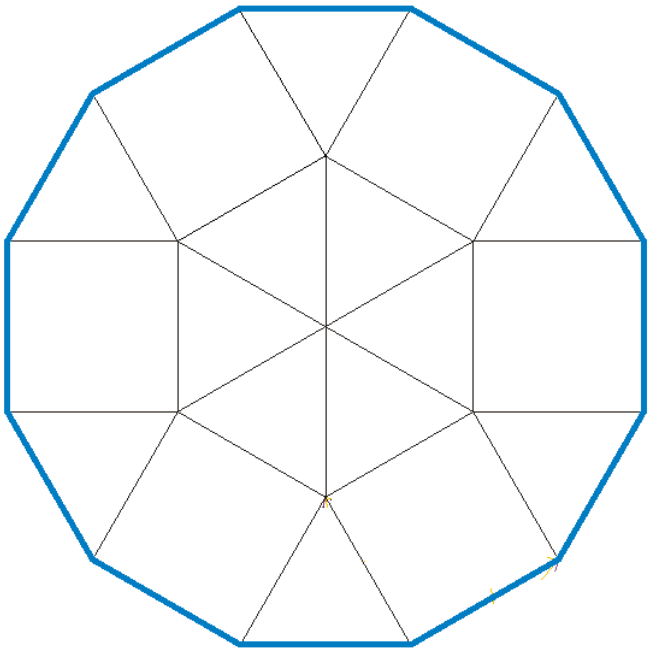
Interactive Game #1

The second method consists of making interactive mathematical games at the end of each class. For example, one game that Sr. Salomon often uses is “The Beep-Plang” game. The rules of the game are that all students begin standing up and gather up in a circle shape, they need to start counting from 0 upwards (in a quick pace) and each time a multiple of 3 comes up the student must say “Beep” instead of the number (eg: when it’s a student turn to say 9 he must say “Beep”). The same happens with multiples of 5 but in this case the student must say “Plang”. Finally when a multiple of 3 and 5 comes up the student must say “Beep-Plang” (like 15). If the student breaks one of the rules he must sit down. The game continues until only one student is left making him the winner.

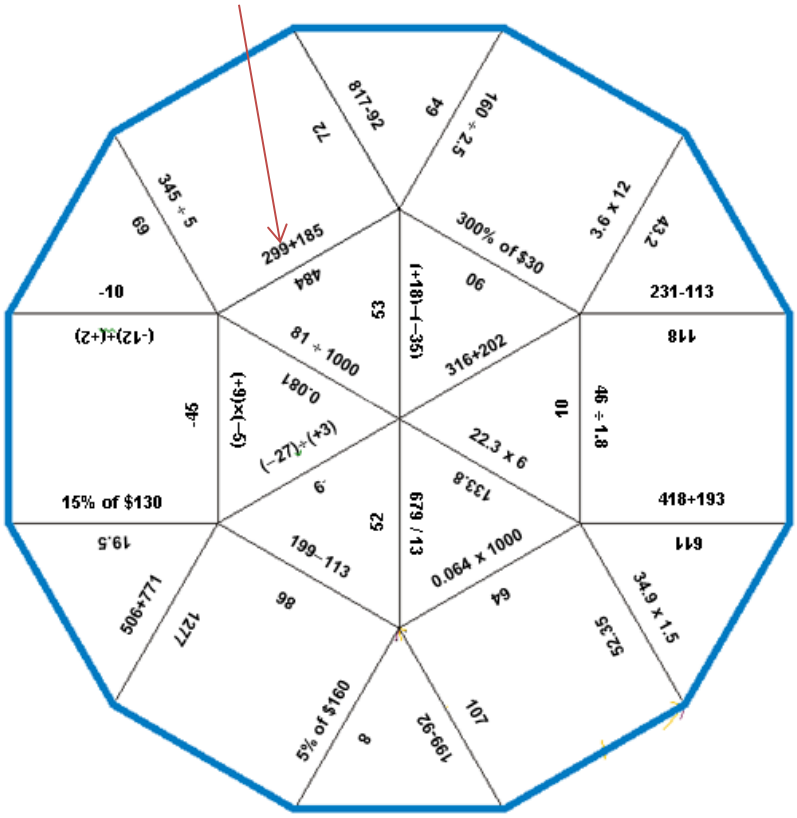
According to Sr. Salomon the problem with this game or others is that some shy students from his class feel a lot of pressure when playing the game; this makes them feel very nervous and disables them from excelling in the course. Furthermore, he reports that “kids with weaker maths are often the ones that sit down first which means that they practice less”. Moreover, “there are kids in his class that don’t take the game very seriously and prefer to lose quickly so they can have a free time in the meantime; these kids tend to distract the others that are still playing the game and Sr. Salomon has to pause everything ruining the whole purpose of the game (quick mathematics).

Interactive game #2

The third method is an interactive game just like the previous example (focuses on mathematical speed) but has a different gameplay. Sr. Salomon uses Microsoft Word in the computer to make a dodecagon (12 sides) and divides it in squares and triangles.

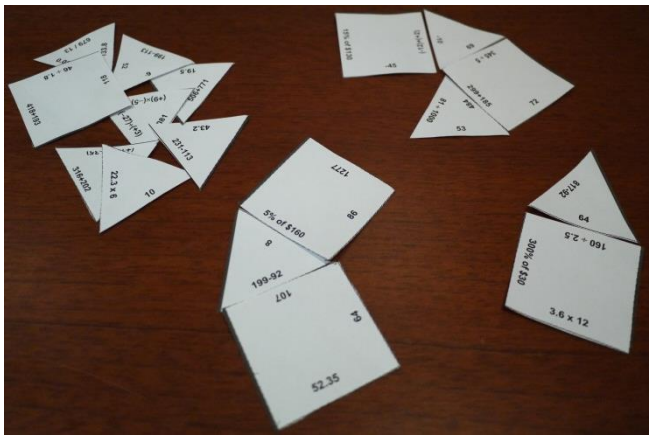


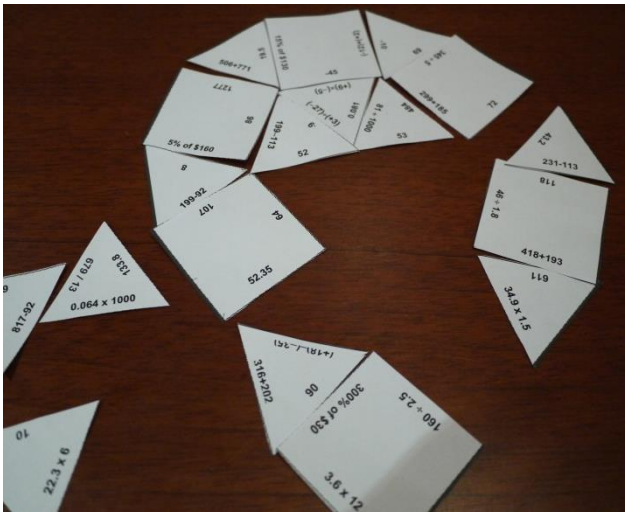
Then he puts math exercises at one of the sides of the triangle or square and puts the answer on the other side of the line. For example (299 +185) and on the other side the answer (484).

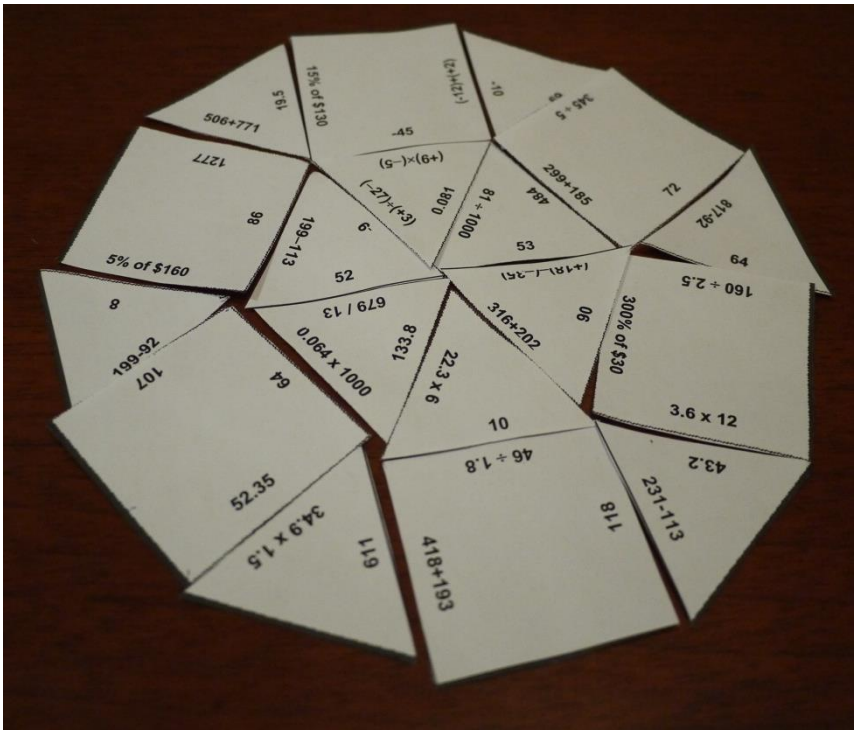


He prints lots of copies and cuts the pieces of squares and triangles and mixes them up. Then, in class, before starting the game, separates the pupils in pairs and gives them the mixed shapes.

The game consists in joining the exercises with the answers and eventually forming the dodecagon. The students are given squared paper to do the working out. The group that finishes first is the winner.







Sr. Salomon states that students really enjoy playing this game but he doesn't do it quite often because producing a new decagon with new exercises is really time consuming. This is also due because, as he recognizes, his knowledge and skill in computing is not very good and finds it difficult to re-edit the dodecagon each time he wants to create a new game. The only disadvantage of the game itself is that students tend to look at the group at the side and copy the answers and shapes because of game competition. By doing this they ruin the whole purpose of the game.

Personal Conclusions:

Sr. Salomon needs the following to solve his problems:

1. Improve the student's mathematical speed in the four operations (addition, subtraction, multiplication, division)
2. A simple system in which the students can log and do the math games with the ability to save and update the records.
3. A high score system that compares all the scores of the students in class
4. Track the students improvement
5. An automatic question generator
6. An automatic correcting system
7. An admin interface for Sr. Salomon

Opinion of user

When I told Sr. Salomon my personal conclusions on what I thought was needed to solve the general problem he was satisfied with them. He made a further proposal in regard to a variation of different games so the students may vary and won't get bored with the same game over and over. At the start the proposal was to do a game to record the time it took for the students to complete 20 math problems. Sr. Salomon proposed two more games

Game 2

-The amount of correct answers the student can achieve in 60 seconds.

Game 3

- Awarded initial 60 seconds and as the questions are answered correctly the student gets an extra bonus time (10 seconds more), if all answers are correct the student can play without limit of time.

My opinion of Sr. Salomon's proposal

I found the proposal to be an excellent idea and it will be implemented into the system as it helps to solve the solution better because a greater variety of games means more fun and more competition between the students.

A2 – Criteria for Success

A computer based solution for the students must allow the following functions to be achieved:

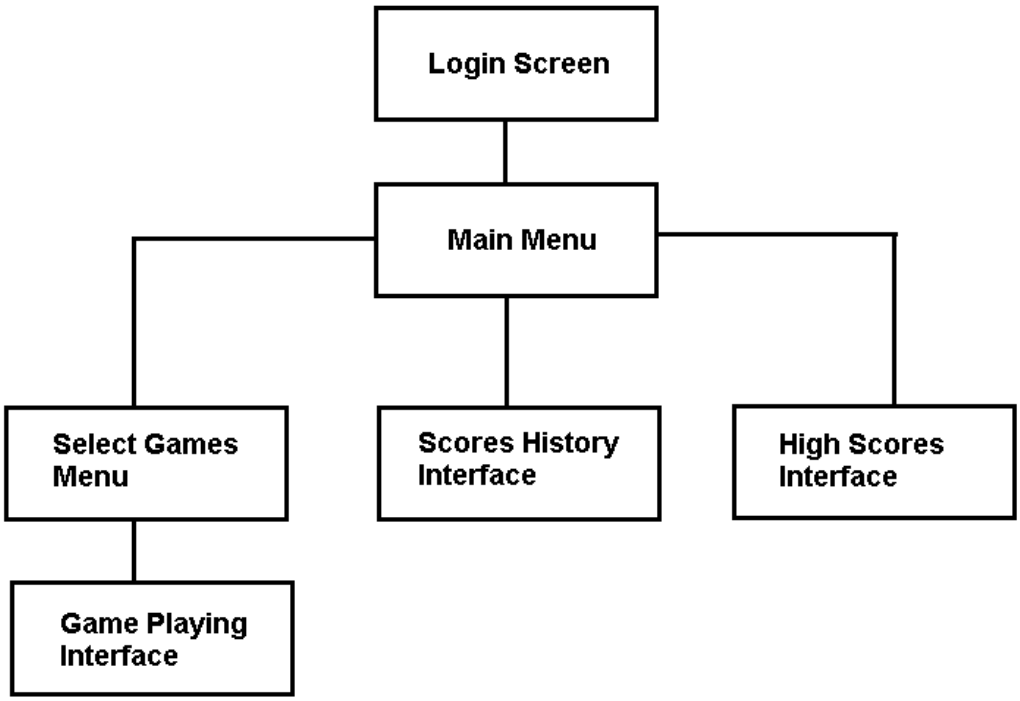
Objective # | Description

1. User Log In interface with username and password provided by Sr. Salomon
    a. Each student has a record with their full names and a unique ID (so students with a same name and last name are not confused)
    b. The record is saved in file. The students may login to the app with their ID and password.
2. An automatic question generator
    a. Short Mathematical questions must be automatically generated in relation to what the p6 syllabus specifies (using the original parameters of Sr. Salomon – Check A1 → Homework → "Generating the Questions" for more information)
    b. The questions are unlimited and constant repetition of the same question will be avoided.
3. A mathematical game for the students to practice quick math questions , three type of games presented
    a. (Game 1) The time it takes for the student to complete 20 questions.
    b. (Game 2) The amount of correct answers the student can achieve in 60 seconds.
    c. Game (3) awarded initial 60 seconds and as the questions are answered correctly the student gets an extra bonus time (10 seconds more), if all answers are correct the student can play without limit of time.
4. An automatic correcting system
    a. As each question is answered the system checks if the answer is correct or incorrect
5. A score generator
    a. For game 1 the student's score will be calculated using the amount of answers correct and the time taken (with the original process Sr. Salomon uses to find the final score of the homework – Check A1 → Homework → "Generating the Score" for more information)
    b. For game 2 the student's score will be the amount of correct answers made in the 60 seconds.
    c. For game 3 the student's score will be the amount of seconds the student survived playing the game
6. A high score list
    a. All the scores of the students are saved in a file. The high scores are generated by selecting the top scores on each game.
7. A login for Sr. Salomon with access to an Admin Interface

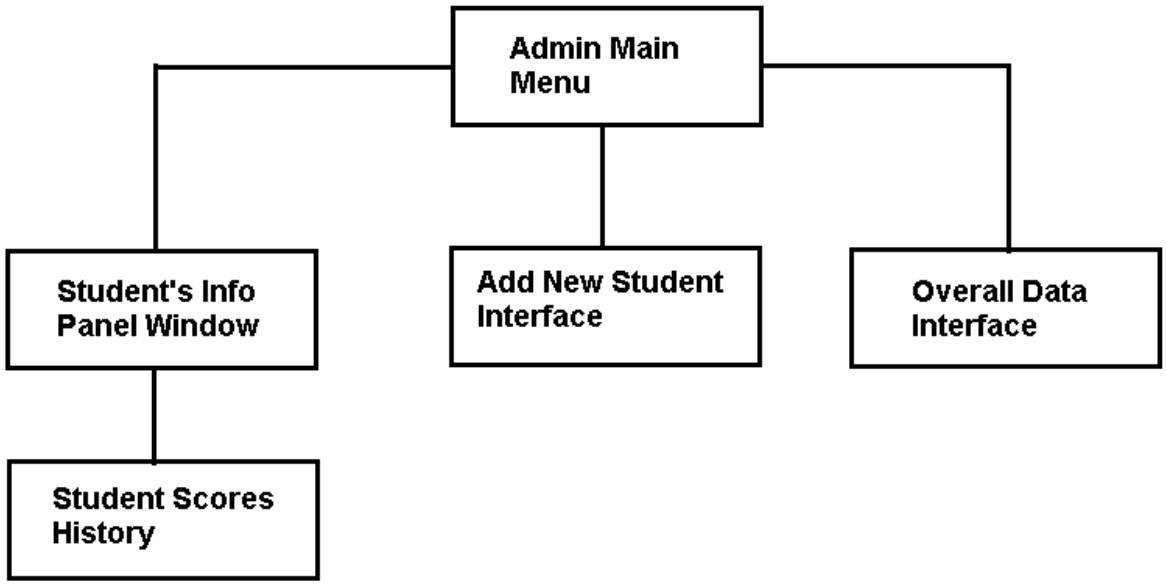
a. He must be able to check if the students have played a game (and when – date and time), see their scores and percentage improvement in relation to the amount of times they have played the game.
b. He may create a new record for a new student
c. The id's are automatically generated by the program in order to avoid the admin giving the same id to students.
d. He may see the class's high scores for each game (in order to award the top students).

A3 Prototypes

Student's application Initial Design



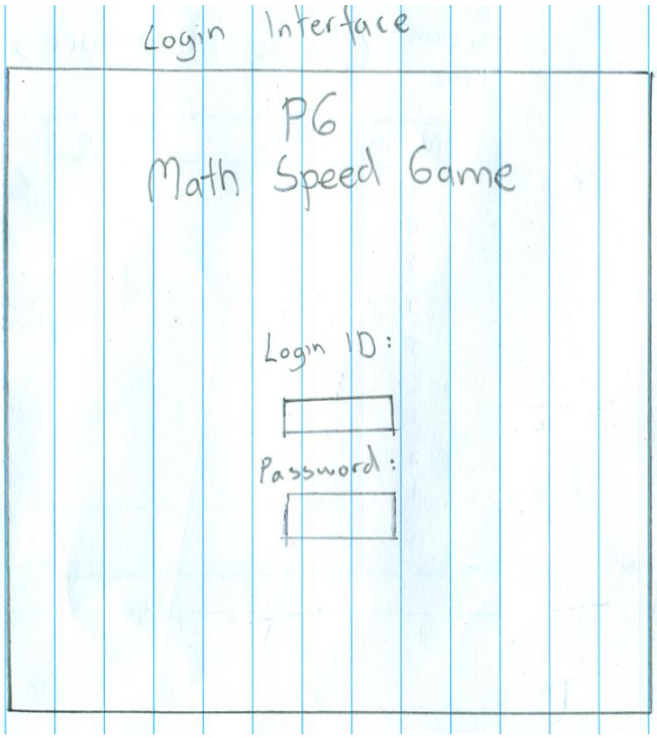
Admin's application Initial Design



The prototypes were developed using paper and pencil. They were created so the user could visualize how the program could look when it's ready; by doing this the user is able to decide if this prototype fits well for the solution and also to see if the user and the developer (me) have a similar idea/picture of the solution. After showing the prototypes a discussion was engaged with the user to see if it suits the needs of the solution and also to ask for any future recommendations so the final constructed program is exactly what the user wants.

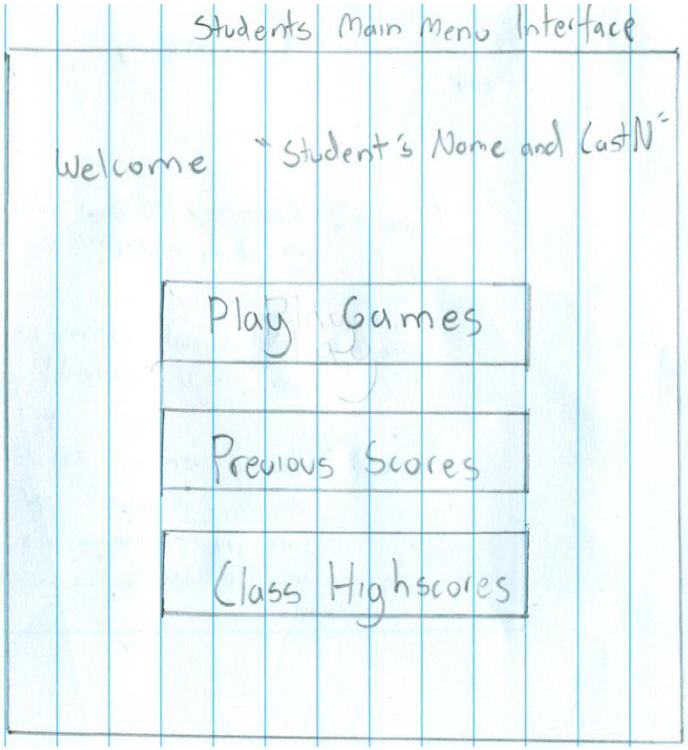
Student's Login Interface

The Student's login interface is quite simple; this is where the student's input the Login ID and password provided by Sr. Salomon. The user approved it and agreed with the idea that the login interface needs to be very simple and straightforward because the p6 students are still children and they may get confused with the first screen that appears in the program which is not a good idea. The user proposed a friendlier login interface with a nice title and some images so the young pupils get attracted by the program and find it fun and entertaining instead of dull like a normal homework.

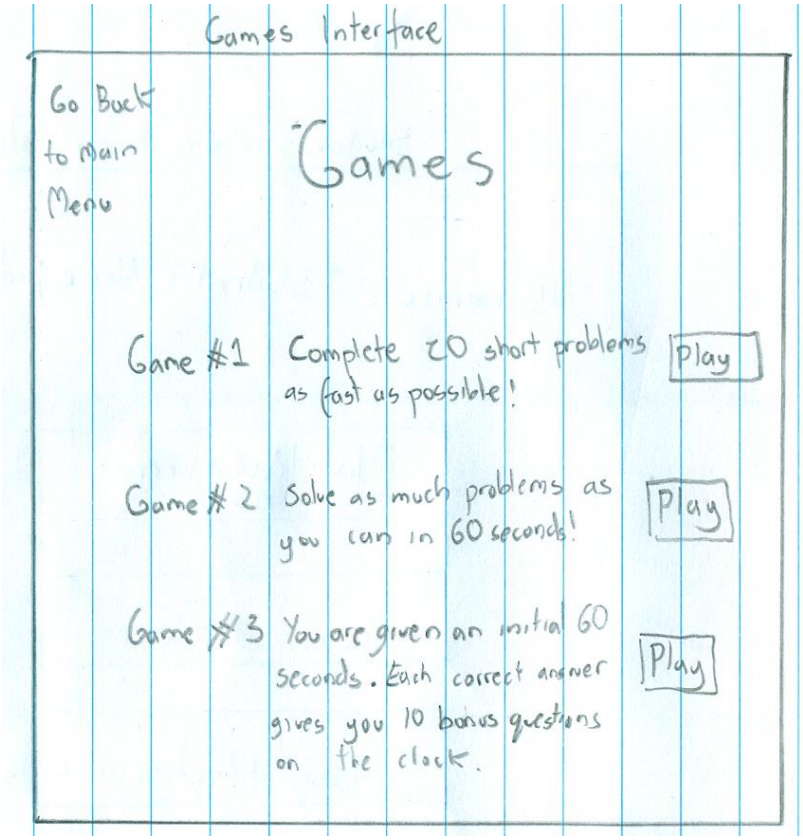


Student's Main Menu Interface

Similarly to the Login Interface the Student's Main Menu is extremely simple. The students are greeted with their name and they are presented with three different options in the forms of buttons. The user approved this Main Menu but suggested the same as in the login interface: a bit friendlier interface with images and maybe some animations. He also suggested for a "Logout Button" which I forgot to add, this way the students can easily logout from the main menu instead of closing the program and rebooting it if another student wants to login in the same computer right after them.

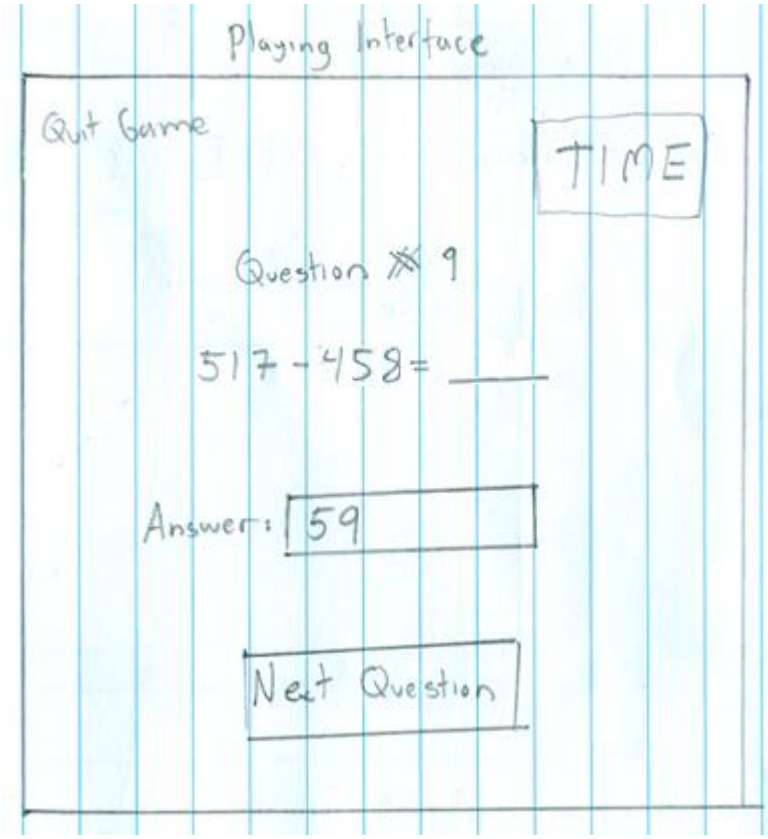


When clicking the “Play Games” button in the Main Menu the student is redirected to the “Select Games Panel” where he/she is presented with the three different games options. There is a small but clear description of each game and the student can choose which game to play by clicking the play button at the side. When the user saw this interface he suggested that it would be better if the students were presented with the description of the different games at the top and in the bottom there would be a combo box where the student can choose between games one, two or three and click a single “Play” button at the side of the combo box. This recommendation was found to be quite useful and will be applied to the final program.



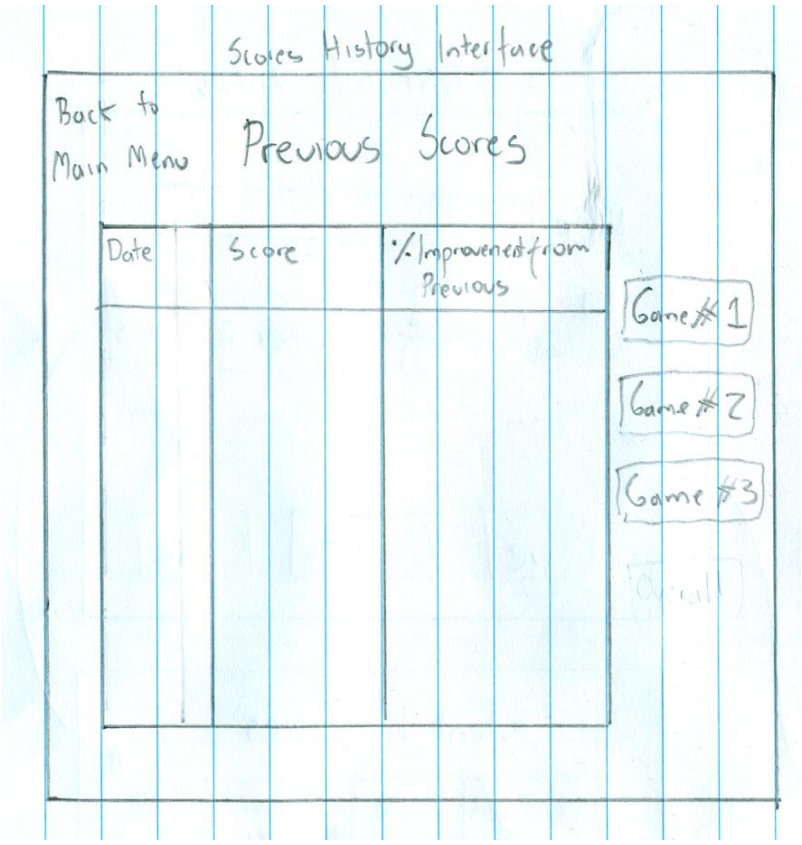
Game Playing Interface

This is the interface where the student’s play the actual game. They are presented with the question and the question number and they must input the answer to the question in the answer box (where only numbers may be inputted – of java type double). The timer is located in the top right, the students will use this as a reference of how much time they have left. When the user was presented with this he approved the simplicity of it but suggested that it is shown exactly what game is the student currently playing (one, two or three) with the description of the game so the students are reminded with the rules of each game at all times. He suggested that the timer should be placed at the bottom right and that it shouldn’t be too big in size so the students don’t get nervous and become less focused with the question with a big timer at the side of each question.



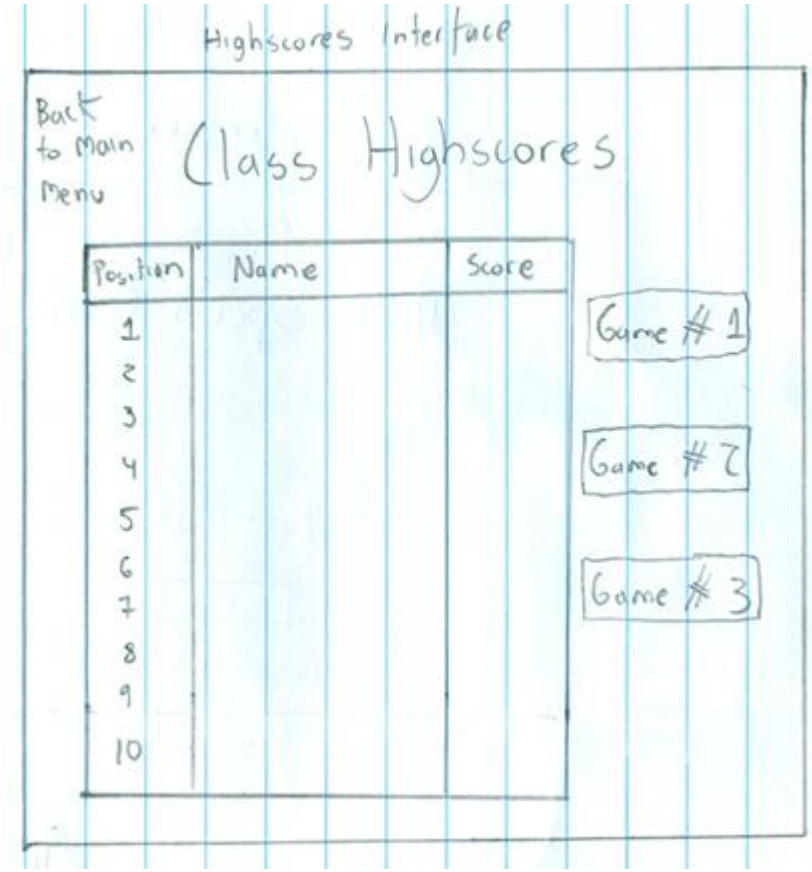
Student Scores History Interface

When the students click the second option in the main menu "Previous Scores" they are redirected to the "Student Scores History Interface". Here they can see their previous scores in each specific game along with the time when they played the game. The students can also see the % percentage Improvement from the previous game played, this way they may see if they are improving in the game or getting worse at it. The user found this interface really great and suggested a really interesting idea. If there is a positive percentage improvement then the record of the game score is coloured with green, otherwise, if there is no improvement the record is coloured red. This idea was found to be very useful and will be implemented to the final program.



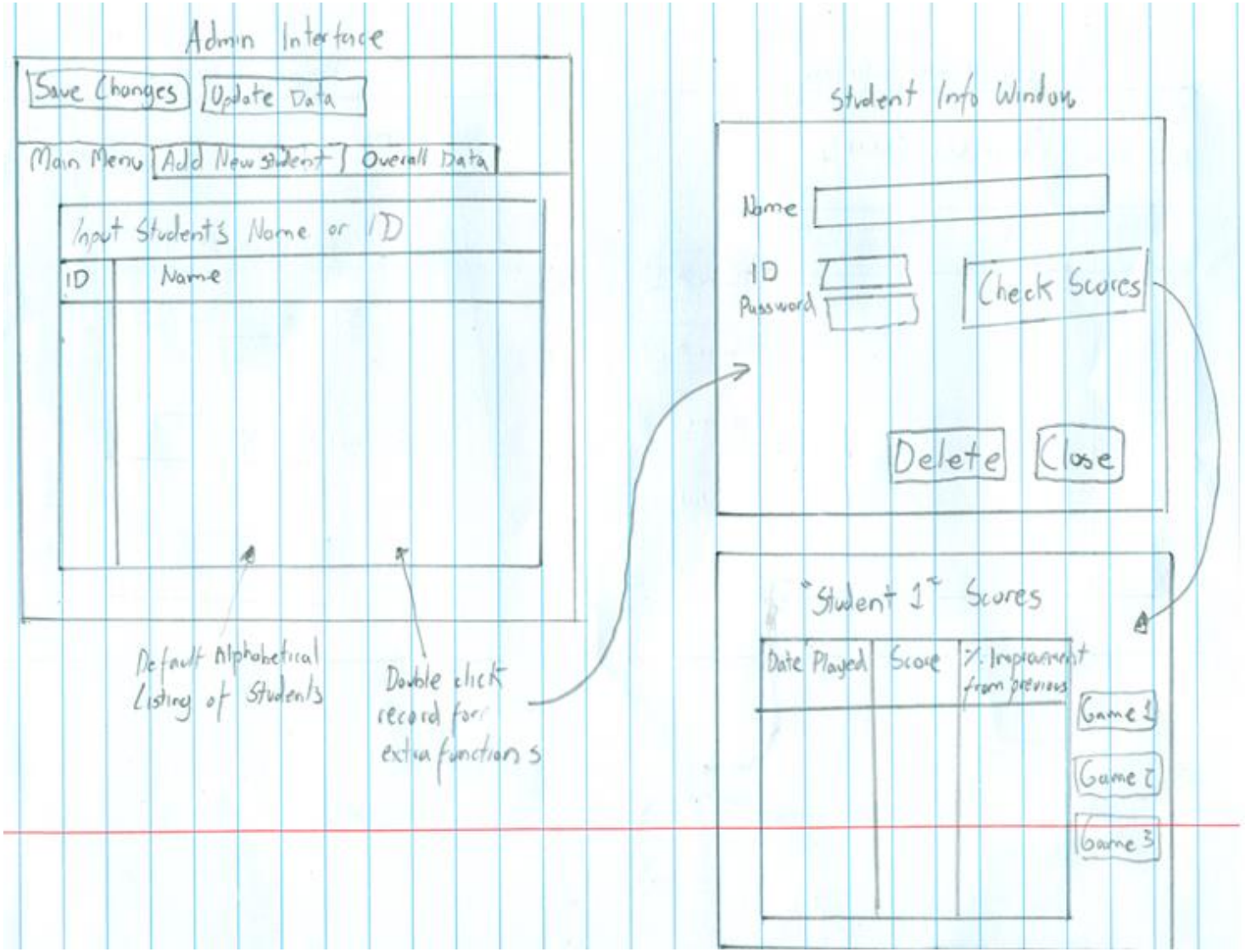
Class High scores Interface

When the student's click the third option in the Main Menu "Class High scores" they are redirected to the "Class High scores Interface". Here they are able to see which students in the class have the best scores for each specific game and finally they are also able to see who has the high scores for the best average score between the three games. The user found this interface really great and his only suggestion was to increase the positions showed from 10 to 15 so at least the half of the class would be on the high scores (there will be more competition among the students to get to the top if more of them are shown in the table). This idea was also found very interesting and will be implemented to the final program.



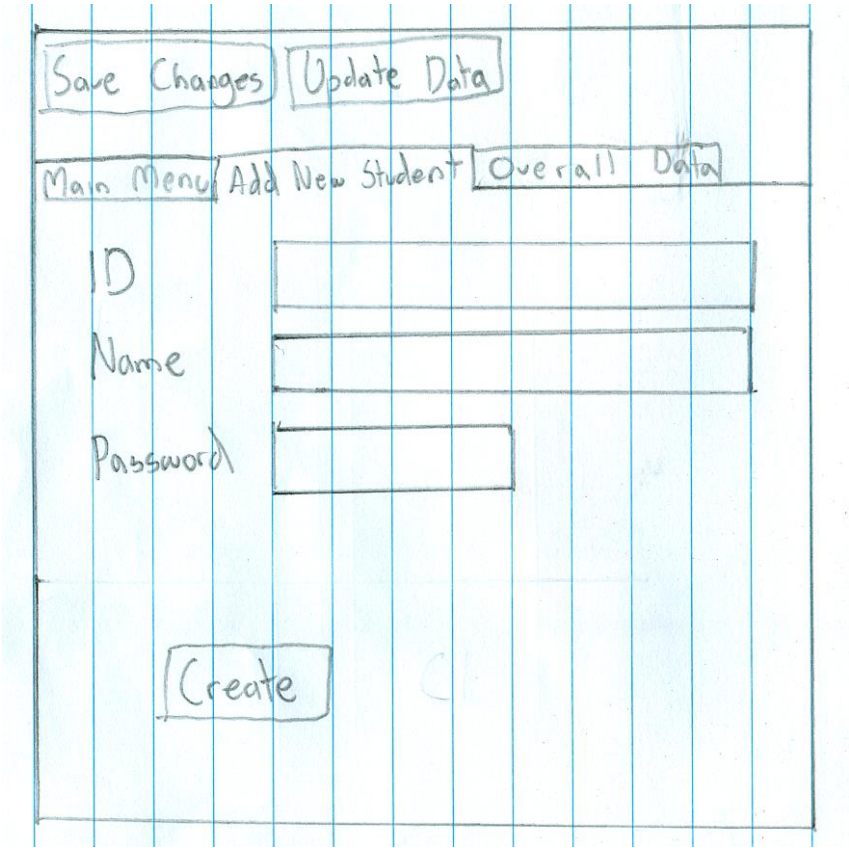
Admin Main Menu Interface

The admin Main Menu interface opens when the user opens the admin program from his computer. In the Main Menu the user can search for a specific student by entering part of their name or their ID in the search box. The results are then shown in the table and he may access the record of a specific student by double clicking the record. When this is done a window pops up with the student info, this window is called “Student Info Window”. Here the user may check the information of the student, delete it or check his previous scores. The previous scores interface is the same as the “Student Scores History Interface” in the student’s program. The user found the Admin interface extremely organized, simple and useful. He liked the idea of searching by name or ID and then opening the record by double clicking. He also liked the idea of separating the different interfaces by tabs. He suggested to take out the save button because it is actually irrelevant (because each time something is changed it is already recorded). He liked the idea of the “Update data button” so he doesn’t have to close and open the program again if he wants to see any changes done. For the “Student Scores History Interface” feedback check previous feedback from the Student’s program.



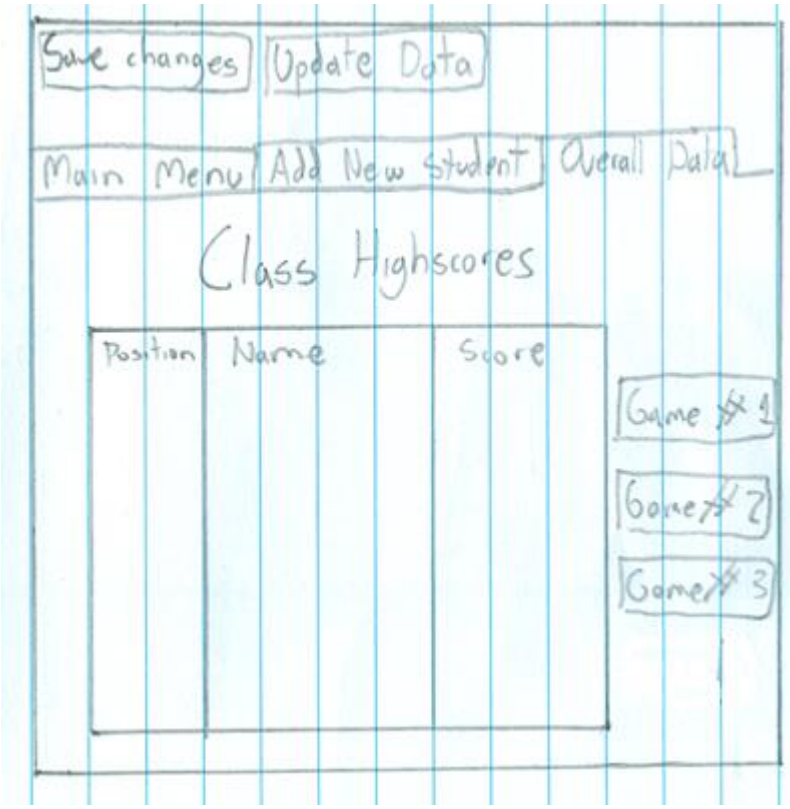
Add New Student Interface

The second tab in the Admin Interface is the “Add New Student Interface”. Here the user may input a new student ID, Name and Password and finally add it to the system. It was explained that the ID input box would only accept numbers (java type int), the Name input box would only accept letters (java type string) and finally the password would accept any alphanumerical value (java type string). The user suggested separating the name between “First Names” and “Last Names” so it would be more organized and there would be no space constraints in this box if the whole name (first name and last name) is too large. This idea was accepted and will be implemented to the program



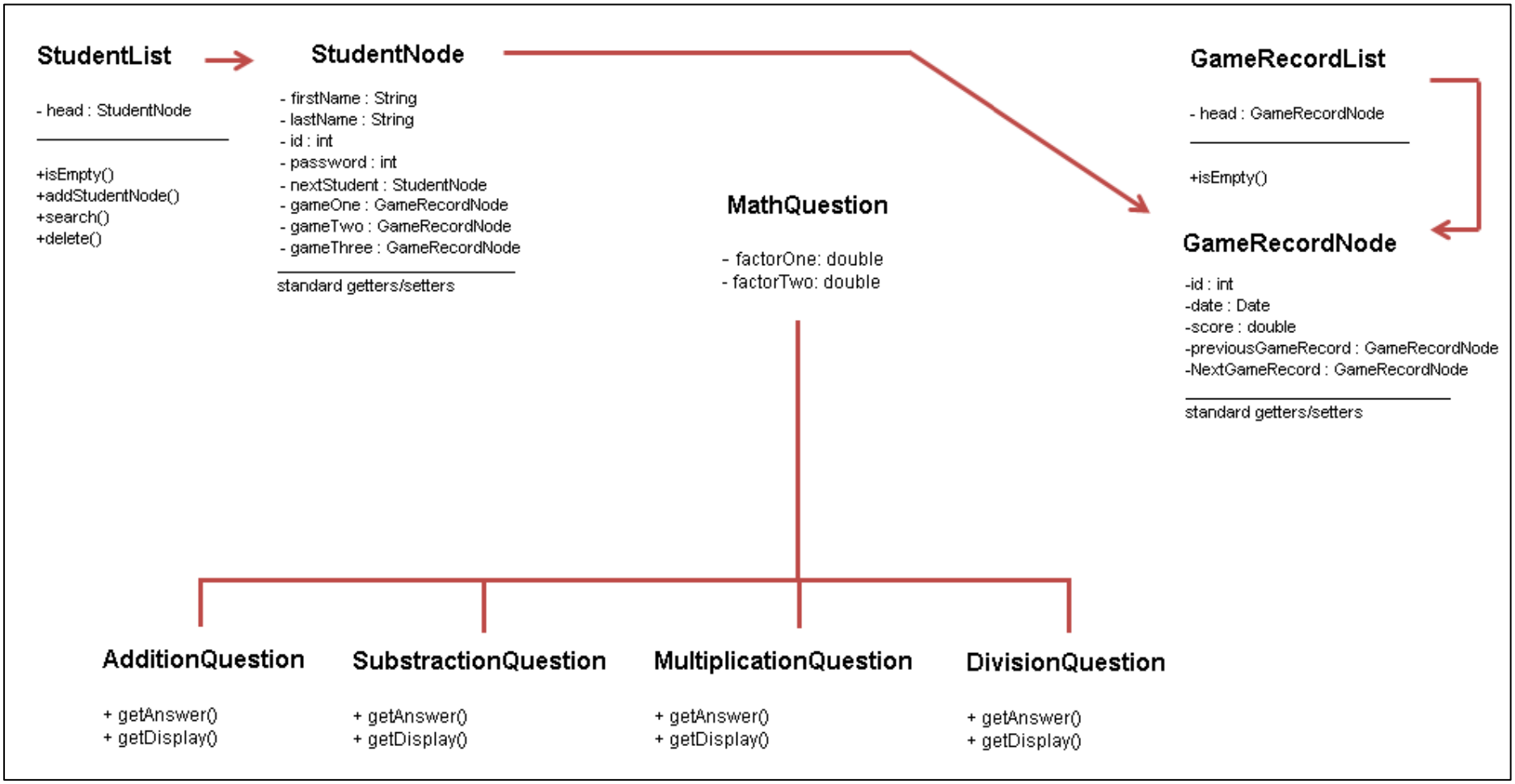
Overall Data Interface

The third tab in the Admin Interface is the “Overall Data Interface”. Here the user is presented with the Class High scores. It is exactly the same interface than the student’s program “Class Highs cores Interface”. For more information about this and user feedback head back to the student program interface and check the “Class Highs scores Interface”



B1 – Data Structures

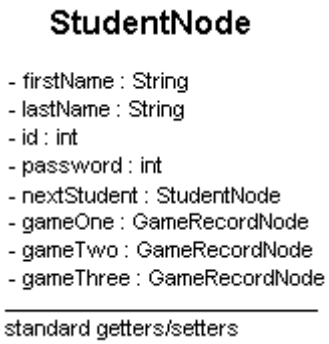
The following diagram shows the java class structure that will be used to solve the problem. Only the important classes (and their methods) are presented. (Getters and setters are implied for all classes)



Below, all the classes are described along with their variables and methods.

StudentNode class

The StudentNode class is a Student object in the Student List (of objects) which holds all the information about the identity of the student. It is also connected to the GameRecordNode class (will be described afterwards) which holds all the records of games played by the students, thanks to this connection the StudentNode class is able to hold aswell all the records of the games played by each specific student.



Variable Name	Description	Type	Sample
firstName	First Name of the Student	String	Gonzalo
lastName	Last Name of the Student	String	Rojas
id	3 digit number id of the student	String	001
password	4 digit Password of the student	String	9u7p
nextStudent	Pointer of the StudentList (see below). Points to the next student on the list	StudentNode	Check StudentNode variables
gameOne	Holds all the records of the games of type one for an specific student	GameRecordNode	Check GameRecordNode variables
gameTwo	Holds all the records of the games of type two for an specific student	GameRecordNode	Check GameRecordNode variables
gameThree	Holds all the records of the games of type three for an specific student	GameRecordNode	Check GameRecordNode variables

StudentList class

This class creates the list of StudentNodes by recording the head variable (the first student in the list). It also has the methods for creating, deleting or searching students.

StudentList

- head : StudentNode

+isEmpty()
+addStudentNode()
+search()
+delete()

Variable Name	Description	Type	Sample
head	This is the head (first object) of the StudentList (Abstract Data Structure - it is described on depth afterwards)	StudentNode	See StudentNode (above) class for variables

GameRecordNode class

This GameRecordNode class is the object in the Game Record list which holds all the records of games played by the students (each time a game is played all the information about it →(the ID of the student that played it, the score obtained, the game played, etc) is recorded in the GameRecordNode object).

GameRecordNode

-id : int
-date : Date
-score : double
-previousGameRecord : GameRecordNode
-NextGameRecord : GameRecordNode

---

standard getters/setters

Variable Name	Description	Type	Sample
id	The ID of the student that played the game	String	001
date	The number of milliseconds that have passed the January the 1st of 1970 00:00:00 GMT.	Date	1335452070097 (26th April 9:56 am 2012)
score	The percentage score obtained in Game 1 or Game 2 OR the total survival time for Game 3	int	87
previousGameRecord	Pointer of the GameRecordList (see below), points to the previous record in the GameRecordList	StudentNode	Check StudentNode variables
nextGameRecord	Pointer of the GameRecordList (see below), points to the next record in the GameRecordList	GameRecordNode	Check GameRecordNode variables

GameRecordList class

This class creates the double linked list of GameRecordNode by recording the head variable (the first record in the list).

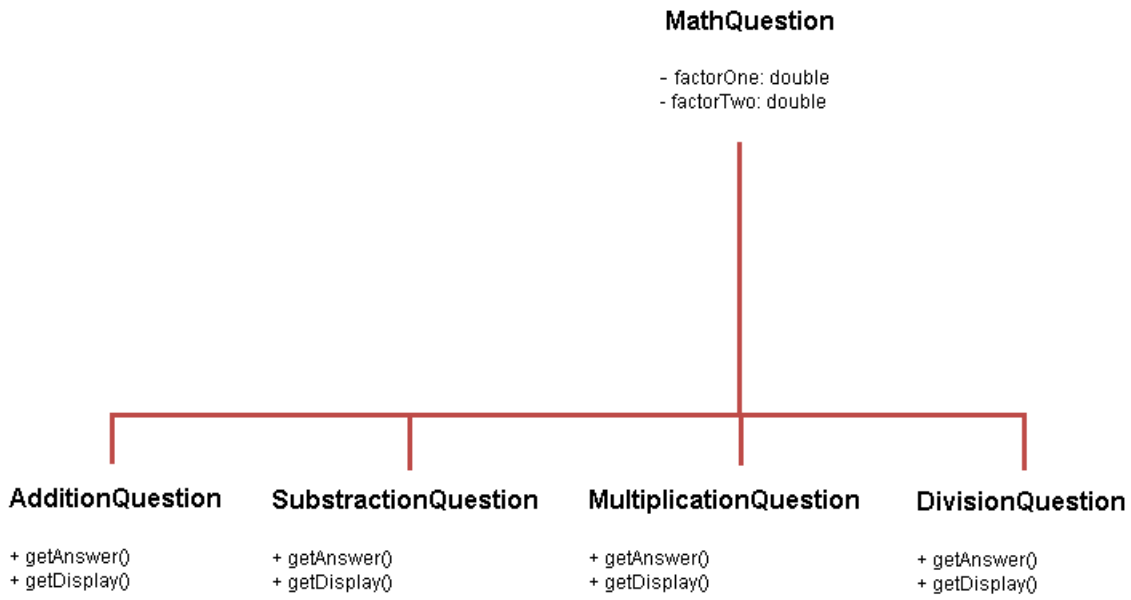
Variable Name	Description	Type	Sample
head	This is the head (first object) of the GameRecordNode double linked list (Abstract Data Structure - it is described on depth afterwards)	StudentNode	See GameRecordNode (above) class for variables

MathQuestion class

This superclass is in charge of generating a mathematical question.It holds the information relating to the question. This superclass is created because in order to generate the questions it is intended to use an array which generates numerous different objects of the MathQuestion subclasses.

Eg : Question [] qs = new MathQuestion [20]

As you can see in the diagram below the MathQuestion superclass has 4 subclasses (AdditionQuestion, SubstractionQuestion, MultiplicationQuestion and DivisionQuestion). Depending on chance it will generate a question based on the rules of the subclass.



Variable Name	Description	Type	Sample
factorOne	This is the first operand in the question.	double	107.5
factorTwo	This is the second operand in the question.	double	79.5

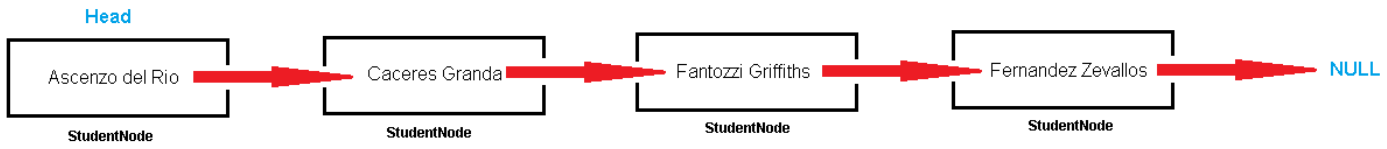
Abstract Data Types

The Student List

There are around 30 students in Sr. Salomon's class. The student's record objects may be stored in a static data type such as an array but because in several occasions all the student's records need to be outputted in Lastname alphabetical order to a table in the GUI interface and also since each bimester there are students who go up and down the math sets (therefore changing the amount of students in Sr. Salomon's class) we would need to change the array's size each bimester. Because of these reasons a more flexible data type is needed.

A linked list was chosen as the appropriate data type to store the student record objects. The student list needs to be ordered in alphabetical order according to the student's last names. Following we have a graphical representation of how the StudentList would look. For further information about the variables in the StudentList class and the StudentNode class check the previous section.

Graphical Representation of StudentList ADT

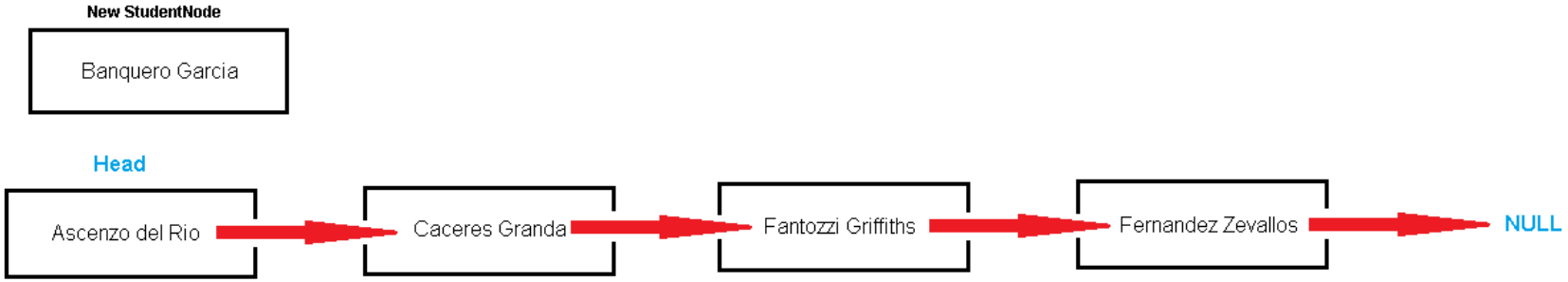


As we can see we have a head StudentNode (the first student in alphabetical order) and there is a pointer in each StudentNode that points to the next student in the StudentNode list. The last student in the Student's list points to "null".

As it has been explained before the students will be added in alphabetical order. Following in the next page we have a Graphical Representation of how a new StudentNode is added to the StudentList in alphabetical order.

Adding a StudentNode to the StudentList in alphabetical order

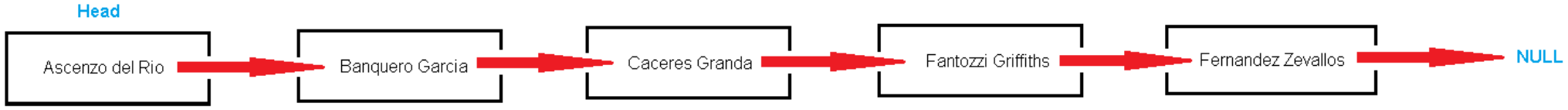
1. We have the original StudentList of StudentNodes and we have the new StudentNode that we want to add.



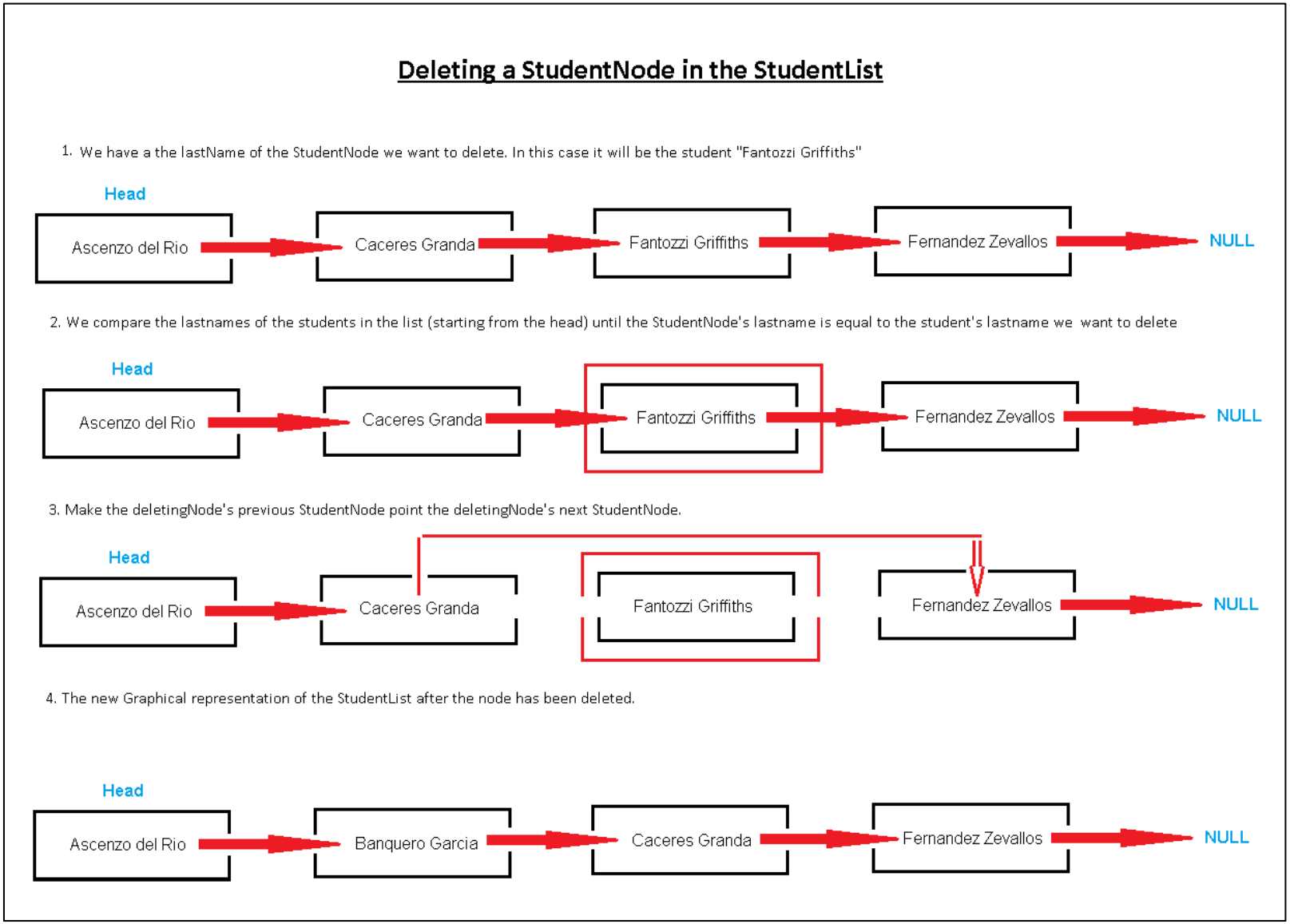
2. Compare the lastnames of the students in the list (starting from the head) until it's correct alphabetical position is found (with the java compareTo method.

1. Banquero Garcia.compareTo (Ascenzo del Rio) --> 1 (This means that "Banquero Garcia" has greater String value than "Ascenzo del Rio". Check the next student.
2. Banquero Garcia.compareTo(Caceres Granda) --> -1 (This means that "Banquero Garcia" has a lower String value than "Caceres Granda". This means that this will be the position of the new StudentNode.
3. Make the StudentNode Banquero Garcia point the StudentNode Caceres Granda.
4. Make the StudentNode Ascenzo del Rio point the StudentNode Banquero Garcia.

3. The new graphical representation of the StudentList after the new node has been added.



In order to delete a StudentNode from the StudentList we have also created a graphical representation to show a more friendly and accurate description.

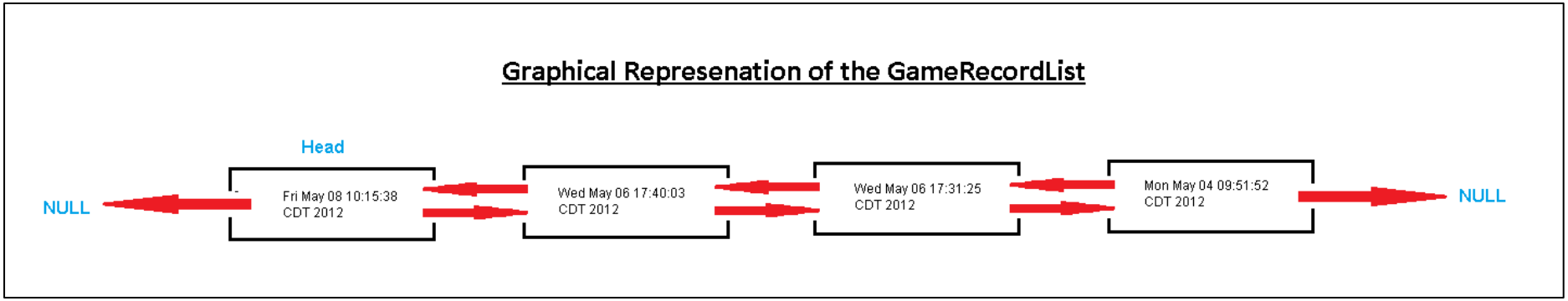


The Game Records

There will be an infinite amount of game records because we don't know how many times the students will play the different games. As we don't know the size of the structure needed it would be irrational to use a static data type such as an array (we would need to set a very big size to ensure it's size wont be exceeded, but this would be a very big waste of space for memory). Because of this reason it would be understandable to say we are also going to use a linked list for the game records but one of the main objectives of the program is to be able to compare the previous game score with the new game played by the student and calculate a percentage improvement. Because of this reason a Double Linked list is needed because we need an ADT that is able to check the previous GameRecordNode as well as the next GameRecordNode.

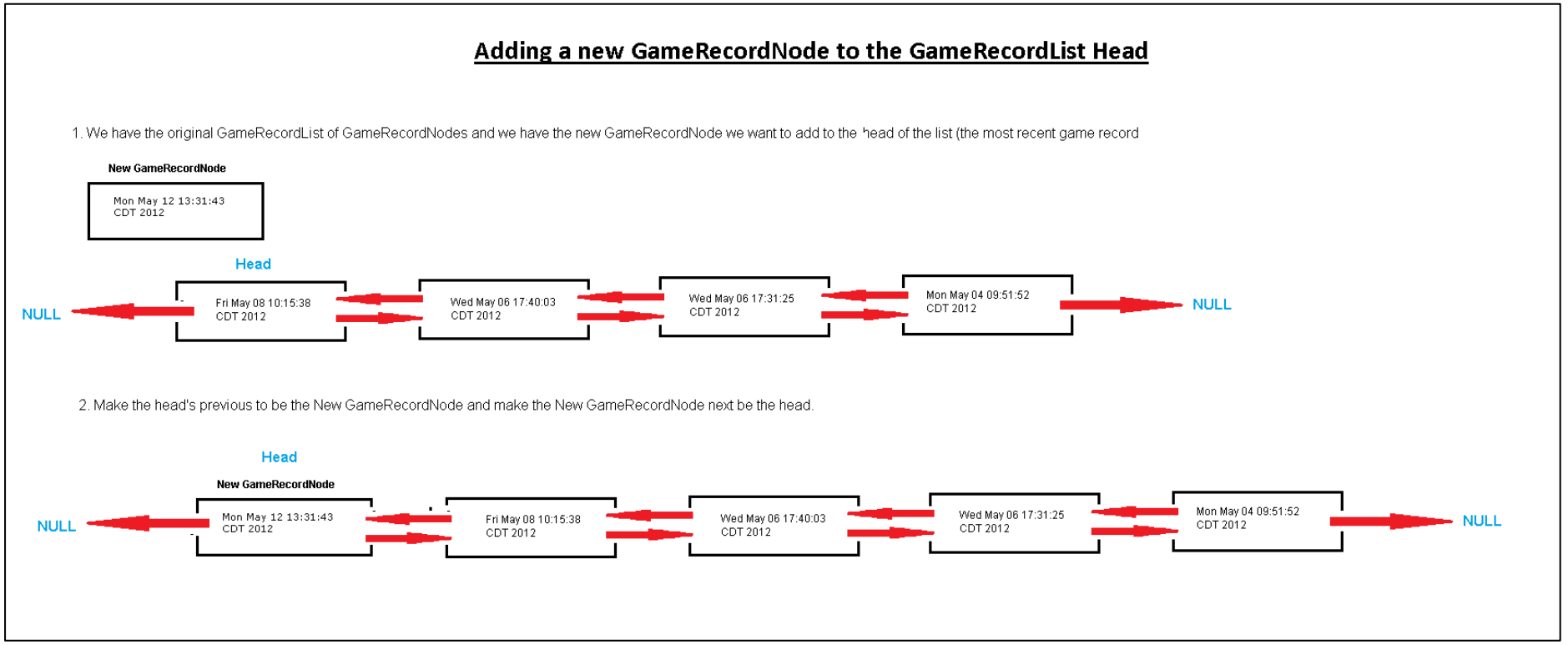
The double link list needs to be ordered in date order, this means that the newest game record will be the head of the list and the oldest game record will be the tail of the list. Because of this reason the GameRecordList will use an addToTail method

Following we have a graphical representation of how the GameRecordList would look. For further information about the variables in the GameRecordList class and the GameRecordNode class check the previous section.



As we can see we have a head GameRecordNode (the lastgame played) and every GameRecordNode has a double pointer (one pointing to the next GameRecordNode and another pointing to the previous GameRecordNode).

As it has been explained before the Game records will be added in date order (this means all the new games go to the head of the GameRecordList). Following in the next page we have a Graphical Representation of how a new GameRecordNode is added to the GameRecordList in date order.



There will be no delete function for the GameRecordNodes because there is no reason to delete a game record.

Files

The data stored in the StudentList and the GameRecordList must be stored in secondary memory because otherwise all the data would be erased as soon as the program is closed. In this section we will design and show the way in which the students and game records will be stored in files.

The Student Records

As stated before, there are 30 students in Sr. Salomon's class. The student records will be stored in a random access file to ensure an extremely fast processing. They will be ordered in ascending order by their ID.

Field Name	Description	Type	Bytes	Sample
firstName	First Name of the Student (Maximum Character Length = 20)	String	40	Gonzalo
lastName	Last Name of the Student (Maximum Character Length 25)	String	40	Rojas Buendía
id	3 digit number id of the student (Maximum Character Length 3)	String	6	001
password	4 digit Password of the student (Maximum Character Length 4)	String	8	9u7p

The total record length for each student is 94 bytes.

The students will be searched using hashing. This will be further explained in the Algorithms section (B2).

The Game Records

The game records will be stored in three different files. These are GameOne.txt (where all the records of Game one will be storred), GameTwo.txt (where all the records of game two will be stored) and finally GameThree.txt (where all the records of game three will be stored). These files will be sequential and they will be ordered in descending order by date. The new game records will be added at the top of the list. This means that at the top of the list we find the newest record and at the bottom of the list we find the oldest record. The records in these files will also be pipe delimited "|".

Field Name	Description	Type	Sample
id	The ID of the student that played the game	String	001
date	The number of milliseconds that have passed the January the 1st of 1970 00:00:00 GMT.	Date	1335452070097 (26th April 9:56 am 2012)
score	The percentage score obtained in Game 1 or Game 2 OR the total survival time for Game 3	int	87

This will be the format of the records:

Id| score|date

Sample Data for GameOne.txt:

001| 65| 1335452070097

003| 87| 1335700123128

Sample Data for GameTwo.txt:

021| 66| 1335452070097

017| 87| 1335700123128

Sample Data for GameThree.txt:

011| 121| 1335452070097

007| 89| 1335700123128

Note: As you can see the score in GameThree exceeds 100, this is because in GameThree records the score is not a percentage (as said before) it is the amount of seconds survived playing the game.

B2 – Algorithms

The following main algorithms will be needed for the system. Comments are highlited in green.

1. **generateQuestions** method

The following algorithm fills the array of MathQuestion objects (the questions presented in the games). The type of questions is decided by chance, a random number from 0 to 3 is generated, the number 0 produces an addition question, 1 produces a substraction question, 2 produces a multiplication question and 3 produces a division question.

```
public void generateQuestions(){
        int randomNumber = 0;
        for (int i = 0; i < 20; i++){ //it loops around 20 times because there are 20 questions for the games
                randomNumber = (int)(Math.random()*4);//random number from 0 to 3 is generated (an int rounds down)
                        if (randomNumber == 0){
                                question [i] = new AdditionQuestion(); // addition question generated
                        } else if (randomNumber == 1){
                                question [i] = new SubstractionQuestion(); // substraction question generated
                        } else if (randomNumber == 2){
                                question [i] = new MultiplicationQuestion(); //  multiplication question generated
                        } else {
                                question [i] = new DivisionQuestion(); // division question generated
                        }
                }
        }
```

2. **gameOneScoreGenerator** method

The following algorithm will be used to calulcate the score for game one, it is base don the time taken to answer the 20 questions and the amount of correct answered questions (the input parameters). Refer to section A for the calculation of the score based on time and correct answers.

```
public void gameOneScoreGenerator(int correctAnswers, int time){

    int percentageCorrectAnswers = correctAnswers*5; // the amount of correct answers is over 20, we multiply it by 5 to get the percentage of correct answers

    double maximumTime = 500;// the maximum time a student should take to complete game one
    double actualTimeTaken = time; // the actual time taken by the student to complete game one
    double bonusScore = (double)(maximumTime/actualTimeTaken); // the bonus (based on time) added to the final score
    double finalScore = (int)(((percentageCorrectAnswers*bonusScore)/2)); // the calculation of the final score

    if (finalScore > 100){
        finalScore = 100; // if the bonus is really big the score may overpass 100 percent, if this happens the score stays in 100
    }
}
```

3. **getSize** method for StudentList and GameRecordList
   This method counts the number of nodes in the list starting from the head. It returns a int value (the amount of nodes in the list.

```
public int getSize (){

    Node current = head; //sets the head as the current node
    int count = 0; //count is set to 0
    while (current != null){ //while the list is not over
        count = count +1; //increase count by 1
        current = current.getNext(); //go to next node on the list
        }
    return count; //return the amount of nodes in the list
    }
```

4. Adding a GameRecordNode to the head of the GameRecordList, **addNodeToHead** method

   This method adds a new GameRecordNode (input as a parameter), to the head of the list.

```
public void addNodeToHead(GameRecordNode node) {
        if (isEmpty()){//if list is empty
                head = node; //if the list is empty then the new node becomes the head
        } else {
                node.setNext(head); // otherwise the head is set as the next of the new node
                head = node; // the new node is set as the new head
                }
        }
```

5. Creating an ordered GameRecordList (from high scores to low scores) in order to set the high scores of each game. The **addNodeInOrder** method

In the following algorithm a new GameRecordNode is being added to the list (as a paremeter). An ordered list (by score from high to low) is generated.

```
public void addNodeInOrder(GameRecordNode node) {

        if (isEmpty()){ //if list is empty set new node as head
                head = node;
        } else{
                GameRecordNode current = head; //sets the head as the node that will be compared
                with the new one
                boolean added = false;
                //check if should go before head
                //parses scores to int so they can be numerically compared
                double nodeScore = Double.parseDouble(node.getScore());
                double currentScore = Double.parseDouble(current.getScore());

                if (nodeScore > currentScore){ //if the new node is bigger than the head then it is set as
                the new head
                        node.setNext(head);
                        head = node;
                        added = true;
                }

                if(!added){//if node hasn't been added yet

                        while (current.getNext() != null){//while the list hasn't finished

                                double nodeScoreTwo =
                                Double.parseDouble(node.getScore());
                                double currentNextScore =
Double.parseDouble(current.getNext().getScore());

                                if (nodeScoreTwo > currentNextScore){ //if the new node score is
                                bigger than the current one then it is set before the current one
                                        node.setNext(current.getNext());
```

```
                                current.setNext(node);
                                added = true;
                                break;

                        }
                        current = current.getNext(); //the next node is checked
                }
                        if (!added){ //if it hasn't been added yet it means that it is the lower score in the last, it places it in the end (tail)
                        current.setNext(node);
                        added = true;
                }
        }
    }
}
```

6. Getting the previous scores for a student. The **getGameRecords** method.

This method uses the ID of the student and game number (as parameters) to search for the specific records needed to retrieve the previous scores of the student for a specific game. This method returnsan ordered GameRecordList (from oldest game records to newest ones as they are stored in the sequential file ordererdly) holding the previous game records of a specific student for a specific game.

```
public GameRecordList getGameRecords(String id, int gameNumber) {


    if (gameNumber == 1){
                gameRecordsFile = new File("GameOneRecords.txt");// check game one records stored in sequential file GameOneRecords.txt
    } else if (gameNumber == 2){
                gameRecordsFile = new File("GameTwoRecords.txt");//check game two records stored in sequential file GameTwoRecords.txt
    } else {
                gameRecordsFile = new File("GameThreeRecords.txt");//check game three records stored in sequential file GameThreeRecords.txt
    }


        FileReader read = new FileReader(gameRecordsFile);
                BufferedReader buffReader = new BufferedReader(read);//buffered reader used to read the sequential file characters


        boolean eof = false;
```

```
                while (!eof){
                        String line = buffReader.readLine();
                        if (line == null){
                                //end of file reached
                                eof = true;//quits reading file
                        } else {
                                StringTokenizer tokcheckid = new StringTokenizer(line, "|");
                                if (id.equals(tokcheckid.nextToken())) {
//if the ID of the student is equal to the ID in the game record then retrieve the game record
                                        StringTokenizer tok = new StringTokenizer(line, "|");
                                        String idRecord = tok.nextToken();
                                        long date = Long.parseLong(tok.nextToken());
                                        String score = tok.nextToken();
//creates a GameRecordNode object with the data in the stored game record
                                        GameRecordNode node = new
GameRecordNode(idRecord, date, score);
                                        list.addNodeToHead(node);
//then the GameRecordNode object is added to the head of the list
                                }
                        }
                }
                return list;//a complete list with the previous records of the student for a specific game is
returned
        }
```

7. Getting the percentage improvement from the previous score to the current one. The **getPercentageImprovement** method.

The following algorithm returns a double which holds the value of the percentage improvement from a previous game to a current one. This is an aspect of the program displayed in the previous scores section, here the student can see if he has improved his scores from game to game.

```
public double getPercentageImprovement(){

        GameRecordNode temp = list.getHead();//the first game played by the student

        String previousscore = temp.getScore();//gets the previous score
        double doublepreviousscore = Double.parseDouble(previousscore);

        String currentscore = temp.getNext().getScore();//gets the current score
        double doublecurrentscore = Double.parseDouble(currentscore);

        double difference = doublepreviousscore - doublecurrentscore;//calculates the
difference

        double percentageimprovement = (difference/doublecurrentscore)*100.0;//calculates
the percentage improvement

        return percentageimprovement;//returns the percentage improvement

}
```

8. Students' login to the application. The **login** method.

This method uses the ID and password provided by the student in the login interface as parameters to check if they match with a record in the random access file StudentRecords.dat.

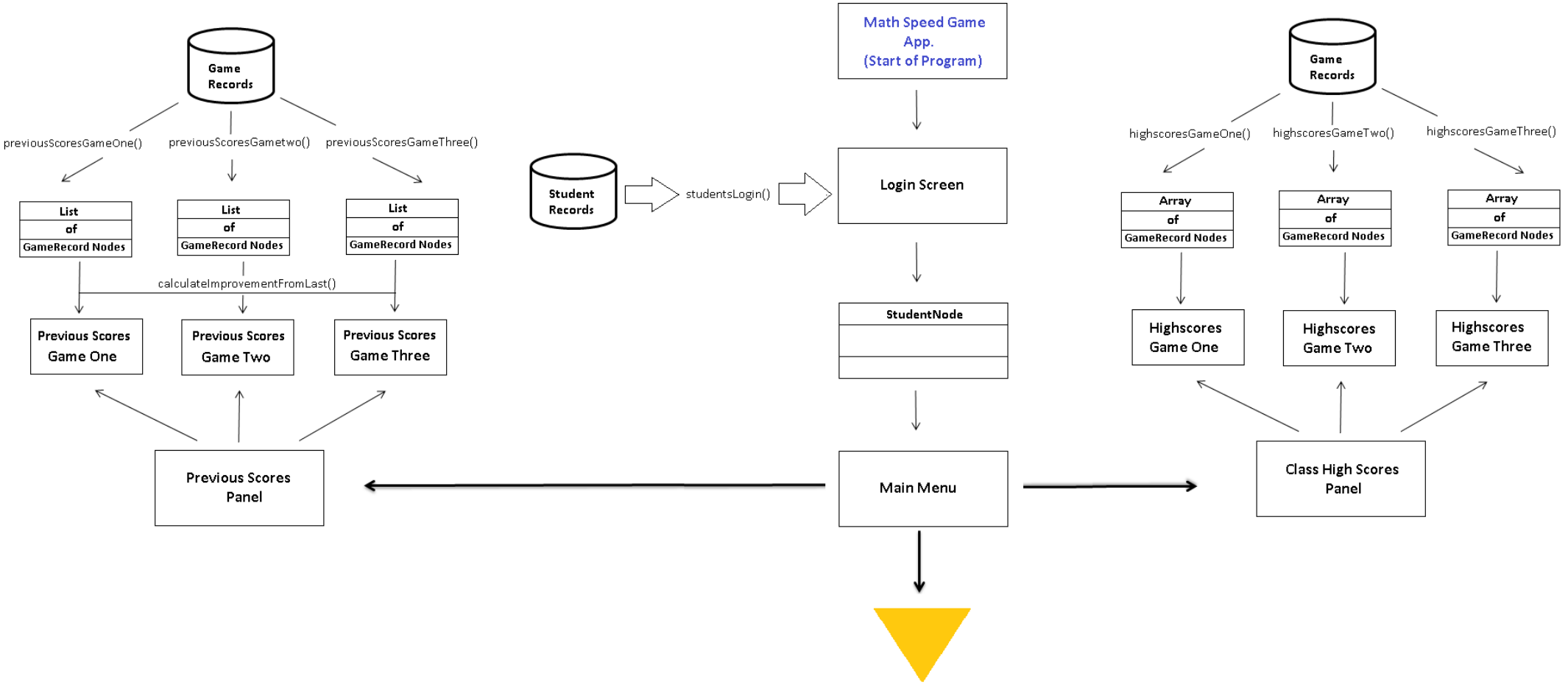
```
public boolean login(String id, String password) {

        file.seek(Integer.parseInt(id)*RECORD_LENGHT);//position of record (searched by id)
        file.readUTF();//skips first name
        file.readUTF();//skips last name

        if (file.readUTF().equals(id) && file.readUTF().equals(password)){
                return true;//login granted
        }
        return false;//login not granted
}
```

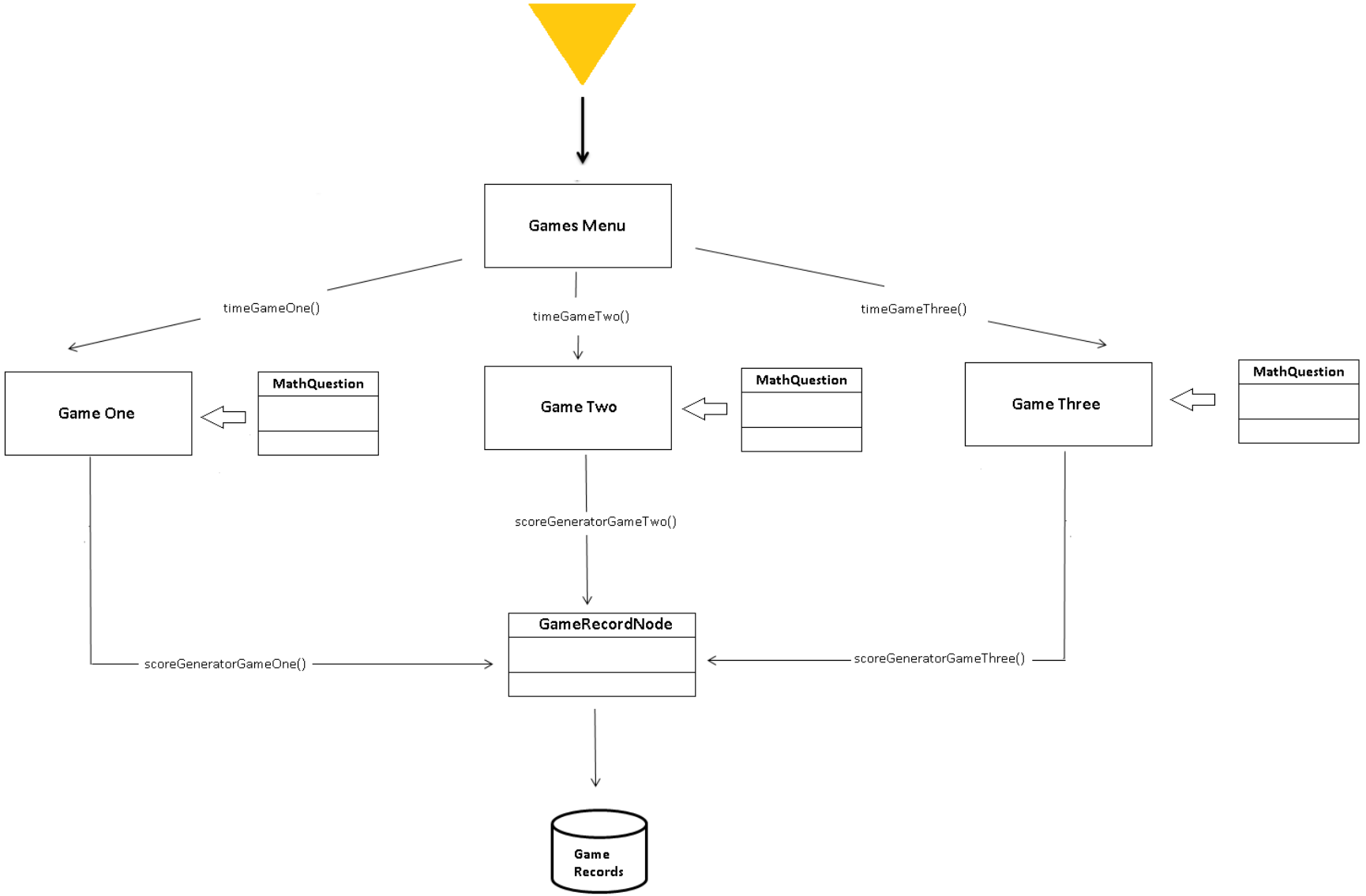
B3 - Modular Organisation

The diagram presented in the following page shows how the prototypes in section A3, along with the data structures and algorithms in section B, combine to compose the entire **students program**. Afterwards, we find a similar diagram, this is the composition of **Sr. Salomon's program** (or admin)

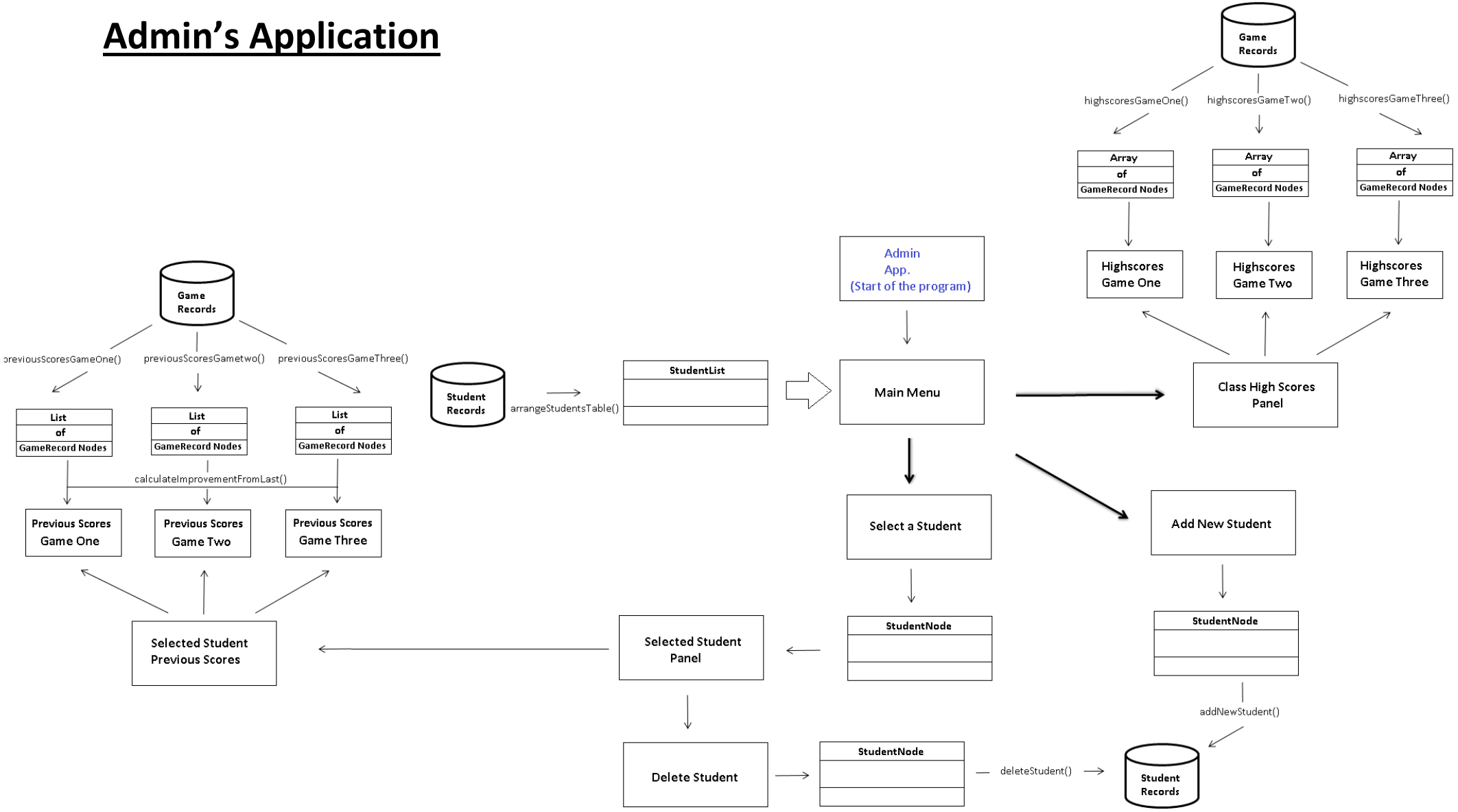
Because of the student's program diagram's size one section has been continued in the next page.

Student's Application




Games Menu
timeGameOne()
timeGameTwo()
timeGameThree()
Game One
MathQuestion
Game Two
MathQuestion
Game Three
MathQuestion
scoreGeneratorGameTwo()
GameRecordNode
scoreGeneratorGameOne()
scoreGeneratorGameThree()
Game Records

Admin's Application



C1 – The Program

Outline

All the code of the program Math Speed Game is listed in this section. The mastery levels achieved are explained within the classes where they are located.

Mastery Levels

#	Level	Class	Lines	Page(s)
1	Adding data to an instance of the RandomAccessFile class by direct manipulation of the file pointer using the seek method	*StudentHashFile* class (*addStudent* method)	56-65	108
3	Searching for specified data in a file.	*GameRecordGenerator* class (*getGameRecords* method)	82-122	94-99
6	Polymorphism	*MathQuestion* class→ *Addition Question* class, *SubtractionQuestion* class, *MultiplicationQuestion* class and *DivisionQuestion* class	All code in classes	80-93
7	Inheritance	*MathQuestion* class→ *Addition Question* class, *SubtractionQuestion* class, *MultiplicationQuestion* class and *DivisionQuestion* class	All code in classes	80-93
8	Encapsulation	*GameRecordNode* class	1-95	104-107
9	Parsing a text file or other data stream	Turning sequential line into a Student object (*StudentHashFile* class - *getStudentRecords* method)	121-142	112-113
11	The use of any five standard level mastery factors	2. User defined objects (*MathQuestion* class) 3. Objects as Data records (*StudentNode* class) 4. Simple Selection (*LoginScreenPan*el class – *login* method) 6. Loops (*StudentHashFile* class – *createFile* method) 7. Complex Loops (*GamePlayingInterfacePanel*		2. 80-91 3.117-119 4. 49-50 6. 110 7. 73-74

		class – *generateQuestion* method)		
12-15	Implementation of ADTs	*GameRecordList* class and *GameRecordNode* class	All code in classes	101-108
16	Use of additional libraries (swing)	*LoginScreenPanel* class		44-50

Classes concerning the Student's application

Classes concerning the Admin's application (the admin application uses some classes from the Student's app → the ones coloured in green)

```
1.  import javax.swing.*;
2.  import java.awt.*;
3.  import java.awt.event.ActionEvent;
4.  import java.awt.event.ActionListener;
5.  import java.io.IOException;
6.
7.  /**
8.   * @program Math Speed Game
9.   * @class LoginScreenPanel
10.  * @author nahel.rifai
11.  * @description This class has auto-generated code by Jigloo, plugin program used to create the GUI interface panel that
12.  * displays the login interface for the students' application. Here the students submit their ID and password to gain access to their
13.  * account in the app.
14.  * to the previousScoresPanel and to the  ClassHighScoresPanel
15.  * @Date 23/8/2012
16.  * @School Markham College, Lima, Peru
17.  * @IDE Eclipse (version Indigo)
18.  * @IBCode 000633054
19.  */
20.
21. /**
22.  * This code was edited or generated using CloudGarden's Jigloo
23.  * SWT/Swing GUI Builder, which is free for non-commercial
24.  * use. If Jigloo is being used commercially (ie, by a corporation,
25.  * company or business for any purpose whatever) then you
26.  * should purchase a license for each developer using Jigloo.
27.  * Please visit www.cloudgarden.com for details.
```

```
28.  * Use of Jigloo implies acceptance of these licensing terms.
29.  * A COMMERCIAL LICENSE HAS NOT BEEN PURCHASED FOR
30.  * THIS MACHINE, SO JIGLOO OR THIS CODE CANNOT BE USED
31.  * LEGALLY FOR ANY CORPORATE OR COMMERCIAL PURPOSE.
32.  */
33.
34. public class LoginScreenPanel extends javax.swing.JPanel {
35.
36.         {
37.                 //Set Look & Feel - code auto generated by Jigloo
38.                 try {
39.                         javax.swing.UIManager.setLookAndFeel("javax.swing.plaf.nimbus.NimbusLookAndFeel");
40.                 } catch(Exception e) {
41.                         e.printStackTrace();
42.                 }
43.         }
44.
45.         //variables auto-generated by Jigloo
46.         private JLabel titleLabel;
47.         private JLabel loginIdLabel;
48.         private JButton loginButton;
49.         private JTextField txtPassword;
50.         private JLabel passwordLabel;
51.         private JTextField txtLoginId;
52.         private  JFrame frame;
53.
54.         /**
55.          * Auto-generated main method to display this
```

```
56.          * JPanel inside a new JFrame.
57.          */
58.         public static void main(String[] args) {
59.                 JFrame frame = new JFrame();
60.                 frame.getContentPane().add(new LoginScreenPanel(frame));
61.                 frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
62.                 frame.pack();
63.                 frame.setLocation(380, 180);
64.                 frame.setVisible(true);
65.         }
66.
67.         /**
68.          * The constructor sets the created JFrame in the main method as the current frame
69.          * @param frame the frame created in the main method
70.          */
71.         public LoginScreenPanel(JFrame frame) {
72.                 super();
73.                 this.frame = frame;
74.                 initGUI();
75.         }
76.
77.         /**
78.          * Code in the initGUI method was automatically generated by Jigloo
79.          * The action listeners for the buttons were not generated by Jigloo
80.          */
81.         private void initGUI() {
82.                 try {
83.                         GridBagLayout thisLayout = new GridBagLayout();
```

```
84.                         this.setPreferredSize(new java.awt.Dimension(500, 350));
85.                         thisLayout.rowWeights = new double[] {0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1,0.1};
86.                         thisLayout.rowHeights = new int[] {7, 7, 7, 7, 7, 7, 7, 7, 7, 7};
87.                         thisLayout.columnWeights = new double[] {0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1,0.1};
88.                         thisLayout.columnWidths = new int[] {7, 7, 7, 7, 7, 7, 7, 7, 7, 7};
89.                         this.setLayout(thisLayout);
90.                         this.setSize(500, 350);
91.                         {
92.                                 titleLabel = new JLabel();
93.                                 this.add(titleLabel, new GridBagConstraints(2, 1, 6, 1, 0.0, 0.0,GridBagConstraints.CENTER, GridBagConstraints
    .NONE, new Insets(0, 0, 0, 0), 0, 0));
94.                                 titleLabel.setText("Math Speed Game");
95.                                 titleLabel.setFont(new java.awt.Font("Calibri",1,28));
96.                         }
97.                         {
98.                                 loginIdLabel = new JLabel();
99.                                 this.add(loginIdLabel, new GridBagConstraints(4, 3, 2, 1, 0.0, 0.0,GridBagConstraints.CENTER, GridBagConstrain
    ts.NONE, new Insets(0, 0, 0, 0), 0, 0));
100.                                        loginIdLabel.setText("Login ID:");
101.                                }
102.                                {
103.                                        txtLoginId = new JTextField();
104.                                        this.add(txtLoginId, new GridBagConstraints(4, 4, 2, 1, 0.0, 0.0,GridBagConstraints.CENTER, GridBagCo
    nstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
105.                                }
106.                                {
107.                                        passwordLabel = new JLabel();
```

```
108.                                    this.add(passwordLabel, new GridBagConstraints(4, 5, 2, 1, 0.0, 0.0,GridBagConstraints.CENTER, GridBa
    gConstraints.NONE, new Insets(0, 0, 0, 0), 0, 0));
109.                                    passwordLabel.setText("Password:");
110.                            }
111.                            {
112.                                    txtPassword = new JPasswordField();
113.                                    this.add(txtPassword, new GridBagConstraints(4, 6, 2, 1, 0.0, 0.0,GridBagConstraints.CENTER, GridBagC
    onstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
114.                            }
115.                            {
116.                                    loginButton = new JButton();
117.                                    this.add(loginButton, new GridBagConstraints(4, 7, 2, 1, 0.0, 0.0,GridBagConstraints.CENTER, GridBagC
    onstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
118.                                    loginButton.setText("Login");
119.                                    /**
120.                                     * Adds an action listener to the "Login Button"
121.                                     *
122.                                     */
123.                                    loginButton.addActionListener(new ActionListener() {
124.                                            public void actionPerformed(ActionEvent evt) {
125.                                                    login();//when the button Login is clicked the method login() is executed
126.                                            }
127.                                    });
128.                            }
129.                    } catch (Exception e) {
130.                            e.printStackTrace();
131.                    }
132.            }
```

```
133.
134.                /**
135.                 * The ID and password input are validated by calling the method constructor of the StudentHashFile (class dealing with the
    student's RandomAcessFile
136.                 * If the ID matches the password then the student is granted access to it's account on the application
137.                 */
138.                public void login(){
139.
140.                        try{
141.                                try {
142.                                        StudentHashFile file = new StudentHashFile();
143.
144.                                        if (file.login(txtLoginId.getText(), txtPassword.getText()) == true){
145.                                                //takes you to the main menu of the app
146.                                                JFrame frame = new JFrame();
147.                                                frame.getContentPane().add(new StudentsMainMenuPanel(frame, txtLoginId.getText()));
148.                                                frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
149.                                                frame.pack();
150.                                                frame.setLocation(380, 180);
151.                                                frame.setVisible(true);
152.                                                this.frame.setVisible(false);
153.                                                this.frame.dispose();
154.                                        } else {
155.                                                JOptionPane.showMessageDialog(this, "Wrong Username or
    Password"  ,"Error", JOptionPane.INFORMATION_MESSAGE);
156.                                        }
157.                                } catch (IOException e){
158.                                        JOptionPane.showMessageDialog(this, "File Error"  , "Error",JOptionPane.INFORMATION_MESSAGE);
```

```
159.                                }
160.                        } catch (NumberFormatException e){
161.                                JOptionPane.showMessageDialog(this, "Wrong Username or Password"  , "Error",JOptionPane.INFORMATION_MESSAGE);
162.                                txtLoginId.setText("");
163.                                txtPassword.setText("");
164.                        }
165.                }
166.        }
```

```
1.  import java.io.*;
2.  import java.awt.*;
3.  import java.awt.event.ActionEvent;
4.  import java.awt.event.ActionListener;
5.  import javax.swing.*;
6.
7.  /**
8.   * @program Math Speed Game
9.   * @class StudentsMainMenuPanel
10.  * @class This class has auto-generated code by Jigloo, plugin program used to create the GUI interface for panel that
11.  * displays the main menu interface of the students program. Here the students can choose to go to the SelectGamesPanel,
12.  * the PreviousScoresPanel or the ClassHighScoresPanel.
13.  * @author nahel.rifai
14.  * @Date 23/8/2012
15.  * @School Markham College, Lima, Peru
16.  * @IDE Eclipse (version Indigo)
17.  * @IBCode 000633054
18.  */
19.
20. /**
21.  * This code was edited or generated using CloudGarden's Jigloo
22.  * SWT/Swing GUI Builder, which is free for non-commercial
23.  * use. If Jigloo is being used commercially (ie, by a corporation,
24.  * company or business for any purpose whatever) then you
25.  * should purchase a license for each developer using Jigloo.
26.  * Please visit www.cloudgarden.com for details.
27.  * Use of Jigloo implies acceptance of these licensing terms.
28.  * A COMMERCIAL LICENSE HAS NOT BEEN PURCHASED FOR
```

```
29.  * THIS MACHINE, SO JIGLOO OR THIS CODE CANNOT BE USED
30.  * LEGALLY FOR ANY CORPORATE OR COMMERCIAL PURPOSE.
31.  */
32. public class StudentsMainMenuPanel extends javax.swing.JPanel {
33. 
34.         //variables auto-generated by Jigloo
35.         private JLabel welcomeLabel;
36.         private JButton playGamesButton;
37.         private JButton previousScoresButton;
38.         private JButton logoutButton;
39.         private JButton highscoresButton;
40.         private JFrame frame;
41. 
42.         //variables not generated by Jigloo
43.         private String id; //the ID of the student logged in into the system
44. 
45.         /**
46.          * The constructor sets the id and frame passed from the LoginScreenPanel and sets them
47.          * the current ones being used in this class
48.          * @param frame the previous frame
49.          * @param id the student's id
50.          */
51.         public StudentsMainMenuPanel(JFrame frame, String id) {
52.                 super();
53.                 this.frame = frame;
54.                 this.id = id;
55.                 initGUI();
56.         }
```

```
57.
58.         /**
59.          * Code in the initGUI method was automatically generated by Jigloo
60.          * The action listeners for the buttons were not generated by Jigloo
61.          */
62.         private void initGUI() {
63.                 try {
64.                         GridBagLayout thisLayout = new GridBagLayout();
65.                         thisLayout.rowWeights = new double[] {0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1};
66.                         thisLayout.rowHeights = new int[] {7, 7, 7, 7, 7, 7, 7, 7, 7, 7};
67.                         thisLayout.columnWeights = new double[] {0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1};
68.                         thisLayout.columnWidths = new int[] {7, 7, 7, 7, 7, 7, 7, 7, 7, 7};
69.                         this.setLayout(thisLayout);
70.                         this.setPreferredSize(new java.awt.Dimension(500, 350));
71.                         this.setSize(500, 350);
72.                         {
73.                                 welcomeLabel = new JLabel();
74.                                 this.add(welcomeLabel, new GridBagConstraints(2, 1, 6, 1, 0.0, 0.0, GridBagConstraints.CENTER, GridBagConstrai
    nts.NONE, new Insets(0, 0, 0, 0), 0, 0));
75.                                 welcomeLabel.setText("Welcome " + getStudentName());
76.                                 welcomeLabel.setFont(new java.awt.Font("Calibri",1,28));
77.                         }
78.                         {
79.                                 playGamesButton = new JButton();
80.                                 this.add(playGamesButton, new GridBagConstraints(4, 3, 2, 1, 0.0, 0.0, GridBagConstraints.CENTER, GridBagConst
    raints.BOTH, new Insets(0, 0, 0, 0), 0, 0));
81.                                 playGamesButton.setText("Play Games!");
82.                                 playGamesButton.addActionListener(new ActionListener() {
```

```
83.                                     public void actionPerformed(ActionEvent evt) {
84.                                             playGames();//button takes you to the SelectGamesPanel
85.                                     }
86.                                 });
87.                         }
88.                         {
89.                                 previousScoresButton = new JButton();
90.                                 this.add(previousScoresButton, new GridBagConstraints(4, 5, 2, 1, 0.0, 0.0, GridBagConstraints.CENTER, GridBag
   Constraints.BOTH, new Insets(0, 0, 0, 0), 0, 0));
91.                                 previousScoresButton.setText("Check Previous Scores");
92.                                 previousScoresButton.addActionListener(new ActionListener() {
93.                                         public void actionPerformed(ActionEvent evt) {
94.                                                 previousScores();//Button takes you to the PreviousScoresPanel
95.                                         }
96.                                 });
97.                         }
98.                         {
99.                                 highscoresButton = new JButton();
100.                                        this.add(highscoresButton, new GridBagConstraints(4, 7, 2, 1, 0.0, 0.0, GridBagConstraints.CENTER, Gr
   idBagConstraints.BOTH, new Insets(0, 0, 0, 0), 0, 0));
101.                                        highscoresButton.setText("Class Highscores");
102.                                        highscoresButton.addActionListener(new ActionListener() {
103.                                                public void actionPerformed(ActionEvent evt) {
104.                                                        highScores();//Button takes you to the ClassHighScoresPanel
105.                                                }
106.                                        });
107.
108.                                }
```

```
109.                                {
110.                                        logoutButton = new JButton();
111.                                        this.add(logoutButton, new GridBagConstraints(4, 9, 2, 1, 0.0, 0.0, GridBagConstraints.CENTER, GridBa
    gConstraints.NONE, new Insets(0, 0, 0, 0), 0, 0));
112.                                        logoutButton.setText("Logout");
113.                                        logoutButton.addActionListener(new ActionListener() {
114.                                                public void actionPerformed(ActionEvent evt) {
115.                                                        logout();//Button logs the student out of the application
116.                                                }
117.                                        });
118.                                }
119.                        } catch (Exception e) {
120.                                e.printStackTrace();
121.                        }
122.                }
123.
124.                //takes you to the SelectGamesPanel
125.                public void playGames(){
126.                        JFrame frame = new JFrame();
127.                        frame.getContentPane().add(new SelectGamesPanel(frame, id));
128.                        frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
129.                        frame.pack();
130.                        frame.setLocation(380, 180);
131.                        frame.setVisible(true);
132.                        this.frame.setVisible(false);
133.                        this.frame.dispose();
134.                }
135.
```

```
136.                //takes you to the PreviousScoresPanel
137.                public void previousScores(){
138.                        JFrame frame = new JFrame();
139.                        frame.getContentPane().add(new PreviousScoresPanel(frame, id));
140.                        frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
141.                        frame.pack();
142.                        frame.setLocation(380, 180);
143.                        frame.setVisible(true);
144.                        this.frame.setVisible(false);
145.                        this.frame.dispose();
146.                }
147.
148.                //takes you the the ClassHighScoresPanel
149.                public void highScores(){
150.                        JFrame frame = new JFrame();
151.                        frame.getContentPane().add(new ClassHighScoresPanel(frame, id));
152.                        frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
153.                        frame.pack();
154.                        frame.setLocation(380, 180);
155.                        frame.setVisible(true);
156.                        this.frame.setVisible(false);
157.                        this.frame.dispose();
158.                }
159.
160.                //takes you to the LoginScreenPanel - logs you out of the app
161.                public void logout(){
162.                        JFrame frame = new JFrame();
163.                        frame.getContentPane().add(new LoginScreenPanel(frame));
```

```
164.                    frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
165.                    frame.pack();
166.                    frame.setLocation(380, 180);
167.                    frame.setVisible(true);
168.                    this.frame.setVisible(false);
169.                    this.frame.dispose();
170.            }
171.
172.            /**
173.             * This method uses the id of the logged in student to
174.             * search for their first name in the sequential file where
175.             * the student's are located. Their first name is used
176.             * for the greeting message "Welcome "example first name"!"
177.             * @return the string first name of the student
178.             */
179.            public String getStudentName(){
180.
181.                    try{
182.                            StudentHashFile student = new StudentHashFile();
183.                            return student.getFirstName(id);//returns the student's first name
184.
185.                    } catch (IOException e){
186.                            return "";
187.                    }
188.            }
189.    }
```

```
1. import java.awt.*;
2. import java.awt.event.ActionEvent;
3. import java.awt.event.ActionListener;
4. import javax.swing.*;
5.
6. /**
7.  * @program Math Speed Game
8.  * @class SelectGamesPanel
9.  * @author nahel.rifai
10. * @description This class has auto-generated code by Jigloo, plugin program used to create the GUI interface for panel that displays the
11. * different games available ot the students, here they can choose which of the three games to play.
12. * @Date 23/8/2012
13. * @School Markham College, Lima, Peru
14. * @IDE Eclipse (version Indigo)
15. * @IBCode 000633054
16. */
17.
18./**
19. * This code was edited or generated using CloudGarden's Jigloo
20. * SWT/Swing GUI Builder, which is free for non-commercial
21. * use. If Jigloo is being used commercially (ie, by a corporation,
22. * company or business for any purpose whatever) then you
23. * should purchase a license for each developer using Jigloo.
24. * Please visit www.cloudgarden.com for details.
25. * Use of Jigloo implies acceptance of these licensing terms.
26. * A COMMERCIAL LICENSE HAS NOT BEEN PURCHASED FOR
27. * THIS MACHINE, SO JIGLOO OR THIS CODE CANNOT BE USED
28. * LEGALLY FOR ANY CORPORATE OR COMMERCIAL PURPOSE.
```

```
29.  */
30. public class SelectGamesPanel extends javax.swing.JPanel {
31. 
32.         //variables auto-generated by Jigloo
33.         private JLabel selectAGameLabel;
34.         private JButton backToMainMenuButton;
35.         private JLabel jLabel1;
36.         private JLabel gameThreeLabel;
37.         private JLabel separatorLabel;
38.         private JLabel gameThreeDescriptionThreeLabel;
39.         private JButton playGameThreeButton;
40.         private JButton playGameTwoButton;
41.         private JButton playGameOneButton;
42.         private JLabel gameThreeDescriptionTwoLabel;
43.         private JLabel gameThreeDescriptionLabel;
44.         private JLabel gameTwoDescriptionLabel;
45.         private JLabel gameOneDescriptionLabel;
46.         private JLabel gameTwoLabel;
47.         private JLabel gameOneLabel;
48.         private JFrame frame;
49. 
50.         //variable not generated by Jigloo
51.         private String id; //the ID of the student logged in into the system
52. 
53.         /**
54.          * The constructor uses the frame from the previous panel- StudentsMainMenuPanel and sets it as the current
55.          * It sets the id of the student that logged in as the current id
56.          * @param frame the previous frame
```

```
57.          * @param id the id of the student
58.          */
59.         public SelectGamesPanel(JFrame frame, String id) {
60.                 super();
61.                 this.frame = frame;
62.                 this.id = id;
63.                 initGUI();
64.         }
65.
66.         /**
67.          * Code in the initGUI method was automatically generated by Jigloo
68.          * The action listeners for the buttons were not generated by Jigloo
69.          */
70.         private void initGUI() {
71.                 try {
72.                         this.setPreferredSize(new java.awt.Dimension(500, 350));
73.                         GridBagLayout thisLayout = new GridBagLayout();
74.                         this.setSize(500, 350);
75.                         thisLayout.rowWeights = new double[] {0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1,
    0.1, 0.1, 0.1, 0.1};
76.                         thisLayout.rowHeights = new int[] {7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7};
77.                         thisLayout.columnWeights = new double[] {0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1};
78.                         thisLayout.columnWidths = new int[] {7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7};
79.                         this.setLayout(thisLayout);
80.                         {
81.                                 selectAGameLabel = new JLabel();
82.                                 this.add(selectAGameLabel, new GridBagConstraints(3, 1, 6, 1, 0.0, 0.0, GridBagConstraints.CENTER, GridBagCons
    traints.NONE, new Insets(0, 0, 0, 0), 0, 0));
```

```
83.                             selectAGameLabel.setText("Select a Game");
84.                             selectAGameLabel.setFont(new java.awt.Font("Calibri",1,28));
85.                         }
86.                         {
87.                             gameOneLabel = new JLabel();
88.                             this.add(gameOneLabel, new GridBagConstraints(1, 6, 2, 1, 0.0, 0.0, GridBagConstraints.CENTER, GridBagConstrai
    nts.VERTICAL, new Insets(0, 0, 0, 0), 0, 0));
89.                             gameOneLabel.setText("Game One:");
90.                             gameOneLabel.setFont(new java.awt.Font("Tahoma",1,14));
91.                         }
92.                         {
93.                             gameTwoLabel = new JLabel();
94.                             this.add(gameTwoLabel, new GridBagConstraints(1, 10, 2, 1, 0.0, 0.0, GridBagConstraints.CENTER, GridBagConstra
    ints.VERTICAL, new Insets(0, 0, 0, 0), 0, 0));
95.                             gameTwoLabel.setText("Game Two:");
96.                             gameTwoLabel.setFont(new java.awt.Font("Tahoma",1,14));
97.                         }
98.                         {
99.                             gameThreeLabel = new JLabel();
100.                                this.add(gameThreeLabel, new GridBagConstraints(1, 14, 2, 1, 0.0, 0.0, GridBagConstraints.CENTER, Gri
    dBagConstraints.VERTICAL, new Insets(0, 0, 0, 0), 0, 0));
101.                                gameThreeLabel.setText("Game Three:");
102.                                gameThreeLabel.setFont(new java.awt.Font("Tahoma",1,14));
103.                            }
104.                            {
105.                                gameOneDescriptionLabel = new JLabel();
106.                                this.add(gameOneDescriptionLabel, new GridBagConstraints(3, 6, 4, 1, 0.0, 0.0, GridBagConstraints.CEN
    TER, GridBagConstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
```

```
107.                                    gameOneDescriptionLabel.setText("Complete 20 short questions as fast as possible!");
108.                            }
109.                            {
110.                                    gameTwoDescriptionLabel = new JLabel();
111.                                    this.add(gameTwoDescriptionLabel, new GridBagConstraints(3, 10, 4, 1, 0.0, 0.0, GridBagConstraints.CE
    NTER, GridBagConstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
112.                                    gameTwoDescriptionLabel.setText("Solve as much problems as you can in 60 seconds!");
113.                            }
114.                            {
115.                                    gameThreeDescriptionLabel = new JLabel();
116.                                    this.add(gameThreeDescriptionLabel, new GridBagConstraints(3, 14, 1, 1, 0.0, 0.0, GridBagConstraints.
    CENTER, GridBagConstraints.NONE, new Insets(0, 0, 0, 0), 0, 0));
117.                                    gameThreeDescriptionLabel.setText("You are given an initial 60 seconds.");
118.                            }
119.                            {
120.                                    gameThreeDescriptionTwoLabel = new JLabel();
121.                                    this.add(gameThreeDescriptionTwoLabel, new GridBagConstraints(3, 15, 1, 1, 0.0, 0.0, GridBagConstrain
    ts.CENTER, GridBagConstraints.NONE, new Insets(0, 0, 0, 0), 0, 0));
122.                                    gameThreeDescriptionTwoLabel.setText(" Each correct answer gives  you 10 bonus");
123.                            }
124.                            {
125.                                    playGameOneButton = new JButton();
126.                                    this.add(playGameOneButton, new GridBagConstraints(7, 6, 6, 1, 0.0, 0.0, GridBagConstraints.CENTER, G
    ridBagConstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
127.                                    playGameOneButton.setText("Play");
128.                                    playGameOneButton.addActionListener(new ActionListener() {
129.                                            public void actionPerformed(ActionEvent evt) {
130.                                                    playGame(1);//play game 1
```

```
131.                                        }
132.                                });
133.                        }
134.                        {
135.                                playGameTwoButton = new JButton();
136.                                this.add(playGameTwoButton, new GridBagConstraints(7, 10, 6, 1, 0.0, 0.0, GridBagConstraints.CENTER,
    GridBagConstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
137.                                playGameTwoButton.setText("Play");
138.                                playGameTwoButton.addActionListener(new ActionListener() {
139.                                        public void actionPerformed(ActionEvent evt) {
140.                                                playGame(2);//play game 2
141.                                        }
142.                                });
143.                        }
144.                        {
145.                                playGameThreeButton = new JButton();
146.                                this.add(playGameThreeButton, new GridBagConstraints(7, 14, 6, 1, 0.0, 0.0, GridBagConstraints.CENTER
    , GridBagConstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
147.                                playGameThreeButton.setText("Play");
148.                                playGameThreeButton.addActionListener(new ActionListener() {
149.                                        public void actionPerformed(ActionEvent evt) {
150.                                                playGame(3);//play game 3
151.                                        }
152.                                });
153.                        }
154.                        {
155.                                gameThreeDescriptionThreeLabel = new JLabel();
```

```
156.                                    this.add(gameThreeDescriptionThreeLabel, new GridBagConstraints(3, 16, 1, 1, 0.0, 0.0, GridBagConstra
    ints.CENTER, GridBagConstraints.NONE, new Insets(0, 0, 0, 0), 0, 0));
157.                                    gameThreeDescriptionThreeLabel.setText(" seconds on the clock. Better hurry!");
158.                            }
159.                            {
160.                                    separatorLabel = new JLabel();
161.                                    this.add(separatorLabel, new GridBagConstraints(0, 8, 16, 1, 0.0, 0.0, GridBagConstraints.CENTER, Gri
    dBagConstraints.NONE, new Insets(0, 0, 0, 0), 0, 0));
162.                                    separatorLabel.setText("_____");
163.                            }
164.                            {
165.                                    jLabel1 = new JLabel();
166.                                    this.add(jLabel1, new GridBagConstraints(1, 12, 12, 1, 0.0, 0.0, GridBagConstraints.CENTER, GridBagCo
    nstraints.NONE, new Insets(0, 0, 0, 0), 0, 0));
167.                                    jLabel1.setText("_____");
168.                            }
169.                            {
170.                                    backToMainMenuButton = new JButton();
171.                                    this.add(backToMainMenuButton, new GridBagConstraints(1, 1, 2, 1, 0.0, 0.0, GridBagConstraints.CENTER
    , GridBagConstraints.NONE, new Insets(0, 0, 0, 0), 0, 0));
172.                                    backToMainMenuButton.setText("Back to Menu");
173.                                    backToMainMenuButton.addActionListener(new ActionListener() {
174.                                            public void actionPerformed(ActionEvent evt) {
175.                                                    backToMainMenu();//back to main menu
176.                                            }
177.                                    });
178.                            }
179.                    } catch (Exception e) {
```

```
180.                            e.printStackTrace();
181.                    }
182.            }
183.
184.            //takes you back to main menu
185.            public void backToMainMenu(){
186.                    JFrame frame = new JFrame();
187.                    frame.getContentPane().add(new StudentsMainMenuPanel(frame, id));
188.                    frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
189.                    frame.pack();
190.                    frame.setLocation(380, 180);
191.                    frame.setVisible(true);
192.                    this.frame.setVisible(false);
193.                    this.frame.dispose();
194.            }
195.
196.            /**
197.             * This method calls the class GamePlayingInterfacePanel (where the game is played)
198.             * The game number is important because it states the rules of the game
199.             * @param i this is the number of the game which has been selected (either 1, 2 or 3)
200.             */
201.            public void playGame(int i){
202.                    JFrame frame = new JFrame();
203.                    frame.getContentPane().add(new GamePlayingInterfacePanel(frame, i, id));
204.                    frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
205.                    frame.pack();
206.                    frame.setLocation(380, 180);
207.                    frame.setVisible(true);
```

```
208.                     this.frame.setVisible(false);
209.                     this.frame.dispose();
210.                 }
211.         }
```

```
1.  import java.awt.*;
2.  import java.awt.event.ActionEvent;
3.  import java.awt.event.ActionListener;
4.  import java.io.IOException;
5.  import javax.swing.*;
6.
7.  /**
8.   * @program Math Speed Game
9.   * @class GamePlayingInterfacePanel
10.  * @author nahel.rifai
11.  * @description This class has auto-generated code by Jigloo, plugin program used to create the GUI interface panel that
12.  * displays the game playing interface panel where the students play the math games and record the answers to the
13.  * mathematical problems presented. This class provides displays the math problem ot the student, provides a blank input
14.  * box for them to enter the answer and also presents a timer, used in the speed games presented in the application.
15.  * @Date 23/8/2012
16.  * @School Markham College, Lima, Peru
17.  * @IDE Eclipse (version Indigo)
18.  * @IBCode 000633054
19.  */
20.
21. /**
22.  * This code was edited or generated using CloudGarden's Jigloo
23.  * SWT/Swing GUI Builder, which is free for non-commercial
24.  * use. If Jigloo is being used commercially (ie, by a corporation,
25.  * company or business for any purpose whatever) then you
26.  * should purchase a license for each developer using Jigloo.
27.  * Please visit www.cloudgarden.com for details.
28.  * Use of Jigloo implies acceptance of these licensing terms.
```

```
29.  * A COMMERCIAL LICENSE HAS NOT BEEN PURCHASED FOR
30.  * THIS MACHINE, SO JIGLOO OR THIS CODE CANNOT BE USED
31.  * LEGALLY FOR ANY CORPORATE OR COMMERCIAL PURPOSE.
32.  */
33. public class GamePlayingInterfacePanel extends javax.swing.JPanel {
34. 
35.         //variables auto-generated by Jigloo
36.         private JLabel questionNumberLabel;
37.         private JLabel equalLabel;
38.         private JLabel displayQuestion;
39.         private JLabel answerLabel;
40.         private JButton backToMenuButton;
41.         private JLabel timerLabel;
42.         private JButton nextQuestionButton;
43.         private JLabel correctAnswerLabel;
44.         private JTextField answerTxt;
45.         private JFrame frame;
46. 
47.         //variables NOT generated by Jigloo
48.         private MathQuestion [] question = new MathQuestion [30]; //the array of MathQuestion objects each used for the math questions
49.         private int currentQuestion = 0; //the current question being attempted
50.         private int correctAnswers; //the counter of correct answer
51.         private int gameNumber = 0; //the current game being played
52.         private long start = System.currentTimeMillis();//the starting time on the timer
53.         int leftTime = 60; //the time left on the timer for Game 3
54.         private String id; //the ID of the student logged in into the system
55. 
56. 
```

```
57.         /**
58.          * The constructor passes the frame from the StudentsMainMenuPanel and sets it as the current.
59.          * It also passes the game type chosen (as the rules for the games are different)
60.          * It passes the id of the student logged in, to know which student is playing again (needed to record the game record with the
    student id).
61.          * @param frame the previous frame from the StudentsMainMenuPanel
62.          * @param gameNumber the game type chosen by the student
63.          * @param id the id of the student logged in
64.          */
65.         public GamePlayingInterfacePanel(JFrame frame, int gameNumber, String id) {
66.                 super();
67.                 this.frame = frame;
68.                 this.gameNumber = gameNumber;
69.                 this.id = id;
70.                 initGUI();
71.                 generateQuestions();//generates new math questions for the current game
72.                 displayNextQuestion();//displays the next question generated
73.         }
74.
75.         /**
76.          * Code in the initGUI method was automatically generated by Jigloo
77.          * The action listeners for the buttons were not generated by Jigloo
78.          */
79.         private void initGUI() {
80.                 try {
81.                         this.setPreferredSize(new java.awt.Dimension(500, 350));
82.                         GridBagLayout thisLayout = new GridBagLayout();
83.                         this.setSize(500, 350);
```

```
84.                         thisLayout.rowWeights = new double[] {0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1};
85.                         thisLayout.rowHeights = new int[] {7, 7, 7, 7, 7, 7, 7, 7, 7, 7};
86.                         thisLayout.columnWeights = new double[] {0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1};
87.                         thisLayout.columnWidths = new int[] {7, 7, 7, 7, 7, 7, 7, 7, 7, 7};
88.                         this.setLayout(thisLayout);
89.                         {
90.                                 questionNumberLabel = new JLabel();
91.                                 this.add(questionNumberLabel, new GridBagConstraints(3, 2, 4, 1, 0.0, 0.0, GridBagConstraints.CENTER, GridBagC
    onstraints.NONE, new Insets(0, 0, 0, 0), 0, 0));
92.                                 questionNumberLabel.setText("Question " + (currentQuestion+1));
93.                                 questionNumberLabel.setFont(new java.awt.Font("Tahoma",1,14));
94.                         }
95.                         {
96.                                 equalLabel = new JLabel();
97.                                 this.add(equalLabel, new GridBagConstraints(6, 4, 1, 1, 0.0, 0.0, GridBagConstraints.CENTER, GridBagConstraint
    s.NONE, new Insets(0, 0, 0, 0), 0, 0));
98.                                 equalLabel.setText("= _________");
99.                         }
100.                                {
101.                                        answerTxt = new JTextField();
102.                                        this.add(answerTxt, new GridBagConstraints(4, 6, 2, 1, 0.0, 0.0, GridBagConstraints.CENTER, GridBagCo
    nstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
103.                                }
104.                                {
105.                                        answerLabel = new JLabel();
106.                                        this.add(answerLabel, new GridBagConstraints(3, 6, 1, 1, 0.0, 0.0, GridBagConstraints.CENTER, GridBag
    Constraints.NONE, new Insets(0, 0, 0, 0), 0, 0));
107.                                        answerLabel.setText("Answer:");
```

```
108.                                }
109.                                {
110.                                        nextQuestionButton = new JButton();
111.                                        this.add(nextQuestionButton, new GridBagConstraints(4, 7, 2, 1, 0.0, 0.0, GridBagConstraints.CENTER,
    GridBagConstraints.BOTH, new Insets(0, 0, 0, 0), 0, 0));
112.                                        nextQuestionButton.setText("Enter");
113.                                        nextQuestionButton.addActionListener(new ActionListener() {
114.
115.                                                /**
116.                                                 * When the button "Enter" is clicked the answer input is checked and
117.                                                 * the next question is displayed
118.                                                 */
119.                                                public void actionPerformed(ActionEvent evt) {
120.
121.                                                        try{
122.                                                                checkAnswer();
123.                                                                displayNextQuestion();
124.                                                        } catch (NumberFormatException e){
125.                                                                JOptionPane.showMessageDialog(frame, "Only input
    numbers!", "Error", JOptionPane.INFORMATION_MESSAGE);
126.                                                                answerTxt.setText("");
127.                                                        }
128.                                                }
129.                                        });
130.                                }
131.                                {
132.                                        timerLabel = new JLabel();
```

```
133.                                    this.add(timerLabel, new GridBagConstraints(6, 0, 4, 2, 0.0, 0.0, GridBagConstraints.CENTER, GridBagC
    onstraints.NONE, new Insets(0, 0, 0, 0), 0, 0));
134.                                    timerLabel.setText("0 seconds");
135.                                    timerLabel.setFont(new java.awt.Font("Tahoma",1,20));
136.                                }
137.                                {
138.                                    backToMenuButton = new JButton();
139.                                    this.add(backToMenuButton, new GridBagConstraints(3, 0, 1, 2, 0.0, 0.0, GridBagConstraints.CENTER, Gr
    idBagConstraints.NONE, new Insets(0, 0, 0, 0), 0, 0));
140.                                    backToMenuButton.setText("Quit Game");
141.                                    backToMenuButton.addActionListener(new ActionListener() {
142.                                            public void actionPerformed(ActionEvent evt) {
143.                                                    quitGame();//when this button is clicked, the student is taken to the main menu and
    the answers are not registered
144.                                            }
145.                                    });
146.                                }
147.                                {
148.                                    displayQuestion = new JLabel();
149.                                    this.add(displayQuestion, new GridBagConstraints(3, 4, 3, 1, 0.0, 0.0, GridBagConstraints.CENTER, Gri
    dBagConstraints.NONE, new Insets(0, 0, 0, 0), 0, 0));
150.                                    displayQuestion.setText("displayQuestion");
151.                                    displayQuestion.setFont(new java.awt.Font("Tahoma",1,18));
152.                                }
153.                                {
154.                                    correctAnswerLabel = new JLabel();
155.                                    this.add(correctAnswerLabel, new GridBagConstraints(6, 6, 1, 1, 0.0, 0.0, GridBagConstraints.CENTER,
    GridBagConstraints.NONE, new Insets(0, 0, 0, 0), 0, 0));
```

```
156.                                    correctAnswerLabel.setText("");
157.                                    correctAnswerLabel.setVisible(false);
158.
159.                            }
160.                    } catch (Exception e) {
161.                            e.printStackTrace();
162.                    }
163.            }
164.
165.            public void quitGame(){
166.                    JFrame frame = new JFrame();
167.                    frame.getContentPane().add(new SelectGamesPanel(frame, id));
168.                    frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
169.                    frame.pack();
170.                    frame.setLocation(380, 180);
171.                    frame.setVisible(true);
172.                    this.frame.setVisible(false);
173.                    this.frame.dispose();
174.            }
175.
176.            /**
177.             * The method generateQuestions generates 30 questions
178.             * There is an equal chance that an addition, subtraction, multiplication or division question is generated
179.             * The array question[] holds 20 MathQuestion objects (the math questions presented to the student)
180.             */
181.            public void generateQuestions(){
182.
183.                    int randomNumber = 0;
```

```
184.
185.                     for (int i = 0; i < 20; i++){
186.                             randomNumber = (int)(Math.random()*4);
187.
188.                             if (randomNumber == 0){
189.                                     question [i] = new AdditionQuestion();
190.                             } else if (randomNumber == 1){
191.                                     question [i] = new SubstractionQuestion();
192.                             } else if (randomNumber == 2){
193.                                     question [i] = new MultiplicationQuestion();
194.                             } else {
195.                                     question [i] = new DivisionQuestion();
196.                             }
197.                     }
198.
199.             }
200.
201.             /**
202.              * The method displayNexQuestion regulates the timer each time a new question is presented
203.              * In game 3 each time the student inputs a right answer they are given 10 additional seconds in the timer
204.              * It also updates the currentQuestion counter
205.              */
206.             private void displayNextQuestion(){
207.
208.                     long finish = System.currentTimeMillis();
209.                     timerLabel.setText(((finish-start)/1000)+ "");
210.
211.                     if (gameNumber == 3){
```

```
                                long checkPoint = System.currentTimeMillis();
                                leftTime = 60 - ((int)(checkPoint-start)/1000) ;
                                leftTime = leftTime + 10*(correctAnswers);
                                timerLabel.setText(leftTime + "");
                            }

                            questionNumberLabel.setText("Question " + (currentQuestion+1));
                            answerTxt.setText("");
                            displayQuestion.setText(question[currentQuestion].getDisplay());
                            correctAnswerLabel.setText(question[currentQuestion].getAnswer()+"");
                            currentQuestion++;
                        }

                        /**
                         * The checkAnswer method is in charge of the following
                         * There is an invisible label called "correctAnswerLabel" holding the exact answer to the question
                         * The answer input by the student is compared to the answer in the "correctAnswerLabel"
                         * The answer calculated by the app always has decimal places, even if the answer is a whole number
                         * As the students won't input a whole number with a .0 decimal place the answer input by the students is converter to a
                    double and then to a string to add the .0
                         * If the student is playing game one and the current question is 20 (last question) then the game ends and the score is
                    calculated and stored
                         * If the student is playing game two it checks if the time is bigger than 60 seconds, then the game ends and the score is
                    calculated and stored
                         * If the student is playing game three it checks if the timer is less than 0 seconds, then the game ends and the score is
                    calculated
                         */
                        private void checkAnswer(){
```

```
236.                        //convert the answer to double and then again to string to add the .0 needed
237.                        String answer = answerTxt.getText();
238.                        //check if answer is empty because an empty string cant be converted to a double
239.                        if (!answer.equals("")){
240.                                double number = Double.parseDouble(answer);
241.                                answer = number + "";
242.                        }
243.
244.                        //check for correct or wrong answers
245.                        if (answer.equals(correctAnswerLabel.getText())){
246.                                correctAnswers++;
247.                                JOptionPane.showMessageDialog(this, "You have " + correctAnswers + " correct answers!", "Correct
    Answer", JOptionPane.INFORMATION_MESSAGE);
248.
249.                        } else {
250.                                JOptionPane.showMessageDialog(this, "Wrong Answer!", "Wrong Answer", JOptionPane.INFORMATION_MESSAGE);
251.                        }
252.
253.                        //check if game one has finished
254.                        if (gameNumber == 1 && currentQuestion == 20){
255.                                long finish = System.currentTimeMillis();
256.                                gameOneScoreGenerator(correctAnswers, (int)((finish-start)/1000));
257.                        }
258.
259.                        long finish = System.currentTimeMillis();
260.                        String timeString = timerLabel.getText();
261.                        int timeInt = Integer.parseInt(timeString);
262.
```

```
263.                    if (gameNumber == 2 && timeInt > 60) {
264.                            gameTwoScoreGenerator(correctAnswers);
265.                    }
266.
267.                    if (gameNumber == 3 && leftTime < 0) {
268.                            gameThreeScoreGenerator((int)((finish-start)/1000));
269.                    }
270.            }
271.
272.            /**
273.             * The method gameOneScoreGenerator calculates the score of game one by using the time taken by the student
274.             * to answer 20 questions and the amount of correct answers achieved.
275.             * As soon as the score is calculated a GameRecordNode object is created with the details of the game played
276.             * The student is given it's score and the game finishes by taking the student to the main menu
277.             * @param correctAnswers the amount of correct answers achieved
278.             * @param time the time taken by the student to complete the 20 seconds
279.             *
280.             */
281.            private void gameOneScoreGenerator(int correctAnswers, int time){
282.
283.                    int percentageCorrectAnswers = correctAnswers*5;
284.
285.                    double maximumTime = 500;
286.                    double actualTimeTaken = time;
287.                    double bonusScore = (double)(maximumTime/actualTimeTaken);
288.                    double finalScore = (int)(((percentageCorrectAnswers*bonusScore)/2));
289.
290.                    if (finalScore > 100){
```

```
291.                            finalScore = 100;
292.                    }
293.
294.                    GameRecordNode gameRecord = new GameRecordNode(id, System.currentTimeMillis(), finalScore + "");
295.                    GameRecordGenerator generator = new GameRecordGenerator();
296.                    try {
297.                            generator.createGameRecord(gameRecord, gameNumber);
298.                    } catch (IOException e){
299.                            e.getMessage();
300.                    }
301.
302.                    JOptionPane.showMessageDialog(this, finalScore , "Final Score Game One", JOptionPane.INFORMATION_MESSAGE);
303.                    backToMainMenu();
304.            }
305.
306.            /**
307.             * The method gameTwoScoreGenerator records the score of the game by storing the amount of correct answers achieved in 60
    seconds.
308.             * As soon as the score is calculated a GameRecordNode object is created with the details of the game played
309.             * The student is given it's score and the game finishes by taking the student to the main menu
310.             * @param correctAnswers the amount of correct answers achieved
311.             *
312.             */
313.            private void gameTwoScoreGenerator(int correctAnswers){
314.
315.                    GameRecordNode gameRecord = new GameRecordNode(id, System.currentTimeMillis(), correctAnswers + "");
316.                    GameRecordGenerator generator = new GameRecordGenerator();
317.                    try {
```

```
318.                            generator.createGameRecord(gameRecord, gameNumber);
319.                    } catch (IOException e){
320.                            e.getMessage();
321.                    }
322.
323.                    JOptionPane.showMessageDialog(this, "You answered " + correctAnswers + " correct answer in 60 seconds!
   Congratulations"  , "Final Score Game Two", JOptionPane.INFORMATION_MESSAGE);
324.
325.                    backToMainMenu();
326.
327.            }
328.
329.            /**
330.             * The method gameThreeScoreGenerator uses the time survived on the game as the score of the student.
331.             * As soon as the score is calculated a GameRecordNode object is created with the details of the game played
332.             * The student is given it's score and the game finishes by taking the student to the main menu
333.             * @param time the time survived by the student in Game 3
334.             *
335.             */
336.            private void gameThreeScoreGenerator(int time){
337.
338.                    GameRecordNode gameRecord = new GameRecordNode(id, System.currentTimeMillis(), time + "");
339.                    GameRecordGenerator generator = new GameRecordGenerator();
340.                    try {
341.                            generator.createGameRecord(gameRecord, gameNumber);
342.                    } catch (IOException e){
343.                            e.getMessage();
344.                    }
```

```
345.
346.                         JOptionPane.showMessageDialog(this, "You survived a total of " + time + "" + " seconds. Congratulations"  , "Final
    Score Game Three", JOptionPane.INFORMATION_MESSAGE);
347.
348.                         backToMainMenu();
349.                 }
350.
351.                 //takes you back to main menu -quits game
352.                 public void backToMainMenu(){
353.                         JFrame frame = new JFrame();
354.                         frame.getContentPane().add(new StudentsMainMenuPanel(frame, id));
355.                         frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
356.                         frame.pack();
357.                         frame.setLocation(380, 180);
358.                         frame.setVisible(true);
359.                         this.frame.setVisible(false);
360.                         this.frame.dispose();
361.                 }
362.
363.         }
```

```
1.  * @program Math Speed Game
2.  * @class MathQuestion
3.  * @author nahel.rifai
4.  * @description This class is used to generate the 20 questions needed for each math game.
5.  * generated by the MathQuestion class.
6.  * @Date 23/8/2012
7.  * @School Markham College, Lima, Peru
8.  * @IDE Eclipse (version Indigo)
9.  * @IBCode 000633054
10. */
11.
12. public class MathQuestion {
13.
14.         protected double operandOne;
15.         protected double operandTwo;
16.
17.         /**
18.          * each time the constructor is called two new double numbers (with no current value) are set as the operands
19.          * that will be used in each of the math questions generated
20.          * @param operandOne the first operand
21.          * @param operandTwo the second operand
22.          */
23.         public MathQuestion(double operandOne, double operandTwo) {
24.                 super();
25.                 this.operandOne = operandOne;
26.                 this.operandTwo = operandTwo;
27.         }
28.
```

```
29.         /**
30.          * The operands are assigned random numbers by executing the generateNumber method on each of them.
31.          */
32.         public MathQuestion(){
33.                 this.operandOne = generateNumber();
34.                 this.operandTwo = generateNumber();
35.         }
36.
37.         public double getAnswer(){
38.                 return 0.0;
39.         }
40.
41.         public String getDisplay(){
42.                 return "";
43.         }
44.
45.         /**
46.          * This method uses the Math.random operation to generate a random number between 0 and 1000
47.          * @return the number that will be assigned to the operand used in the operation
48.          */
49.         public double generateNumber(){
50.                 return ((int)(Math.random()*100000))/100.0;
51.         }
52. }
53. import java.io.IOException;
54.
55. /**
56.  * @program Math Speed Game
```

```
57.  * @class Addition Question
58.  * @author nahel.rifai
59.  * @description This class is used to generate an addition question using the two operands (random numbers)
60.  * generated by the Math Question class.
61.  * @Date 23/8/2012
62.  * @School Markham College, Lima, Peru
63.  * @IDE Eclipse (version Indigo)
64.  * @IBCode 000633054
65.  */
66.
67. public class AdditionQuestion extends MathQuestion {
68.
69.         /**
70.          * This method uses the two random numbers generated by the Math Question class (where the AdditionQuestion class was called)
71.          * It creates an addition operation which outputs a number with less than 2 decimal places.
72.          */
73.         public AdditionQuestion(){
74.                 super();
75.
76.                 double answer = operandOne + operandTwo;
77.
78.                 String text = Double.toString(Math.abs(answer));
79.                 int integerPlaces = text.indexOf('.');
80.                 int decimalPlaces = (text.length() - integerPlaces - 1);
81.
82.                 while (decimalPlaces > 2){
83.
84.                         this.operandOne = generateNumber();
```

```
85.                         this.operandTwo = generateNumber();
86.
87.                         double newAnswer = operandOne + operandTwo;
88.
89.                         text = Double.toString(Math.abs(newAnswer));
90.                         integerPlaces = text.indexOf('.');
91.                         decimalPlaces = (text.length() - integerPlaces - 1);
92.
93.
94.
95.                 }
96.         }
97.
98.         /**
99.          * This method returns the answer of the addition operation
100.                 */
101.                public double getAnswer(){
102.                        return operandOne + operandTwo;
103.                }
104.
105.                /**
106.                 * This method returns the display of the addition operation for the user to read and calculate (in the GUI interface)
107.                 */
108.                public String getDisplay (){
109.                        return operandOne + " + " + operandTwo + " = ";
110.
111.                }
112.        }
```

```
1.  import java.io.IOException;
2.
3.  /**
4.   * @program Math Speed Game
5.   * @class Subtraction Question
6.   * @author nahel.rifai
7.   * @description This class is used to generate a subtraction question using the two operands (random numbers)
8.   * generated by the Math Question class.
9.   * @Date 23/8/2012
10.  * @School Markham College, Lima, Peru
11.  * @IDE Eclipse (version Indigo)
12.  * @IBCode 000633054
13.  */
14.
15. public class SubstractionQuestion extends MathQuestion {
16.
17.         /**
18.          * This method uses the two random numbers generated by the Math Question class (where the SubstractionQuestion class was called)
19.          * It creates an subtraction operation which outputs an answer with less than 2 decimal places.  It also requires that the operand one
    (the first operand)
20.          * is bigger than the second operand so the answer is a number bigger than 1 (reducing the complexity of the problem)
21.          */
22.         public SubstractionQuestion(){
23.
24.                 super();
25.
26.                 double holdtemp = 0;
27.
```

```
28.                 if (operandTwo > operandOne){
29.                         holdtemp = operandTwo;
30.                         operandTwo = operandOne;
31.                         operandOne = holdtemp;
32.                 }
33.
34.                 double answer = operandOne - operandTwo;
35.
36.                 String text = Double.toString(Math.abs(answer));
37.                 int integerPlaces = text.indexOf('.');
38.                 int decimalPlaces = (text.length() - integerPlaces - 1);
39.
40.                 while (decimalPlaces > 2){
41.
42.                         this.operandOne = generateNumber();
43.                         this.operandTwo = generateNumber();
44.
45.                         double newAnswer = operandOne - operandTwo;
46.
47.                         text = Double.toString(Math.abs(newAnswer));
48.                         integerPlaces = text.indexOf('.');
49.                         decimalPlaces = (text.length() - integerPlaces - 1);
50.
51.
52.                 }
53.         }
54.
55.         /**
```

```
56.          * Returns the answer of the division question
57.          */
58.         public double getAnswer(){
59.                 return operandOne - operandTwo;
60.         }
61.
62.         /**
63.          * This method returns the display of the subtraction operation for the user to read and calculate (in the GUI interface)
64.          */
65.         public String getDisplay (){
66.                 return operandOne + " - " + operandTwo + " = ";
67.
68.         }
69. }
```

```
1. import java.io.IOException;
2. 
3. /**
4.  * @program Math Speed Game
5.  * @class MultiplicationQuestion
6.  * @author nahel.rifai
7.  * @description This class is used to generate a multiplication question using the two operands (random numbers)
8.  * generated by the MathQuestion class.
9.  * @Date 23/8/2012
10. * @School Markham College, Lima, Peru
11. * @IDE Eclipse (version Indigo)
12. * @IBCode 000633054
13. */
14. 
15. public class MultiplicationQuestion extends MathQuestion {
16. 
17.         /**
18.          * This constructor uses the two random numbers generated by the Math Question class (where the MultiplicationQuestion class was
    called)
19.          * It creates a multiplication problem with no more than 2 decimal places in the answer by setting the appropriate operands.
20.          */
21.         public MultiplicationQuestion(){
22. 
23.                 super();
24. 
25.                 double answer = operandOne * operandTwo;
26. 
27.                 String text = Double.toString(Math.abs(answer));
```

```
28.                 int integerPlaces = text.indexOf('.');
29.                 int decimalPlaces = (text.length() - integerPlaces - 1);
30.
31.                 while (decimalPlaces > 2){
32.
33.                         this.operandOne = generateNumber();
34.                         this.operandTwo = generateNumber();
35.
36.                         double newAnswer = operandOne * operandTwo;
37.
38.                         text = Double.toString(Math.abs(newAnswer));
39.                         integerPlaces = text.indexOf('.');
40.                         decimalPlaces = (text.length() - integerPlaces - 1);
41.
42.
43.                 }
44.         }
45.
46.         /**
47.          * This method returns the answer of the multiplication operation
48.          */
49.         public double getAnswer(){
50.                 return operandOne * operandTwo;
51.         }
52.
53.         /**
54.          * This method returns the display of the division operation for the user to read and calculate (in the GUI interface)
55.          */
```

```
56.         public String getDisplay (){
57.                 return (int)operandOne + " x " + (int)operandTwo + " = ";
58.
59.         }
60.
61.         /**
62.          * This method generates a new random number if the multiplication operation results in a number with more than 2 decimal places.
63.          */
64.         public double generateNumber(){
65.                 return ((int)(Math.random()*100));
66.         }
67. }
```

```
1.  import java.io.IOException;
2.
3.  /**
4.   * @program Math Speed Game
5.   * @class DivisionQuestion
6.   * @author nahel.rifai
7.   * @description This class is used to generate a division question using the two operands (random numbers)
8.   * generated by the MathQuestion class.
9.   * @Date 23/8/2012
10.  * @School Markham College, Lima, Peru
11.  * @IDE Eclipse (version Indigo)
12.  * @IBCode 000633054
13.  */
14.
15. public class DivisionQuestion extends MathQuestion {
16.
17.
18.         /**
19.          * This method uses the two random numbers generated by the Math Question class (where the DivisionQuestion class was called)
20.          * It creates a division problem with no decimal places in the answer. It also requires that the operand one (the first operand)
21.          * is bigger than the second operand so the answer is a number bigger than 1 (reducing the complexity of the problem)
22.          */
23.         public DivisionQuestion(){
24.
25.                 super();
26.
27.                 double holdtemp = 0;
28.
```

```
29.             if (operandTwo > operandOne){
30.                     holdtemp = operandTwo;
31.                     operandTwo = operandOne;
32.                     operandOne = holdtemp;
33.             }
34.
35.             double answer = operandOne / operandTwo;
36.
37.             String text = Double.toString(Math.abs(answer));
38.             int integerPlaces = text.indexOf('.');
39.             int decimalPlaces = (text.length() - integerPlaces - 1);
40.
41.             while (decimalPlaces > 2){
42.
43.                     this.operandOne = generateNumber();
44.                     this.operandTwo = generateNumber();
45.
46.                     if (operandTwo > operandOne){
47.                             holdtemp = operandTwo;
48.                             operandTwo = operandOne;
49.                             operandOne = holdtemp;
50.                     }
51.
52.                     double newAnswer = operandOne / operandTwo;
53.
54.                     text = Double.toString(Math.abs(newAnswer));
55.                     integerPlaces = text.indexOf('.');
56.                     decimalPlaces = (text.length() - integerPlaces - 1);
```

```
57.
58.
59.                 }
60.         }
61.
62.         /**
63.          * This method returns the answer of the division operation
64.          */
65.         public double getAnswer(){
66.
67.                 return operandOne/operandTwo;
68.
69.         }
70.
71.
72.         /**
73.          * This method returns the display of the division operation for the user to read and calculate (in the GUI interface)
74.          */
75.         public String getDisplay (){
76.
77.                 return (int)operandOne + " ÷ " + (int)operandTwo + " = ";
78.         }
79.
80.         /**
81.          * This method generates a new random number with less than 2 decimal places.
82.          * If the number is equal to zero it generates a new one
83.          */
84.         public double generateNumber(){
```

```
85.
86.                 double number = ((int)(Math.random()*200)/2);
87.                 while (number == 0){
88.                         number = ((int)(Math.random()*100)/2);
89.                 }
90.                 return number;
91.         }
92. }
```

```
1. import java.io.*;
2. import java.util.StringTokenizer;
3.
4. import javax.swing.JFrame;
5.
6. /**
7.  * @program Math Speed Game
8.  * @class Game Record Generator
9.  * @class This class is used to save each game played by a student in it's corresponding file. The creategameRecord method
10. * is responsible of this.It is also used to get all the game records from a specific game type (either game 1, 2 or 3)
11. * through the getGameRecords method. It may also be used to get all the game records from a specific game type in order
12. * (from high to low) through the getGameRecordsInOrder method (used for high scores).
13. * @author nahel.rifai
14. * @Date 23/8/2012
15. * @School Markham College, Lima, Peru
16. * @IDE Eclipse (version Indigo)
17. * @IBCode 000633054
18. *
19. */
20.
21. public class GameRecordGenerator  {
22.
23.         /**
24.          * This method adds an new game record to it's corresponding game record file (either game 1, 2 or 3).
25.          * @param gameRecord is the game record object including the date, time, student id and score of the game trial
26.          * @param gameNumber the game type played (game 1, 2 or 3)
27.          * @throws IOException if file error occurs
28.          */
```

```
29.         public void createGameRecord(GameRecordNode gameRecord, int gameNumber) throws IOException {
30.                 File gameRecordsFile = null;
31.                 if (gameNumber == 1){
32.                         gameRecordsFile = new File("GameOneRecords.txt");
33.                 } else if (gameNumber ==2){
34.                         gameRecordsFile = new File("GameTwoRecords.txt");
35.                 } else {
36.                         gameRecordsFile = new File("GameThreeRecords.txt");
37.                 }
38.
39.                 File tempFile = new File("GameRecordsTemp.txt");
40.
41.                 FileReader read = new FileReader(gameRecordsFile);
42.                 BufferedReader buffReader = new BufferedReader(read);
43.
44.                 FileWriter write = new FileWriter(tempFile);
45.                 BufferedWriter buffWrite = new BufferedWriter(write);
46.
47.                 boolean eof = false;
48.                 while (!eof){
49.                         String line = buffReader.readLine();
50.                         if (line == null){
51.                                 //end of file reached
52.                                 eof = true;
53.                                 buffWrite.write(gameRecord.getId()+"|"+ gameRecord.getDate() +"|" + gameRecord.getScore());
54.                                 buffWrite.newLine();
55.
56.                         } else {
```

```
57.                             buffWrite.write(line);
58.                             buffWrite.newLine();
59.                         }
60.                 }
61.
62.                 //close readers
63.                 buffReader.close();
64.                 read.close();
65.                 //close writers
66.                 buffWrite.close();
67.                 write.close();
68.
69.                 //Replace gameRecords file with temp file
70.                 gameRecordsFile.delete();
71.                 tempFile.renameTo(gameRecordsFile);
72.         }
73.
74.         /**
75.          * This method creates a list of all the records played in a specific game type (1, 2 or 3) by a specific student.
76.          * @param id the identification number of the student
77.          * @param gameNumber the game type played (game 1, 2 or 3)
78.          * @return a list of all the games (of the type chosen - either 1, 2 or 3) played by a student
79.          * @throws IOException if file error occurs
80.          */
81.
82.         public GameRecordList getGameRecords(String id, int gameNumber) throws IOException {
83.
84.                 File gameRecordsFile = null;
```

```
85.                 GameRecordList list = new GameRecordList();
86.
87.                 if (gameNumber == 1){
88.                         gameRecordsFile = new File("GameOneRecords.txt");
89.                 } else if (gameNumber == 2){
90.                         gameRecordsFile = new File("GameTwoRecords.txt");
91.                 } else {
92.                         gameRecordsFile = new File("GameThreeRecords.txt");
93.                 }
94.
95.                 FileReader read = new FileReader(gameRecordsFile);
96.                 BufferedReader buffReader = new BufferedReader(read);
97.
98.                 boolean eof = false;
99.
100.                        while (!eof){
101.                                String line = buffReader.readLine();
102.                                if (line == null){
103.                                        //end of file reached
104.                                        eof = true;
105.                                } else {
106.
107.                                        StringTokenizer tokcheckid = new StringTokenizer(line, "|");
108.
109.                                        if (id.equals(tokcheckid.nextToken())) {
110.
111.                                                StringTokenizer tok = new StringTokenizer(line, "|");
112.                                                String idRecord = tok.nextToken();
```

```
113.                                    long date = Long.parseLong(tok.nextToken());
114.                                    String score = tok.nextToken();
115.
116.                                    GameRecordNode node = new GameRecordNode(idRecord, date, score);
117.                                    list.addNodeToHead(node);
118.                                }
119.                            }
120.                    }
121.                    return list;
122.            }
123.
124.            /**
125.             * This method generates a high scores game record list of one game type
126.             * @param gameNumber is the game type from which the high scores will be generated (either game 1, 2 or 3)
127.             * @return a ordered list (from highest score to lowest score) of a game type
128.             * @throws IOException if file error occurs
129.             */
130.            public GameRecordList getGameRecordsInOrder(int gameNumber) throws IOException {
131.
132.                    File gameRecordsFile = null;
133.                    GameRecordList list = new GameRecordList();
134.
135.                    if (gameNumber == 1){
136.                            gameRecordsFile = new File("GameOneRecords.txt");
137.                    } else if (gameNumber == 2){
138.                            gameRecordsFile = new File("GameTwoRecords.txt");
139.                    } else {
140.                            gameRecordsFile = new File("GameThreeRecords.txt");
```

```
141.                        }
142.
143.                        FileReader read = new FileReader(gameRecordsFile);
144.                        BufferedReader buffReader = new BufferedReader(read);
145.
146.                        boolean eof = false;
147.                        while (!eof){
148.                                String line = buffReader.readLine();
149.
150.                                if (line == null){
151.                                        //end of file reached
152.                                        eof = true;
153.
154.                                } else {
155.
156.                                        StringTokenizer tok = new StringTokenizer(line, "|");
157.                                        String idRecord = tok.nextToken();
158.                                        long date = Long.parseLong(tok.nextToken());
159.                                        String score = tok.nextToken();
160.
161.                                        GameRecordNode node = new GameRecordNode(idRecord, date, score);
162.                                        list.addNodeInOrder(node);
163.                                }
164.                        }
165.                        return list;
166.                }
167.        }
168.
```

```
169.  /**
170.   * @program Math Speed Game
171.   * @class GameRecordList
172.   * @author nahel.rifai
173.   * @description This class holds the head of the GameRecordList and the methods involving this list
174.   * @Date 23/8/2012
175.   * @School Markham College, Lima, Peru
176.   * @IDE Eclipse (version Indigo)
177.   * @IBCode 000633054
178.   */
179.
180.  public class GameRecordList {
181.
182.          private GameRecordNode head;
183.
184.          /**
185.           * Counts the number of nodes in the GameRecordList
186.           * @return the number of nodes in the list
187.           */
188.          public int getSize (){
189.
190.                  GameRecordNode current = head;
191.                  int count = 0;
192.                  while (current != null){
193.                          count = count +1;
194.                          current = current.getNext();
195.                  }
196.                  return count;
```

```
197.            }
198.
199.            /**
200.             * Returns the head Node of the GameRecordList
201.             * @return the head Node
202.             */
203.            public GameRecordNode getHead(){
204.                    return head;
205.            }
206.
207.            /**
208.             * Checks if the list if empty
209.             * @return if it is empty it returns true, otherwise it returns false
210.             */
211.            public boolean isEmpty() {
212.                    return head == null;
213.            }
214.
215.            /**
216.             * Adds a new node to the head of the list
217.             * @param node the node being added to the list
218.             */
219.            public void addNodeToHead(GameRecordNode node) {
220.                    if (isEmpty()){
221.                            head = node;
222.                    } else {
223.                            node.setNext(head);
224.                            head = node;
```

```
225.                         }
226.                 }
227.
228.                 /**
229.                  * This method creates a list of Game Record Nodes in order by score (from highest to lowest), it is used for the
    display of high scores
230.                  * @param node the new node being added
231.                  */
232.                 public void addNodeInOrder(GameRecordNode node) {
233.
234.                         if (isEmpty()){
235.                                 //empty list
236.                                 head = node;
237.                         } else{
238.                                 //not empty
239.                                 GameRecordNode current = head;
240.                                 boolean added = false;
241.                                 //check if should go before head
242.
243.                                 double nodeScore = Double.parseDouble(node.getScore());
244.                                 double currentScore = Double.parseDouble(current.getScore());
245.
246.                                 if (nodeScore > currentScore){
247.                                         node.setNext(head);
248.                                         head = node;
249.                                         added = true;
250.                                 }
251.
```

```
252.                            if(!added){
253.
254.                                    while (current.getNext() != null){
255.
256.                                            double nodeScoreTwo = Double.parseDouble(node.getScore());
257.                                            double currentNextScore = Double.parseDouble(current.getNext().getScore());
258.
259.                                            if (nodeScoreTwo > currentNextScore){
260.                                                    node.setNext(current.getNext());
261.                                                    current.setNext(node);
262.                                                    added = true;
263.                                                    break;
264.
265.                                            }
266.                                            current = current.getNext();
267.                                    }
268.                                    if (!added){
269.                                            current.setNext(node);
270.                                            added = true;
271.                                    }
272.                            }
273.                    }
274.            }
275.    }
```

```
1. /**
2.  * @program Math Speed Game
3.  * @class GameRecordNode
4.  * @author nahel.rifai
5.  * @description This class holds the information recorded in each Game Record Node object, including the pointer to the next Game Record Node
    on the list
6.  * @Date 23/8/2012
7.  * @School Markham College, Lima, Peru
8.  * @IDE Eclipse (version Indigo)
9.  * @IBCode 000633054
10. */
11.
12. public class GameRecordNode {
13.
14.         private String id;
15.         private long date;
16.         private String score;
17.         private GameRecordNode next;
18.
19.         /**
20.          * The constructor sets the id, date and score of the new GameRecordNode object
21.          * @param id the id of the student who played the game
22.          * @param date the date in which the game was played
23.          * @param score the score of the played game
24.          */
25.         public GameRecordNode(String id, long date, String score) {
26.                 super();
27.                 this.id = id;
```

```
28.                 this.date = date;
29.                 this.score = score;
30.         }
31.
32.         /**
33.          * Returns the id of the student that played the game
34.          * @return the student id
35.          */
36.         public String getId() {
37.                 return id;
38.         }
39.
40.         /**
41.          * sets the id of the student that played the game
42.          * @param id id of the student
43.          */
44.         public void setId(String id) {
45.                 this.id = id;
46.         }
47.
48.         /**
49.          * gets the exact date in which the game was played
50.          * @return the date
51.          */
52.         public long getDate() {
53.                 return date;
54.         }
55.
```

```
56.         /**
57.          * Sets the date in which the game was played
58.          * @param date
59.          */
60.         public void setDate(int date) {
61.                 this.date = date;
62.         }
63.
64.         /**
65.          * gets the score obtained by the student in the game
66.          * @return the score
67.          */
68.         public String getScore() {
69.                 return score;
70.         }
71.
72.         /**
73.          * sets the score obtained by the student in the game
74.          * @param the score
75.          */
76.         public void setScore(String score) {
77.                 this.score = score;
78.         }
79.
80.         /**
81.          * Gets the next GameRecordNode in the list
82.          * @return the next GameRecordNode
83.          */
```

```
84.         public GameRecordNode getNext() {
85.                 return next;
86.         }
87.
88.         /**
89.          * Sets the pointer to the next GameRecordNode
90.          * @param next the next GameRecordNode that follows in the list
91.          */
92.         public void setNext(GameRecordNode next) {
93.                 this.next = next;
94.         }
95. }
```

```
1.  import java.io.*;
2.
3.  /**
4.   * @program Math Speed Game
5.   * @class StudentHashFile
6.   * @class This class is used to manage the records of the students in the random access file StudentRecords.dat. This class
7.   * creates a new random access file if it doesn't exist. It sets the correct size and record length and fills with the char "x"
8.   * all the empty fields in the file. It involves processes such as login (where a boolean is returned) and the add student method.
9.   * @author nahel.rifai
10.  * @Date 23/8/2012
11.  * @School Markham College, Lima, Peru
12.  * @IDE Eclipse (version Indigo)
13.  * @IBCode 000633054
14.  *
15.  */
16.
17. public class StudentHashFile {
18.
19.         private static final int SIZE = 50; //the size of the record
20.         private static final int RECORD_LENGHT = 94;//the record length of each record
21.         private static final String FILE_NAME = "StudentRecords.dat";//the name of the random access file
22.         private RandomAccessFile file;
23.
24.         /**
25.          * The constructor looks for the random access file
26.          * if it doesn't exist it creates a new one
27.          * @throws IOException If there is a file error
28.          */
```

```
29.         public StudentHashFile() throws IOException {
30.                 File f = new File(FILE_NAME);
31.
32.                 if(!f.exists()){
33.                         createFile();//if file doesn't exist the file is created
34.                 }
35.                 file = new RandomAccessFile(f, "rw");
36.         }
37.
38.         /**
39.          * A new file is created by this method
40.          * the empty fields are filled with "x"
41.          */
42.         private void createFile(){
43.
44.                 try{
45.                         file = new RandomAccessFile(FILE_NAME, "rw");
46.
47.                         for (int i = 0; i < SIZE; i++){
48.                                 file.seek(i*RECORD_LENGHT);
49.                                 file.writeUTF("x"); //first name of student
50.                                 file.writeUTF("x"); // last name
51.                                 file.writeUTF("x"); // id
52.                                 file.writeUTF("x"); //password
53.                         }
54.
55.                         file.close();
56.
```

```
57.                 } catch (IOException e){
58.                         System.out.println("error adding record");
59.                 }
60.         }
61.
62.         // mastery Higher Level 1 - Random Access File
63.         /**
64.          * Adds a student object into the random access file
65.          * @param newStudent the student object being added
66.          * @throws IOException if there is a file error
67.          */
68.         public void addStudent (StudentNode newStudent) throws IOException{
69.
70.                 int id = Integer.parseInt(newStudent.getiD());  //id of the student
71.
72.                 file.seek(id*RECORD_LENGHT);//position of record in random access file
73.                 file.writeUTF(newStudent.getFirstName());//first name
74.                 file.writeUTF(newStudent.getLastName());//last name
75.                 file.writeUTF(newStudent.getiD());//id
76.                 file.writeUTF(newStudent.getPassword());//password
77.         }
78.
79.         /**
80.          * This method generates a new id for the created student (this is done to ensure every student has a different id)
81.          * @return the id that will be assigned to the created student
82.          * @throws IOException if there is a file error
83.          */
84.         public int generateID() throws IOException {
```

```
85.
86.                 for (int i = 0; i < SIZE; i++){//searches the random access file
87.
88.                         file.seek(i*RECORD_LENGHT);
89.
90.                         if (file.readUTF().equals("x")){
91.                                 return i;//as soon as it finds an empty record it returns the position of the record (equals to the id of the
    new student)
92.                         }
93.                 }
94.                 return -1;//if there are no empty positions a "-1" is returned
95.         }
96.
97.         /**
98.          * This method validates the login of the student into the application
99.          * @param id the id of the student (serves as user name)
100.                 * @param password the password of the student
101.                 * @return if id and password match a "true" boolean is returned, otherwise a "false" boolean is returned
102.                 * @throws IOException if there is a file error
103.                 */
104.                public boolean login(String id, String password) throws IOException{
105.
106.                        file.seek(Integer.parseInt(id)*RECORD_LENGHT);//position of record (searched by id)
107.                        file.readUTF();//skips first name
108.                        file.readUTF();//skips last name
109.
110.                        if (file.readUTF().equals(id) && file.readUTF().equals(password)){
111.                                return true;//login granted
```

```
112.                     }
113.                     return false;//login not granted
114.             }
115.
116.             /**
117.              * returns the
118.              * @param id the id of the student which name must be returned
119.              * @return the first name of the student
120.              * @throws IOException if there is a file error
121.              */
122.             public String getFirstName(String id) throws IOException {
123.
124.                     file.seek(Integer.parseInt(id)*RECORD_LENGHT);
125.                     return file.readUTF();//returns first name
126.             }
127.
128.             /**
129.              * Creates a StudentList with the records in the random access file storing all the students in the class
130.              * @return a list of students (in order by id - as they are stored in order)
131.              * @throws IOException if there is a file error
132.              */
133.             public StudentList getStudentRecords() throws IOException {
134.
135.                     StudentList list = new StudentList();
136.
137.                     for (int i = 0; i < SIZE; i++){
138.
139.                             file.seek(i*RECORD_LENGHT);
```

```
140.                            String firstName = file.readUTF();
141.
142.                            if (!firstName.equals("x")){//if the field in not empty
143.
144.                                    String LastName = file.readUTF();
145.                                    String id = file.readUTF();
146.                                    String password = file.readUTF();
147.
148.                                    StudentNode node = new StudentNode(firstName, LastName, id, password);// a student object is created
149.
150.                                    list.addNodeInOrder(node);//it is added to the list
151.                            }
152.                    }
153.                    return list;//the complete list of students is returned
154.            }
155.
156.            /**
157.             * Gets the full name of the student by giving the id
158.             * @param id the id of the student
159.             * @return the full name of the student (LastName, FirstName)
160.             * @throws IOException if there is a file error
161.             */
162.            public String getNamebyID(String id) throws IOException {
163.
164.                    file.seek(Integer.parseInt(id)*RECORD_LENGHT);
165.
166.                    String firstName = file.readUTF();
167.                    String lastName = file.readUTF();
```

```
168.
169.                         return lastName + ", " + firstName;
170.                 }
171.         }
```

```java
/**
 * @program Math Speed Game
 * @class StudentList
 * @author nahel.rifai
 * @description This class holds the head of the StudentList and the methods involving this list such as the getSize, the
 * addNodeInOrder and the getHead methods.
 * @Date 23/8/2012
 * @School Markham College, Lima, Peru
 * @IDE Eclipse (version Indigo)
 * @IBCode 000633054
 */
public class StudentList {

	private StudentNode head;// the hea noded of the StudentList

	/**
	 * If the head is null (empty) - there are no StudentNode objects on the list, this means the list is empty
	 * @return true is returned if the list is empty, otherwise a false boolean is returned.
	 */
	public boolean isEmpty() {
		return head == null;
	}

	/**
	 * This method returns the first element in the StudentList
	 * @return the head of the list (StudentNode object)
	 */
	public StudentNode getHead() {
```

```
29.                 return head;
30.         }
31.
32.         /**
33.          * This method counts the number of elements in the list
34.          * @return the amount of StudentNode objects currently in the list
35.          */
36.         public int getSize (){
37.
38.                 StudentNode current = head;
39.                 int count = 0;
40.                 while (current != null){
41.                         count = count +1;
42.                         current = current.getNext();
43.                 }
44.                 return count;
45.         }
46.
47.         /**
48.          * The following method adds a StudentNode in it's corresponding position in the StudentList
49.          * @param node the student node being added to the list
50.          */
51.         public void addNodeInOrder(StudentNode node) {
52.
53.                 if (isEmpty()){
54.                         //empty list
55.                         head = node; //the new node becomes the head of the list
56.                 } else{
```

```
57.                         //not empty
58.                         StudentNode current = head; //the current node being compared to
59.                         int currentID = Integer.parseInt(current.getiD()); //the IDs are parsed into int form so they can be numerically
    compared
60.                         int newNodeID = Integer.parseInt(node.getiD());
61.                         boolean added = false;
62.                         //check if should go before head
63.                         if (newNodeID < currentID){ //if the new node ID is smaller than the current one being compared to then it is placed
    before it
64.                                 node.setNext(head);
65.                                 head = node;
66.                                 added = true;
67.                         }
68.
69.                         if(!added){
70.                                 //while the list has not ended yet
71.                                 while (current.getNext() != null){
72.                                         int nextID = Integer.parseInt(current.getNext().getiD());
73.                                         if (newNodeID < nextID){ //if the new node ID is smaller than the current one being compared to then
    it is placed before it
74.                                                 node.setNext(current.getNext());
75.                                                 current.setNext(node);
76.                                                 added = true;
77.                                                 break;
78.                                         }
79.                                         current = current.getNext();
80.                                 }
81.                                 if (!added){
```

```
82.                                     current.setNext(node);//if it has not been added yet it means that it belongs at the end of the list
83.                                     added = true;
84.                                 }
85.                             }
86.
87.                     }
88.
89.             }
90.
91.             /**
92.              * Constructs the student node object
93.              * @param firstName the first name of the student
94.              * @param lastName the last name of the student
95.              * @param iD the ID of the student
96.              * @param password the password of the student
97.              */
98.             public void addStudent(String firstName, String lastName, String iD, String password) {
99.                     StudentNode node = new StudentNode(firstName, lastName, iD, password);
100.                            addNodeInOrder(node);
101.                    }
102.
103.            }
```

```
1. /**
2.  * @program Math Speed Game
3.  * @class StudentNode
4.  * @author nahel.rifai
5.  * @description This class holds the information recorded in each StudentNode object, including the pointer to the next StudentNode on the
    list
6.  * @Date 23/8/2012
7.  * @School Markham College, Lima, Peru
8.  * @IDE Eclipse (version Indigo)
9.  * @IBCode 000633054
10. */
11.
12.
13. public class StudentNode {
14.
15.         private String firstName; //the first name of the student
16.         private String lastName;//the last name of the student
17.         private String iD;//the ID of the student
18.         private String password;//the password of the student
19.         private StudentNode next;//the next StudentNode object in the list
20.
21.         /**
22.          * This method constructs a student object by introducing the adequate parameters
23.          * @param firstName the first name of the student
24.          * @param lastName the last name of the student
25.          * @param iD the ID of the student
26.          * @param password the password of the student
27.          */
```

```
28.         public StudentNode(String firstName, String lastName, String iD, String password) {
29.                 super();
30.                 this.firstName = firstName;
31.                 this.lastName = lastName;
32.                 this.iD = iD;
33.                 this.password = password;
34.         }
35.
36.         //returns the first name of a student object
37.         public String getFirstName() {
38.                 return firstName;
39.         }
40.
41.         //returns the last name of the student object
42.         public String getLastName() {
43.                 return lastName;
44.         }
45.
46.         //returns the full name of the student object
47.         public String getName() {
48.                 return firstName + " " + lastName;
49.         }
50.
51.         //returns the ID of the student object
52.         public String getiD() {
53.                 return iD;
54.         }
55.
```

```
56.         //returns the password of the student object
57.         public String getPassword() {
58.                 return password;
59.         }
60.
61.         //returns the next StudetNode object in the list
62.         public StudentNode getNext() {
63.                 return next;
64.         }
65.
66.         //sets the next StudentNode object in the list
67.         public void setNext (StudentNode next) {
68.                 this.next = next;
69.         }
70.
71. }
```

```
1.
2.  import java.io.*;
3.  import java.awt.*;
4.  import java.awt.event.ActionEvent;
5.  import java.awt.event.ActionListener;
6.
7.  import javax.swing.*;
8.
9.  /**
10.  * @program Math Speed Game
11.  * @class PreviousScoresPanel
12.  * @author nahel.rifai
13.  * @description This class has auto-generated code by Jigloo, plugin program used to create the GUI interface panel that
14.  * displays the previous scores achieved in each type of game for each student. It creates a new table model
15.  * by using a list of the game records played by a specific student in a specific game type.
16.  * @Date 23/8/2012
17.  * @School Markham College, Lima, Peru
18.  * @IDE Eclipse (version Indigo)
19.  * @IBCode 000633054
20.  */
21.
22. /**
23.  * This code was edited or generated using CloudGarden's Jigloo
24.  * SWT/Swing GUI Builder, which is free for non-commercial
25.  * use. If Jigloo is being used commercially (ie, by a corporation,
26.  * company or business for any purpose whatever) then you
27.  * should purchase a license for each developer using Jigloo.
28.  * Please visit www.cloudgarden.com for details.
```

```
29.  * Use of Jigloo implies acceptance of these licensing terms.
30.  * A COMMERCIAL LICENSE HAS NOT BEEN PURCHASED FOR
31.  * THIS MACHINE, SO JIGLOO OR THIS CODE CANNOT BE USED
32.  * LEGALLY FOR ANY CORPORATE OR COMMERCIAL PURPOSE.
33.  */
34. public class PreviousScoresPanel extends javax.swing.JPanel {
35. 
36.         //variables auto-generated by Jigloo
37.         private JLabel previousScoresLabel;
38.         private JButton gameOneButton;
39.         private JButton gameTwoButton;
40.         private JButton gameThreeButton;
41.         private JButton backToMenuButton;
42.         private JFrame frame;
43.         private JTable previousScoresTable;
44.         private JScrollPane scrollPane;
45. 
46.         //variables not generated by Jigloo
47.         private String id; //the ID of the student logged in into the system
48.         private int gameNumber = 1;   //the game number which previous scores want to be accessed
49.         private PreviousScoresTableModel previousScoresTableModel = new PreviousScoresTableModel();
50.         private GameRecordGenerator gameRecords = new GameRecordGenerator();
51. 
52.         /**
53.          * Sets the previous frame, from the StudentsMainMenuPanel, as the current frame
54.          * The id of the student is passed from the StudentsMainMenuPanel and set as the current id in order to
55.          * get the game records of that specific student
56.          * @param frame the previous frame
```

```
57.          * @param id the student id
58.          */
59.         public PreviousScoresPanel(JFrame frame, String id) {
60.                 super();
61.                 this.frame = frame;
62.                 this.id = id;
63.                 try {
64.                         /**
65.                          * The previous scores table is generated by using the getGameRecords method in the GameRecordGenerator class.
66.                          * It outputs a list of the game records played by a specific students in a specific game type (game 1, 2 or 3).
67.                          */
68.                         previousScoresTableModel.setData(gameRecords.getGameRecords(id, gameNumber));
69.                 } catch (IOException e){
70.                         System.out.println("error setting table data");
71.                 }
72.                 initGUI();
73.         }
74.
75.         /**
76.          * Code in the initGUI method was automatically generated by Jigloo
77.          * The action listeners for the buttons were not generated by Jigloo
78.          */
79.         private void initGUI() {
80.                 try {
81.                         this.setPreferredSize(new java.awt.Dimension(500, 350));
82.                         GridBagLayout thisLayout = new GridBagLayout();
83.                         this.setSize(500, 350);
84.                         thisLayout.rowWeights = new double[] {0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1};
```

```
85.                         thisLayout.rowHeights = new int[] {7, 7, 7, 7, 7, 7, 7, 7, 7, 7};
86.                         thisLayout.columnWeights = new double[] {0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.
    1};
87.                         thisLayout.columnWidths = new int[] {7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7};
88.                         this.setLayout(thisLayout);
89.                         {
90.                                 previousScoresLabel = new JLabel();
91.                                 this.add(previousScoresLabel, new GridBagConstraints(6, 0, 4, 1, 0.0, 0.0, GridBagConstraints.CENTER, GridBagC
    onstraints.NONE, new Insets(0, 0, 0, 0), 0, 0));
92.                                 previousScoresLabel.setText("Previous Scores");
93.                                 previousScoresLabel.setFont(new java.awt.Font("Tahoma",1,22));
94.                         }
95.                         {
96.                                 gameOneButton = new JButton();
97.                                 this.add(gameOneButton, new GridBagConstraints(13, 3, 2, 1, 0.0, 0.0, GridBagConstraints.CENTER, GridBagConstr
    aints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
98.                                 gameOneButton.setText("Game One");
99.                                 gameOneButton.addActionListener(new ActionListener() {
100.                                        public void actionPerformed(ActionEvent evt) {
101.                                                gameNumber = 1;
102.                                                generateRecordsTable();
103.                                                //when this button is clicked the previous scores for game 1 are displayed
104.                                        }
105.                                });
106.                        }
107.                        {
108.                                gameTwoButton = new JButton();
```

```
109.                                    this.add(gameTwoButton, new GridBagConstraints(13, 5, 2, 1, 0.0, 0.0, GridBagConstraints.CENTER, Grid
    BagConstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
110.                                    gameTwoButton.setText("Game Two");
111.                                    gameTwoButton.addActionListener(new ActionListener() {
112.                                            public void actionPerformed(ActionEvent evt) {
113.                                                    gameNumber = 2;
114.                                                    generateRecordsTable();
115.                                                    //when this button is clicked the previous scores for game 2 are displayed
116.                                            }
117.                                    });
118.                            }
119.                            {
120.                                    gameThreeButton = new JButton();
121.                                    this.add(gameThreeButton, new GridBagConstraints(13, 7, 2, 1, 0.0, 0.0, GridBagConstraints.CENTER, Gr
    idBagConstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
122.                                    gameThreeButton.setText("Game Three");
123.                                    gameThreeButton.addActionListener(new ActionListener() {
124.                                            public void actionPerformed(ActionEvent evt) {
125.                                                    gameNumber = 3;
126.                                                    generateRecordsTable();
127.                                                    //when this button is clicked the previous scores for game 3 are displayed
128.                                            }
129.                                    });
130.                            }
131.                            {
132.                                    backToMenuButton = new JButton();
133.                                    this.add(backToMenuButton, new GridBagConstraints(0, 0, 1, 1, 0.0, 0.0, GridBagConstraints.CENTER, Gr
    idBagConstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
```

```
134.                                    backToMenuButton.setText("Back to Menu");
135.                                    backToMenuButton.addActionListener(new ActionListener() {
136.                                            public void actionPerformed(ActionEvent evt) {
137.                                                    backToMainMenu();//back to main menu
138.                                            }
139.                                    });
140.                            }
141.                            {
142.                                    scrollPane = new JScrollPane();
143.                                    this.add(scrollPane, new GridBagConstraints(0, 2, 11, 7, 0.0, 0.0, GridBagConstraints.CENTER, GridBag
    Constraints.BOTH, new Insets(0, 0, 0, 0), 0, 0));
144.                                    {
145.                                            previousScoresTable = new JTable();
146.                                            scrollPane.setViewportView(previousScoresTable);
147.                                            previousScoresTable.setModel(previousScoresTableModel);
148.                                    }
149.                            }
150.                    } catch (Exception e) {
151.                            e.printStackTrace();
152.                    }
153.            }
154.
155.            //takes you back to main menu
156.            public void backToMainMenu(){
157.                    JFrame frame = new JFrame();
158.                    frame.getContentPane().add(new StudentsMainMenuPanel(frame, id));
159.                    frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
160.                    frame.pack();
```

```
161.                    frame.setLocation(380, 180);
162.                    frame.setVisible(true);
163.                    this.frame.setVisible(false);
164.                    this.frame.dispose();
165.            }
166.
167.            /**
168.             * When the buttons to display the previous scores are clicked a new table model will be created, presenting the
169.             * records of the game type desired.
170.             */
171.            public void generateRecordsTable(){
172.                    try {
173.                            previousScoresTableModel.setData(gameRecords.getGameRecords(id, gameNumber));
174.                    } catch (IOException e){
175.                            System.out.println("error setting table data");
176.                    }
177.
178.            }
179.    }
```

```
1.  import javax.swing.table.*;
2.  import java.text.*;
3.  import java.util.*;
4.
5.  /**
6.   * @program Math Speed Game
7.   * @class PreviousScoresTableModel
8.   * @author nahel.rifai
9.   * @description This class constructs the previous scores table displayed to each student on it's account and to the admin
10.  * @Date 23/8/2012
11.  * @School Markham College, Lima, Peru
12.  * @IDE Eclipse (version Indigo)
13.  * @IBCode 000633054
14.  */
15.
16. public class PreviousScoresTableModel extends AbstractTableModel{
17.
18.         private GameRecordList list;//the list of game record objects
19.
20.         /**
21.          * sets the amount of columns in the table
22.          */
23.         public int getColumnCount() {
24.                 return 3;//3 columns
25.         }
26.
27.         /**
28.          * sets the columns names
```

```
29.          */
30.         public String getColumnName(int col){
31.                 if (col == 0){
32.                         return "Date";
33.                 } else if(col == 1){
34.                         return "Score";
35.                 } else {
36.                         return "% Improvement";
37.                 }
38.         }
39.
40.         /**
41.          * sets the amount of rows in the table
42.          */
43.         public int getRowCount() {
44.                 return list.getSize();//the amount of game record objects of each student
45.         }
46.
47.         /**
48.          * This method fills the previous scores table with the information of each record
49.          * provided by the GameRecordList (list of games from a specific game (either 1, 2 or 3) played by a student)
50.          */
51.         public Object getValueAt(int row, int col) {
52.
53.                 GameRecordNode temp = list.getHead();//the first game played by the student
54.
55.                 for (int i = 0; i < row; i++){
56.                         temp = temp.getNext();
```

```
57.                 }
58.
59.                 if (col == 0){
60.                         //gets the date in which the game was played
61.                         SimpleDateFormat sdf = new SimpleDateFormat("MMM dd,yyyy HH:mm");
62.                         Date resultdate = new Date(temp.getDate());
63.                         return sdf.format(resultdate);
64.
65.                 } else if ( col == 1){
66.                         //gets the score of the game
67.                         return temp.getScore();
68.                 } else if (col == 2){
69.                         //gets the percentage improvement from the previous game to the current one
70.                         if(temp.getNext() != null){
71.
72.                                 String previousscore = temp.getScore();//gets the previous score
73.                                 double doublepreviousscore = Double.parseDouble(previousscore);
74.
75.                                 String currentscore = temp.getNext().getScore();//gets the current score
76.                                 double doublecurrentscore = Double.parseDouble(currentscore);
77.
78.                                 double difference = doublepreviousscore - doublecurrentscore;//calculates the difference
79.
80.                                 double percentageimprovement = (difference/doublecurrentscore)*100.0;//calculates the percentage
    improvement
81.
82.                                 DecimalFormat twdf = new DecimalFormat("#.##");//lowers it down to two decimal places
83.
```

```
84.                                 return twdf.format(percentageimprovement);
85.                         }
86.                         return "First Game Played";//if it is the first game played by the student there is no percentage improvement
87.                 }
88.                 return null;
89.         }
90.
91.         /**
92.          * Resets the data in the table each time the program is refreshed
93.          * @param list the GameRecordList
94.          */
95.         public void setData(GameRecordList list){
96.                 this.list = list;
97.                 fireTableDataChanged();
98.         }
99. }
```

```
1.  import java.awt.*;
2.  import java.awt.event.ActionEvent;
3.  import java.awt.event.ActionListener;
4.  import java.io.IOException;
5.
6.  import javax.swing.*;
7.
8.  /**
9.   * @program Math Speed Game
10.  * @class ClassHighScoresPanel
11.  * @author nahel.rifai
12.  * @description This class has auto-generated code by Jigloo, plugin program used to create the GUI interface panel that
13.  * displays the high scores for the class in each type of game. It creates a new table model by using a list of the
14.  * game records in order (from high to low) from a specific game type. It displays the top 15 scores for each game.
15.  * Unlike the AminClassHighScoresPanel, the ClassHighScoresPanel is presented in the student's application. When they login,
16.  * they may choose to see the class' high scores and see if they have a achieved a top score on any of the games.
17.  * @Date 23/8/2012
18.  * @School Markham College, Lima, Peru
19.  * @IDE Eclipse (version Indigo)
20.  * @IBCode 000633054
21.  */
22.
23. /**
24.  * This code was edited or generated using CloudGarden's Jigloo
25.  * SWT/Swing GUI Builder, which is free for non-commercial
26.  * use. If Jigloo is being used commercially (ie, by a corporation,
27.  * company or business for any purpose whatever) then you
28.  * should purchase a license for each developer using Jigloo.
```

```
29.  * Please visit www.cloudgarden.com for details.
30.  * Use of Jigloo implies acceptance of these licensing terms.
31.  * A COMMERCIAL LICENSE HAS NOT BEEN PURCHASED FOR
32.  * THIS MACHINE, SO JIGLOO OR THIS CODE CANNOT BE USED
33.  * LEGALLY FOR ANY CORPORATE OR COMMERCIAL PURPOSE.
34.  */
35. public class ClassHighScoresPanel extends javax.swing.JPanel {
36. 
37.         //variables auto-generated by Jigloo
38.         private JLabel classHighScoresLabel;
39.         private JButton gameOneButton;
40.         private JButton gameTwoButton;
41.         private JButton gameThreeButton;
42.         private JButton backToMenuButton;
43.         private JFrame frame;
44.         private JTable highscoresTable;
45.         private JScrollPane scrollPane;
46. 
47.         //variables not generated by Jigloo
48.         private String id; //the ID of the student logged in into the system
49.         private int gameNumber = 1; //the game number which high scores want to be accessed
50.         private HighScoresTableModel HighScoresTableModel = new HighScoresTableModel();
51.         private GameRecordGenerator gameRecords = new GameRecordGenerator();
52. 
53. 
54. 
55.         /**
56.          * This constructor sets the previous frame, from the StudentMainMenuPanel, as the current frame.
```

```
57.          * It also passes the id of the student, and sets it as the current id, in order to know which student
58.          * is currently navigating the application.
59.          * @param frame the GUI frame from the previous panel
60.          * @param id the id of the student that has logged into the application
61.          */
62.         public ClassHighScoresPanel(JFrame frame, String id) {
63.                 super();
64.                 this.frame = frame;
65.                 this.id = id;
66.
67.                 /**
68.                  * The high scores table model is generated based on a list with all the game records in order (from high to low)
69.                  * from a specific game type file and also all the Student Records (used to convert the id's from the ordered game records
70.                  * file to the student's name)
71.                  */
72.                 try {
73.                         StudentHashFile studentRecords = new StudentHashFile();
74.                         HighScoresTableModel.setData(gameRecords.getGameRecordsInOrder(gameNumber), studentRecords.getStudentRecords());
75.                 } catch (IOException e){
76.                         System.out.println("error setting table data");
77.                 }
78.
79.                 initGUI();
80.         }
81.
82.         /**
83.          * Code in the initGUI method was automatically generated by Jigloo
84.          * The action listeners for the buttons were not generated by Jigloo
```

```
85.          */
86.         private void initGUI() {
87.                 try {
88.                         this.setPreferredSize(new java.awt.Dimension(500, 350));
89.                         GridBagLayout thisLayout = new GridBagLayout();
90.                         this.setSize(500, 350);
91.                         thisLayout.rowWeights = new double[] {0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1};
92.                         thisLayout.rowHeights = new int[] {7, 7, 7, 7, 7, 7, 7, 7, 7, 7};
93.                         thisLayout.columnWeights = new double[] {0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1,
    0.1};
94.                         thisLayout.columnWidths = new int[] {7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7};
95.                         this.setLayout(thisLayout);
96.                         {
97.                                 classHighScoresLabel = new JLabel();
98.                                 this.add(classHighScoresLabel, new GridBagConstraints(6, 0, 4, 1, 0.0, 0.0, GridBagConstraints.CENTER, GridB
    agConstraints.NONE, new Insets(0, 0, 0, 0), 0, 0));
99.                                 classHighScoresLabel.setText("Class HighScores");
100.                                        classHighScoresLabel.setFont(new java.awt.Font("Tahoma",1,22));
101.                                }
102.                                {
103.                                        gameOneButton = new JButton();
104.                                        this.add(gameOneButton, new GridBagConstraints(13, 3, 2, 1, 0.0, 0.0, GridBagConstraints.CENTER, Grid
    BagConstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
105.                                        gameOneButton.setText("Game One");
106.                                        gameOneButton.addActionListener(new ActionListener() {
107.                                                public void actionPerformed(ActionEvent evt) {
108.                                                        gameNumber = 1;
109.                                                        generateHighScoresTable();
```

```
110.                                            //when this button is clicked the high scores for game 1 are displayed
111.
112.                                    }
113.                            });
114.                    }
115.                    {
116.                            gameTwoButton = new JButton();
117.                            this.add(gameTwoButton, new GridBagConstraints(13, 5, 2, 1, 0.0, 0.0, GridBagConstraints.CENTER, Grid
    BagConstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
118.                            gameTwoButton.setText("Game Two");
119.                            gameTwoButton.addActionListener(new ActionListener() {
120.                                    public void actionPerformed(ActionEvent evt) {
121.                                            gameNumber = 2;
122.                                            generateHighScoresTable();
123.                                            //when this button is clicked the high scores for game 2 are displayed
124.
125.                                    }
126.                            });
127.                    }
128.                    {
129.                            gameThreeButton = new JButton();
130.                            this.add(gameThreeButton, new GridBagConstraints(13, 7, 2, 1, 0.0, 0.0, GridBagConstraints.CENTER, Gr
    idBagConstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
131.                            gameThreeButton.setText("Game Three");
132.                            gameThreeButton.addActionListener(new ActionListener() {
133.                                    public void actionPerformed(ActionEvent evt) {
134.                                            gameNumber = 3;
135.                                            generateHighScoresTable();
```

```
136.                                                    //when this button is clicked the high scores for game 3 are displayed
137.
138.                                            }
139.                                    });
140.                            }
141.                            {
142.                                    backToMenuButton = new JButton();
143.                                    this.add(backToMenuButton, new GridBagConstraints(1, 0, 1, 1, 0.0, 0.0, GridBagConstraints.CENTER, Gr
    idBagConstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
144.                                    backToMenuButton.setText("Back to Menu");
145.                                    backToMenuButton.addActionListener(new ActionListener() {
146.                                            public void actionPerformed(ActionEvent evt) {
147.                                                    backToMainMenu();//button takes you back to main menu
148.
149.
150.                                            }
151.                                    });
152.                            }
153.                            {
154.                                    scrollPane = new JScrollPane();
155.                                    this.add(scrollPane, new GridBagConstraints(1, 2, 11, 7, 0.0, 0.0, GridBagConstraints.CENTER, GridBag
    Constraints.BOTH, new Insets(0, 0, 0, 0), 0, 0));
156.                                    {
157.                                            highscoresTable = new JTable();
158.                                            scrollPane.setViewportView(highscoresTable);
159.                                            highscoresTable.setModel(HighScoresTableModel);
160.                                    }
161.                            }
```

```
162.                     } catch (Exception e) {
163.                             e.printStackTrace();
164.                     }
165.             }
166.
167.             //takes you back to main menu
168.             public void backToMainMenu(){
169.                     JFrame frame = new JFrame();
170.                     frame.getContentPane().add(new StudentsMainMenuPanel(frame, id));
171.                     frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
172.                     frame.pack();
173.                     frame.setLocation(380, 180);
174.                     frame.setVisible(true);
175.                     this.frame.setVisible(false);
176.                     this.frame.dispose();
177.             }
178.
179.             /**
180.              * When the buttons to display the high scores are clicked a new table model will be created
181.              */
182.             public void generateHighScoresTable(){
183.
184.                     try {
185.                             StudentHashFile studentRecords = new StudentHashFile();
186.                             HighScoresTableModel.setData(gameRecords.getGameRecordsInOrder(gameNumber),
    studentRecords.getStudentRecords());
187.                     } catch (IOException e){
188.                             System.out.println("error setting table data");
```

```
189.                        }
190.
191.                }
192.
193.        }
```

```
1.  import javax.swing.table.AbstractTableModel;
2.  import java.io.*;
3.
4.  /**
5.   * @program Math Speed Game
6.   * @class HighScoresTableModel
7.   * @author nahel.rifai
8.   * @description This class constructs the high scores table displayed to the students and the admin
9.   * @Date 23/8/2012
10.  * @School Markham College, Lima, Peru
11.  * @IDE Eclipse (version Indigo)
12.  * @IBCode 000633054
13.  */
14.
15. public class HighScoresTableModel extends AbstractTableModel{
16.
17.         private GameRecordList list;
18.         private StudentList studentList;//get student name
19.
20.         public int getColumnCount() {
21.                 //
22.                 return 3;
23.         }
24.
25.         public String getColumnName(int col){
26.                 if (col == 0){
27.                         return "Rank";
28.                 } else if(col == 1){
```

```
29.                         return "Name";
30.                 } else {
31.                         return "Score";
32.                 }
33.         }
34.
35.         /**
36.          * It only shows the top 15 scores in each game
37.          */
38.         public int getRowCount() {
39.
40.                 if(list.getSize() > 15){
41.                         return 15;
42.                 }
43.
44.                 return list.getSize();
45.         }
46.
47.         /**
48.          * This method fills the high scores table with the information of each record
49.          * provided by the GameRecordList
50.          */
51.         public Object getValueAt(int row, int col) {
52.                 try{
53.                         GameRecordNode temp = list.getHead();
54.                         StudentHashFile studentFile = new StudentHashFile();
55.
56.                         for (int x = 0; x < getRowCount(); x++){
```

```
57.
58.                             for (int i = 0; i < row; i++){
59.                                     temp = temp.getNext();
60.                             }
61.                             if (col == 0){
62.                                     return row+1;
63.                             } else if ( col == 1){
64.                                     return studentFile.getNamebyID(temp.getId());//gets the name of the student by using the id
65.
66.                             } else if (col == 2){
67.                                     return temp.getScore();//gets the score of the student
68.                             }
69.                     }
70.                     return null;
71.             } catch (IOException e){
72.                     return null;
73.             }
74.     }
75.
76.     /**
77.      * Resets the data in the table each time the program is refreshed
78.      * @param list the GameRecordList
79.      * @param studentList the StudentList
80.      */
81.     public void setData(GameRecordList list, StudentList studentList){
82.             this.list = list;
83.             this.studentList = studentList;
84.             fireTableDataChanged();
```

```
85.         }
86.
87. }
```

```
1.  import java.io.*;
2.  import java.awt.*;
3.  import java.awt.event.ActionEvent;
4.  import java.awt.event.ActionListener;
5.
6.  import javax.swing.*;
7.
8.  /**
9.   * @program Math Speed Game
10.  * @class AdminMainMenuPanel
11.  * @class This class has auto-generated code by Jigloo, plugin program used to create the GUI interface for panel that
12.  * displays the main menu interface of the admin program. This class generates a table with all the students in the class.
13.  * A student may be selected by double clicking the record of the student in the table. this opens the previous scores of a student.
14.  * The high scores may also be accessed through this class. Finally, a student may be created by calling the addStudentPanel
15.  * with the "Add Student" button.
16.  * @author nahel.rifai
17.  * @Date 23/8/2012
18.  * @School Markham College, Lima, Peru
19.  * @IDE Eclipse (version Indigo)
20.  * @IBCode 000633054
21.  */
22.
23. /**
24.  * This code was edited or generated using CloudGarden's Jigloo
25.  * SWT/Swing GUI Builder, which is free for non-commercial
26.  * use. If Jigloo is being used commercially (ie, by a corporation,
27.  * company or business for any purpose whatever) then you
28.  * should purchase a license for each developer using Jigloo.
```

```
29.  * Please visit www.cloudgarden.com for details.
30.  * Use of Jigloo implies acceptance of these licensing terms.
31.  * A COMMERCIAL LICENSE HAS NOT BEEN PURCHASED FOR
32.  * THIS MACHINE, SO JIGLOO OR THIS CODE CANNOT BE USED
33.  * LEGALLY FOR ANY CORPORATE OR COMMERCIAL PURPOSE.
34.  */
35. public class AdminMainMenuPanel extends javax.swing.JPanel {
36.
37.         {
38.                 //Set Look & Feel - Code auto-generated by Jigloo
39.                 try {
40.                         javax.swing.UIManager.setLookAndFeel("javax.swing.plaf.nimbus.NimbusLookAndFeel");
41.                 } catch(Exception e) {
42.                         e.printStackTrace();
43.                 }
44.         }
45.
46.         //variables auto-generated by Jigloo
47.         private JScrollPane scrollPane;
48.         private JTable searchStudentTable;
49.         private JButton overallDataButton;
50.         private JButton addStudentButton;
51.         private JTextField searchStudentTextField;
52.
53.
54.         private JFrame frame;
55.
56.         SearchStudentTableModel searchStudentTableModel = new SearchStudentTableModel();
```

```
57.
58.
59.         /**
60.          * Auto-generated main method to display this
61.          * JPanel inside a new JFrame.
62.          */
63.         public static void main(String[] args) {
64.                 JFrame frame = new JFrame();
65.                 frame.getContentPane().add(new AdminMainMenuPanel(frame, "x"));//sample id
66.                 frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
67.                 frame.pack();
68.                 frame.setLocation(380, 180);
69.                 frame.setVisible(true);
70.         }
71.
72.         public AdminMainMenuPanel(JFrame frame, String id) {
73.                 super();
74.
75.                 this.frame = frame;
76.
77.                 try {
78.                         StudentHashFile studentRecords = new StudentHashFile();
79.
80.                         /**
81.                          * The student table displays all the students in the class in order by ID.
82.                          */
83.                         searchStudentTableModel.setData(studentRecords.getStudentRecords());
84.
```

```
85.                 } catch (IOException e){
86.                         System.out.println("error setting table data");
87.                 }
88.
89.                 initGUI();
90.                 /**
91.                  * A mouse listener is used to display the information of a specific student when it's record in the table is double clicked.
92.                  */
93.                 searchStudentTable.addMouseListener(new TableMouseListener(searchStudentTable, frame));
94.         }
95.
96.         /**
97.          * Code in the initGUI method was automatically generated by Jigloo
98.          * The action listeners for the buttons were not generated by Jigloo
99.          */
100.                 private void initGUI() {
101.                         try {
102.                                 GridBagLayout thisLayout = new GridBagLayout();
103.                                 this.setPreferredSize(new java.awt.Dimension(400, 340));
104.                                 thisLayout.rowWeights = new double[] {0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1};
105.                                 thisLayout.rowHeights = new int[] {7, 7, 7, 7, 7, 7, 7, 7, 7, 7};
106.                                 thisLayout.columnWeights = new double[] {0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1};
107.                                 thisLayout.columnWidths = new int[] {7, 7, 7, 7, 7, 7, 7, 7, 7, 7};
108.                                 this.setLayout(thisLayout);
109.                                 this.setSize(400, 340);
110.                                 this.setOpaque(false);
111.                                 {
112.                                         scrollPane = new JScrollPane();
```

```
113.                                     this.add(scrollPane, new GridBagConstraints(1, 2, 8, 7, 0.0, 0.0, GridBagConstraints.CENTER, GridBagC
    onstraints.BOTH, new Insets(0, 0, 0, 0), 0, 0));
114.                                     {
115.
116.                                             searchStudentTable = new JTable();
117.                                             scrollPane.setViewportView(searchStudentTable);
118.                                             searchStudentTable.setModel(searchStudentTableModel);
119.                                     }
120.                             }
121.
122.                             {
123.                                     addStudentButton = new JButton();
124.                                     this.add(addStudentButton, new GridBagConstraints(2, 0, 2, 2, 0.0, 0.0, GridBagConstraints.CENTER, Gr
    idBagConstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
125.                                     addStudentButton.setText("Add Student");
126.                                     addStudentButton.addActionListener(new ActionListener() {
127.                                             public void actionPerformed(ActionEvent evt) {
128.                                                     addStudentPanel();
129.                                                     //Opens the panel addStudentPanel, used to create a new
    student.
130.                                             }
131.                                     });
132.                             }
133.                             {
134.                                     overallDataButton = new JButton();
135.                                     this.add(overallDataButton, new GridBagConstraints(5, 0, 2, 2, 0.0, 0.0, GridBagConstraints.CENTER, G
    ridBagConstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
136.                                     overallDataButton.setText("Overall Data");
```

```
137.                                    overallDataButton.addActionListener(new ActionListener() {
138.                                            public void actionPerformed(ActionEvent evt) {
139.                                                    openHighScoresPanel();
140.                                                    //Opens the panel AdminClassHighScoresPanel, used to display the high scores for each
    game.
141.
142.                                            }
143.                                    });
144.                            }
145.                    } catch (Exception e) {
146.                            e.printStackTrace();
147.                    }
148.            }
149.
150.            //takes you to the addStudentPanel
151.            public void addStudentPanel(){
152.                    JFrame frame = new JFrame();
153.                    frame.getContentPane().add(new AddStudentPanel(frame));
154.                    frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
155.                    frame.pack();
156.                    frame.setLocation(380, 180);
157.                    frame.setVisible(true);
158.                    this.frame.setVisible(false);
159.                    this.frame.dispose();
160.            }
161.
162.            //takes you to the ClassHighScoresPanel
163.            public void openHighScoresPanel(){
```

```
164.
165.                     JFrame frame = new JFrame();
166.                     frame.getContentPane().add(new AdminClassHighScoresPanel(frame, "1"));
167.                     frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
168.                     frame.pack();
169.                     frame.setLocation(380, 180);
170.                     frame.setVisible(true);
171.                     this.frame.setVisible(false);
172.                     this.frame.dispose();
173.                 }
174.         }
```

```
1. import javax.swing.table.*;
2. 
3. /**
4.  * @program Math Speed Game
5.  * @class SearchStudentTableModel
6.  * @author nahel.rifai
7.  * @description This class constructs a table including all the students in the math class, here the admin can see the student records
8.  * and select them to see details of their scores in the games they played
9.  * @Date 23/8/2012
10. * @School Markham College, Lima, Peru
11. * @IDE Eclipse (version Indigo)
12. * @IBCode 000633054
13. */
14. 
15. public class SearchStudentTableModel extends AbstractTableModel{
16. 
17.         private StudentList list;//the list of student objects in the class
18. 
19.         /**
20.          * The amount of columns in the table
21.          */
22.         public int getColumnCount() {
23.                 return 2;
24.         }
25. 
26.         /**
27.          * Get the information that appers as headers of columns
28.          */
```

```
29.         public String getColumnName(int col){
30.                 if (col == 0){
31.                         return "ID";// the id of the student
32.                 } else {
33.                         return "Student Name"; //the name of the student
34.                 }
35.         }
36.
37.         /**
38.          * The amount of rows in the table
39.          */
40.         public int getRowCount() {
41.                 return list.getSize();//the size of the student's list
42.         }
43.
44.         /**
45.          * This method fills the students list table with the information of each record
46.          * provided by the StudentList.
47.          */
48.         public Object getValueAt(int row, int col) {
49.
50.                 StudentNode temp = list.getHead();//the first student in the StudentList
51.
52.                 for (int i = 0; i < row; i++){
53.                         temp = temp.getNext();
54.                 }
55.
56.                 if (col == 0){
```

```
57.                         return temp.getiD();//the id of the student
58.                 } else if (col == 1) {
59.                         return temp.getLastName() + ", " + temp.getFirstName();//the name of the student
60.
61.                 }
62.                 return null;
63.         }
64.
65.         /**
66.          * Resets the data in the table each time the program is refreshed
67.          * @param list the StudentList
68.          */
69.         public void setData(StudentList list){
70.                 this.list = list;
71.                 fireTableDataChanged();
72.         }
73. }
```

```
1. import java.awt.event.MouseEvent;
2. import java.awt.event.MouseAdapter;
3. import javax.swing.*;
4.
5. /**
6.  * @program Math Speed Game
7.  * @class TableMouseListener
8.  * @class This class is used to create a Mouse listener to be used in the Students search table in the class AdminMainMenuPanel
9.  * When a record is double clicked the previous scores of the student (AdminPreviousScoresPanel) is launched.
10. * @author nahel.rifai
11. * @Date 23/8/2012
12. * @School Markham College, Lima, Peru
13. * @IDE Eclipse (version Indigo)
14. * @IBCode 000633054
15. */
16.
17. public class TableMouseListener extends MouseAdapter {
18.
19.         private JTable searchTable;     //the search table
20.
21.         private JFrame frame;//the frame from the AdminMainMenuPanel
22.
23.         /**
24.          * The search table and frame is passed from the AdminMainMenuPanel and they are set as the current SearchTable and frame in this
    class
25.          * @param SearchTable the search table with the records of the students
26.          * @param frame the previous frame
27.          */
```

```
28.         public TableMouseListener(JTable SearchTable, JFrame frame){
29.                 this.frame = frame;
30.                 this.searchTable = SearchTable;
31.         }
32.
33.         /**
34.          * If the record if the student is clicked twice this method opens the panel AdminPreviousScoresPanel
35.          * Here the admin is able to see the previous scores of the students on each game played.
36.          */
37.
38.         public void mouseClicked(MouseEvent e){
39.                 if(e.getClickCount()>1){//checks if record was clicked more than twice
40.                         int row = searchTable.getSelectedRow();//gets the row number from the SearchTable
41.                         String id = (String)searchTable.getModel().getValueAt(row, 0);
42.
43.                         JFrame frame = new JFrame();
44.                         frame.getContentPane().add(new AdminPreviousScoresPanel(frame, id));
45.                         frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
46.                         frame.pack();
47.                         frame.setLocation(380, 180);
48.                         frame.setVisible(true);
49.                         this.frame.setVisible(false);
50.                         this.frame.dispose();
51.
52.                 }
53.         }
54. }
```

```
1.  import java.io.*;
2. import java.awt.*;
3. import java.awt.event.ActionEvent;
4. import java.awt.event.ActionListener;
5. import javax.swing.*;
6.
7. /**
8.  * @program Math Speed Game
9.  * @class AdminPreviousScoresPanel
10. * @author nahel.rifai
11. * @description This class has auto-generated code by Jigloo, plugin program used to create the GUI interface panel that
12. * displays the previous scores achieved in each type of game for a specific student. It creates a new table model
13. * by using a list of the game records played by a specific student in a specific game type.
14. * @Date 23/8/2012
15. * @School Markham College, Lima, Peru
16. * @IDE Eclipse (version Indigo)
17. * @IBCode 000633054
18. */
19.
20. /**
21. * This code was edited or generated using CloudGarden's Jigloo
22. * SWT/Swing GUI Builder, which is free for non-commercial
23. * use. If Jigloo is being used commercially (ie, by a corporation,
24. * company or business for any purpose whatever) then you
25. * should purchase a license for each developer using Jigloo.
26. * Please visit www.cloudgarden.com for details.
27. * Use of Jigloo implies acceptance of these licensing terms.
28. * A COMMERCIAL LICENSE HAS NOT BEEN PURCHASED FOR
```

```
29.  * THIS MACHINE, SO JIGLOO OR THIS CODE CANNOT BE USED
30.  * LEGALLY FOR ANY CORPORATE OR COMMERCIAL PURPOSE.
31.  */
32. public class AdminPreviousScoresPanel extends javax.swing.JPanel {
33. 
34.         //variables auto-generated by Jigloo
35.         private JLabel previousScoresLabel;
36.         private JButton gameOneButton;
37.         private JButton gameTwoButton;
38.         private JButton gameThreeButton;
39.         private JButton backToMenuButton;
40.         private JFrame frame;
41.         private JTable previousScoresTable;
42.         private JScrollPane scrollPane;
43. 
44.         //variables NOT generated by Jigloo
45.         private String id; //the ID of the student which previous scores want to be accessed
46.         private int gameNumber = 1; //automatically the previous scores of game one are displayed when this panel is opened
47.         private PreviousScoresTableModel previousScoresTableModel = new PreviousScoresTableModel();
48.         private GameRecordGenerator gameRecords = new GameRecordGenerator();
49. 
50.         /**
51.          * This constructor sets the id of the clicked student as the current id. It also sets the previous frame (from the main menu)
52.          * as the current frame.
53.          * @param frame the previous frame used in the Admin Main Menu
54.          * @param id the id of the student record clicked in the table displaying all the students in the class in Main Menu.
55.          */
56.         public AdminPreviousScoresPanel(JFrame frame, String id) {
```

```
57.                 super();
58.                 this.frame = frame;
59.                 this.id = id;
60.                 try {
61.                         /**
62.                          * The previous scores table is generated by using the getGameRecords method in the GameRecordGenerator class.
63.                          * It outputs a list of the game records played by a specific students in a specific game type (game 1, 2 or 3).
64.                          */
65.                         previousScoresTableModel.setData(gameRecords.getGameRecords(id, gameNumber));
66.                 } catch (IOException e){
67.                         System.out.println("error setting table data");
68.                 }
69.                 initGUI();
70.         }
71.
72.         /**
73.          * Code in the initGUI method was automatically generated by Jigloo
74.          * The action listeners for the buttons were not generated by Jigloo
75.          */
76.         private void initGUI() {
77.                 try {
78.                         this.setPreferredSize(new java.awt.Dimension(500, 350));
79.                         GridBagLayout thisLayout = new GridBagLayout();
80.                         this.setSize(500, 350);
81.                         thisLayout.rowWeights = new double[] {0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1};
82.                         thisLayout.rowHeights = new int[] {7, 7, 7, 7, 7, 7, 7, 7, 7, 7};
83.                         thisLayout.columnWeights = new double[] {0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1,
    0.1};
```

```
84.                         thisLayout.columnWidths = new int[] {7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7};
85.                         this.setLayout(thisLayout);
86.                         {
87.                                 previousScoresLabel = new JLabel();
88.                                 this.add(previousScoresLabel, new GridBagConstraints(6, 0, 4, 1, 0.0, 0.0, GridBagConstraints.CENTER, GridBa
    gConstraints.NONE, new Insets(0, 0, 0, 0), 0, 0));
89.                                 previousScoresLabel.setText("Previous Scores");
90.                                 previousScoresLabel.setFont(new java.awt.Font("Tahoma",1,22));
91.                         }
92.                         {
93.                                 gameOneButton = new JButton();
94.                                 this.add(gameOneButton, new GridBagConstraints(13, 3, 2, 1, 0.0, 0.0, GridBagConstraints.CENTER, GridBagCons
    traints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
95.                                 gameOneButton.setText("Game One");
96.                                 gameOneButton.addActionListener(new ActionListener() {
97.                                         public void actionPerformed(ActionEvent evt) {
98.                                                 gameNumber = 1;
99.                                                 generateRecordsTable();
100.                                                        //when this button is clicked the previous scores for game 1 are
    displayed
101.                                                }
102.                                        });
103.                                }
104.                                {
105.                                        gameTwoButton = new JButton();
106.                                        this.add(gameTwoButton, new GridBagConstraints(13, 5, 2, 1, 0.0, 0.0, GridBagConstraints.CENTER, Grid
    BagConstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
107.                                        gameTwoButton.setText("Game Two");
```

```
108.                                    gameTwoButton.addActionListener(new ActionListener() {
109.                                            public void actionPerformed(ActionEvent evt) {
110.                                                    gameNumber = 2;
111.                                                    generateRecordsTable();
112.                                                    //when this button is clicked the previous scores for game 2 are
    displayed
113.                                            }
114.                                    });
115.                            }
116.                            {
117.                                    gameThreeButton = new JButton();
118.                                    this.add(gameThreeButton, new GridBagConstraints(13, 7, 2, 1, 0.0, 0.0, GridBagConstraints.CENTER, Gr
    idBagConstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
119.                                    gameThreeButton.setText("Game Three");
120.                                    gameThreeButton.addActionListener(new ActionListener() {
121.                                            public void actionPerformed(ActionEvent evt) {
122.                                                    gameNumber = 3;
123.                                                    generateRecordsTable();
124.                                                    //when this button is clicked the previous scores for game 3 are
    displayed
125.                                            }
126.                                    });
127.                            }
128.                            {
129.                                    backToMenuButton = new JButton();
130.                                    this.add(backToMenuButton, new GridBagConstraints(0, 0, 1, 1, 0.0, 0.0, GridBagConstraints.CENTER, Gr
    idBagConstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
131.                                    backToMenuButton.setText("Back to Admin Menu");
```

```
132.                                    backToMenuButton.addActionListener(new ActionListener() {
133.                                            public void actionPerformed(ActionEvent evt) {
134.                                                    backToAdminMainMenu();//back to main
    menu
135.                                            }
136.                                    });
137.                            }
138.                            {
139.                                    scrollPane = new JScrollPane();
140.                                    this.add(scrollPane, new GridBagConstraints(0, 2, 11, 7, 0.0, 0.0, GridBagConstraints.CENTER, GridBag
    Constraints.BOTH, new Insets(0, 0, 0, 0), 0, 0));
141.                                    {
142.                                            previousScoresTable = new JTable();
143.                                            scrollPane.setViewportView(previousScoresTable);
144.                                            previousScoresTable.setModel(previousScoresTableModel);
145.                                    }
146.                            }
147.                    } catch (Exception e) {
148.                            e.printStackTrace();
149.                    }
150.            }
151.
152.            //takes you back to main menu
153.            public void backToAdminMainMenu(){
154.                    JFrame frame = new JFrame();
155.                    frame.getContentPane().add(new AdminMainMenuPanel(frame, id));
156.                    frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
157.                    frame.pack();
```

```
158.                    frame.setLocation(380, 180);
159.                    frame.setVisible(true);
160.                    this.frame.setVisible(false);
161.                    this.frame.dispose();
162.            }
163.
164.            /**
165.             * When the buttons to display the previous scores are clicked a new table model will be created, presenting the
166.             * records of the game type desired.
167.             */
168.            public void generateRecordsTable(){
169.
170.                    try {
171.                            previousScoresTableModel.setData(gameRecords.getGameRecords(id, gameNumber));
172.                    } catch (IOException e){
173.                            System.out.println("error setting table data");
174.                    }
175.            }
176.    }
```

```
1.
2.
3.  import java.awt.*;
4.  import java.awt.event.ActionEvent;
5.  import java.awt.event.ActionListener;
6.  import java.io.IOException;
7.  import javax.swing.*;
8.
9.  /**
10.  * @program Math Speed Game
11.  * @class AdminClassHighScoresPanel
12.  * @author nahel.rifai
13.  * @description This class has auto-generated code by Jigloo, plugin program used to create the GUI interface panel that
14.  * displays the high scores for the class in each type of game. It creates a new table model by using a list of the
15.  * game records in order (from high to low) from a specific game type. It displays the top 15 scores for each game.
16.  * @Date 23/8/2012
17.  * @School Markham College, Lima, Peru
18.  * @IDE Eclipse (version Indigo)
19.  * @IBCode 000633054
20.  */
21.
22. /**
23.  * This code was edited or generated using CloudGarden's Jigloo
24.  * SWT/Swing GUI Builder, which is free for non-commercial
25.  * use. If Jigloo is being used commercially (ie, by a corporation,
26.  * company or business for any purpose whatever) then you
27.  * should purchase a license for each developer using Jigloo.
28.  * Please visit www.cloudgarden.com for details.
```

```
 * Use of Jigloo implies acceptance of these licensing terms.
 * A COMMERCIAL LICENSE HAS NOT BEEN PURCHASED FOR
 * THIS MACHINE, SO JIGLOO OR THIS CODE CANNOT BE USED
 * LEGALLY FOR ANY CORPORATE OR COMMERCIAL PURPOSE.
 */
public class AdminClassHighScoresPanel extends javax.swing.JPanel {

        //variables auto-generated by Jigloo
        private JLabel classHighScoresLabel;
        private JButton gameOneButton;
        private JButton gameTwoButton;
        private JButton gameThreeButton;
        private JButton backToMenuButton;
        private JFrame frame;
        private JTable highscoresTable;
        private JScrollPane scrollPane;


        //variables not generated by Jigloo

        private int gameNumber = 1; // As this panel is opened the high scores for Game type 1 are automatically displayed
        private HighScoresTableModel HighScoresTableModel = new HighScoresTableModel();
        private GameRecordGenerator gameRecords = new GameRecordGenerator();

        /**
         * This constructor sets the previous frame, from the AdminMainMenuPanel, as the current frame.
         * @param frame the GUI frame from the previous panel
         */
```

```
57.         public AdminClassHighScoresPanel(JFrame frame, String id) {
58.
59.                 super();
60.                 this.frame = frame;
61.
62.                 try {
63.                         StudentHashFile studentRecords = new StudentHashFile();
64.
65.                         /**
66.                          * The high scores table model is generated based on a list with all the game records in order (from high to low)
67.                          * from a specific game type file and also all the Student Records (used to convert the id's from the ordered game
    records
68.                          * file to the student's name)
69.                          */
70.                         HighScoresTableModel.setData(gameRecords.getGameRecordsInOrder(gameNumber), studentRecords.getStudentRecords());
71.                 } catch (IOException e){
72.                         System.out.println("error setting table data");
73.                 }
74.
75.                 initGUI();
76.         }
77.
78.         /**
79.          * Code in the initGUI method was automatically generated by Jigloo
80.          * The action listeners for the buttons were not generated by Jigloo
81.          */
82.         private void initGUI() {
83.                 try {
```

```
84.                         this.setPreferredSize(new java.awt.Dimension(500, 350));
85.                         GridBagLayout thisLayout = new GridBagLayout();
86.                         this.setSize(500, 350);
87.                         thisLayout.rowWeights = new double[] {0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1};
88.                         thisLayout.rowHeights = new int[] {7, 7, 7, 7, 7, 7, 7, 7, 7, 7};
89.                         thisLayout.columnWeights = new double[] {0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1,
    0.1};
90.                         thisLayout.columnWidths = new int[] {7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7};
91.                         this.setLayout(thisLayout);
92.                         {
93.                                 classHighScoresLabel = new JLabel();
94.                                 this.add(classHighScoresLabel, new GridBagConstraints(6, 0, 4, 1, 0.0, 0.0, GridBagConstraints.CENTER, GridB
    agConstraints.NONE, new Insets(0, 0, 0, 0), 0, 0));
95.                                 classHighScoresLabel.setText("Class HighScores");
96.                                 classHighScoresLabel.setFont(new java.awt.Font("Tahoma",1,22));
97.                         }
98.                         {
99.                                 gameOneButton = new JButton();
100.                                    this.add(gameOneButton, new GridBagConstraints(13, 3, 2, 1, 0.0, 0.0, GridBagConstraints.CENTER, Grid
    BagConstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
101.                                    gameOneButton.setText("Game One");
102.                                    gameOneButton.addActionListener(new ActionListener() {
103.                                            public void actionPerformed(ActionEvent evt) {
104.                                                    gameNumber = 1;
105.                                                    generateHighScoresTable();
106.                                                    //when this button is clicked the high scores for game 1 are displayed
107.
108.                                            }
```

```
109.                                    });
110.                            }
111.                            {
112.                                    gameTwoButton = new JButton();
113.                                    this.add(gameTwoButton, new GridBagConstraints(13, 5, 2, 1, 0.0, 0.0, GridBagConstraints.CENTER, Grid
    BagConstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
114.                                    gameTwoButton.setText("Game Two");
115.                                    gameTwoButton.addActionListener(new ActionListener() {
116.                                            public void actionPerformed(ActionEvent evt) {
117.                                                    gameNumber = 2;
118.                                                    generateHighScoresTable();
119.                                                    //when this button is clicked the high scores for game 2 are displayed
120.
121.                                            }
122.                                    });
123.                            }
124.                            {
125.                                    gameThreeButton = new JButton();
126.                                    this.add(gameThreeButton, new GridBagConstraints(13, 7, 2, 1, 0.0, 0.0, GridBagConstraints.CENTER, Gr
    idBagConstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
127.                                    gameThreeButton.setText("Game Three");
128.                                    gameThreeButton.addActionListener(new ActionListener() {
129.                                            public void actionPerformed(ActionEvent evt) {
130.                                                    gameNumber = 3;
131.                                                    generateHighScoresTable();
132.                                                    //when this button is clicked the high scores for game 3 are displayed
133.
134.                                            }
```

```
135.                                    });
136.                            }
137.                            {
138.                                    backToMenuButton = new JButton();
139.                                    this.add(backToMenuButton, new GridBagConstraints(1, 0, 1, 1, 0.0, 0.0, GridBagConstraints.CENTER, Gr
    idBagConstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
140.                                    backToMenuButton.setText("Return");
141.                                    backToMenuButton.addActionListener(new ActionListener() {
142.                                            public void actionPerformed(ActionEvent evt) {
143.                                                    backToAdminMainMenu();//button takes you back to main menu
144.
145.
146.                                            }
147.                                    });
148.                            }
149.                            {
150.                                    scrollPane = new JScrollPane();
151.                                    this.add(scrollPane, new GridBagConstraints(1, 2, 11, 7, 0.0, 0.0, GridBagConstraints.CENTER, GridBag
    Constraints.BOTH, new Insets(0, 0, 0, 0), 0, 0));
152.                                    {
153.
154.                                            highscoresTable = new JTable();
155.                                            scrollPane.setViewportView(highscoresTable);
156.                                            highscoresTable.setModel(HighScoresTableModel);
157.                                    }
158.                            }
159.                    } catch (Exception e) {
160.                            e.printStackTrace();
```

```
161.                    }
162.            }
163.
164.            //takes you back to the admin main menu
165.            public void backToAdminMainMenu(){
166.                    JFrame frame = new JFrame();
167.                    frame.getContentPane().add(new AdminMainMenuPanel(frame, id));
168.                    frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
169.                    frame.pack();
170.                    frame.setLocation(380, 180);
171.                    frame.setVisible(true);
172.                    this.frame.setVisible(false);
173.                    this.frame.dispose();
174.            }
175.
176.            /**
177.             * When the buttons to display the high scores are clicked a new table model will be created
178.             */
179.            public void generateHighScoresTable(){
180.
181.                    try {
182.                            StudentHashFile studentRecords = new StudentHashFile();
183.                            HighScoresTableModel.setData(gameRecords.getGameRecordsInOrder(gameNumber),
    studentRecords.getStudentRecords());
184.                    } catch (IOException e){
185.                            System.out.println("error setting table data");
186.                    }
187.
```

```
188.                }
189.
190.        }
```

```
1. import java.io.*;
2. import java.awt.*;
3. import java.awt.event.ActionEvent;
4. import java.awt.event.ActionListener;
5. import javax.swing.*;
6.
7. /**
8.  * @program Math Speed Game
9.  * @class AddStudentPanel
10. * @author nahel.rifai
11. * @description This class has auto-generated code by Jigloo, plugin program used to create the GUI interface for panel that creates new
    students
12. * in the admin program. The method createStudent creates a new student object based on the information provided by the admin and calls
13. * the addStudent method in the StudentHashFile class where the created student record is added to the Random Access File.
14. * @Date 23/8/2012
15. * @School Markham College, Lima, Peru
16. * @IDE Eclipse (version Indigo)
17. * @IBCode 000633054
18. */
19.
20. /**
21. * This code was edited or generated using CloudGarden's Jigloo
22. * SWT/Swing GUI Builder, which is free for non-commercial
23. * use. If Jigloo is being used commercially (ie, by a corporation,
24. * company or business for any purpose whatever) then you
25. * should purchase a license for each developer using Jigloo.
26. * Please visit www.cloudgarden.com for details.
27. * Use of Jigloo implies acceptance of these licensing terms.
```

```
28.  * A COMMERCIAL LICENSE HAS NOT BEEN PURCHASED FOR
29.  * THIS MACHINE, SO JIGLOO OR THIS CODE CANNOT BE USED
30.  * LEGALLY FOR ANY CORPORATE OR COMMERCIAL PURPOSE.
31.  */
32.
33. public class AddStudentPanel extends javax.swing.JPanel {
34.
35.         //variables auto-generated by Jigloo
36.         private JLabel studentNameLabel;
37.         private JLabel studentIDLabel;
38.         private JLabel studentPasswordLabel;
39.         private JButton createStudentButton;
40.         private JButton backToMainMenuButton;
41.         private JLabel addStudentLabel;
42.         private JTextField studentPasswordTextField;
43.         private JTextField studentIDTextField;
44.         private JTextField studentLastNameTextField;
45.         private JTextField studentNameTextField;
46.         private JLabel studentLastNameLabel;
47.         private JFrame frame;
48.
49.         /**
50.          * The constructor is called to set the previous frame from the AdminMainMenuPanel and open the AddStudentPanel
51.          * @param frame the previous frame is set as the current frame in this panel
52.          */
53.         public AddStudentPanel(JFrame frame) {
54.                 super();
55.                 this.frame = frame;
```

```
56.                 initGUI();
57.         }
58.
59.         /**
60.          * Code in the initGUI method was automatically generated by Jigloo
61.          * The action listeners for the buttons were not generated by Jigloo
62.          */
63.         private void initGUI() {
64.                 try {
65.                         GridBagLayout thisLayout = new GridBagLayout();
66.                         setPreferredSize(new Dimension(400, 300));
67.                         thisLayout.rowWeights = new double[] {0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1};
68.                         thisLayout.rowHeights = new int[] {7, 7, 7, 7, 7, 7, 7, 7, 7, 7};
69.                         thisLayout.columnWeights = new double[] {0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1};
70.                         thisLayout.columnWidths = new int[] {7, 7, 7, 7, 7, 7, 7, 7, 7, 7};
71.                         this.setLayout(thisLayout);
72.                         {
73.                                 studentNameLabel = new JLabel();
74.                                 this.add(studentNameLabel, new GridBagConstraints(1, 3, 1, 1, 0.0, 0.0, GridBagConstraints.CENTER, GridBagCo
    nstraints.NONE, new Insets(0, 0, 0, 0), 0, 0));
75.                                 studentNameLabel.setText("Student Name:");
76.                         }
77.                         {
78.                                 studentLastNameLabel = new JLabel();
79.                                 this.add(studentLastNameLabel, new GridBagConstraints(1, 4, 1, 1, 0.0, 0.0, GridBagConstraints.CENTER, GridB
    agConstraints.NONE, new Insets(0, 0, 0, 0), 0, 0));
80.                                 studentLastNameLabel.setText("Student LastName:");
81.                         }
```

```
82.
83.                         {
84.                                 studentPasswordLabel = new JLabel();
85.                                 this.add(studentPasswordLabel, new GridBagConstraints(1, 5, 1, 1, 0.0, 0.0, GridBagConstraints.CENTER, GridB
    agConstraints.NONE, new Insets(0, 0, 0, 0), 0, 0));
86.                                 studentPasswordLabel.setText("Password");
87.                         }
88.                         {
89.                                 studentNameTextField = new JTextField();
90.                                 this.add(studentNameTextField, new GridBagConstraints(2, 3, 7, 1, 0.0, 0.0, GridBagConstraints.CENTER, GridB
    agConstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
91.                         }
92.                         {
93.                                 studentLastNameTextField = new JTextField();
94.                                 this.add(studentLastNameTextField, new GridBagConstraints(2, 4, 7, 1, 0.0, 0.0, GridBagConstraints.CENTER, G
    ridBagConstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
95.                         }
96.
97.                         {
98.                                 studentPasswordTextField = new JTextField();
99.                                 this.add(studentPasswordTextField, new GridBagConstraints(2, 5, 7, 1, 0.0, 0.0, GridBagConstraints.CENTER, G
    ridBagConstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
100.                            }
101.                            {
102.                                    createStudentButton = new JButton();
103.                                    this.add(createStudentButton, new GridBagConstraints(2, 7, 1, 1, 0.0, 0.0, GridBagConstraints.CENTER,
     GridBagConstraints.NONE, new Insets(0, 0, 0, 0), 0, 0));
104.                                    createStudentButton.setText("Create Student");
```

```
105.                                createStudentButton.addActionListener(new ActionListener() {
106.                                        public void actionPerformed(ActionEvent evt) {
107.
108.                                            if (studentNameTextField.getText().matches("[a-zA-
    z]*") && studentLastNameTextField.getText().matches("[a-zA-z]*")) {
109.                                                createStudent();
110.                                            }
111.                                            else {
112.                                                JOptionPane.showMessageDialog(frame, "Only input letters for Name and Last
    Name"  , "Error", JOptionPane.INFORMATION_MESSAGE);
113.                                                studentNameTextField.setText("");
114.                                                studentLastNameTextField.setText("");
115.                                            }
116.                                        }
117.                                });
118.                            }
119.                            {
120.                                addStudentLabel = new JLabel();
121.                                this.add(addStudentLabel, new GridBagConstraints(2, 1, 5, 1, 0.0, 0.0, GridBagConstraints.CENTER, Gri
    dBagConstraints.NONE, new Insets(0, 0, 0, 0), 0, 0));
122.                                addStudentLabel.setText("Add Student to Class");
123.                            }
124.                            {
125.                                backToMainMenuButton = new JButton();
126.                                this.add(backToMainMenuButton, new GridBagConstraints(1, 1, 1, 1, 0.0, 0.0, GridBagConstraints.CENTER
    , GridBagConstraints.NONE, new Insets(0, 0, 0, 0), 0, 0));
127.                                backToMainMenuButton.setText("Return");
128.                                backToMainMenuButton.addActionListener(new ActionListener() {
```

```
129.                                        public void actionPerformed(ActionEvent evt) {
130.                                                backToAdminMainMenu();//button takes you back to main menu
131.
132.                                        }
133.                                    });
134.                            }
135.                    } catch (Exception e) {
136.                            e.printStackTrace();
137.                    }
138.            }
139.
140.
141.            public void backToAdminMainMenu(){
142.                    JFrame frame = new JFrame();
143.                    frame.getContentPane().add(new AdminMainMenuPanel(frame, "x"));//sample id
144.                    frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
145.                    frame.pack();
146.                    frame.setLocation(380, 180);
147.                    frame.setVisible(true);
148.                    this.frame.setVisible(false);
149.                    this.frame.dispose();
150.            }
151.
152.            /**
153.             * This method generates a new ID for the created Student, constructs a new StudentNode object (including ID, first name,
    last name and password)
154.             * It then calls the method addStudent in the class StudentHashFile, where the created student is added to the Random Access
    File.
```

```
155.                * @param gameRecord is the game record object including the date, time, student id and score of the game trial
156.                * @param gameNumber the game type played (game 1, 2 or 3)
157.                * @throws IOException if file error occurs
158.                */
159.               public void createStudent(){
160.                       try {
161.                               StudentHashFile file = new StudentHashFile();
162.
163.                               int newiD = file.generateID();
164.
165.                               if (newiD != -1){
166.
167.                                       StudentNode newStudent = new StudentNode(studentNameTextField.getText(),
    studentLastNameTextField.getText(), newiD + "", studentPasswordTextField.getText());
168.                                       file.addStudent(newStudent);
169.                                       //takes you to the CreatedStudentPanel
170.                                       JFrame frame = new JFrame();
171.                                       frame.getContentPane().add(new CreatedStudentPanel(frame, newStudent));
172.                                       frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
173.                                       frame.pack();
174.                                       frame.setLocation(380, 180);
175.                                       frame.setVisible(true);
176.                                       this.frame.setVisible(false);
177.                                       this.frame.dispose();
178.                               } else {
179.                                       JOptionPane.showMessageDialog(frame, "No free space available in file, student can't be added
    "   , "Error", JOptionPane.INFORMATION_MESSAGE);                        }
180.                       } catch (IOException e){
```

```
181.                             System.out.println("error creating file");
182.                         }
183.                 }
184.         }
```

```
1.  import java.awt.*;
2.  import java.awt.event.ActionEvent;
3.  import java.awt.event.ActionListener;
4.  import javax.swing.*;
5.
6.  /**
7.   * @program Math Speed Game
8.   * @class CreatedStudentPanel
9.   * @author nahel.rifai
10.  * @description This class has auto-generated code by Jigloo, plugin program used to create the GUI interface panel that
11.  * displays the information about the student that has just been created by the admin.
12.  * @Date 23/8/2012
13.  * @School Markham College, Lima, Peru
14.  * @IDE Eclipse (version Indigo)
15.  * @IBCode 000633054
16.  *
17.  */
18.
19. /**
20.  * This code was edited or generated using CloudGarden's Jigloo
21.  * SWT/Swing GUI Builder, which is free for non-commercial
22.  * use. If Jigloo is being used commercially (ie, by a corporation,
23.  * company or business for any purpose whatever) then you
24.  * should purchase a license for each developer using Jigloo.
25.  * Please visit www.cloudgarden.com for details.
26.  * Use of Jigloo implies acceptance of these licensing terms.
27.  * A COMMERCIAL LICENSE HAS NOT BEEN PURCHASED FOR
28.  * THIS MACHINE, SO JIGLOO OR THIS CODE CANNOT BE USED
```

```
 * LEGALLY FOR ANY CORPORATE OR COMMERCIAL PURPOSE.
 */

public class CreatedStudentPanel extends javax.swing.JPanel {

	//variables auto-generated by Jigloo
	private JFrame frame;
	private StudentNode newStudent; //the StudentNode object of the new student being added
	private JButton backToMainButton;
	private JLabel passwordLabel;
	private JLabel idLabel;
	private JLabel studentLastNameLabel;
	private JLabel studentNameLabel;
	private JLabel createdLabel;

	/**
	 * The constructor sets the previous frame, from the AddStudentPanel, as the current frame, and passes the student object (the new student)
	 * in order to present the new student's information in this panel
	 * @param frame the GUI frame from the previous panel
	 * @param newStudent the student object created belonging to the new student in class
	 */
	public CreatedStudentPanel(JFrame frame, StudentNode newStudent) {
		super();
		this.frame = frame;
		this.newStudent = newStudent;
		initGUI();
	}
```

```
56.
57.         /**
58.          * Code in the initGUI method was automatically generated by Jigloo
59.          * The action listeners for the buttons were not generated by Jigloo
60.          */
61.         private void initGUI() {
62.                 try {
63.                         GridBagLayout thisLayout = new GridBagLayout();
64.                         setPreferredSize(new Dimension(400, 300));
65.                         thisLayout.rowWeights = new double[] {0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1};
66.                         thisLayout.rowHeights = new int[] {7, 7, 7, 7, 7, 7, 7, 7, 7, 7};
67.                         thisLayout.columnWeights = new double[] {0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1};
68.                         thisLayout.columnWidths = new int[] {7, 7, 7, 7, 7, 7, 7, 7, 7, 7};
69.                         this.setLayout(thisLayout);
70.                         {
71.                                 createdLabel = new JLabel();
72.                                 this.add(createdLabel, new GridBagConstraints(3, 1, 5, 1, 0.0, 0.0, GridBagConstraints.CENTER, GridBagConstr
    aints.NONE, new Insets(0, 0, 0, 0), 0, 0));
73.                                 createdLabel.setText("Succesfully Created New Student " );
74.                                 createdLabel.setFont(new java.awt.Font("Tahoma",1,12));
75.                         }
76.                         {
77.                                 studentNameLabel = new JLabel();
78.                                 this.add(studentNameLabel, new GridBagConstraints(2, 3, 2, 1, 0.0, 0.0, GridBagConstraints.CENTER, GridBagCo
    nstraints.NONE, new Insets(0, 0, 0, 0), 0, 0));
79.                                 studentNameLabel.setText("Student Name: " + newStudent.getFirstName());
80.                                 studentNameLabel.setFont(new java.awt.Font("Tahoma",1,11));
81.                         }
```

```
82.                         {
83.                                 studentLastNameLabel = new JLabel();
84.                                 this.add(studentLastNameLabel, new GridBagConstraints(2, 4, 2, 1, 0.0, 0.0, GridBagConstraints.CENTER, GridB
    agConstraints.NONE, new Insets(0, 0, 0, 0), 0, 0));
85.                                 studentLastNameLabel.setText("Last Name: " + newStudent.getLastName());
86.                                 studentLastNameLabel.setFont(new java.awt.Font("Tahoma",1,11));
87.                         }
88.                         {
89.                                 idLabel = new JLabel();
90.                                 this.add(idLabel, new GridBagConstraints(2, 5, 2, 1, 0.0, 0.0, GridBagConstraints.CENTER, GridBagConstraints
    .NONE, new Insets(0, 0, 0, 0), 0, 0));
91.                                 idLabel.setText("Generated ID: "+ newStudent.getiD());
92.                                 idLabel.setFont(new java.awt.Font("Tahoma",1,11));
93.                         }
94.                         {
95.                                 passwordLabel = new JLabel();
96.                                 this.add(passwordLabel, new GridBagConstraints(2, 6, 2, 1, 0.0, 0.0, GridBagConstraints.CENTER, GridBagConst
    raints.NONE, new Insets(0, 0, 0, 0), 0, 0));
97.                                 passwordLabel.setText("Password: " + newStudent.getPassword());
98.                                 passwordLabel.setFont(new java.awt.Font("Tahoma",1,11));
99.                         }
100.                        {
101.                                backToMainButton = new JButton();
102.                                this.add(backToMainButton, new GridBagConstraints(3, 8, 4, 1, 0.0, 0.0, GridBagConstraints.CENTER, Gr
    idBagConstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0, 0));
103.                                backToMainButton.setText("OK");
104.                                backToMainButton.addActionListener(new ActionListener() {
105.                                        public void actionPerformed(ActionEvent evt) {
```

```
106.                                                    backToAdminMainMenu();//button takes you back to the admin main menu
107.
108.                                            }
109.                                    });
110.                            }
111.                    } catch (Exception e) {
112.                            e.printStackTrace();
113.                    }
114.            }
115.
116.            //takes you back to the admin main menu panel
117.            public void backToAdminMainMenu(){
118.                    JFrame frame = new JFrame();
119.                    frame.getContentPane().add(new AdminMainMenuPanel(frame, "x"));//sample id
120.                    frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
121.                    frame.pack();
122.                    frame.setLocation(380, 180);
123.                    frame.setVisible(true);
124.                    this.frame.setVisible(false);
125.                    this.frame.dispose();
126.            }
127.
128.    }
```

C2 – Handling Errors

The following section provides a fully documented error-handling of the input and output methods in the program Math Speed Game. This will be presented class by class, in the same class order as in the program listing. In some cases the error handling for a class is located in a different one. To avoid repetition we will include all the error handling concerning a method once and in full detail.

Moreover, some error handling is also controlled by the GUI components in the program. This can be seen in the length of the text fields so the user won't be able to fill more characters than allowed.

Error Handling in *LoginScreenPanel* class

The user logs in and the username (ID) and password entered are validated. There is an if statement which calls for the method "login" in te StudentHashFile class.

```
167.              /**
168.               * The ID and password input are validated by calling the method constructor of the StudentHashFile (class dealing with the
   student's RandomAcessFile
169.               * If the ID matches the password then the student is granted access to it's account on the application
170.               */
171.              public void login(){
172.
173.                      try{
174.                              try {
175.                                      StudentHashFile file = new StudentHashFile();
176.
177.                                      if (file.login(txtLoginId.getText(), txtPassword.getText()) == true){
178.                                              //takes you to the main menu of the app
```

```
179.                                        JFrame frame = new JFrame();
180.                                        frame.getContentPane().add(new StudentsMainMenuPanel(frame, txtLoginId.getText()));
181.                                        frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
182.                                        frame.pack();
183.                                        frame.setLocation(380, 180);
184.                                        frame.setVisible(true);
185.                                        this.frame.setVisible(false);
186.                                        this.frame.dispose();
187.                                    } else {
188.                                            JOptionPane.showMessageDialog(this, "Wrong Username or
    Password"  ,"Error", JOptionPane.INFORMATION_MESSAGE);
189.                                    }
190.                                } catch (IOException e){
191.                                        JOptionPane.showMessageDialog(this, "File Error"  , "Error",JOptionPane.INFORMATION_MESSAGE);
192.                                }
193.                            } catch (NumberFormatException e){
194.                                    JOptionPane.showMessageDialog(this, "Wrong Username or Password"  , "Error",JOptionPane.INFORMATION_MESSAGE);
195.                                    txtLoginId.setText("");
196.                                    txtPassword.setText("");
197.                            }
198.                    }
199.            }
```

The login method in the *StudentHashFile* class is used to search for the input ID, if it is found it is compared with the password, if they match the user is granted the login, otherwise the login is not granted and a pop message saying “Wrong username and password” is displayed on screen.

```
1.          /**
2.           * This method validates the login of the student into the application
3.           * @param id the id of the student (serves as user name)
4.           * @param password the password of the student
5.           * @return if id and password match a "true" boolean is returned, otherwise a "false" boolean is returned
6.           * @throws IOException if there is a file error
7.           */
8.          public boolean login(String id, String password) throws IOException{
9.
10.                 file.seek(Integer.parseInt(id)*RECORD_LENGHT);//position of record (searched by id)
11.                 file.readUTF();//skips first name
12.                 file.readUTF();//skips last name
13.
14.                 if (file.readUTF().equals(id) && file.readUTF().equals(password)){
15.                         return true;//login granted
16.                 }
17.                 return false;//login not granted
18.         }
```

***GamePlayingInterfacePanel* class general error handling (for games 1, 2 and 3)**

Only numbers are accepted as answers in all of the games. Each time an answer is submitted this is validated before the answer itself is processed. There is a try and a catch. If there is a NumberFormatException this is trapped by the catch, a pop up message saying “Only input numbers” is displayed on screen and the answer field is blanked out.

```
1.                              nextQuestionButton = new JButton();
2.                              this.add(nextQuestionButton, new GridBagConstraints(4, 7, 2, 1, 0.0, 0.0, GridBagConstraints.CENTER, GridBagCo
    nstraints.BOTH, new Insets(0, 0, 0, 0), 0, 0));
3.                              nextQuestionButton.setText("Enter");
4.                              nextQuestionButton.addActionListener(new ActionListener() {
5.
6.                                      /**
7.                                       * When the button "Enter" is clicked the answer input is checked and
8.                                       * the next question is displayed
9.                                       */
10.                                     public void actionPerformed(ActionEvent evt) {
11.
12.                                             try{
13.                                                     checkAnswer();
14.                                                     displayNextQuestion();
15.                                             } catch (NumberFormatException e){
16.                                                     JOptionPane.showMessageDialog(frame, "Only input
    numbers!", "Error", JOptionPane.INFORMATION_MESSAGE);
17.                                                     answerTxt.setText("");
18.                                             }
19.                                     }
20.                             });
```

Each time an answer is submitted it is compared to the actual answer of the object *MathQuestion*, if they match the student is given a +1 in the correct answers count. This is controlled by an IF and ELSE condition. If the answer is not correct the user is output a pop message on screen saying “Wrong Answer”.

```
21.                 //check for correct or wrong answers
22.                 if (answer.equals(correctAnswerLabel.getText())){
23.                         correctAnswers++;
24.                         JOptionPane.showMessageDialog(this, "You have " + correctAnswers + " correct answers!", "Correct
    Answer", JOptionPane.INFORMATION_MESSAGE);
25.
26.                 } else {
27.                         JOptionPane.showMessageDialog(this, "Wrong Answer!", "Wrong Answer", JOptionPane.INFORMATION_MESSAGE);
28.                 }
```

***GamePlayingInterface* class. Error handling for game 1.**

As game 1 only consists of 20 questions after the 20th *MathQuestion* object is reached the score is generated and the game ends. This is controlled by an IF condition. If the question number equals 20 then it is the last question attempted for Game 1 and as soon as the user inputs the answer the score is calculated, the game record is saved and the game ends.

```
29.
30.                 //check if game one has finished
31.                 if (gameNumber == 1 && currentQuestion == 20){
32.                         long finish = System.currentTimeMillis();
33.                         gameOneScoreGenerator(correctAnswers, (int)((finish-start)/1000));
34.                 }
```

***GamePlayingInterface* class. Error handling for game 2.**

Game 2 lasts until the 60 seconds pass on the timer. After this point the score is generated and the game ends. This is controlled by an IF condition. As soon as the 60 questions pass and the user inputs the last number, the score is calculated, the game record is saved and the game ends.

```
35.             if (gameNumber == 2 && timeInt > 60) {
36.                     gameTwoScoreGenerator(correctAnswers);
37.             }
38.
```

***GamePlayingInterface class.* Error handling for game 3.**

Game 3 finished when time reaches 0. After this point the game score is calculated and the game ends. This is also controlled by an IF condition. As soon as the left time is equal to 0 and the user attempts the last question left, the score is calculated, the game record is saved and the game ends.

```
39.             if (gameNumber == 3 && leftTime < 0) {
40.                     gameThreeScoreGenerator((int)((finish-start)/1000));
41.             }
```

***PreviousScorePanel* class error handling (validation comes from the *PreviousScoresTableModel* class which sets the values for all the data in the table)**

All the scores from the games played by the student are displayed in this panel. There is a % improvement displayed from the previous score to the current one. If the score is the first game played by the student there can't be a percentage improvement because there was no previous game. This is controlled by an IF condition, if it is the first game played, the field where the percentage improvement should be displayed will now show "First Game Played".

```
101.
102.             /**
103.              * This method fills the previous scores table with the information of each record
104.              * provided by the GameRecordList (list of games from a specific game (either 1, 2 or 3) played by a student)
105.              */
106.             public Object getValueAt(int row, int col) {
107.
108.                     GameRecordNode temp = list.getHead();//the first game played by the student
109.
110.                     for (int i = 0; i < row; i++){
111.                             temp = temp.getNext();
112.                     }
113.
114.                     if (col == 0){
115.                             //gets the date in which the game was played
116.                             SimpleDateFormat sdf = new SimpleDateFormat("MMM dd,yyyy HH:mm");
117.                             Date resultdate = new Date(temp.getDate());
118.                             return sdf.format(resultdate);
119.
120.                     } else if ( col == 1){
121.                             //gets the score of the game
122.                             return temp.getScore();
123.                     } else if (col == 2){
```

```
124.                            //gets the percentage improvement from the previous game to the current one
125.                            if(temp.getNext() != null){
126.
127.                                    String previousscore = temp.getScore();//gets the previous score
128.                                    double doublepreviousscore = Double.parseDouble(previousscore);
129.
130.                                    String currentscore = temp.getNext().getScore();//gets the current score
131.                                    double doublecurrentscore = Double.parseDouble(currentscore);
132.
133.                                    double difference = doublepreviousscore - doublecurrentscore;//calculates the difference
134.
135.                                    double percentageimprovement = (difference/doublepreviousscore) * 10;//calculates the percentage
    improvement
136.
137.                                    DecimalFormat twdf = new DecimalFormat("#.##");//lowers it down to two decimal places
138.
139.                                    return twdf.format(percentageimprovement);
140.                            }
141.                            return "First Game Played";//if it is the first game played by the student there is no percentage improvement
142.                    }
143.                    return null;
144.            }
145.
```

***AddStudentPanel* class (from the admin program) error handling.**

The name and last name input for the student can oly be letters. This is controlled by an IF and else condition. If the student name and last name are letters then the student may be creater. Else, a pop up message will appear on screen saying “Only input letters for Name and Last Name”.

```
185.       createStudentButton.setText("Create Student");
186.                                       createStudentButton.addActionListener(new ActionListener() {
187.                                               public void actionPerformed(ActionEvent evt) {
188.
189.                                                       if (studentNameTextField.getText().matches("[a-zA-
    z]*") && studentLastNameTextField.getText().matches("[a-zA-z]*")) {
190.                                                               createStudent();
191.                                                       }
192.                                                       else {
193.                                                               JOptionPane.showMessageDialog(frame, "Only input letters for Name and Last
    Name"  , "Error", JOptionPane.INFORMATION_MESSAGE);
194.                                                               studentNameTextField.setText("");
195.                                                               studentLastNameTextField.setText("");
196.                                                       }
197.                                               }
198.                                       });
```

To generate the id of the student (needs to be a new one – no other student may have the same id) the following error handling is implemented . The method generateID is called from the StudentHashFile class. If there are no spaces in the random access file then this method will return a -1. If a -1 is not returned then the student will eb successfully added to the random access file with it's new corresponding ID. Else, a pop up message will be displayed sying "No free space available in file, student can't be added".

```
199.       public void createStudent(){
200.                       try {
201.                               StudentHashFile file = new StudentHashFile();
202.
203.                               int newiD = file.generateID();
204.
205.                               if (newiD != -1){
206.
207.                                       StudentNode newStudent = new StudentNode(studentNameTextField.getText(),
    studentLastNameTextField.getText(), newiD + "", studentPasswordTextField.getText());
208.                                       file.addStudent(newStudent);
209.                                       //takes you to the CreatedStudentPanel
210.                                       JFrame frame = new JFrame();
211.                                       frame.getContentPane().add(new CreatedStudentPanel(frame, newStudent));
212.                                       frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
213.                                       frame.pack();
214.                                       frame.setLocation(380, 180);
215.                                       frame.setVisible(true);
216.                                       this.frame.setVisible(false);
217.                                       this.frame.dispose();
218.                               } else {
219.                                       JOptionPane.showMessageDialog(frame, "No free space available in file, student can't be
    added"  , "Error", JOptionPane.INFORMATION_MESSAGE);
```

```
                                }
                        } catch (IOException e){
                                System.out.println("error creating file");
                        }
                }
        }
```

Note: The method that generated the ID of the student is called generateID and is located in the StudentHashFile class.

```
/**
         * This method generates a new id for the created student (this is done to ensure every student has a different id)
         * @return the id that will be assigned to the created student
         * @throws IOException if there is a file error
         */
        public int generateID() throws IOException {

                for (int i = 0; i < SIZE; i++){//searches the random access file

                        file.seek(i*RECORD_LENGHT);

                        if (file.readUTF().equals("x")){
                                return i;//as soon as it finds an empty record it returns the position of the record (equals to the id of the new student)
                        }
                }
                return -1;//if there are no empty positions a "-1" is returned
        }

```

C3 – Success of the Program

The following section gives an insight of the success of the application constructed. This is measured in terms of the fulfilment of the objectives stated in criterion A2.

Objective	Testing of Objective (Page #)	Level of Achievement
1 a.	197-199	Complete
1 b.	197-199	Complete
2 a.	204-224	Complete
2 b.	204-224	Complete
3 a.	205-208	Complete
3 b.	209-213	Complete
3 c.	214-224	Complete
4 a.	200-204	Complete
5 a.	205-208	Complete
5 b.	209-213	Complete
5 c.	214-224	Complete
6 a.	235-240	Complete
7 a.	247-250	Complete
7 b.	241-246	Complete
7 c.	243-245	Complete
7 d.	251-252	Complete

D1 – Test Output

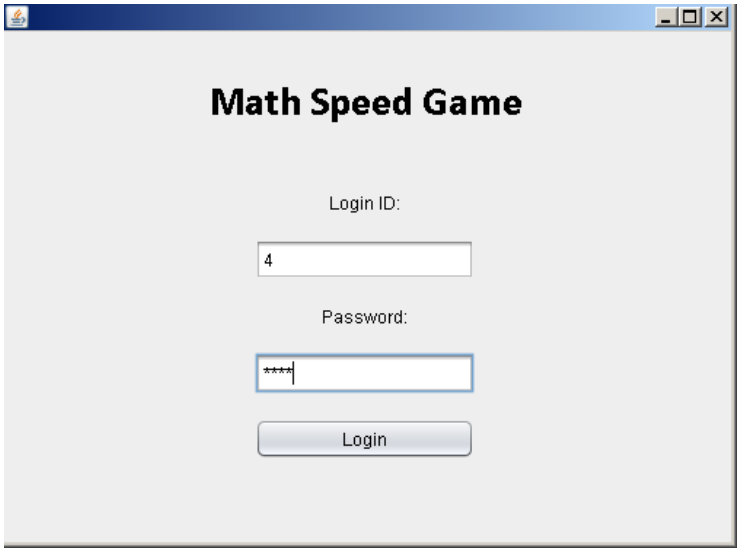
These are the records of all the students in the class. Excel file is in possession of Sr. Salomon. In this testing section we will randomly use some students in the class to test the different features of this application.

Math Students P6 - Sr Rafael Salomon class		
ID - User	**Name**	**Password**
0	ARENAS ARROBA Mateo	1111
1	BELLONI OTAYZA Abrielle	7654
2	BERTOLI MONTAÑO Antonella	5432
3	BUSTAMANTE GINOCCHIO Maria Pierina	7564
4	CAMET VEGA Arturo	8272
5	DUCATO HOFFMANN Giacomo	9992
6	GALDOS CÁRDENAS Diego Gabriel	6312
7	GIHA BRIGNETI Juan Pablo	7721
8	MENDOZA MANNARELLI Santiago	8371
9	OLAECHEA PRADO Fermín	0123
10	QUEIROLO DE PICASSA Pietro	8273
11	RODRÍGUEZ DE RIDDER Felipe Santiago	5212
12	ROEKAERT AICARDI Nicole Marie	1212
13	SÁNCHEZ VILLARÁN Andrés Raúl	7182
14	SILVA CAMACHO Juan Sebastian	9121
15	TORRES-LLOSA POOLEY Patrick	2321
16	VARGAS MONTESDEOCA Alfredo Gabriel	123
17	VARGAS SILVA Daniella Maria	7763
18	VELASCO HUARANGA Brenda	8212
19	VILLA BUSTAMANTE Lorenzo	2211
20	WOODMAN MORALES Andrea	9954
21	ZALDÍVAR DEL AGUILA Gabriel Ignacio	8210

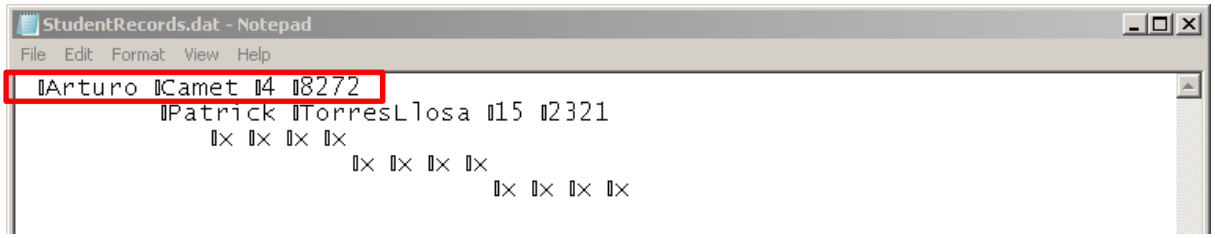
Login Testing

Important Note: Rectangular Boxes marked with red are not part of the GUI interface.

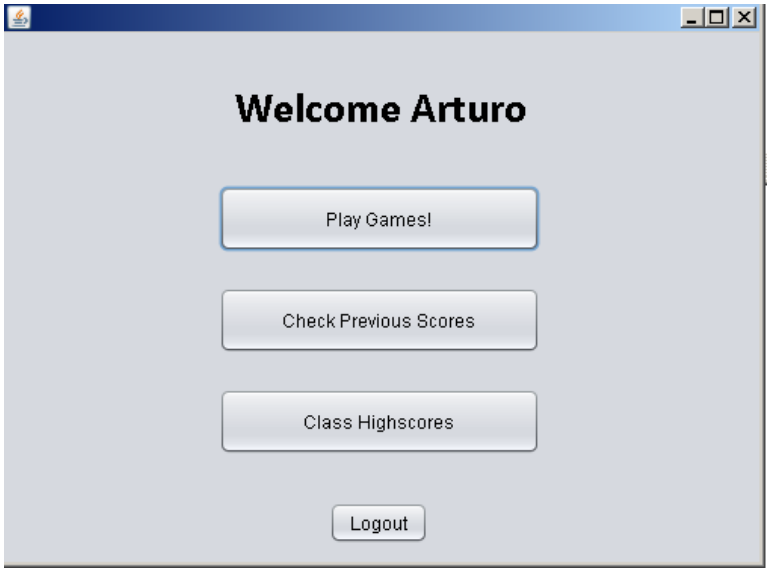
The user “Arturo Camet” with id “4” and password “8272” will be used to test the login.



The button login is clicked and the program searches the RandomAccessFile named “StudentRecords.dat”.



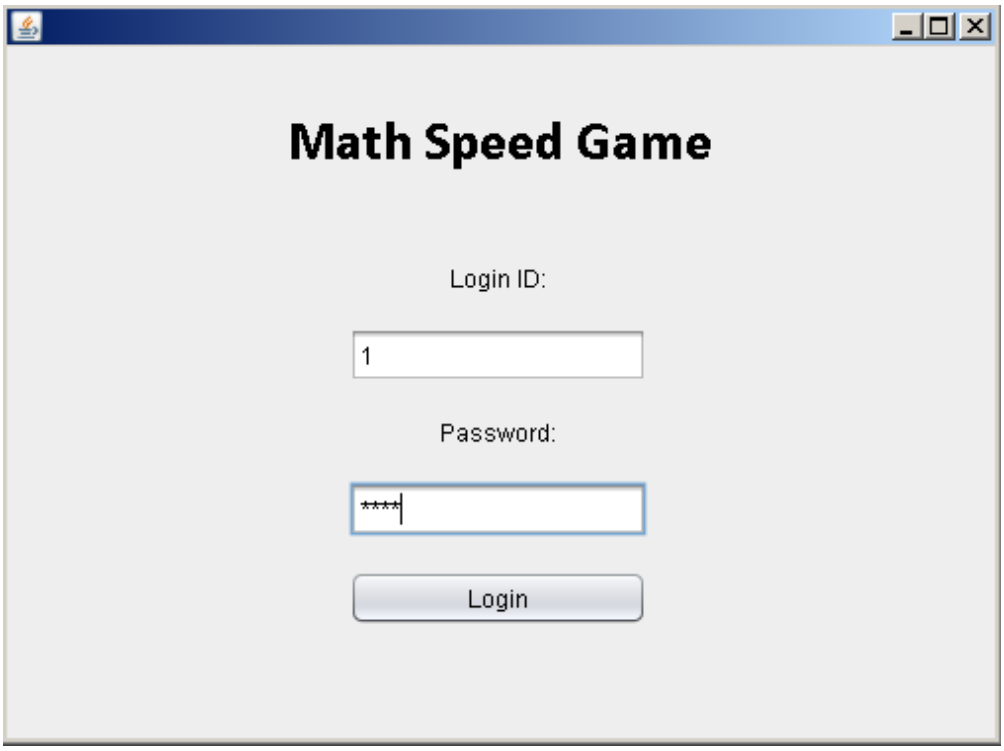
As the ID and password matches the user is granted access.



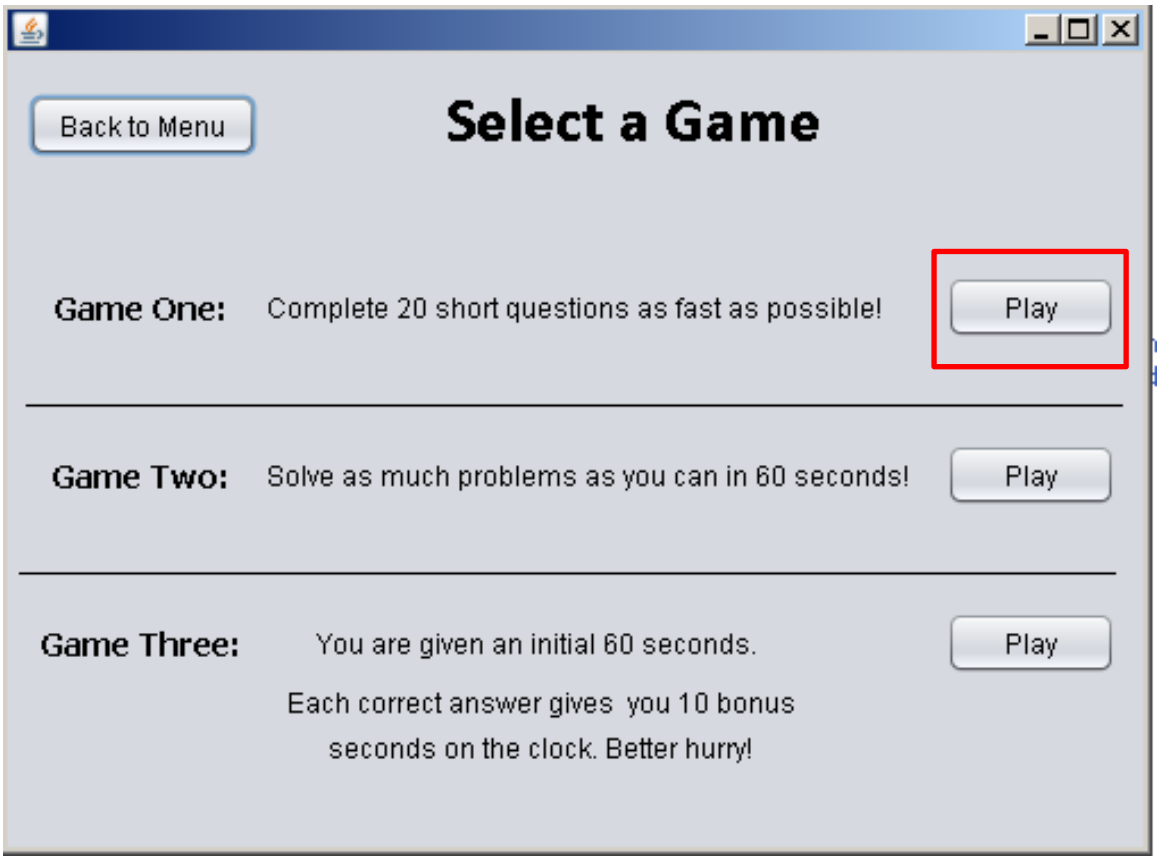
If an incorrect ID or password is entered, for example entering the ID “4” with a wrong password “1111” the user will be denied access to the application.

Game Testing – Checking the Answer

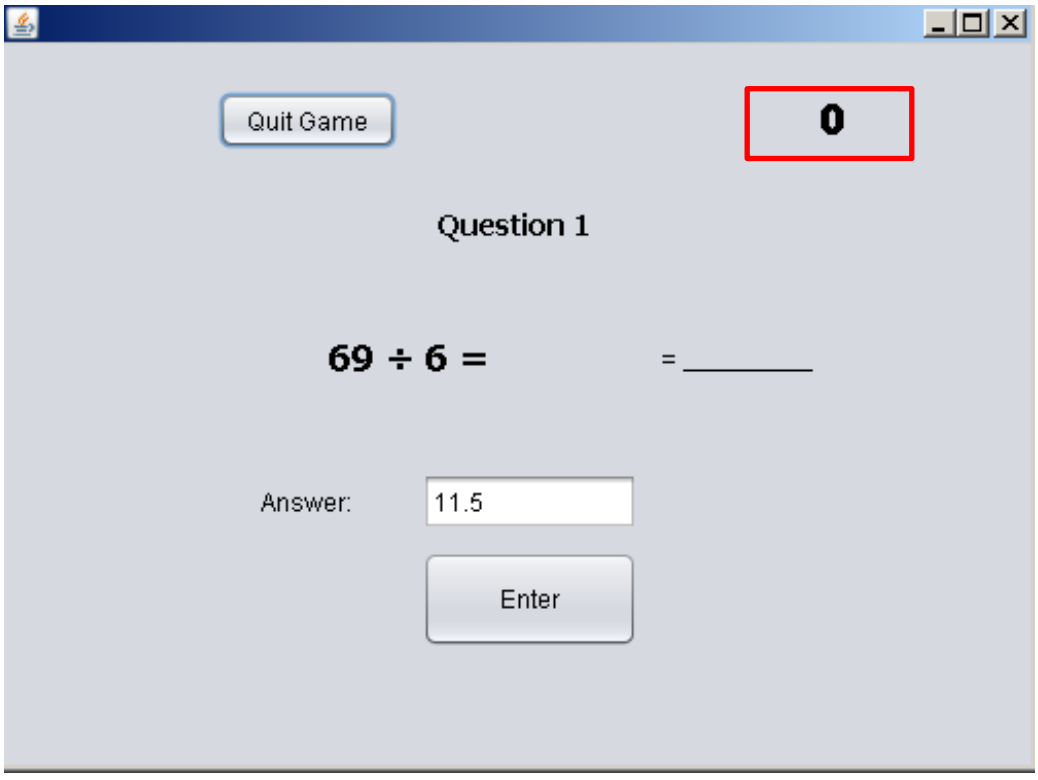
We are going to use the student Abrielle Belloni, with id “1” to test Game 1.



The user is taken to the “SelectGamesPanel”

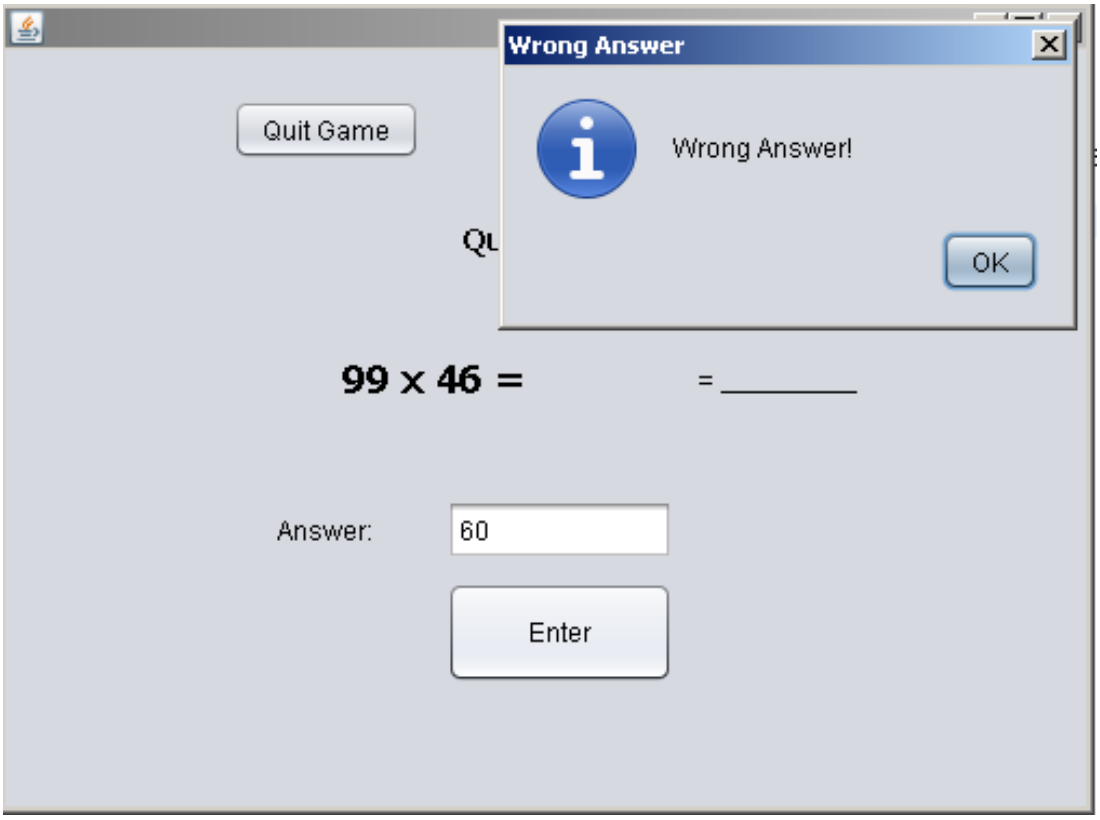


Abrielle is taken to the GamePlayingInterfacePanel. Here she plays the Math Game starting with a 0 in the timer.

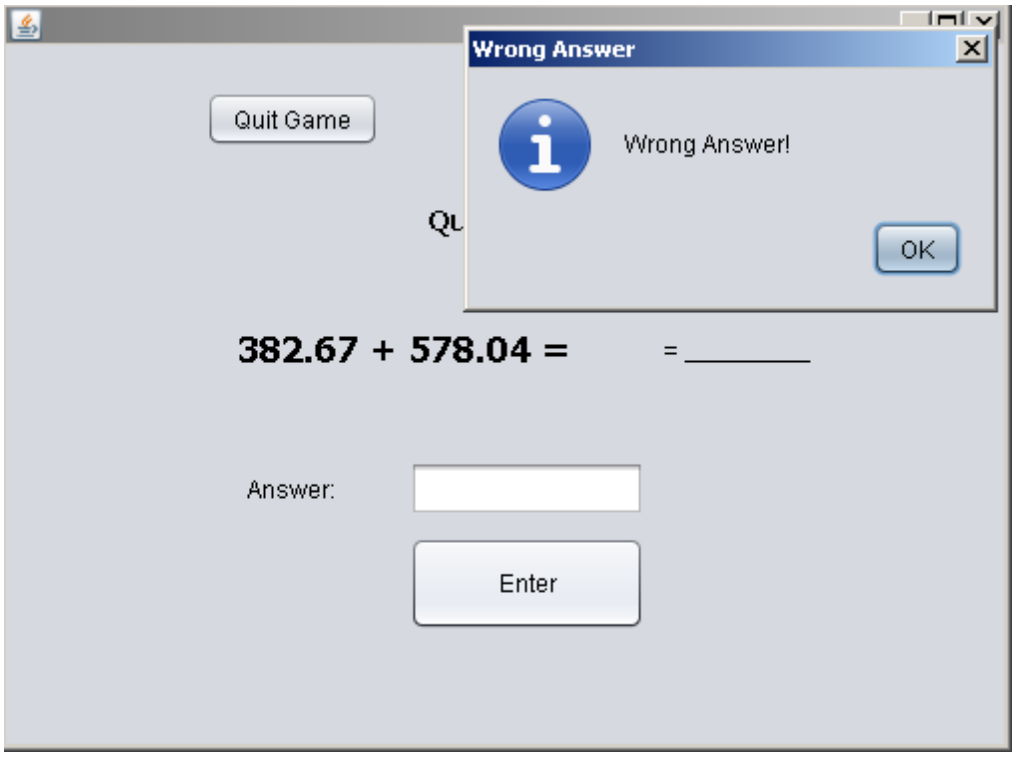


If the user inputs the right answer: 11.5. Then she wins +1 in the correctAnswers counter and she moves to the next question.

In the next question the user is presented with 99x46 which is equal to 4554. If the user inputs a wrong answer, for example 60, the following error is presented. The user is taken to the next question and the correctAnswers counter stays the same.

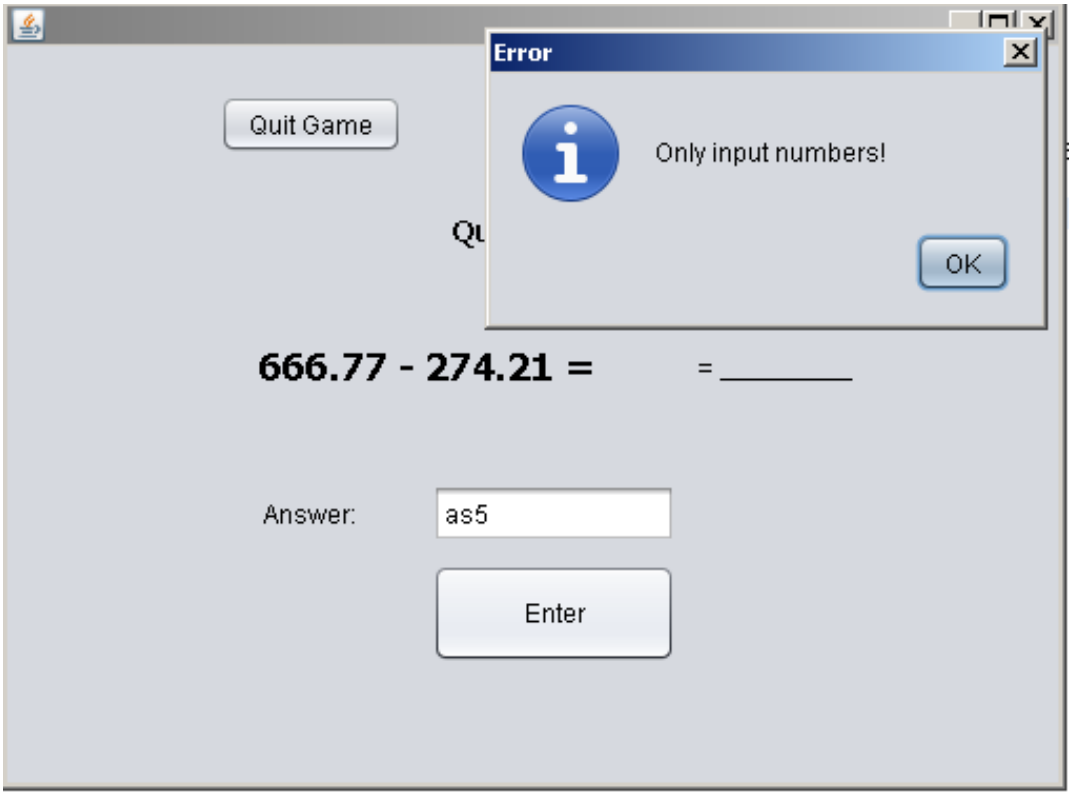


If the user leaves an answer in blank this means that the answer is not known and the student wished to proceed to the next question. This is why an empty answer is recognized as a wrong answer.

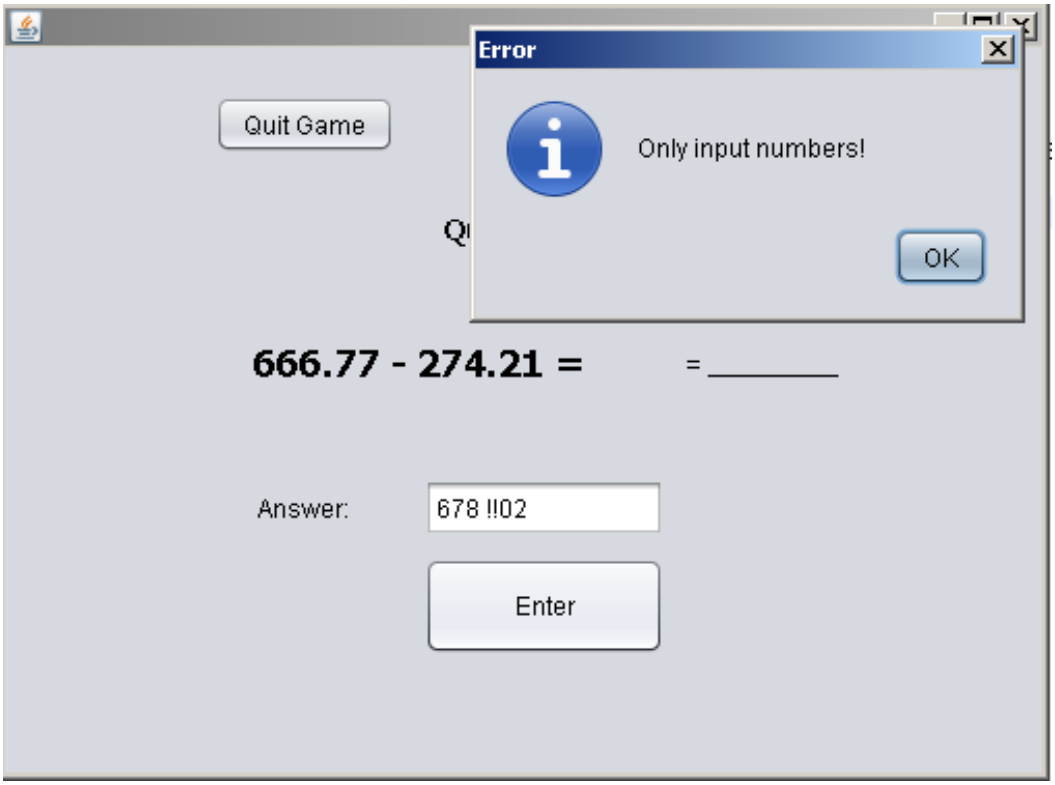


Nevertheless, if the user inputs a character that is not a number (typo) a warning message will appear and the user will be able to fill the answer correctly.

Example 1:



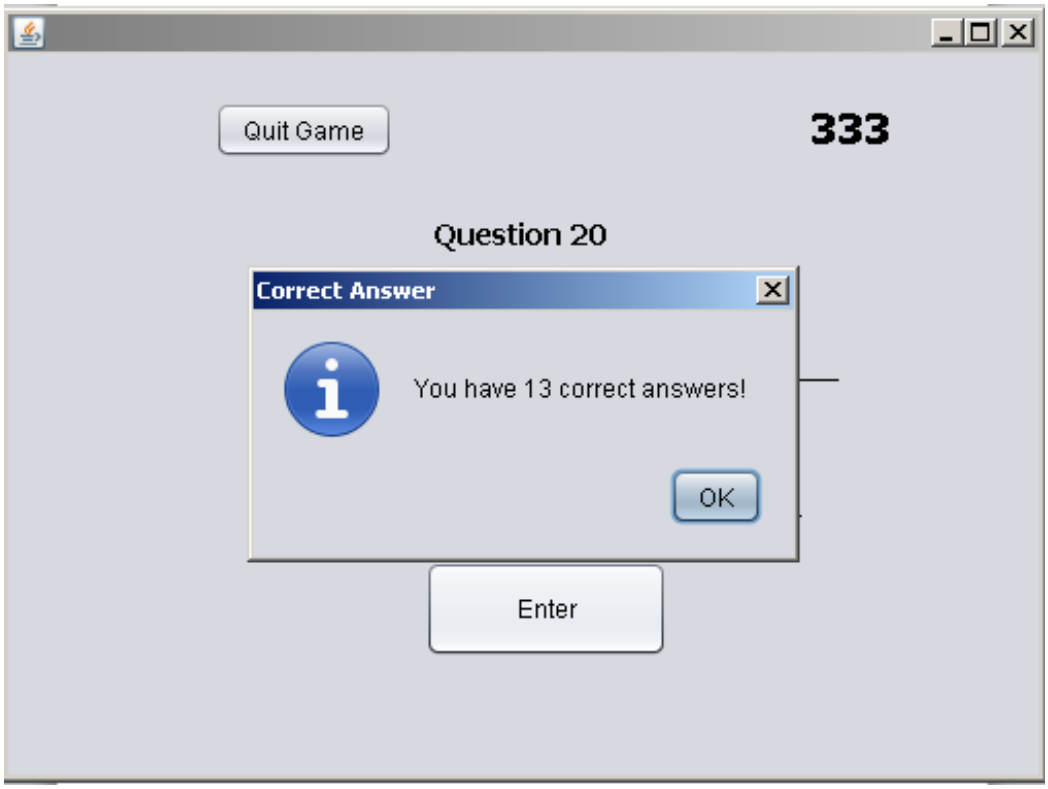
Example 2:


Quit Game
Error
Only input numbers!
OK
666.77 - 274.21 =
= _____
Answer:
678 !!02
Enter

Game Testing – The Game

User with ID = 4 plays a complete trial of Game 1.

The user achieves 13 correct answers over 20 (65%) with a time of 333 seconds.



The way that the score for game one is generated is explained in the previous section A1 (refer back for details).

The user has taken 333 seconds to complete the game.

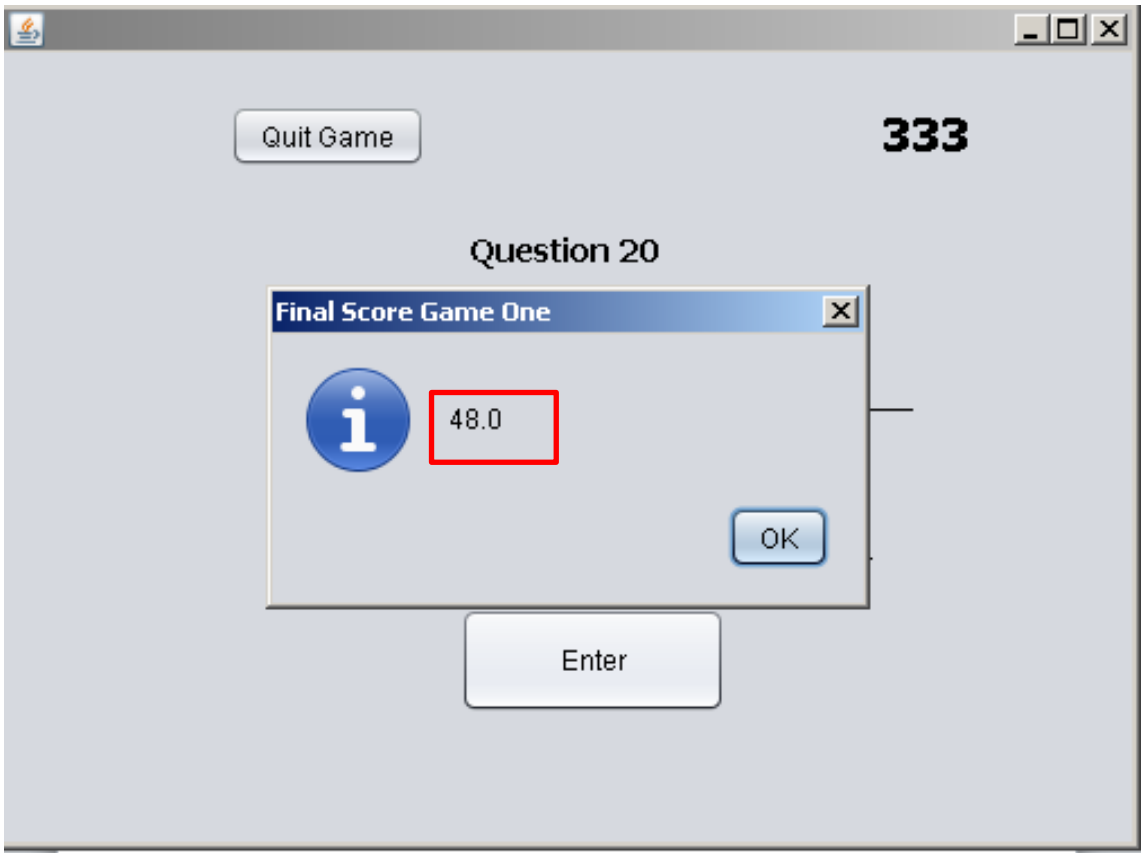
The maximum time (500 seconds) is divided by 333 and the bonus factor is calculated (1.5).

The percentage of correct answers (65%) is multiplied by the bonus factor and the answer is divided by two →

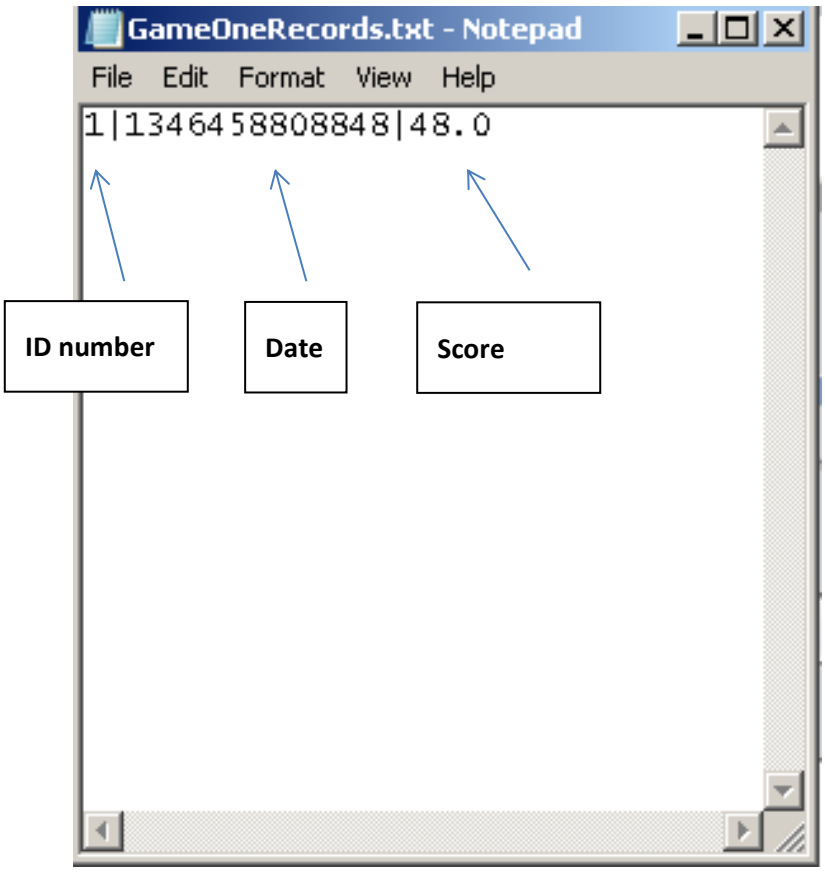
(65 * 1.5)/2 = 48.75

The answer is casted to an int so the number is rounded down to 48

The final score = **48**

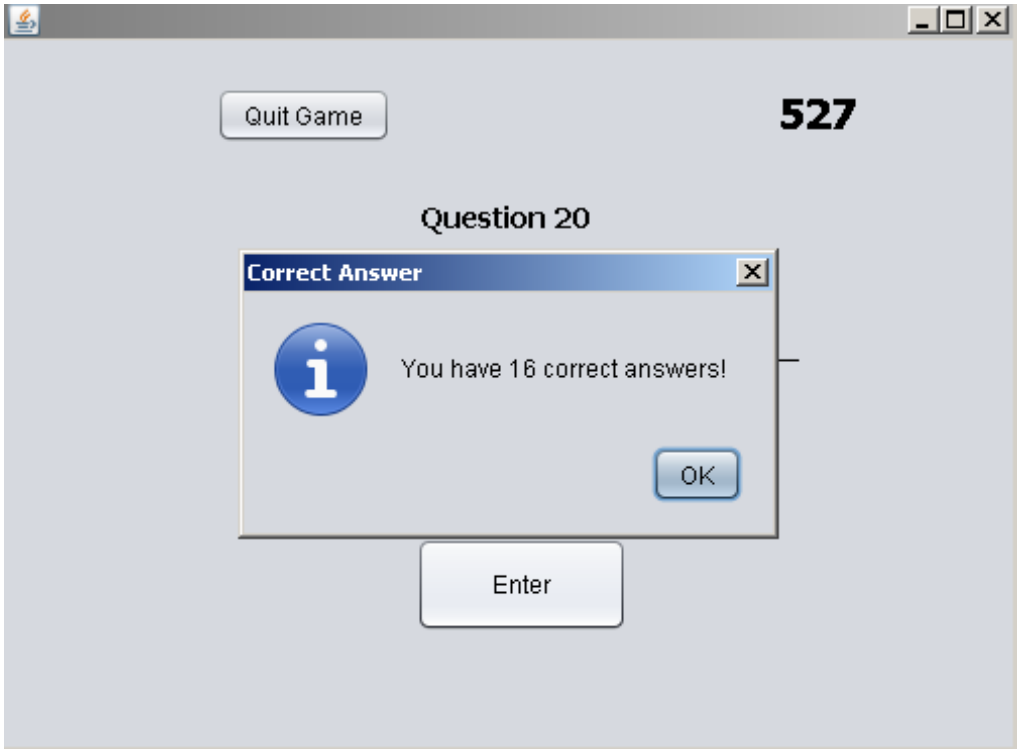


This is saved to the currently empty sequential file “GameOneRecords.txt”



We will now try adding to the file when there is **more than 1 existing record**. We will use the user account Pietro Quierolo, with ID number 10 to play another trial of Game 1.

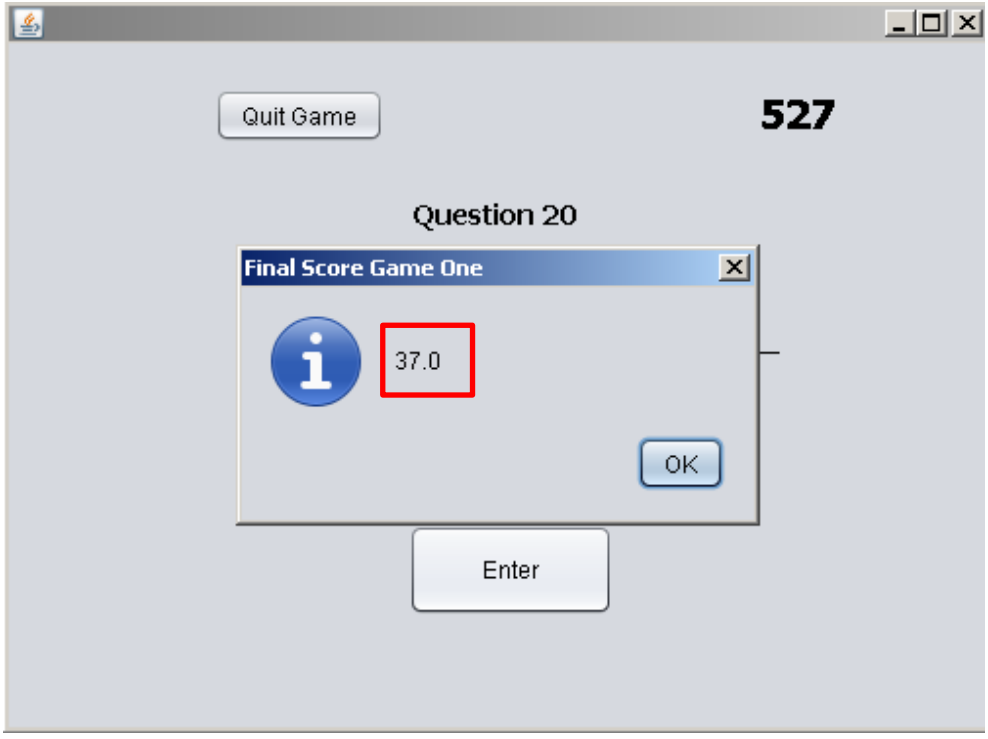
This user achieved 16 correct answers over 20. (80%) within 527 seconds.



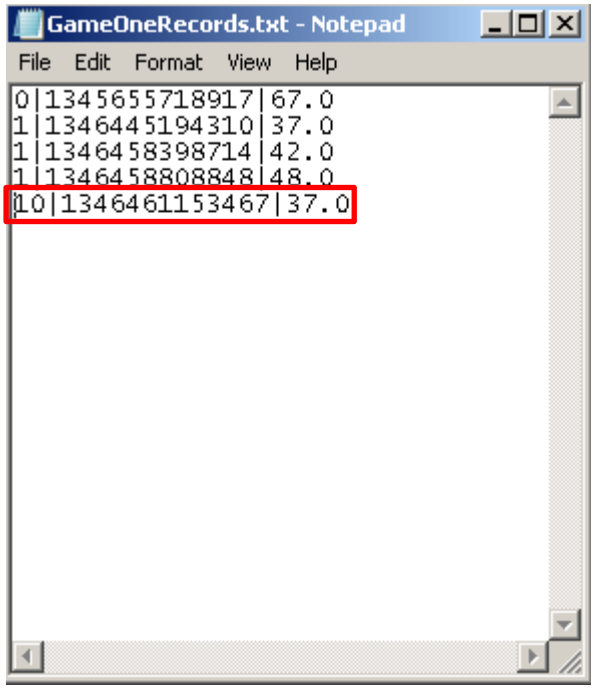
500/527 = 0.95 (The bonus factor now becomes a penalty factor for taking so much time)

(80% * 0.95)/2 = 37.9

The final score is **37.**

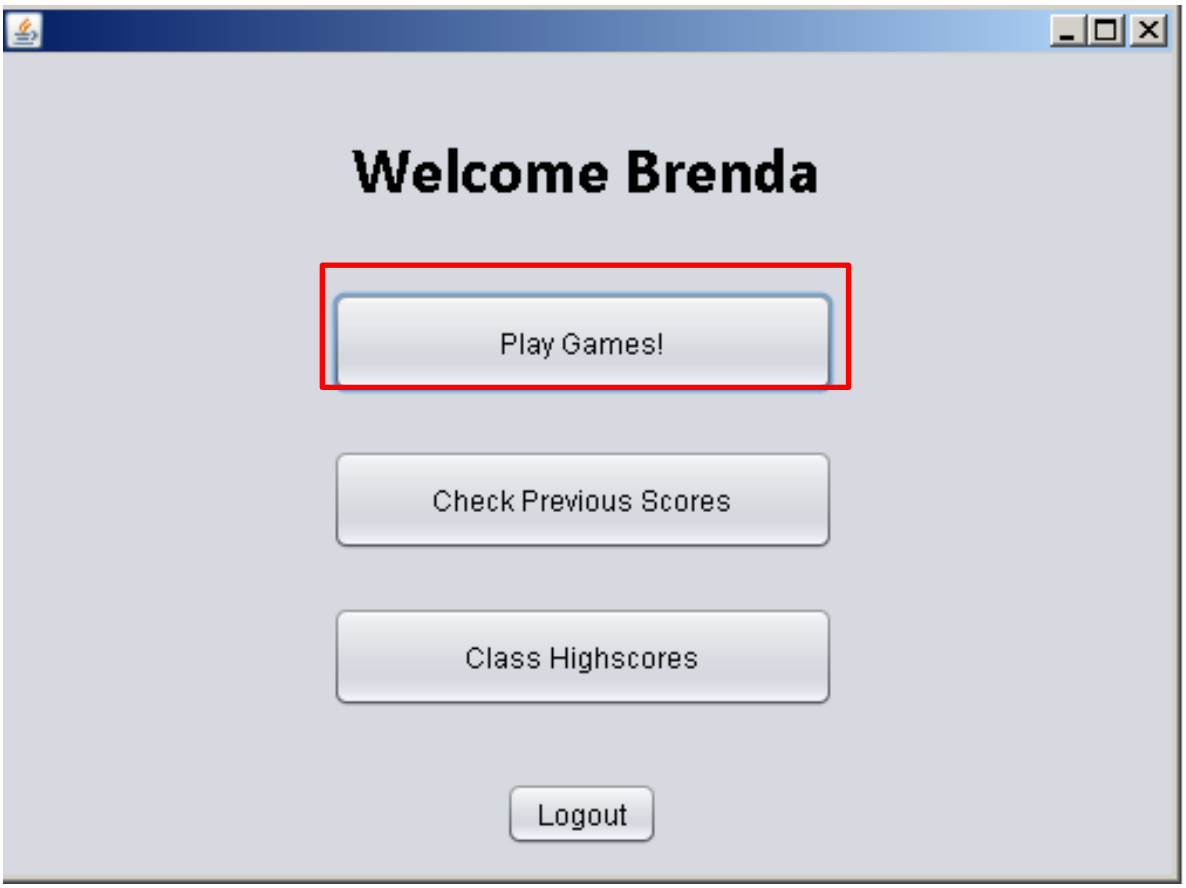


This GameRecordNode object is now saved in the sequential file “GameOneRecords.txt” along with the other existing game one records.



Game Two

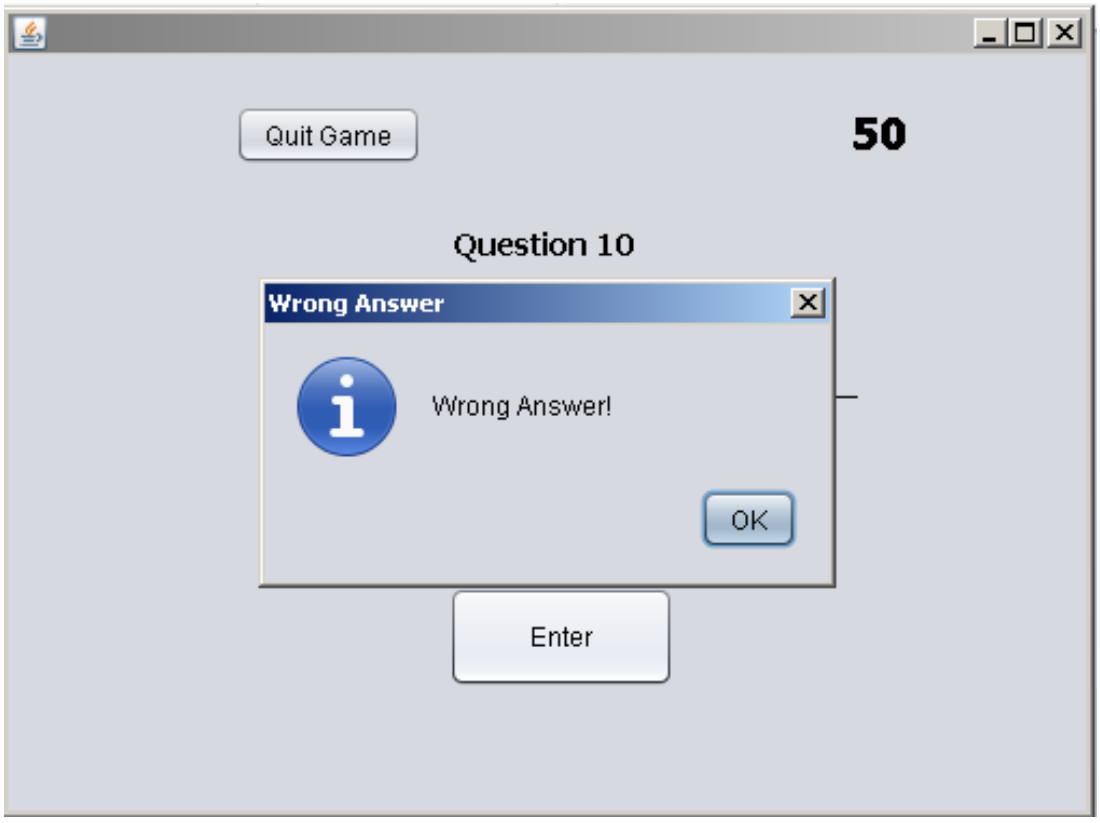
We will now use another user, Brenda Velasco with ID 18, to test Game two.



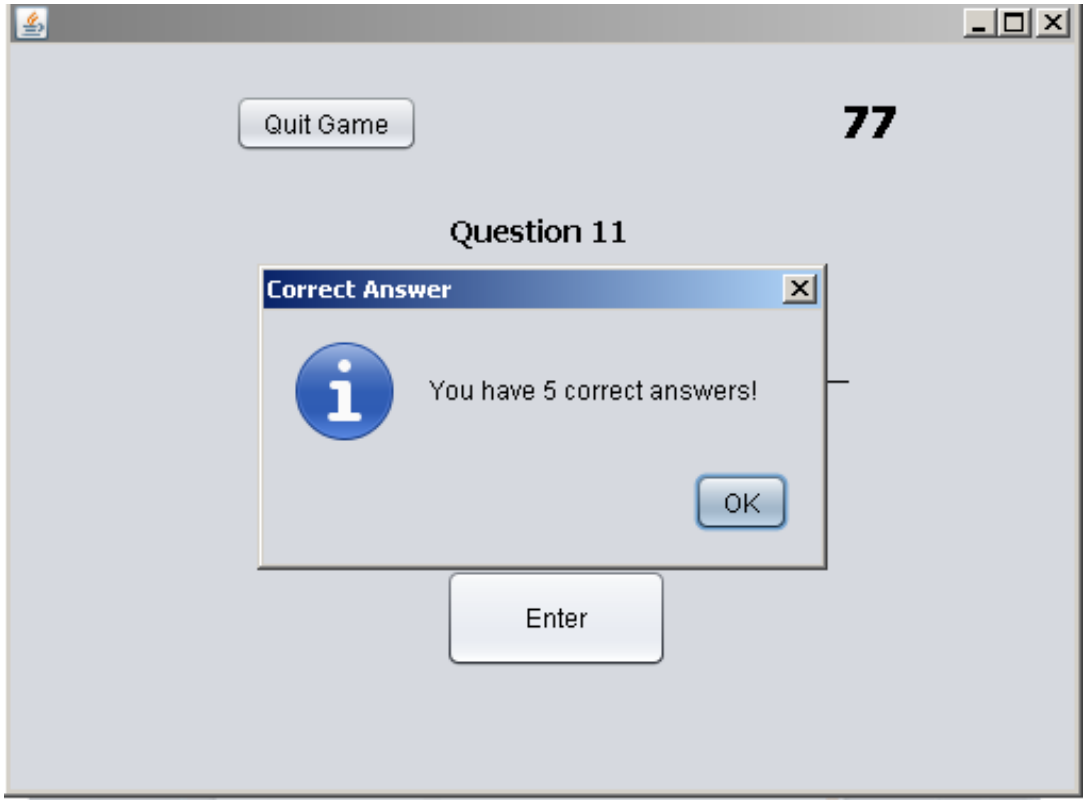
The user is taken to the SelectGamesPanel

The same panel is used when playing Game two so the testing of the common elements of the panel will not be tested (such as checking the answer or not inputting numbers).

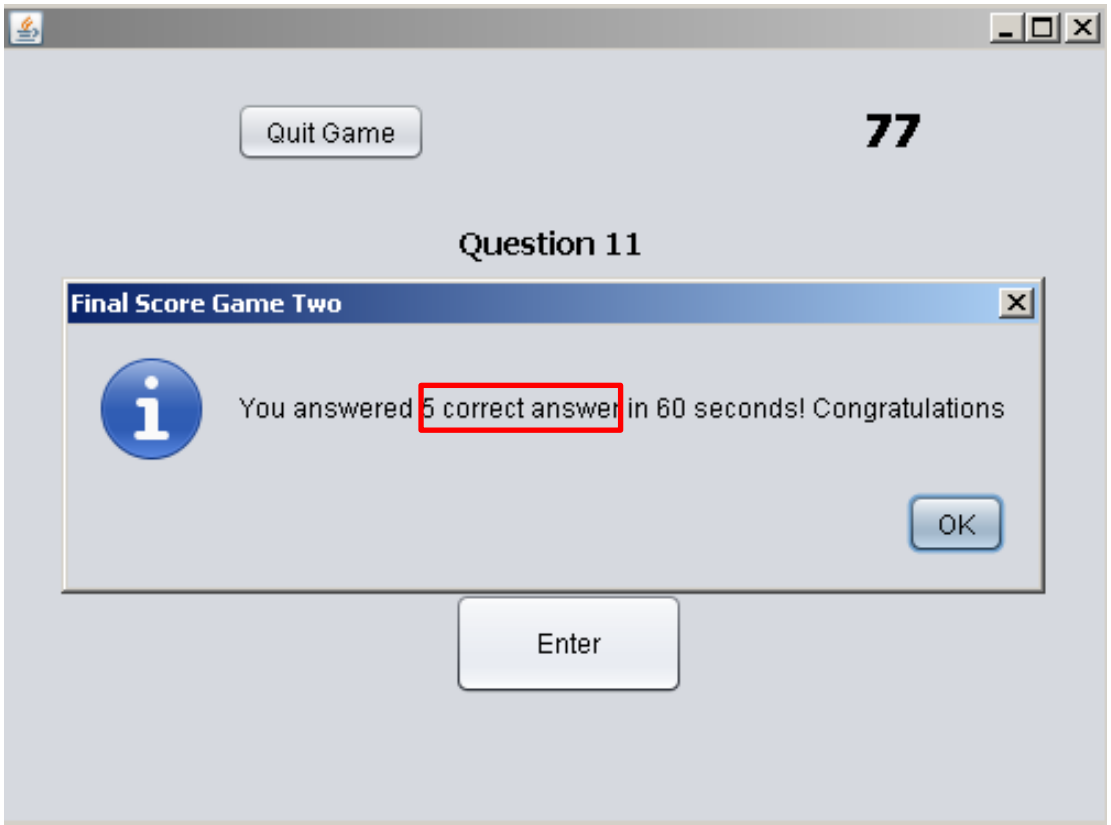
As long as the timer does not pass 60 the user is allowed to keep attempting as much questions as possible.



When the timer passes 60 the user is allowed ONE more question before the game finishes and the score is calculated.

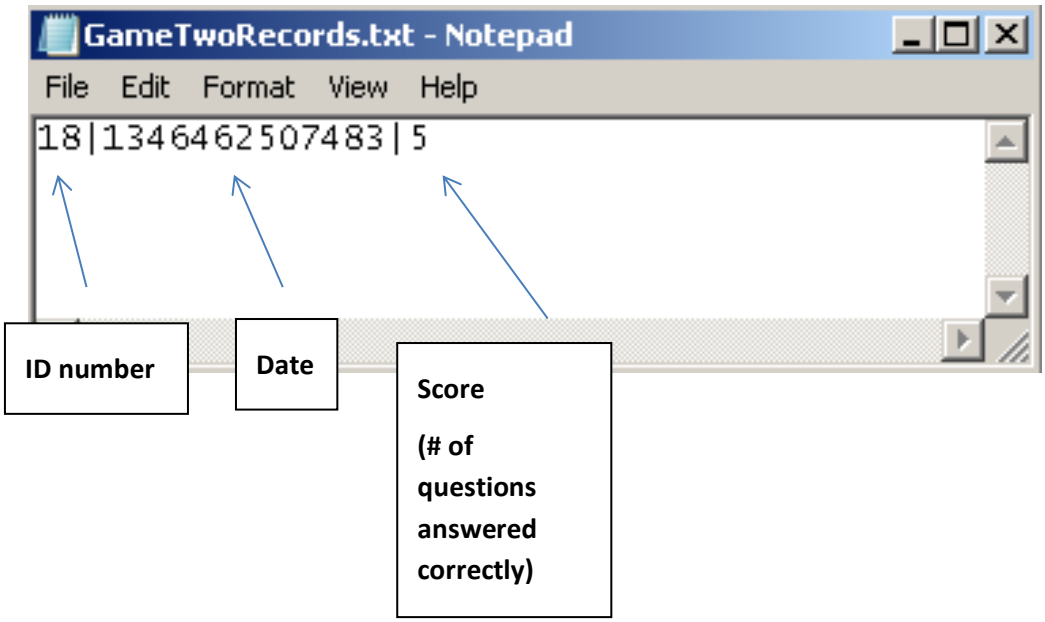


The game now stops.

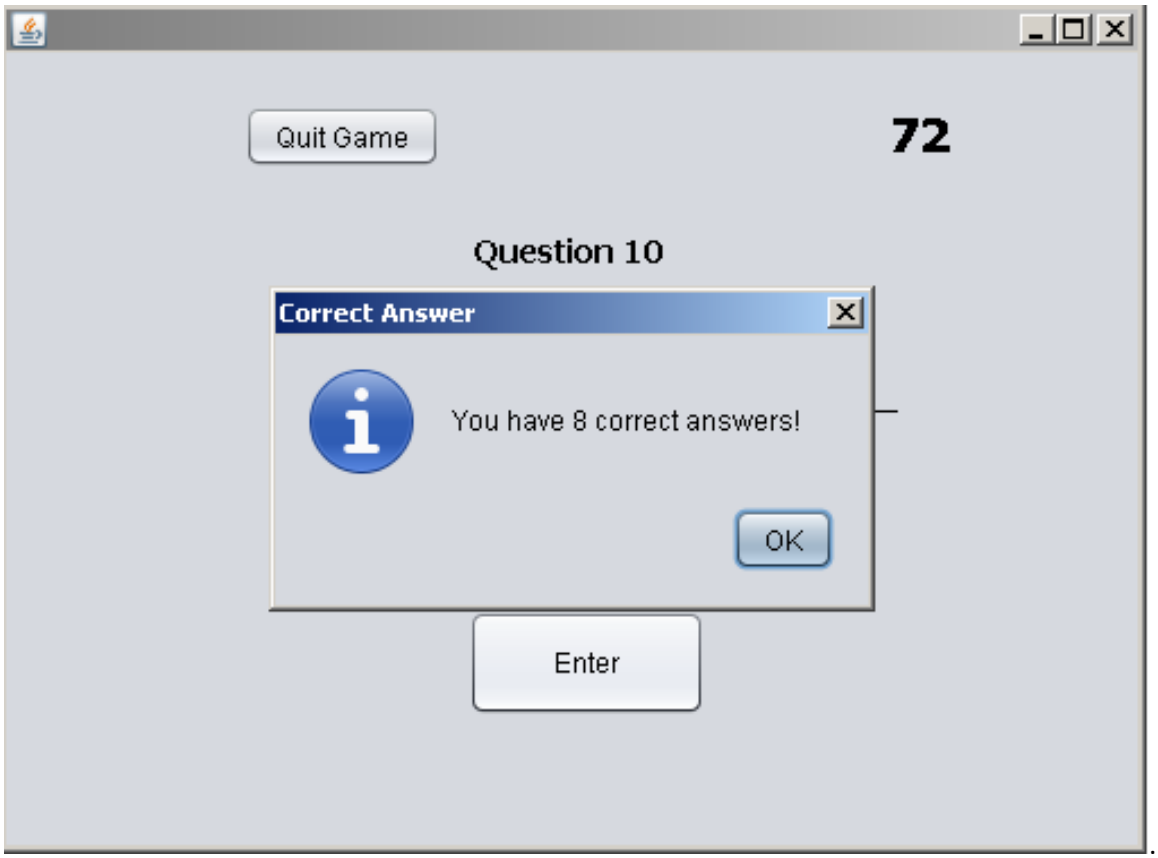


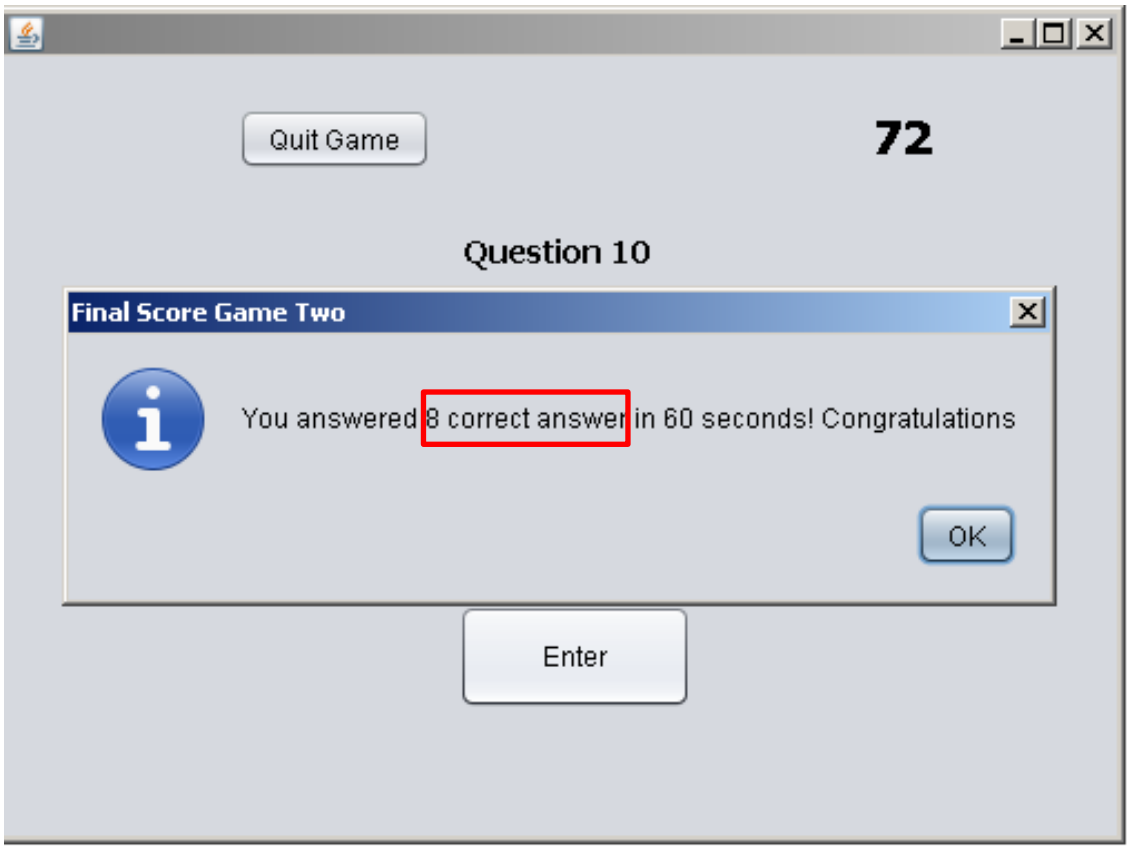
A GameRecordNode object is constructed with the data regarding the played game and the score.

This object is saved as a record in the currently empty sequential file “GameTwoRecords.txt”.

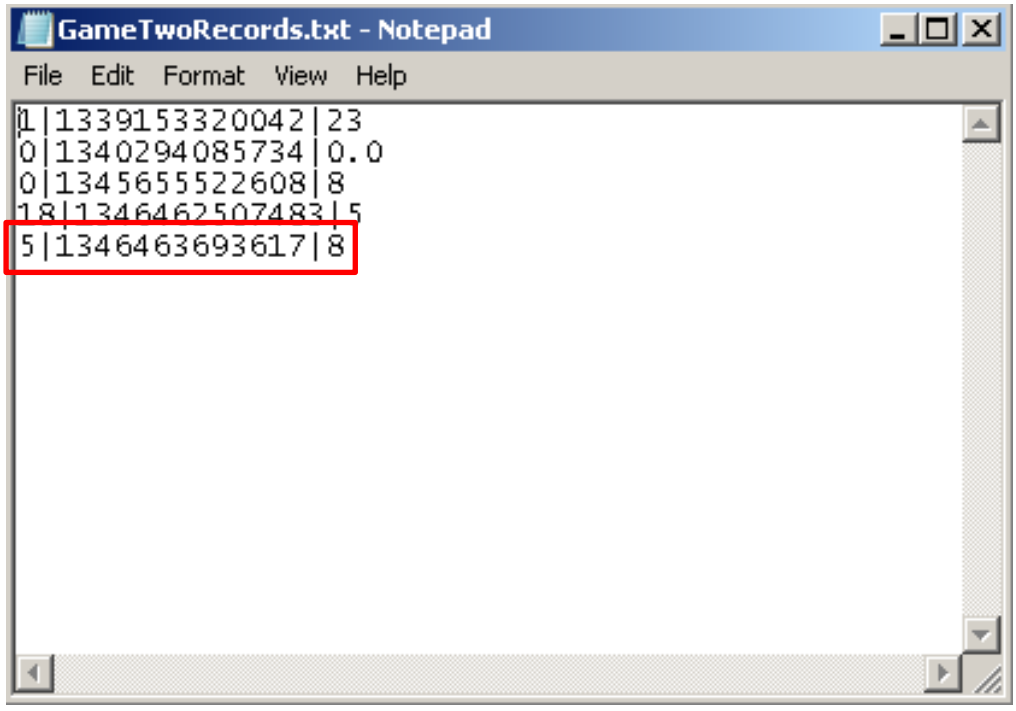


We will now try adding to the file where there is **more than 1 existing record**. We will use the user account Giacomo Ducato, with ID number 5 to play another trial of Game 2



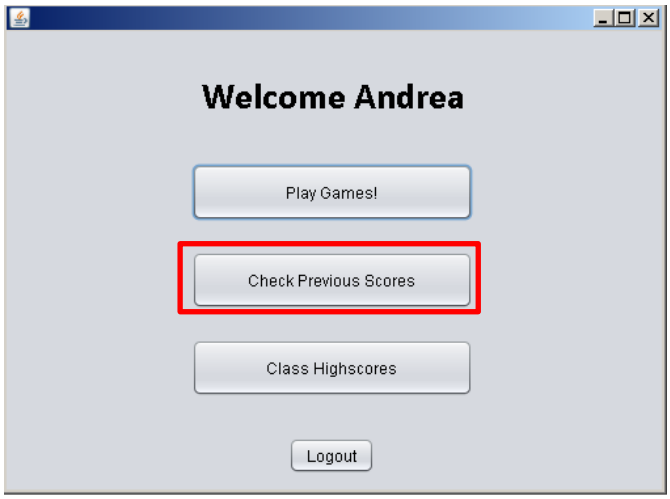


This GameRecordNode object is now saved in the sequential file “GameTwoRecords.txt” along with the other existing game one records.



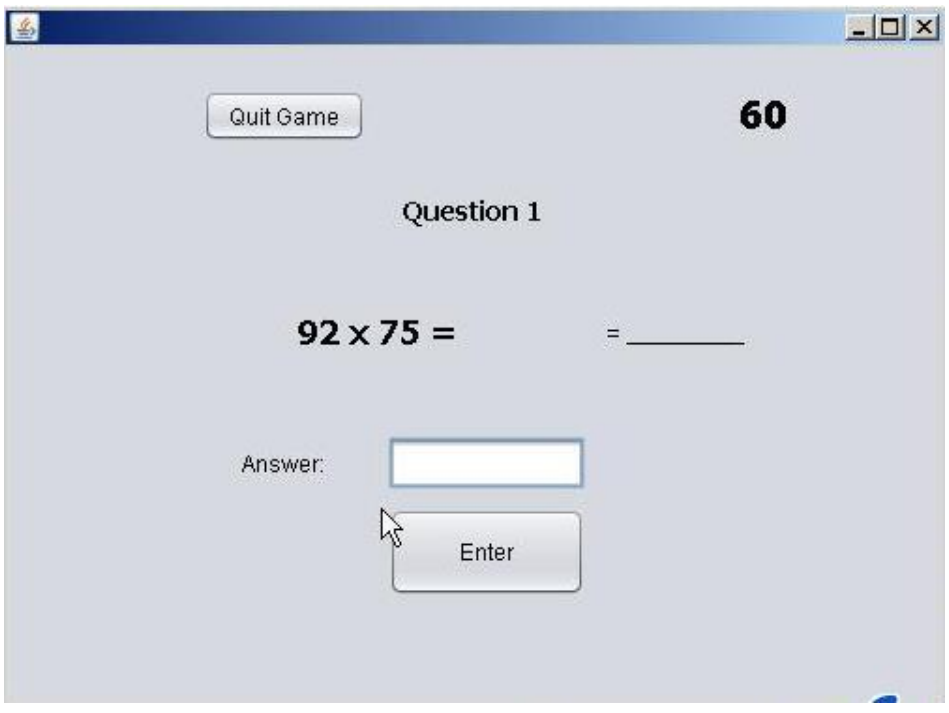
Game Three

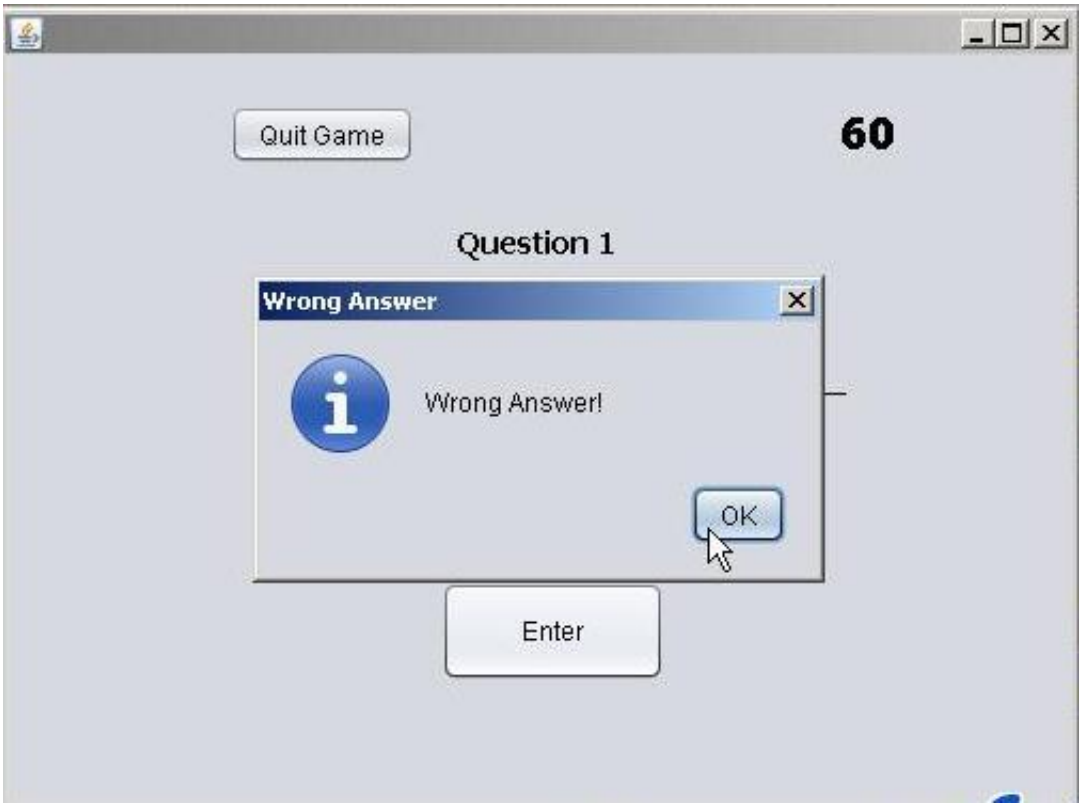
Now Game 3 will be tested. We will use another user, Andrea Woodman with ID 20, to test this game and see how the record is saved in the currently empty sequential file "GameThreeRecords.txt".



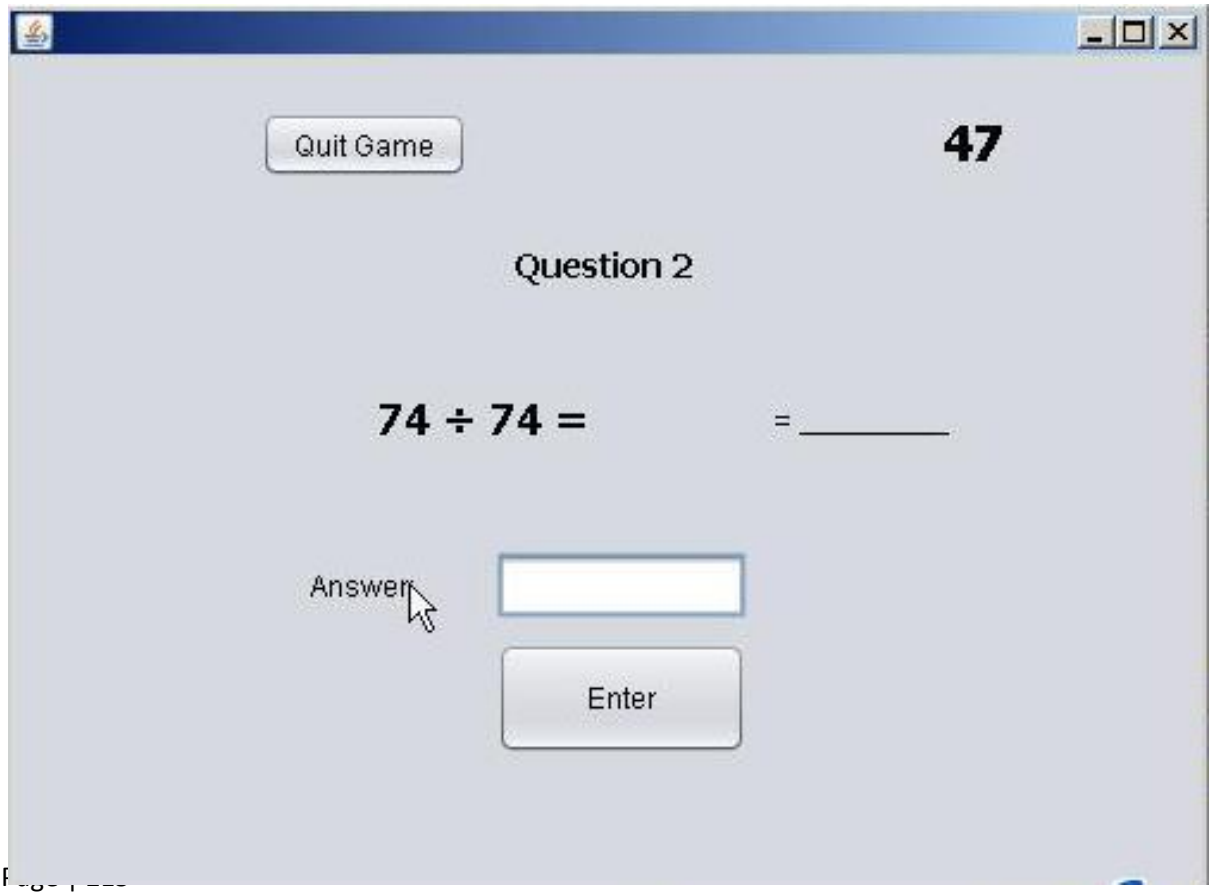
Game Three also uses the common GamePlayingInterfacePanel class as an interface when playing the game except that the timer is not in a stop watch form that goes from 0 to 60, it is a countdown timer that goes from 60 to 0.

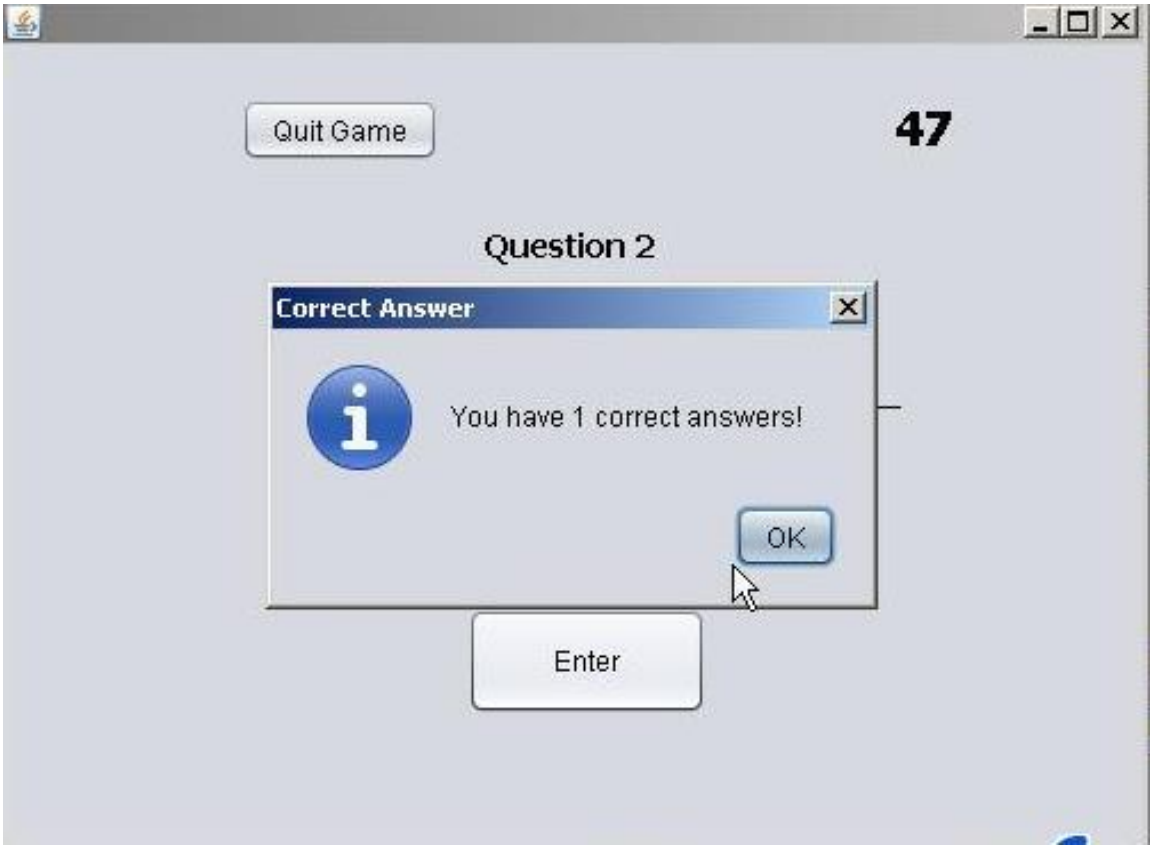
In Game Three, as long as the timer doesn't reach 0, the student is still able to attempt as many questions as possible. Each correct answer gives 10 bonus seconds in the countdown timer. When the timer reaches 0 the user is allowed to answer 1 more question before the game ends (this is why at the end of the games you will see a negative time timer).



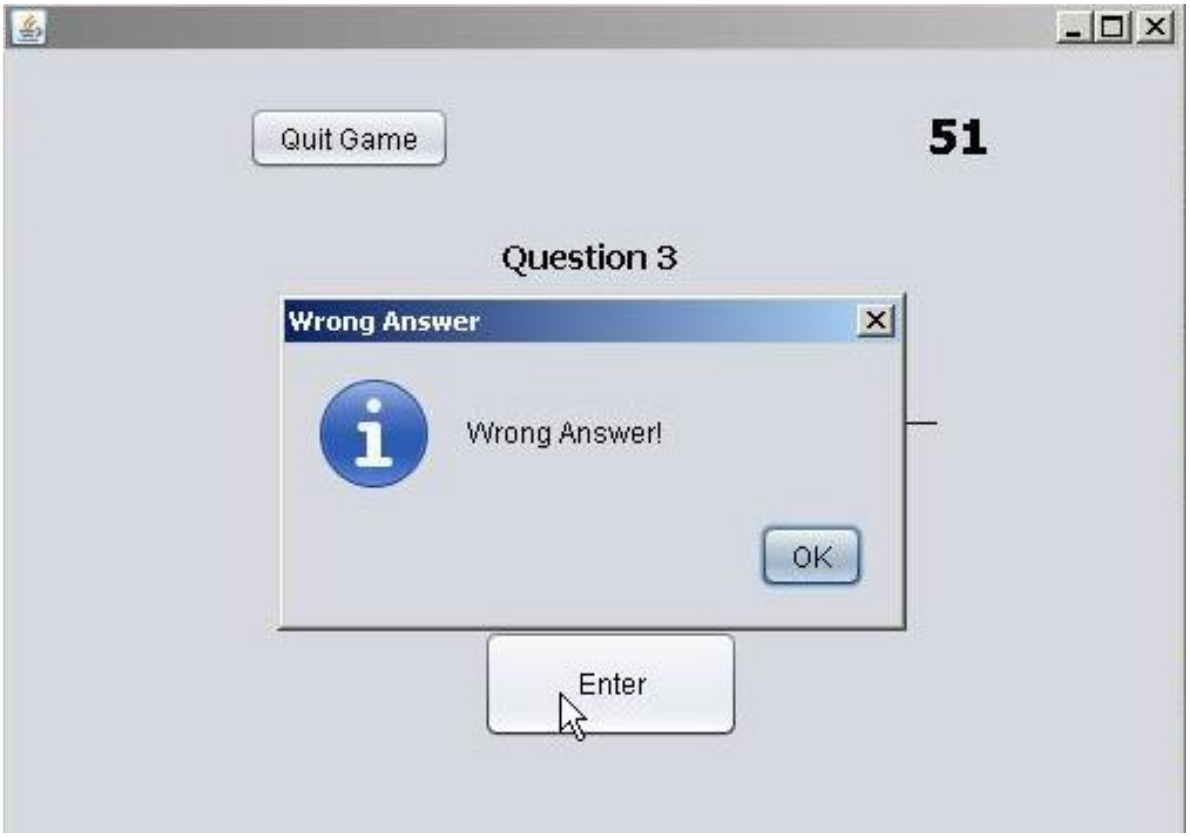


We see that time decreases normally when the questions are not answered properly (there is no 10 second bonus).

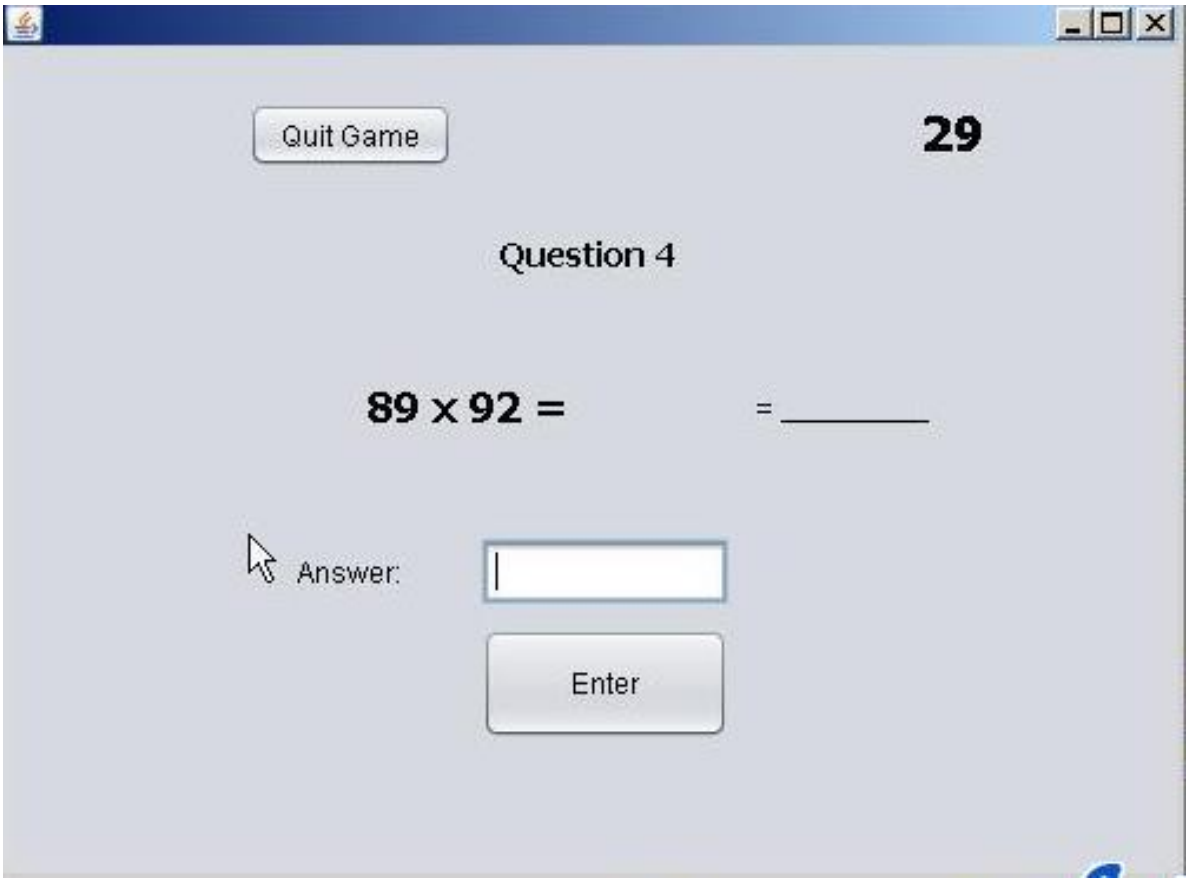


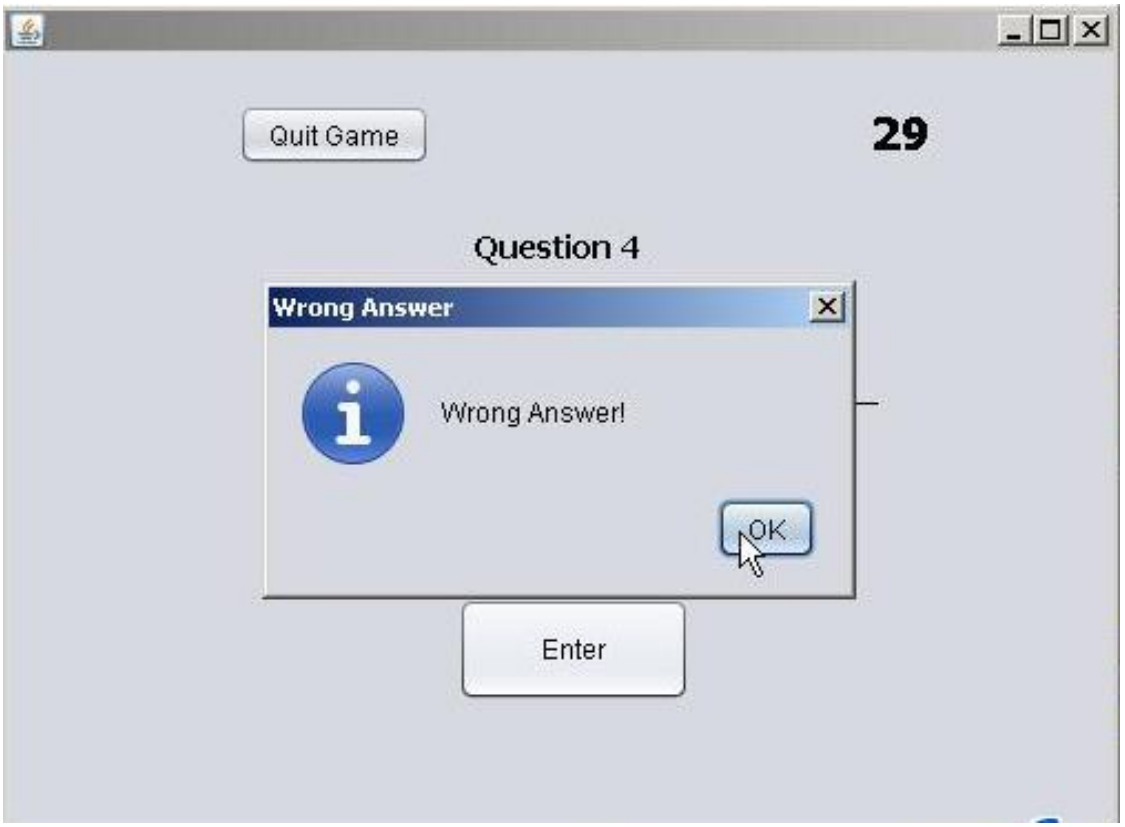


Note: As the questions was answered correctly the Timer went 10 seconds up (in this screenshot we see it is not 10 seconds exactly but this is because it took time to go to the next question and take the screenshot) The same applies to the other screenshots.

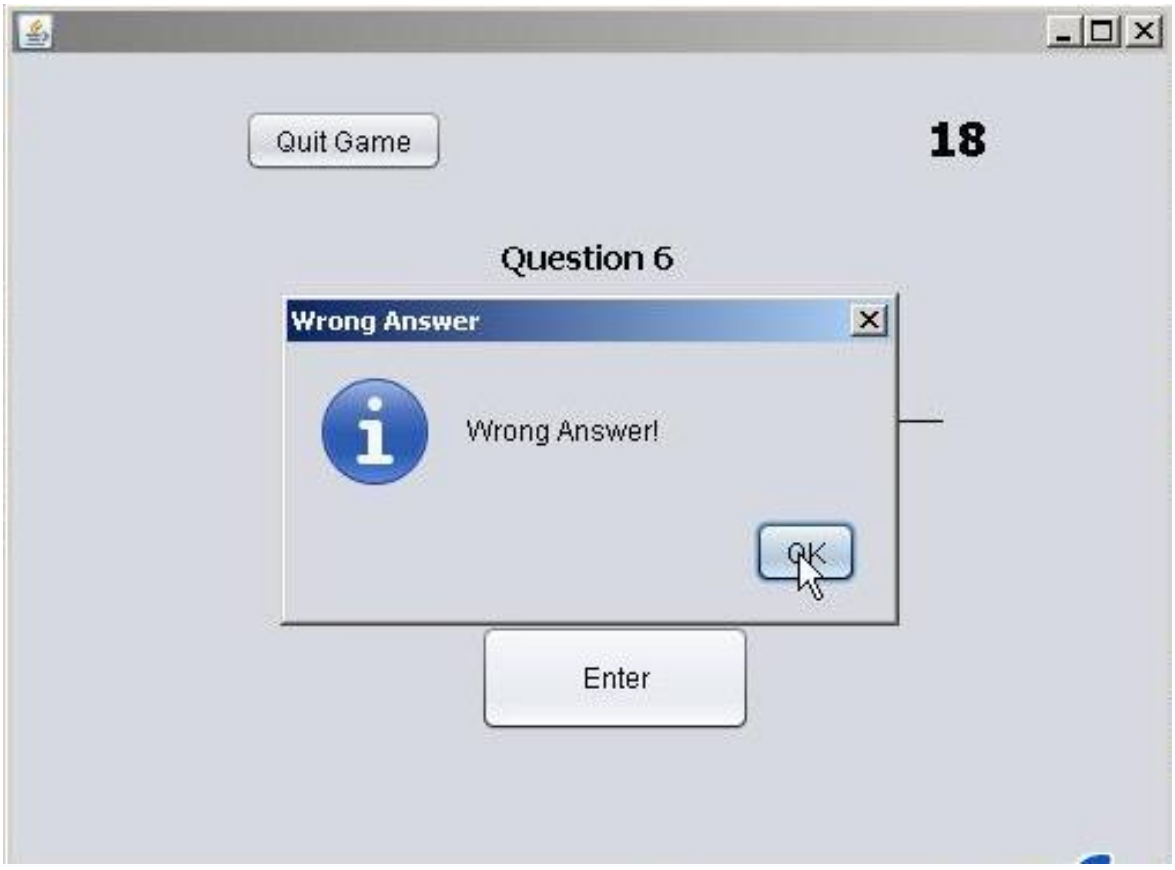


Time continues to decrease as the user fails to answer correctly. The countdown becomes closer and closer to 0.

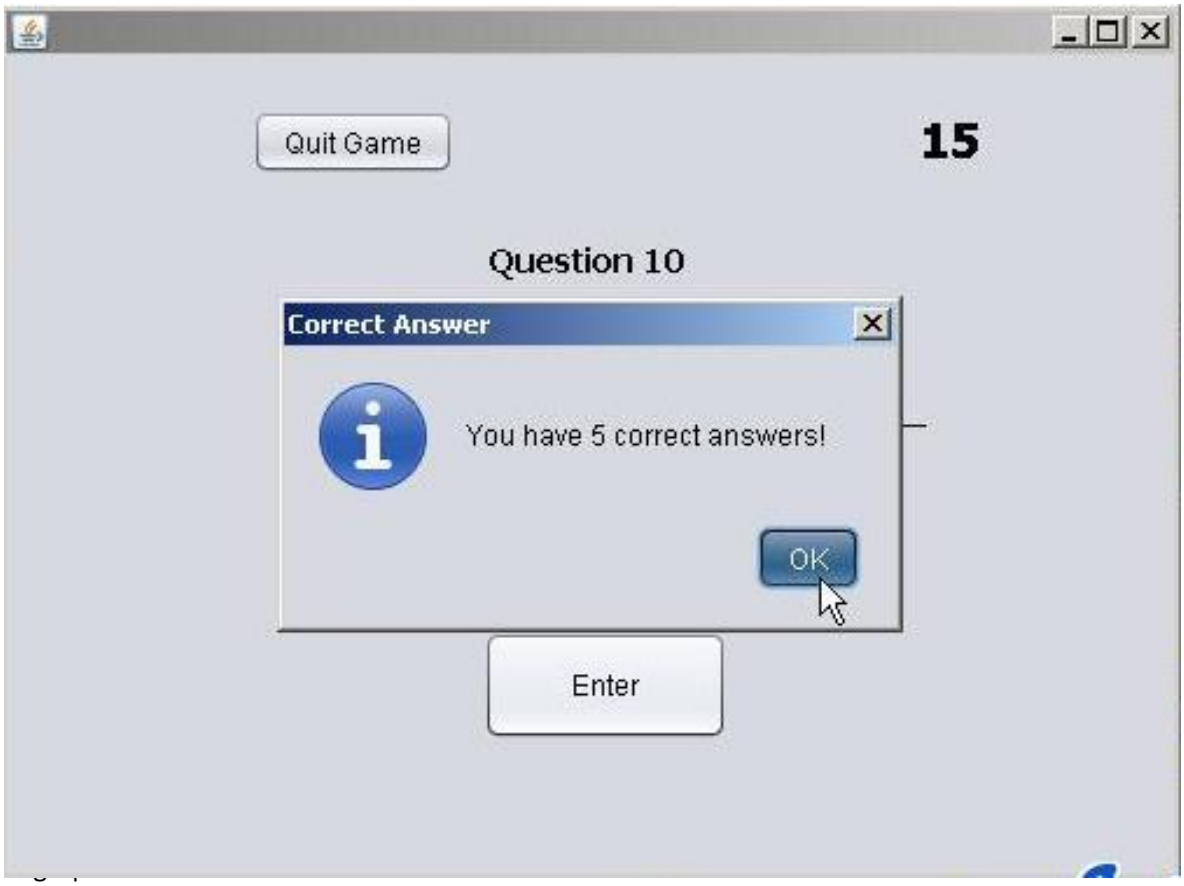




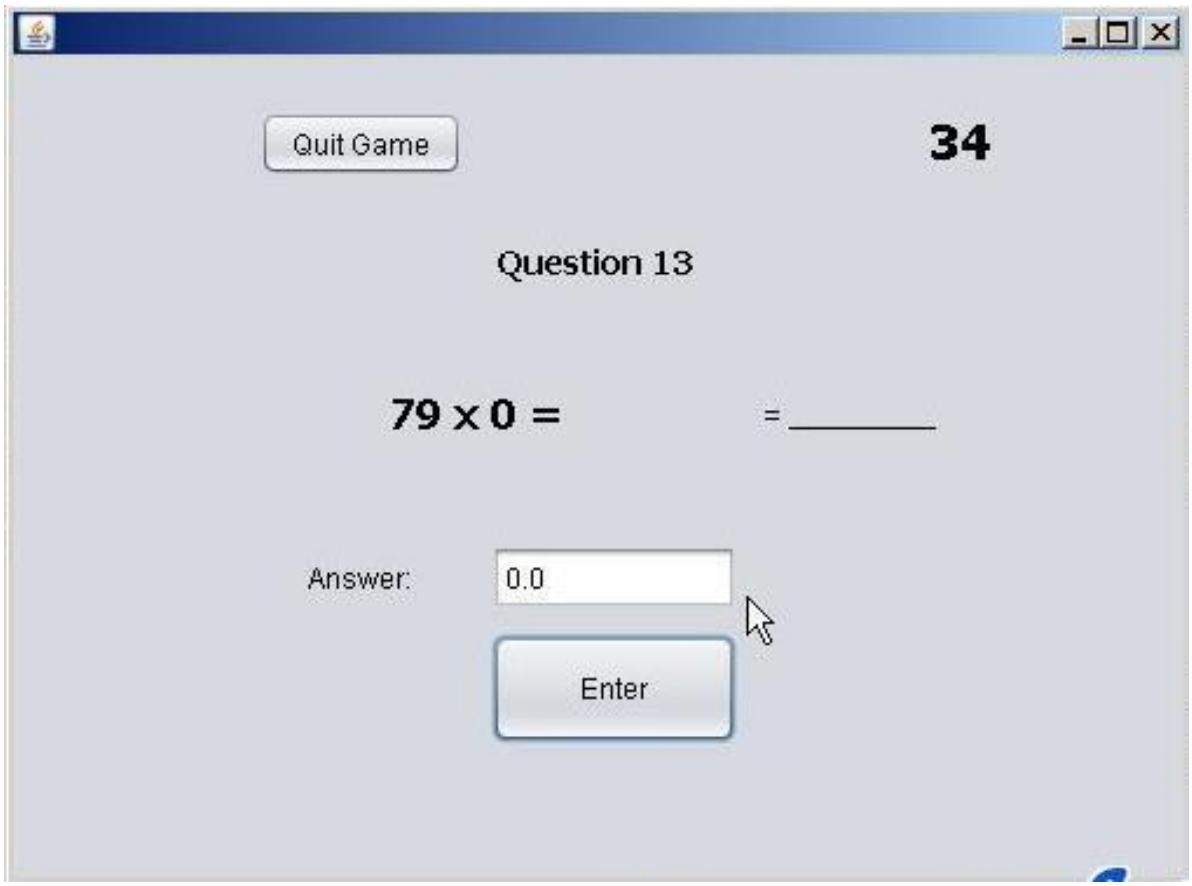
As the user keeps failing to answer any questioncorrectly



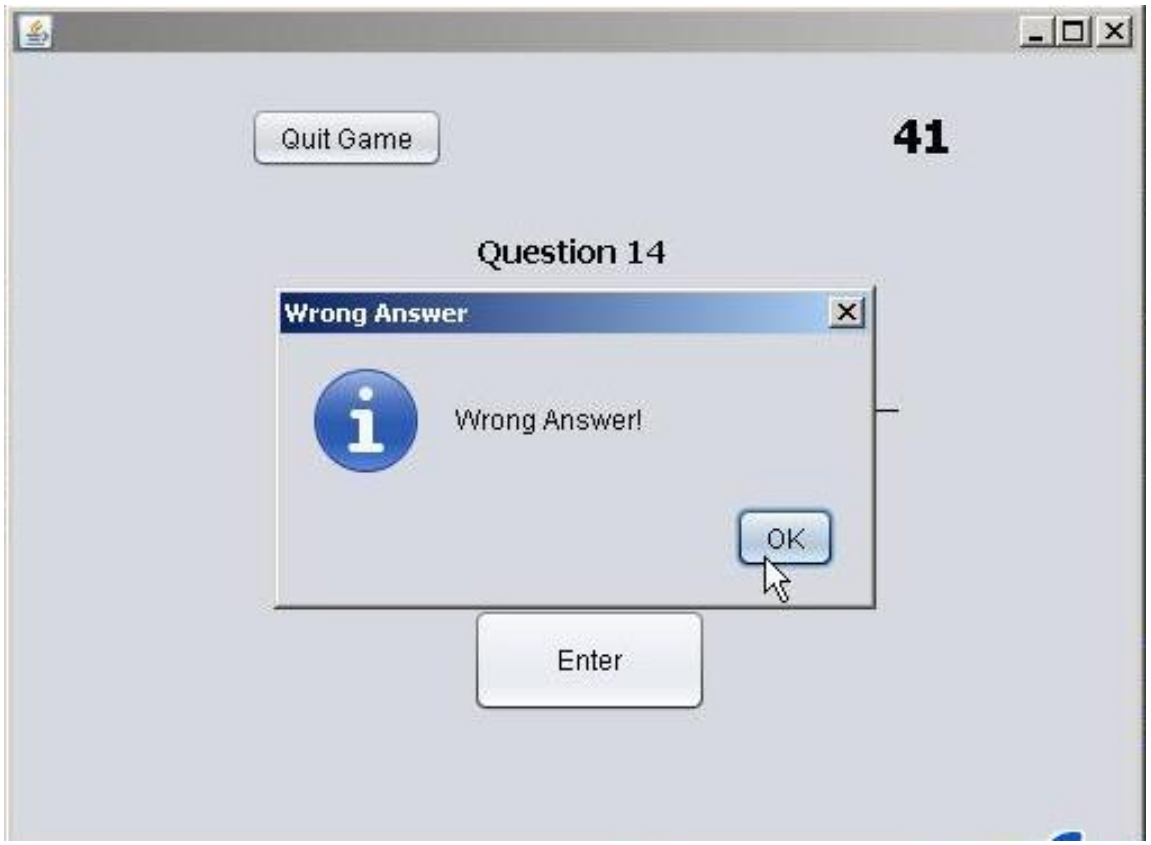
We see the situation in which the countdown came really close to 0 but then suddenly the user starts to answer correctly.

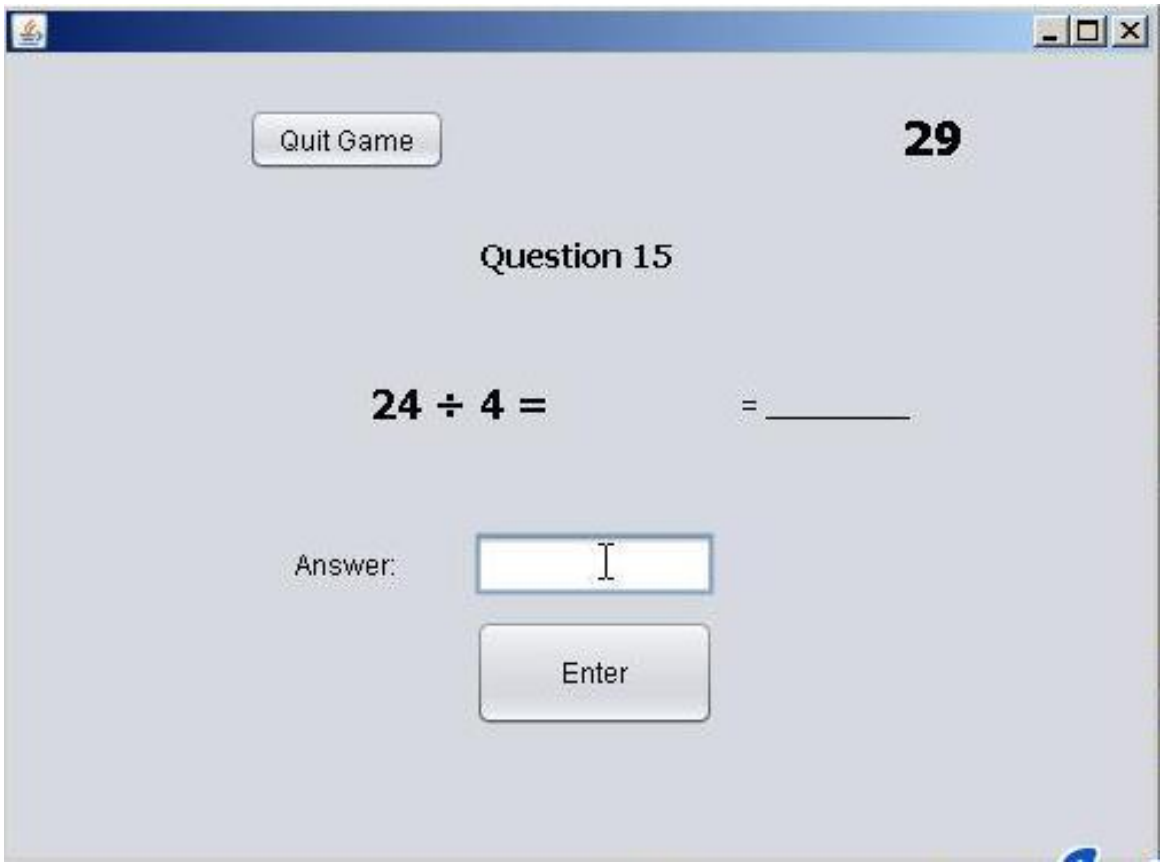


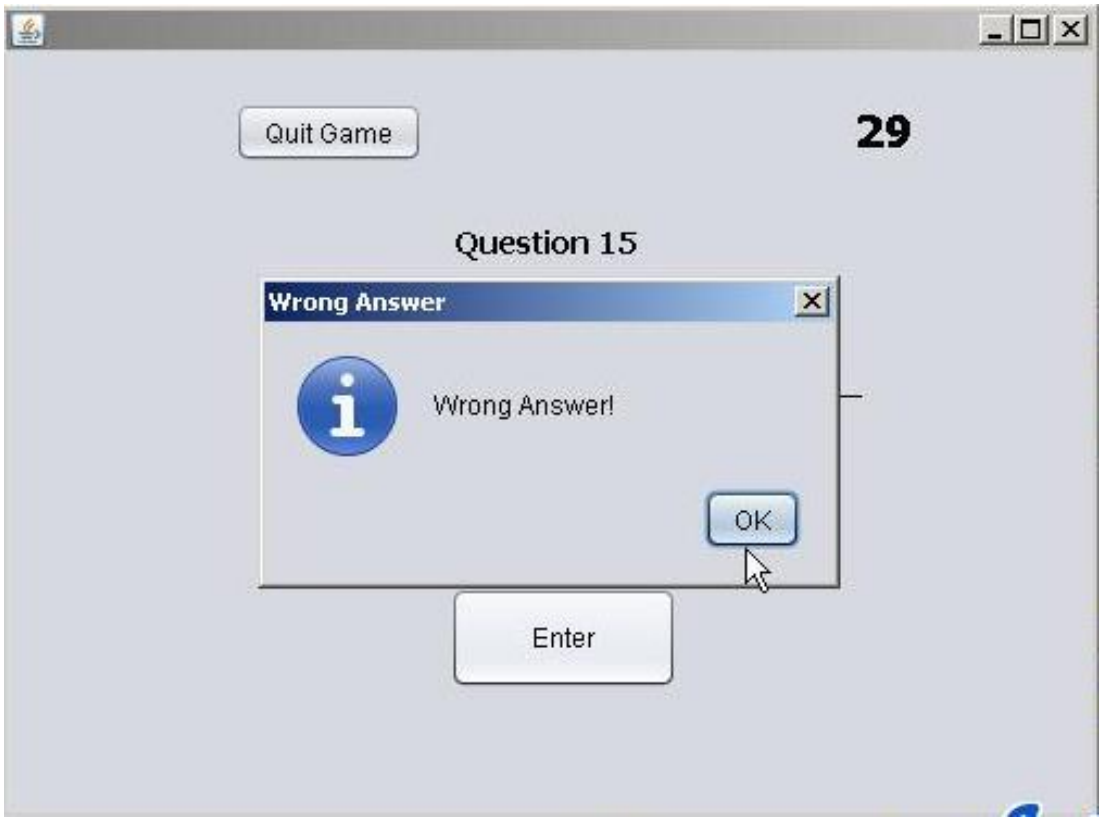

Quit Game
22
Question 11
Correct Answer
You have 6 correct answers!
OK
Enter


Quit Game
34
Question 13
79 x 0 =
= _____
Answer:
0.0
Enter

The user gets to a peak of 41 seconds and then suddenyl fails to answer all questions from this point forward.





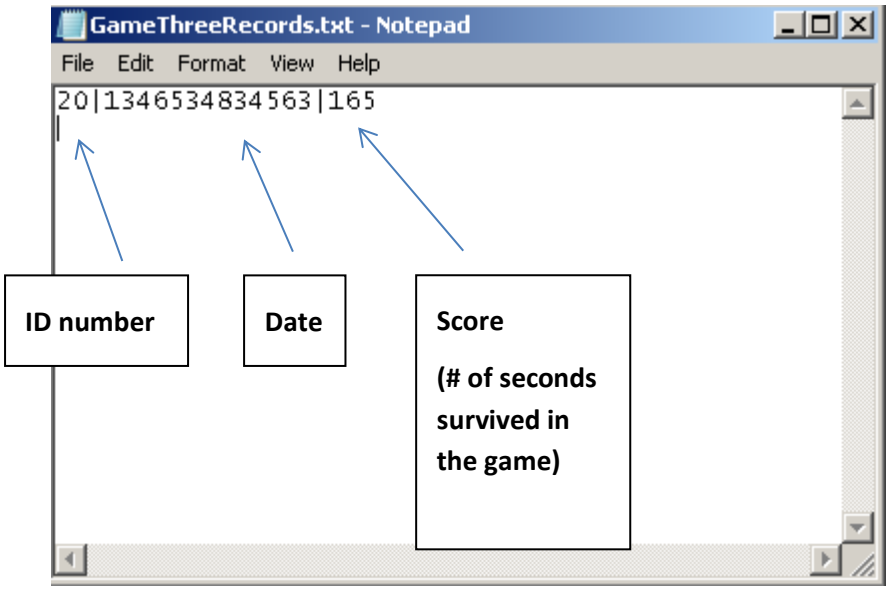


As the timer is now below 0 the game allows the user to attempt one more question before the game finishes.

The game now stops.

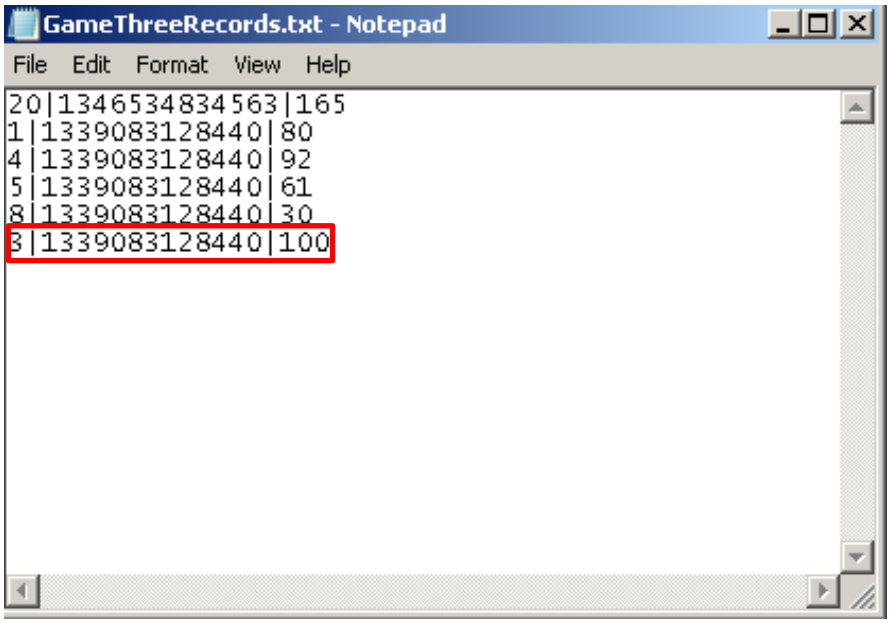
The user survives a total of 165 seconds (this is his final score).

We see that the GameRecordNode object constructed for this game trial is recorded in the currently empty sequential file “GameThreeRecords.txt”.



We are now going to use a different user, Maria Bustamante with ID 3, to test how the program adds a new Game three record to the sequential file “GameThreeRecords.txt” with a few existing records stored.

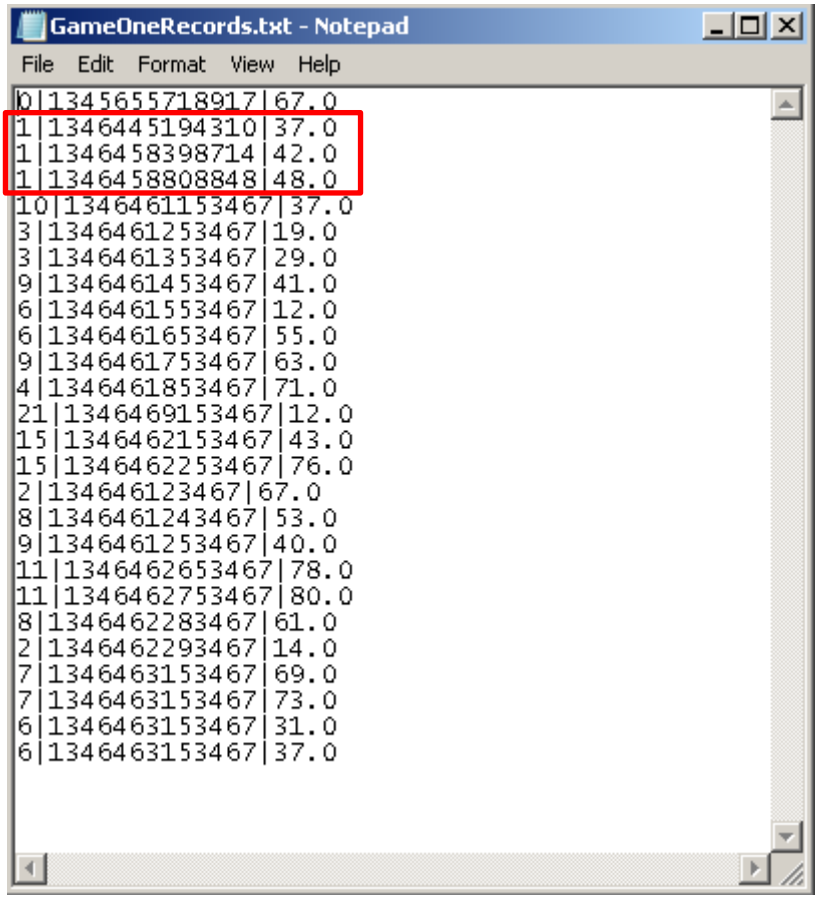
We see how the file gets recorded in the sequential file “GameThreeRecords.txt.”



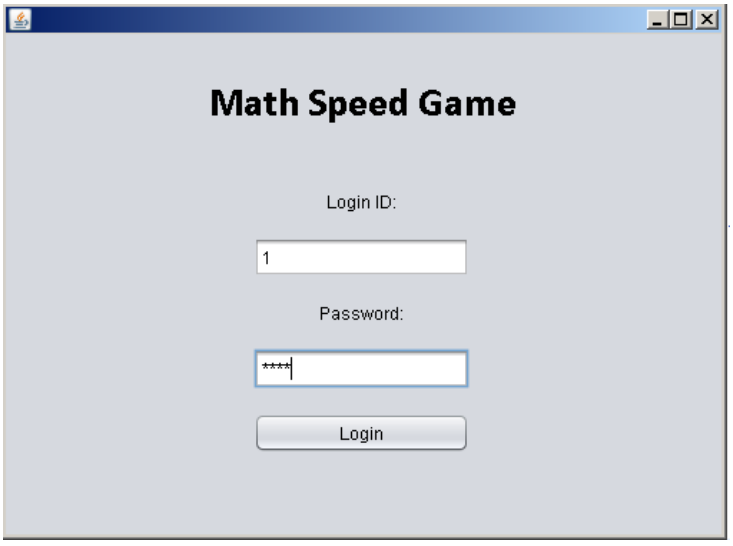
Now that the three games have been fully tested we will proceed to test the previous scores listing of the users.

Previous Scores testing – Game One

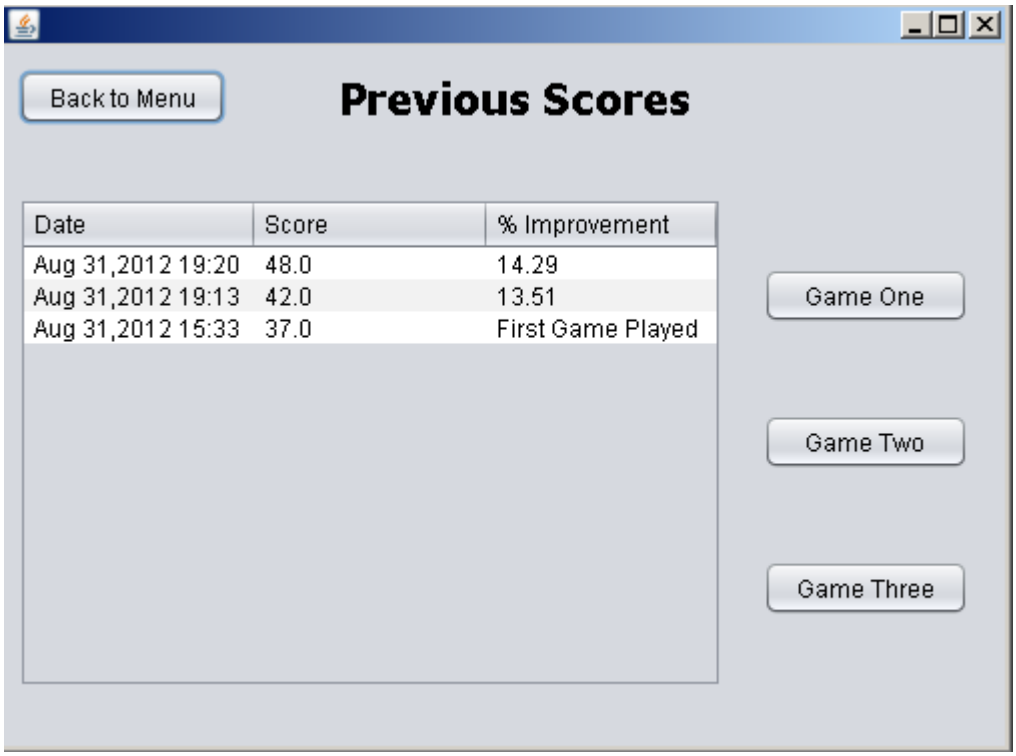
We will access the account of user with ID 1, a female student named Abrielle Belloni. We will then test the previous scores listing for Game One. The sequential file "GameOneRecords.txt" shows all the list of records played by the students. All the records with ID 1 should be output to the previous scores panel in the personal account of Abrielle.



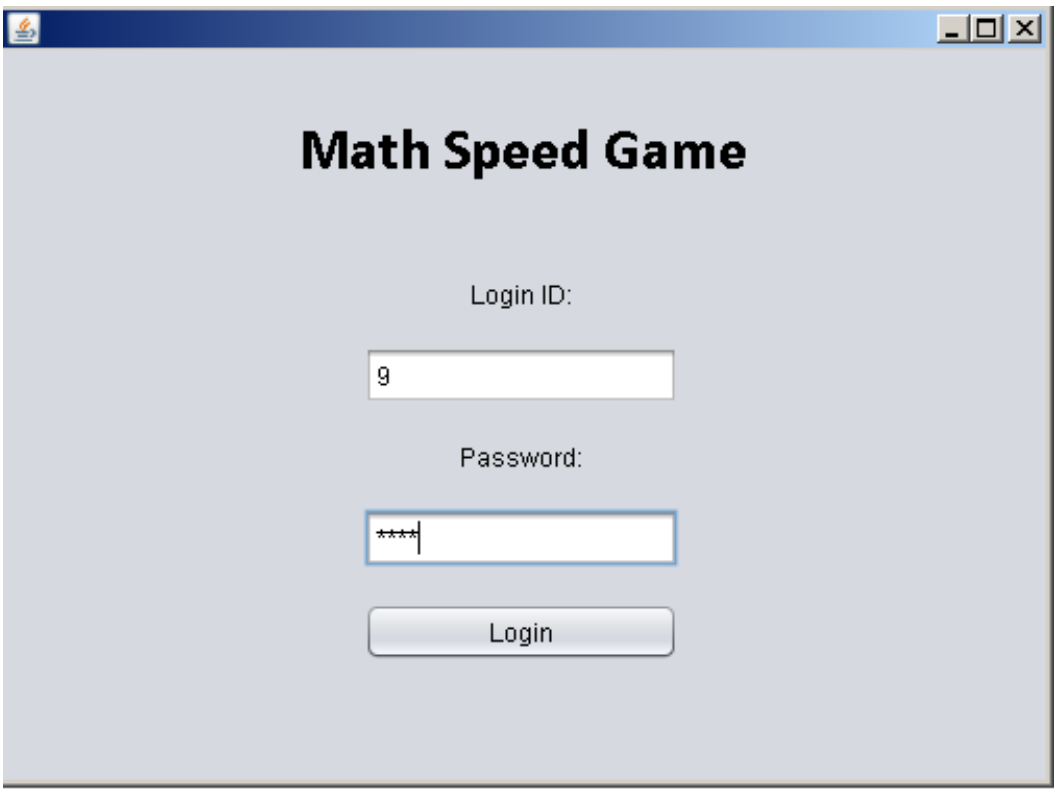
So we login as the student.

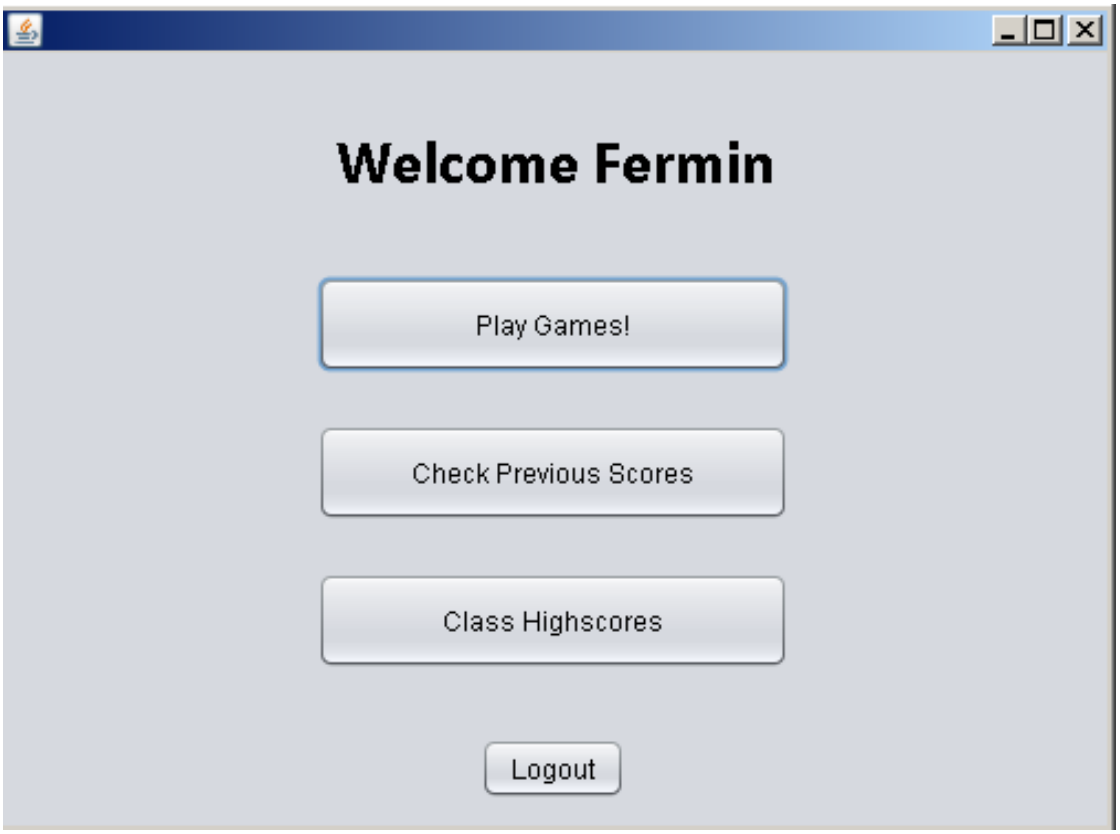


We see that the program successfully outputs the previous scores of the current logged student and calculates the percentage improvement.

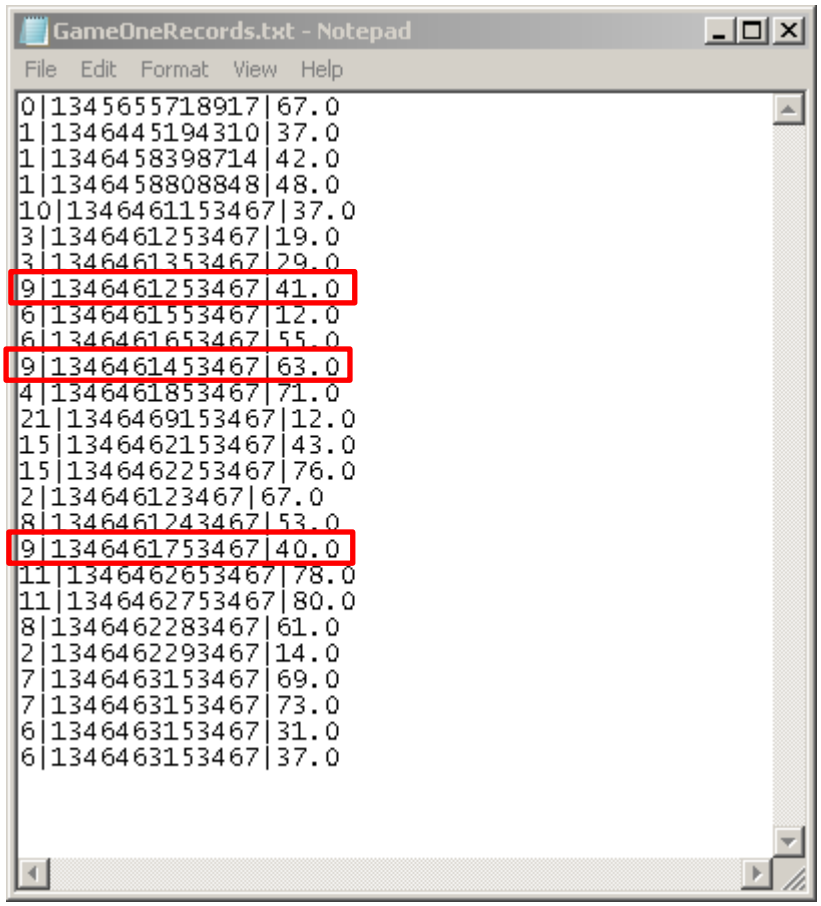


We will now login with user Fermín Olaechea with ID 9 to check his previous scores.

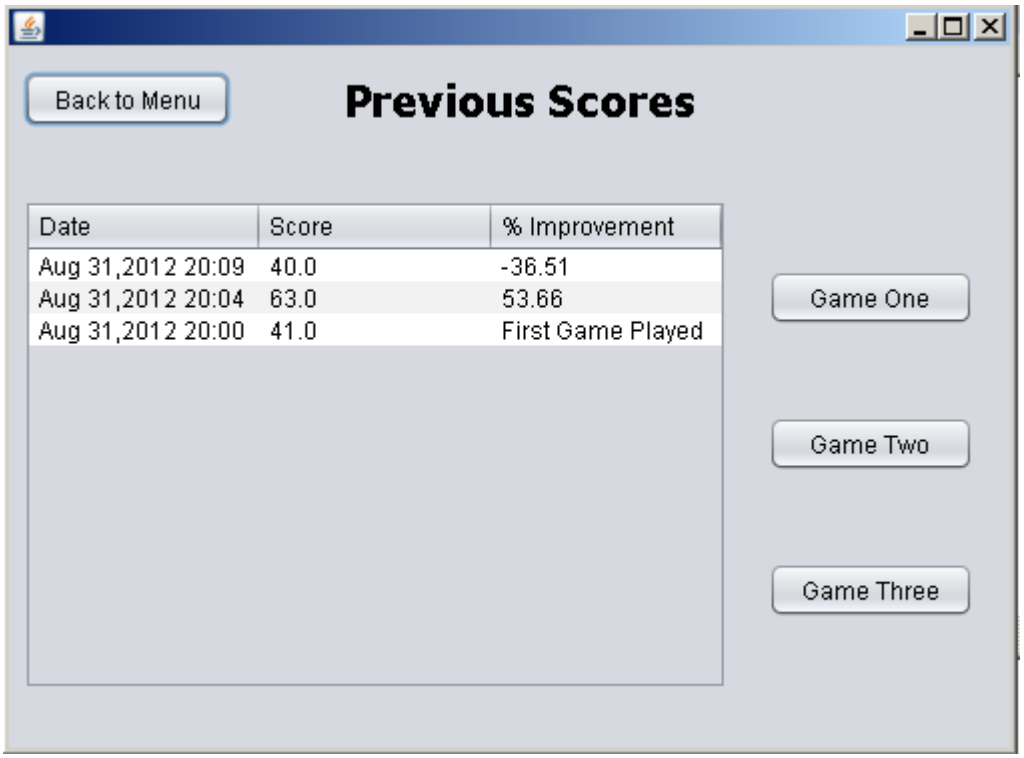




The following previous game records should be displayed for this user.

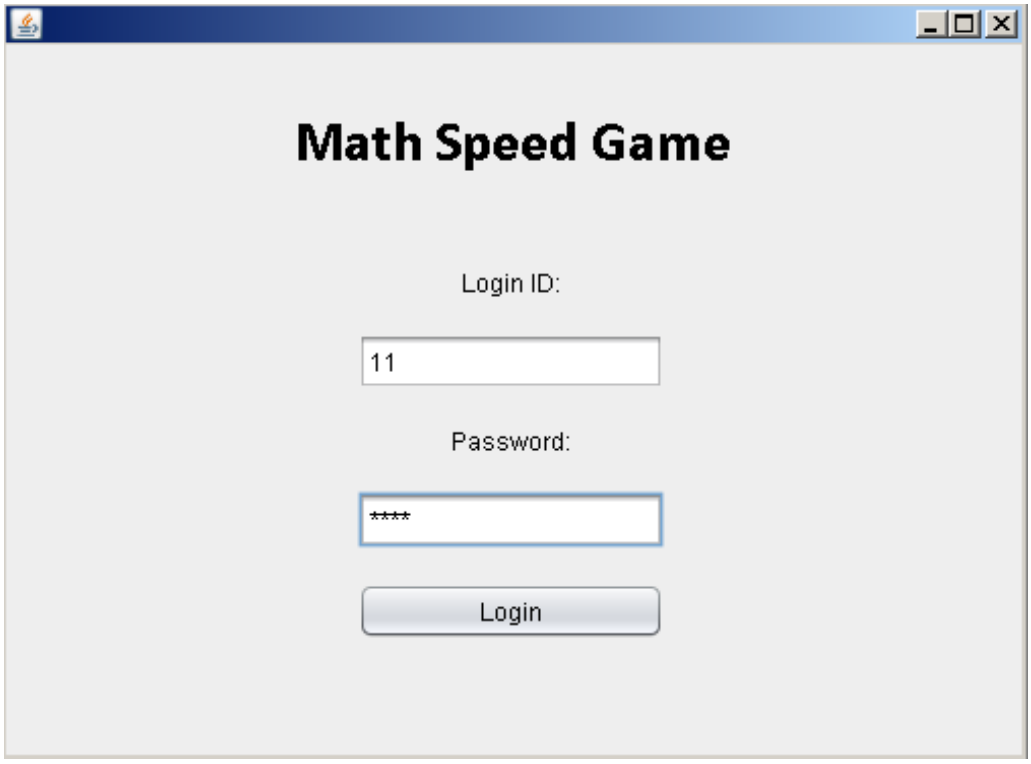


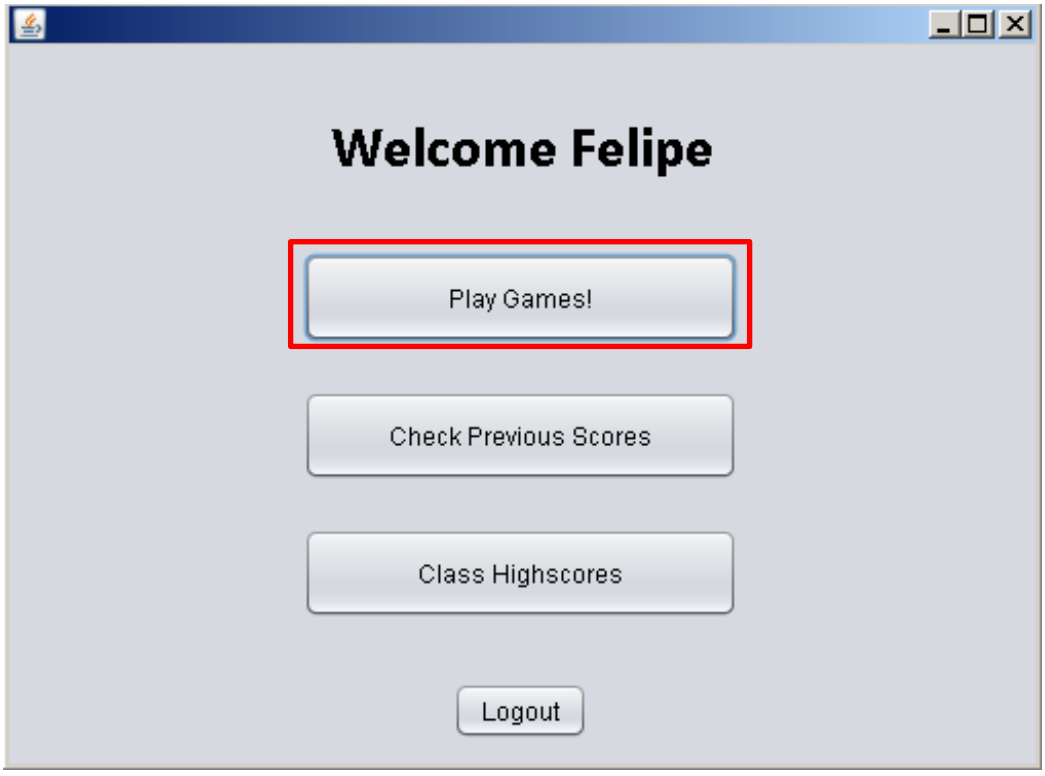
We see that the scores are successfully displayed.



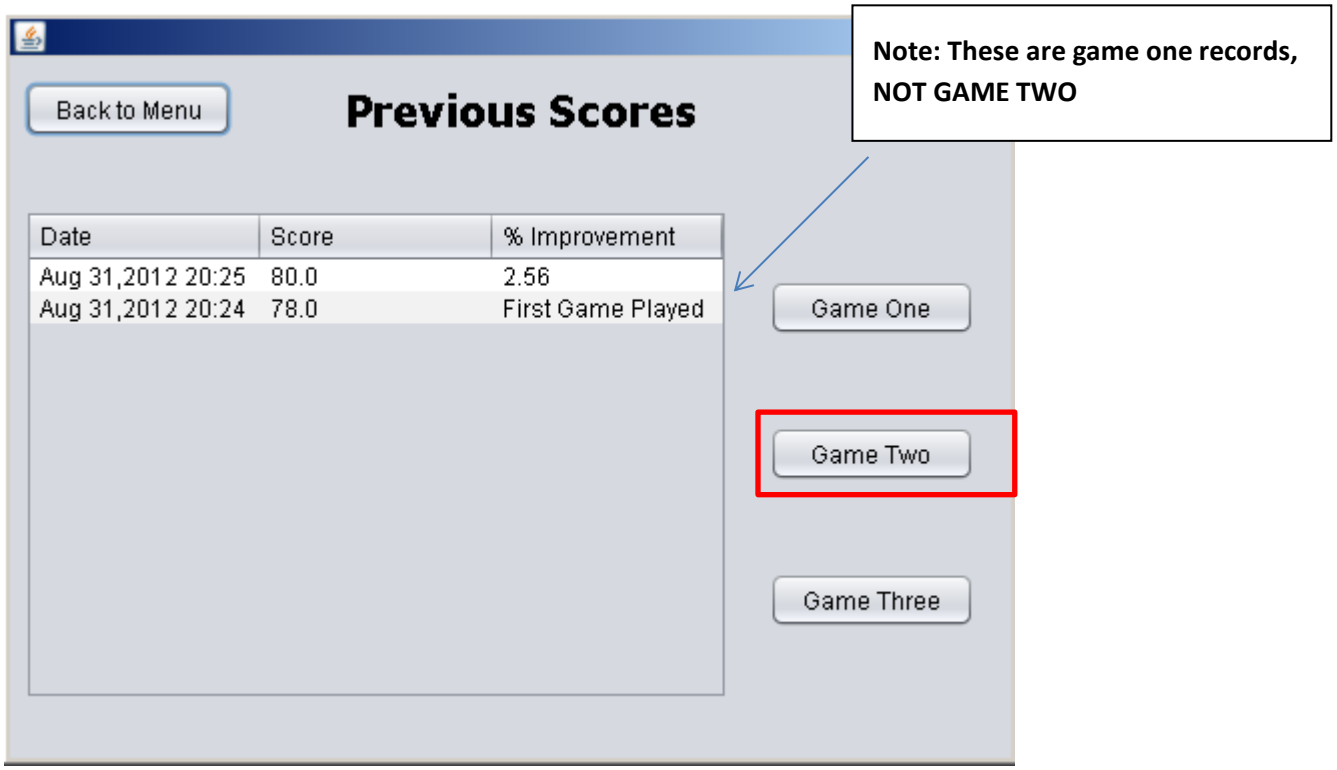
Game Two

We will use the user account of Felipe Rodriguez with ID 11 to test the previous scores of game two

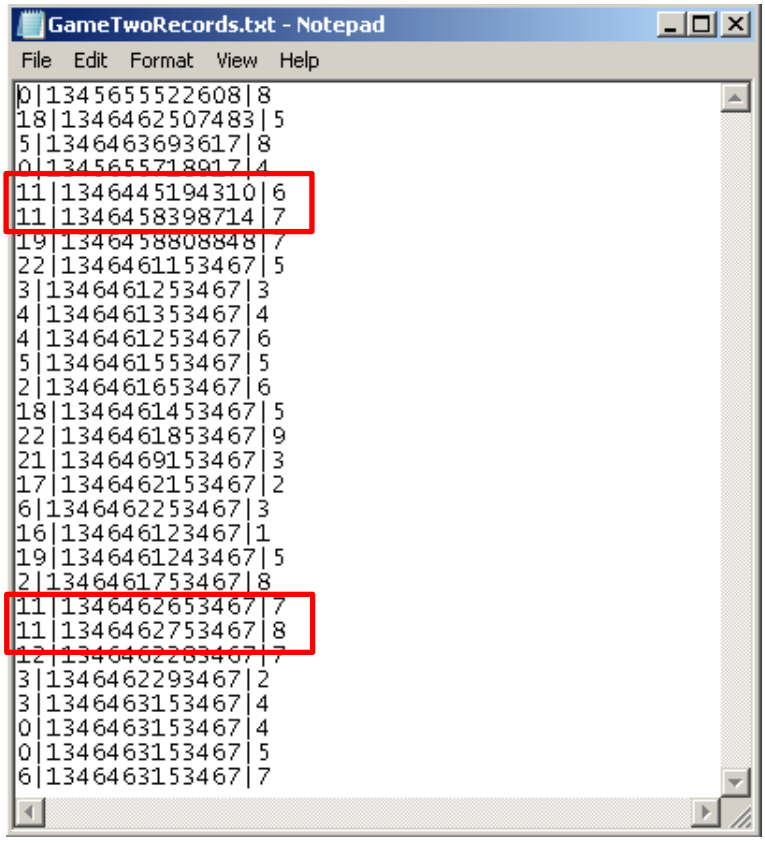




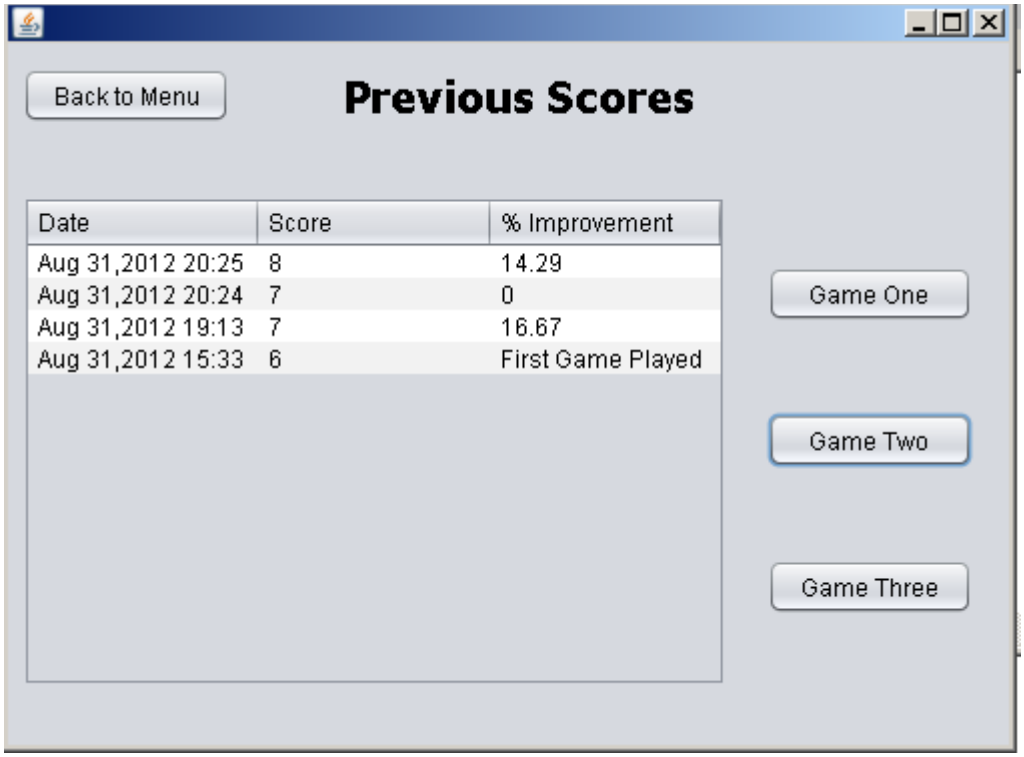
Automatically we are given the previous scores for game one, we need game two.



Felipe Rodriguez, ID "11" should have the following previous scores for Game 2.

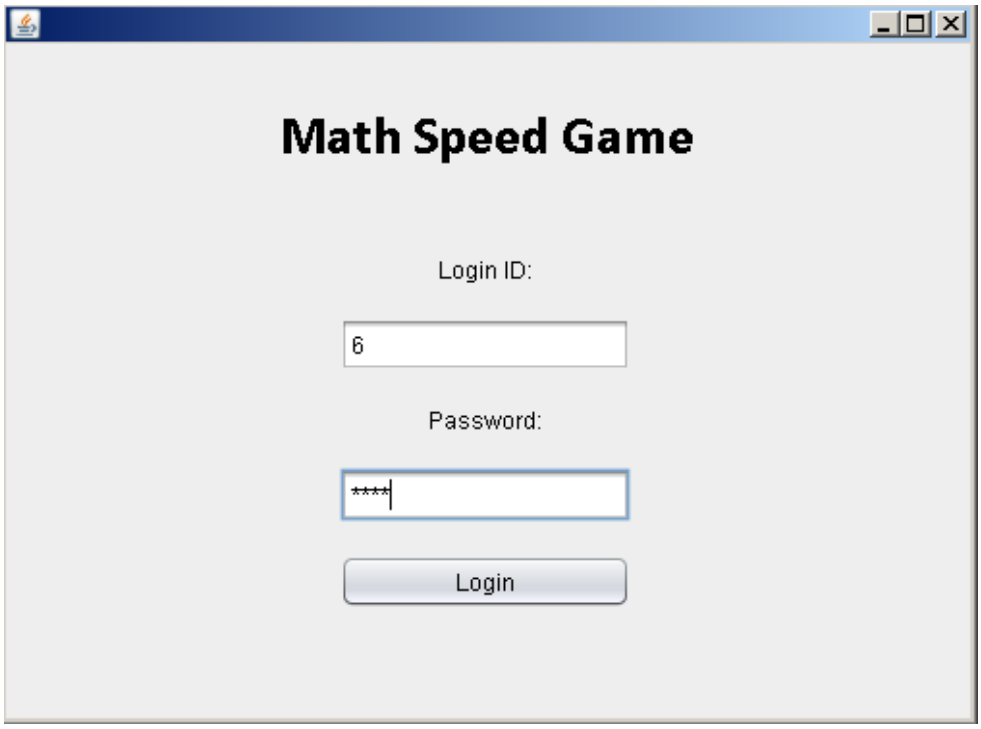


We see how the previous scores for game 2 are successfully displayed.



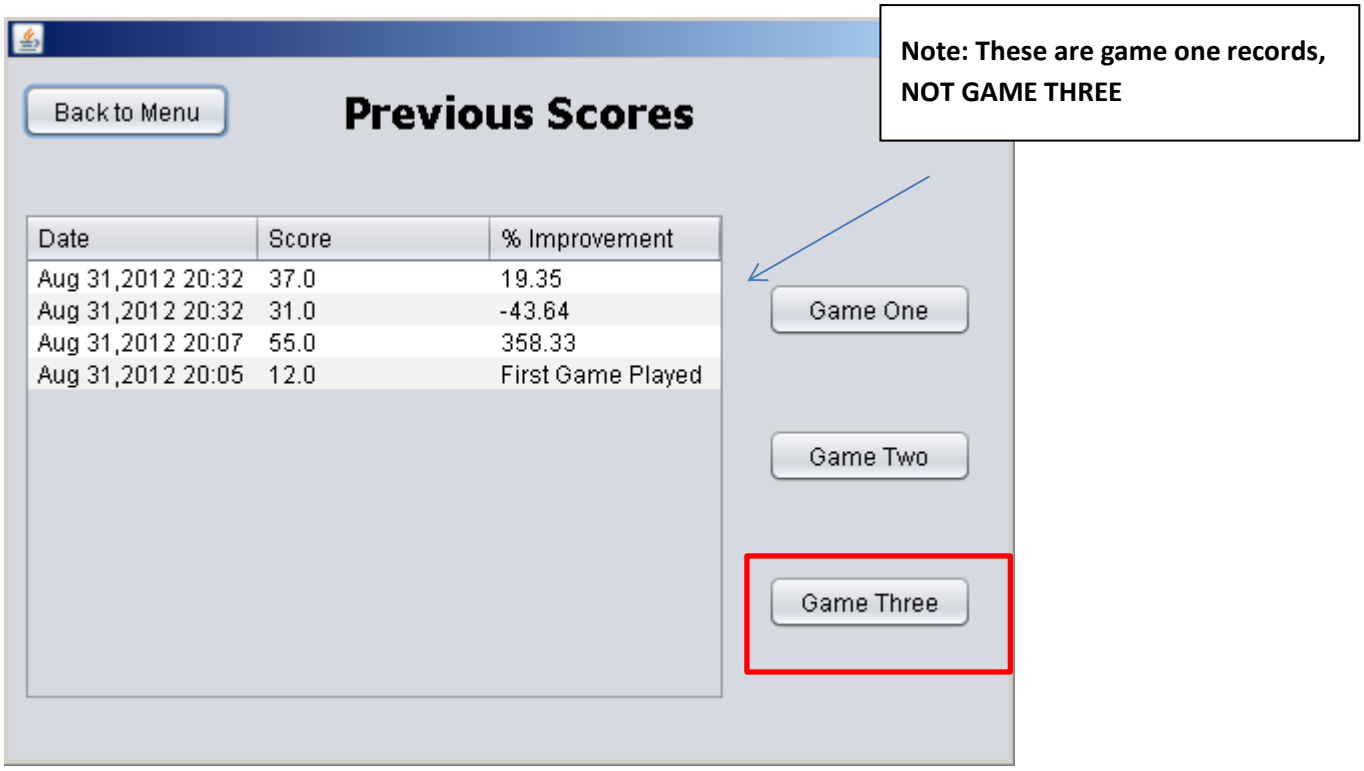
Game Three

We will now use the user account of the student Gabriel Galdos, with ID “6”, to test the previous scores for Game Three.

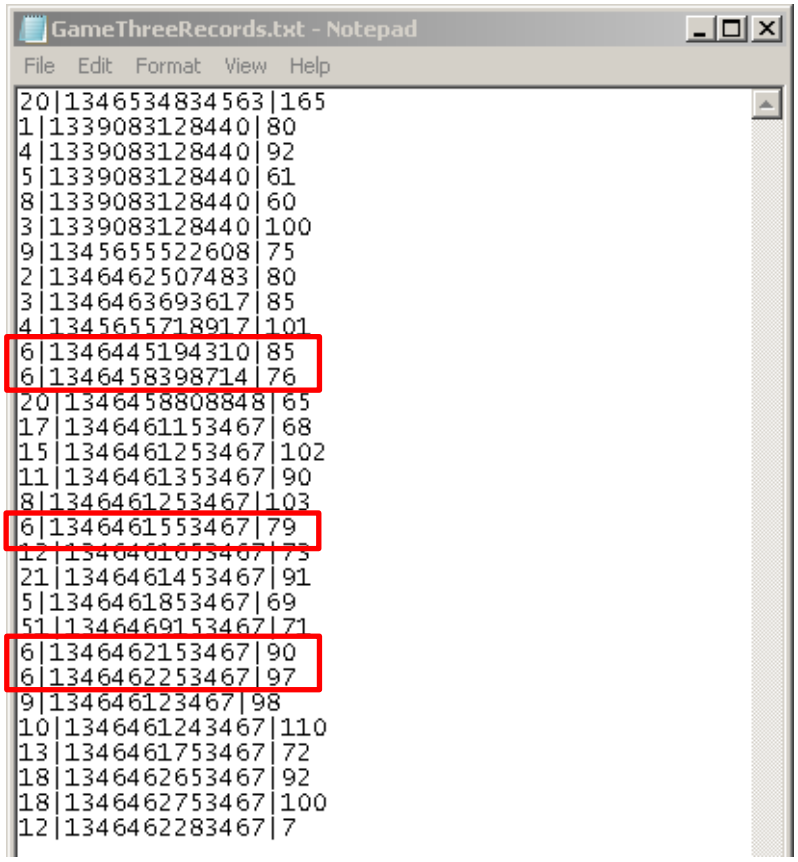


The Previous Scores are accessed

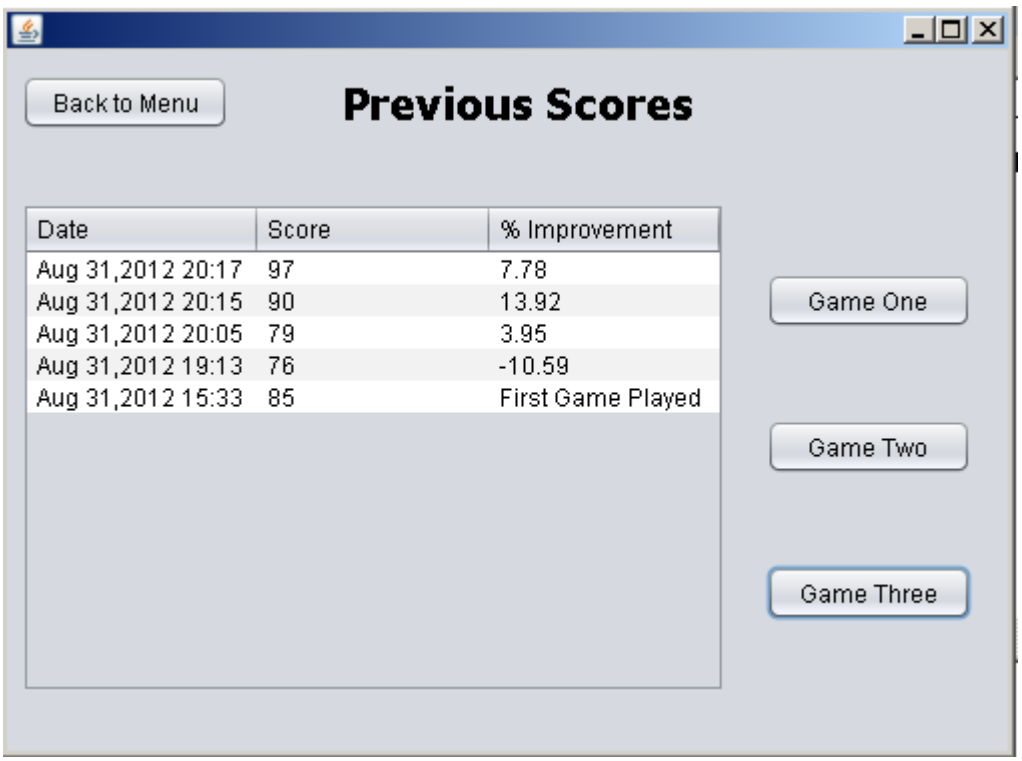
Again, the previous scores for game one are automatically displayed; we need to click the “Game Three” button in order to see the previous scores for Game Three.



The following game records for the user with ID “6” should be displayed.

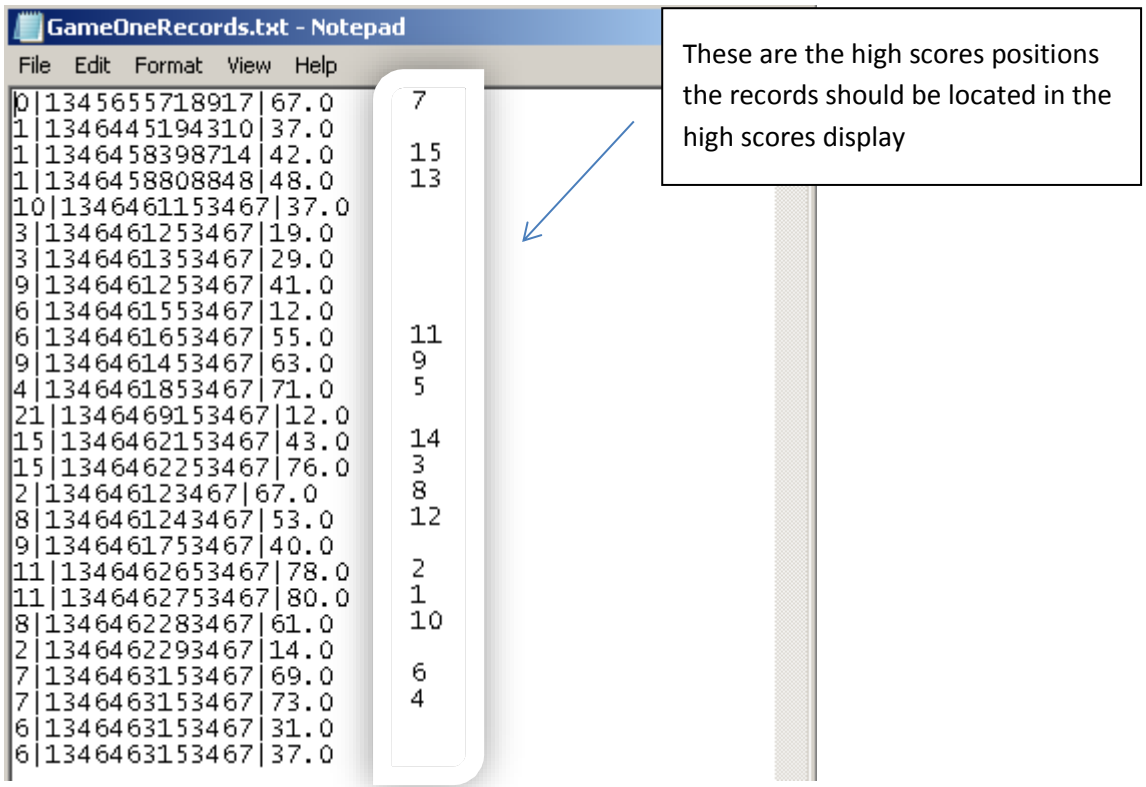


We see how they are successfully displayed.



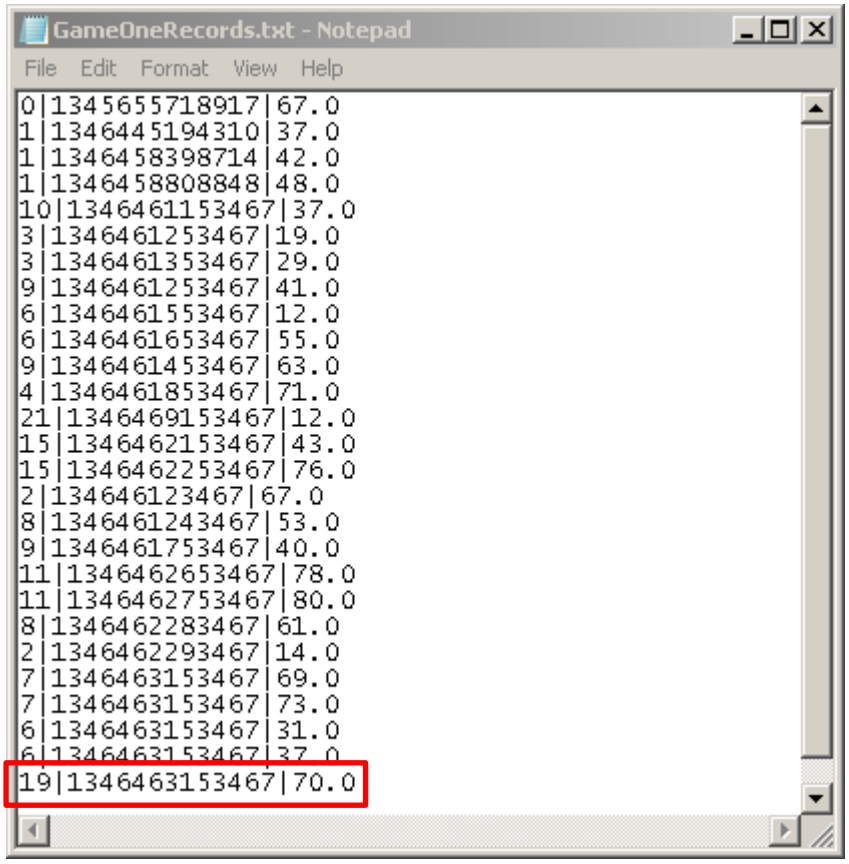
Class High Scores Testing – Game One

We will now proceed to test the display of the top 15 scores for each game. We will start by testing game one.

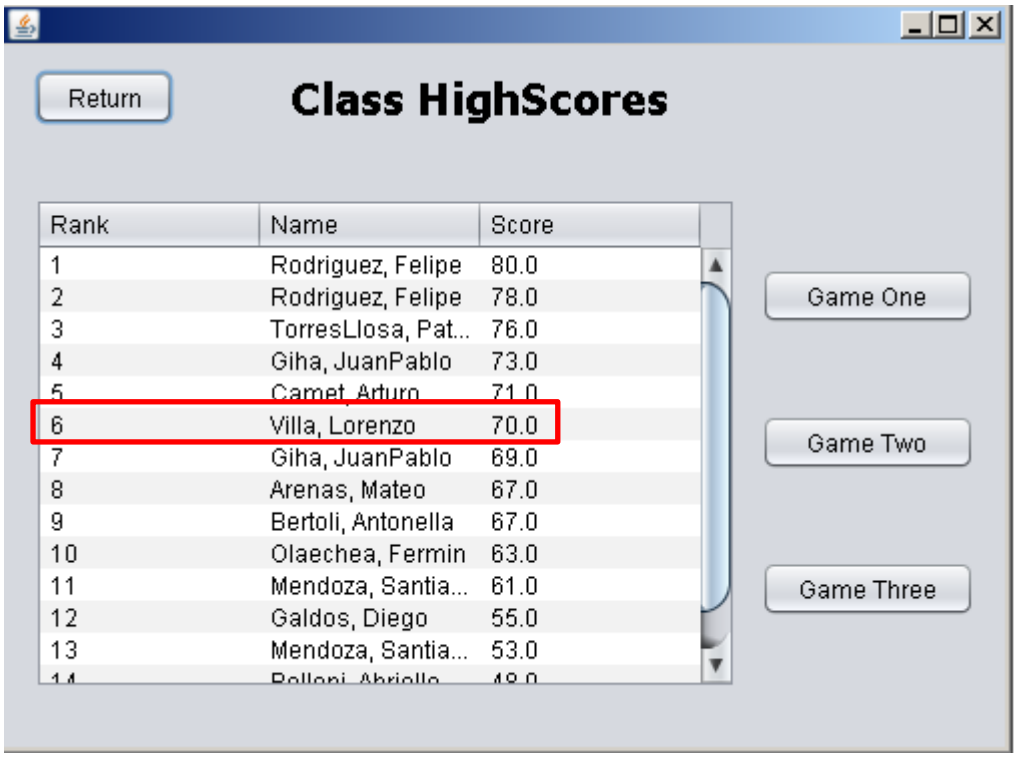


We see that the high scores are successfully displayed

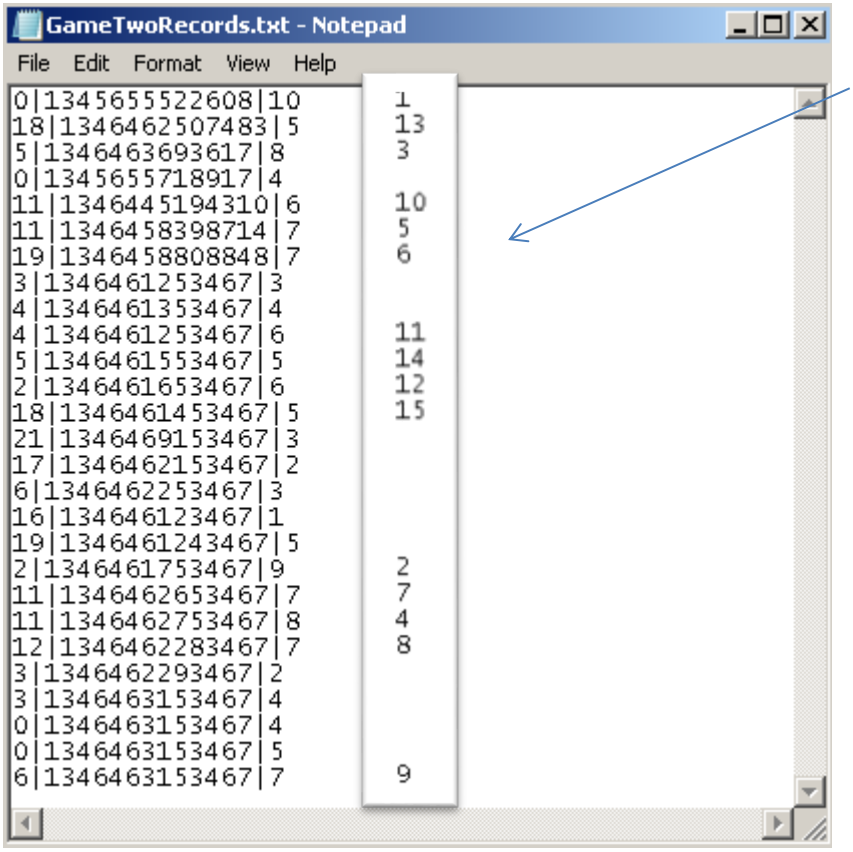
We will now see the situation that the user Lorenzo Villa, ID 19, plays a Game One and scores 70. He now should be placed in position 6 in the high scores and the scores below this should scroll down by one position.



We see how he the program successfully displays the new high scores list for Game One.

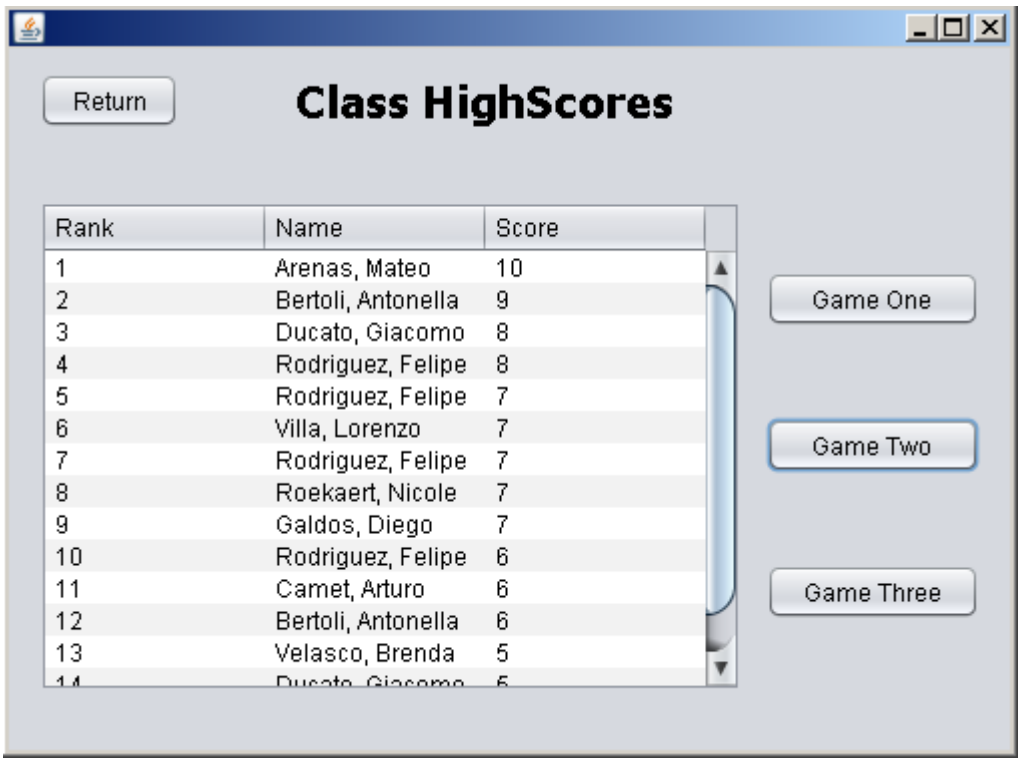


Game Two

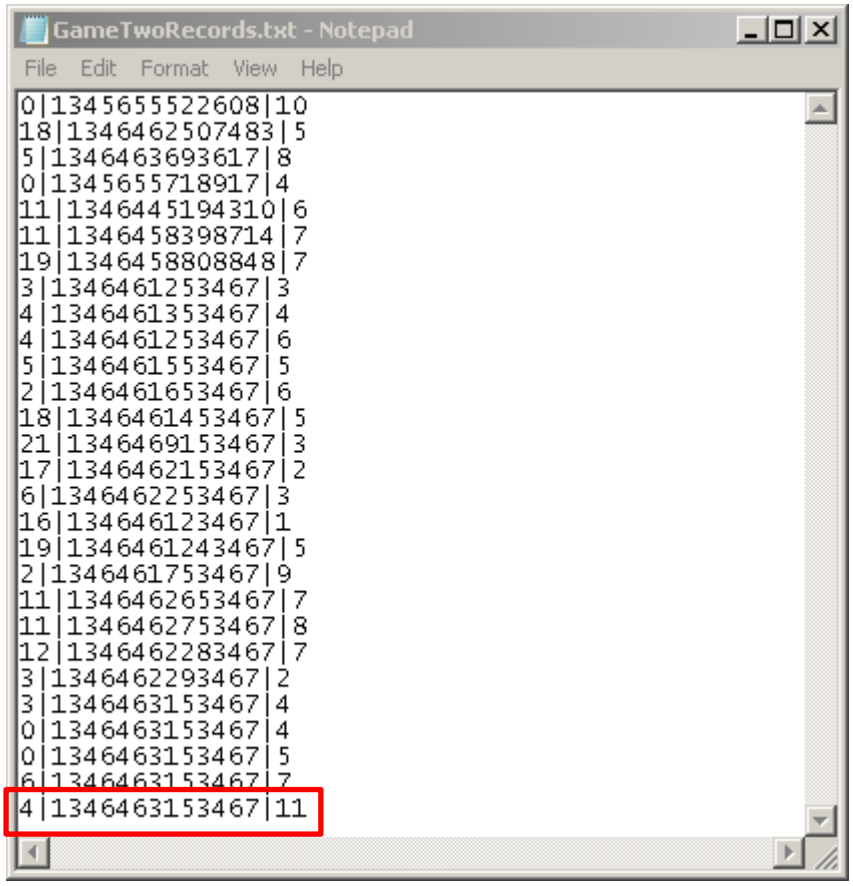


These are the high scores positions the records should be located in the high scores display

We see that the high scores are successfully displayed



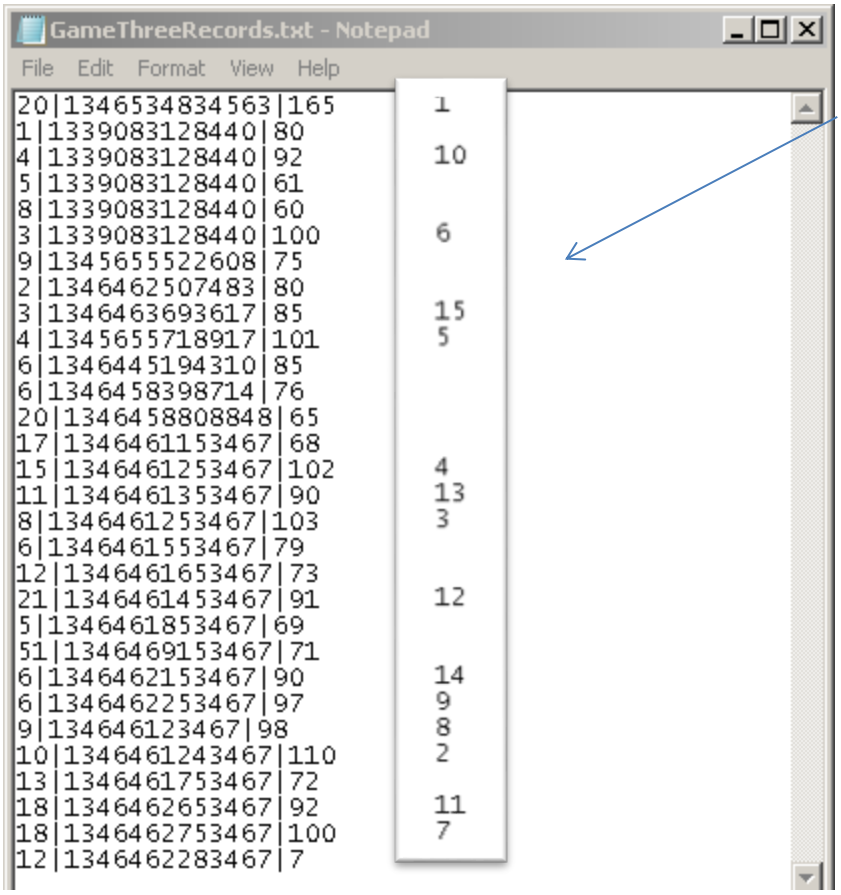
We will now see the situation in which the student Arturo Camet, ID 4, plays a Game Two and scores a high score of 11. He should now be placed at the top of the high score list and all the scores below should scroll one position down.



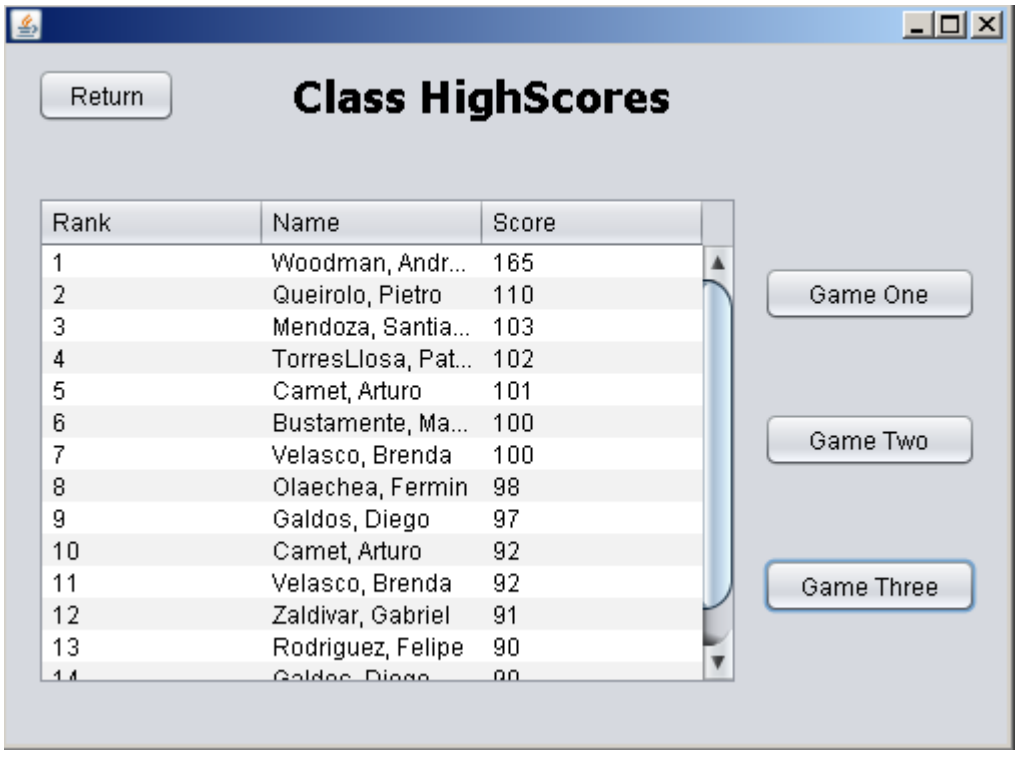
We now see how the program displays the new high score list.

Game Three

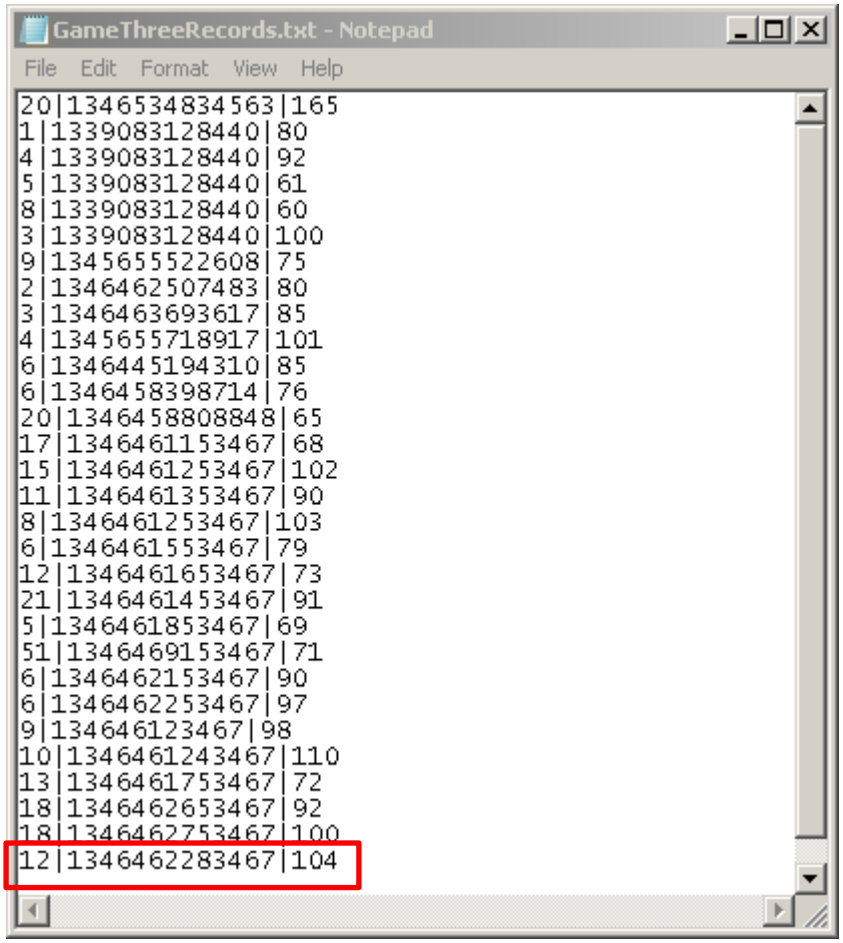
These are the high scores positions the records should be located in the high scores display



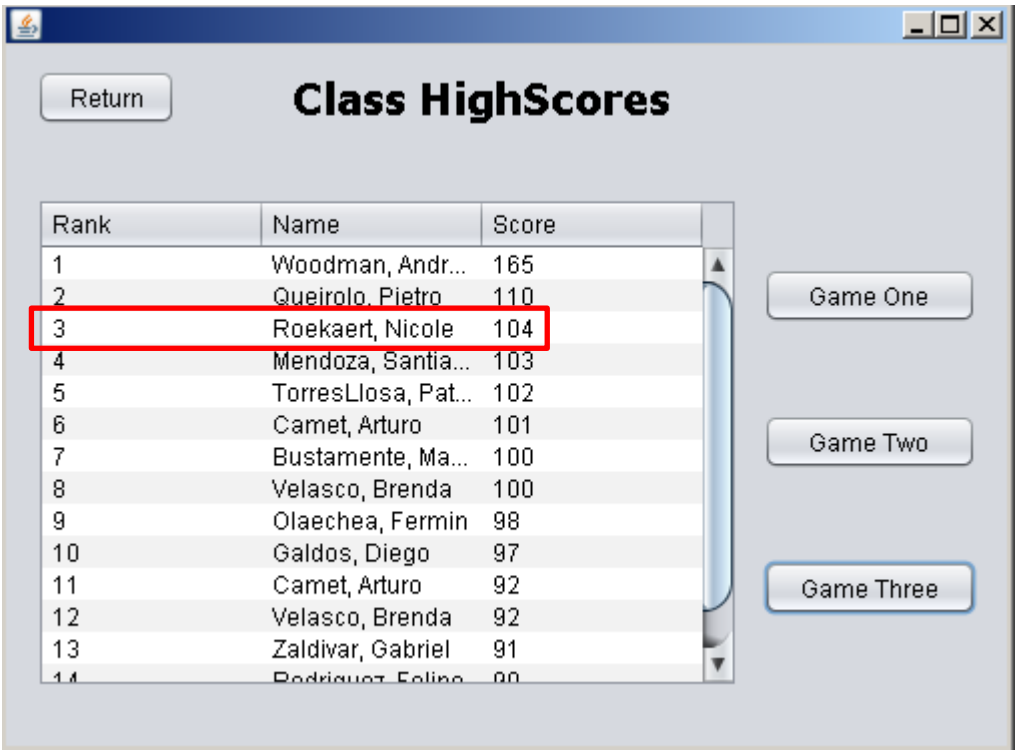
We see that the high scores are successfully displayed



We will now see the situation in which the user Nicole Roekaert, ID 12, plays game three and lasts 104, she should be placed in position 3 in the high scores and all the scores below of it should scroll down one position in the list.

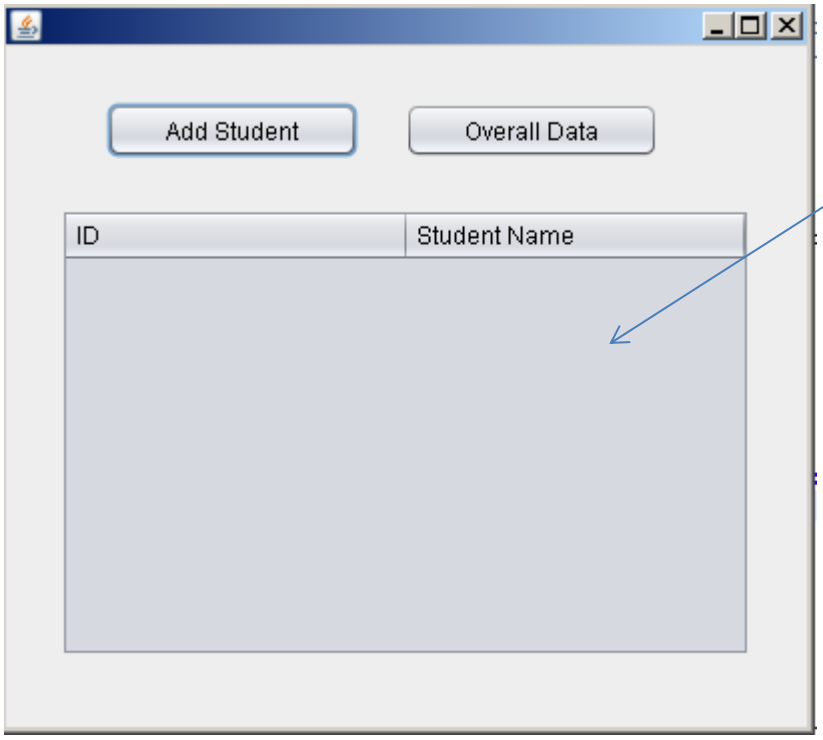


We now see the updated high scores list. The program successfully placed this game record in its correct position.



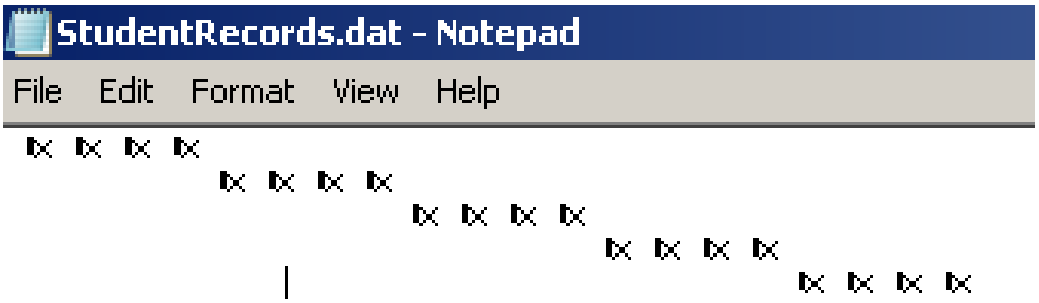
Admin Application – Adding students to the system

We are now going to test how the admin may add students to the system. We will start by testing how a student is added to an empty random access file.



Here are no students registered in the system. The student records should appear in this table.

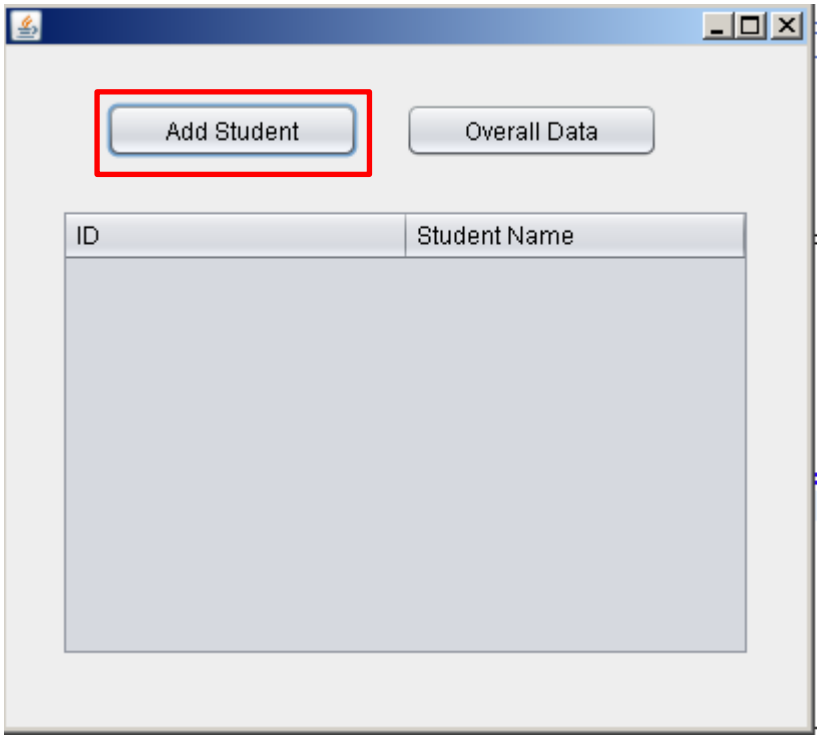
We see that the random access file holding the records of the students: "StudentRecords.dat" is empty (the character "x" marks an empty field.



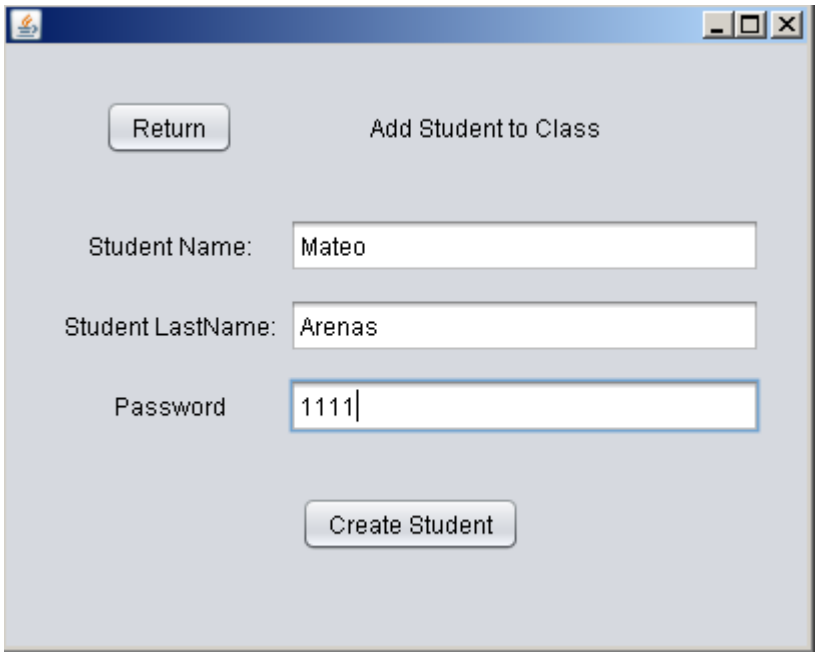
We will now proceed to add the first student from the Sr. Salomon's class

ID - User	Name	Password
0	ARENAS ARROBA Mateo	1111

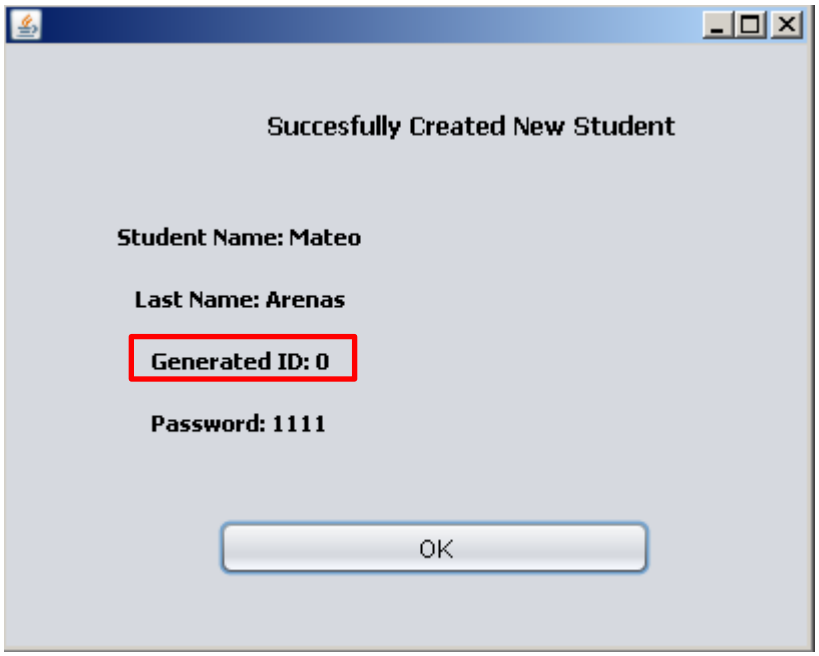
We access the Admin application



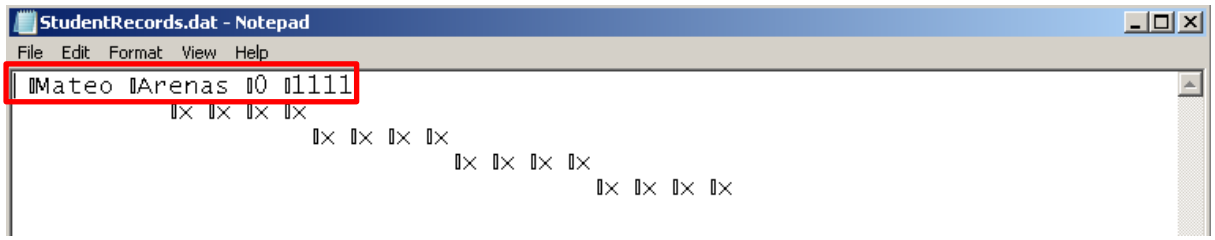
Click the button add student and fill the details of the student being added.



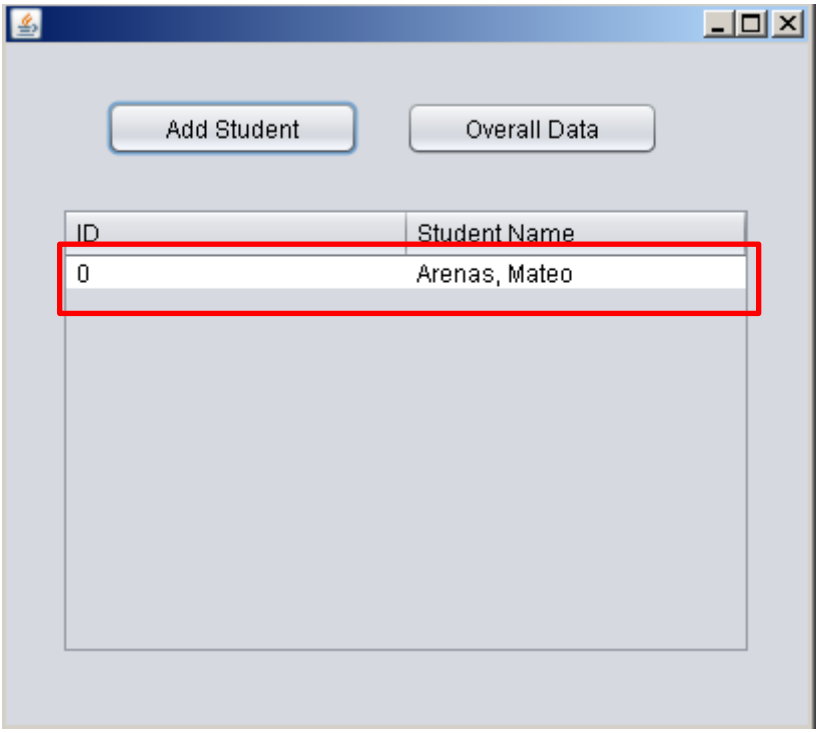
We now click the “Create student” button and an ID for the student should be generated. The ID generated will be 0 because he is the first student in the class.



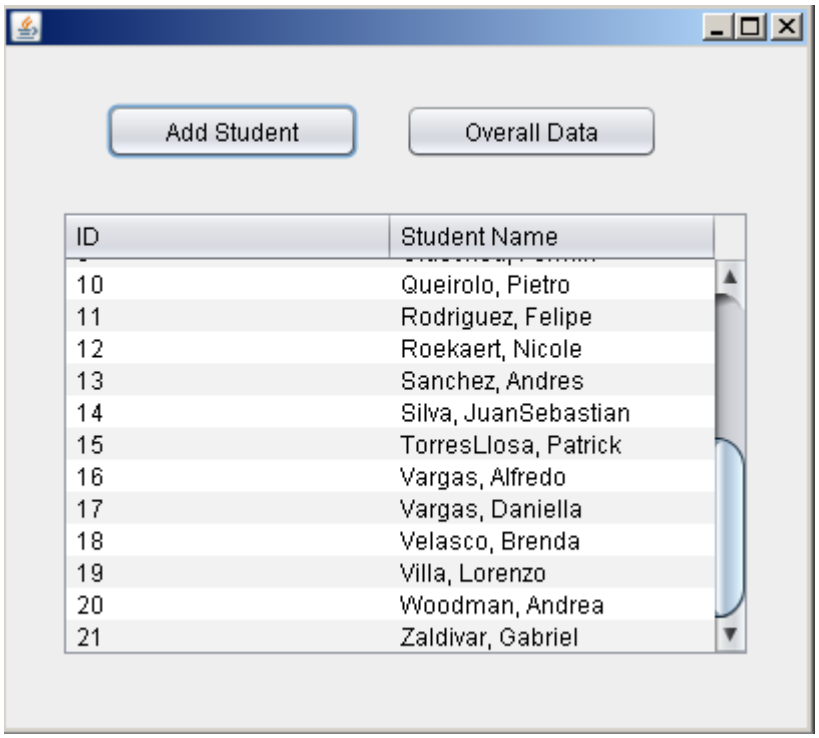
We now see how the record gets successfully added to the random access file “StudentRecords.dat”



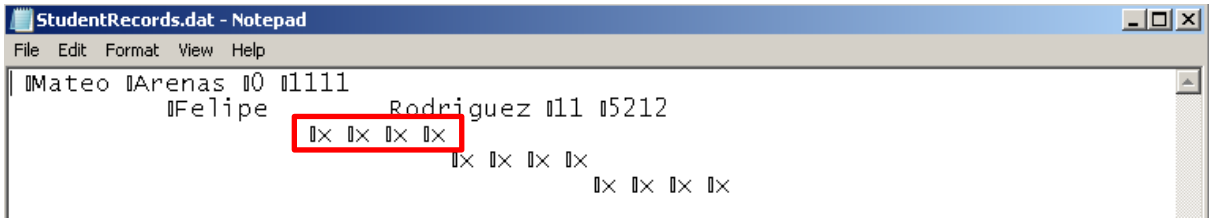
And we see how the student gets displayed in the admin application



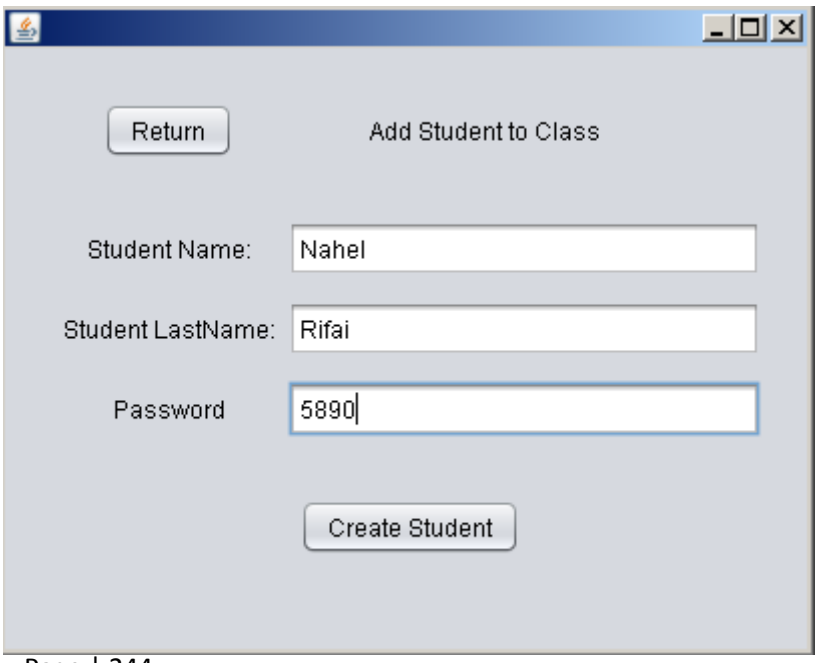
We will now proceed to add a new student to the already existing math class (with all the student records).



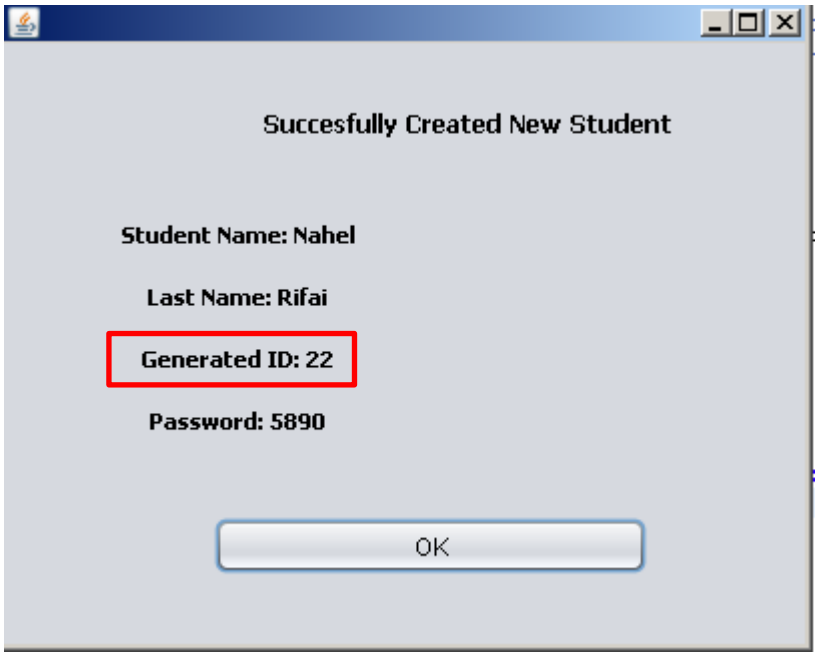
We see the 22 (counting from 0) records of all the students in the class. We will now proceed to add the 23rd record.



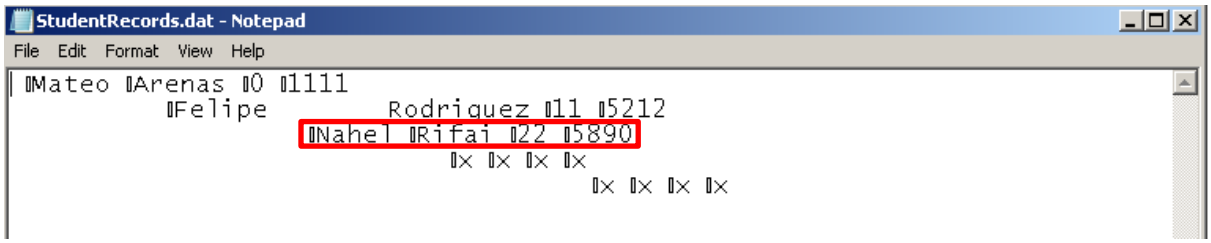
The new student should appear in this empty record position.

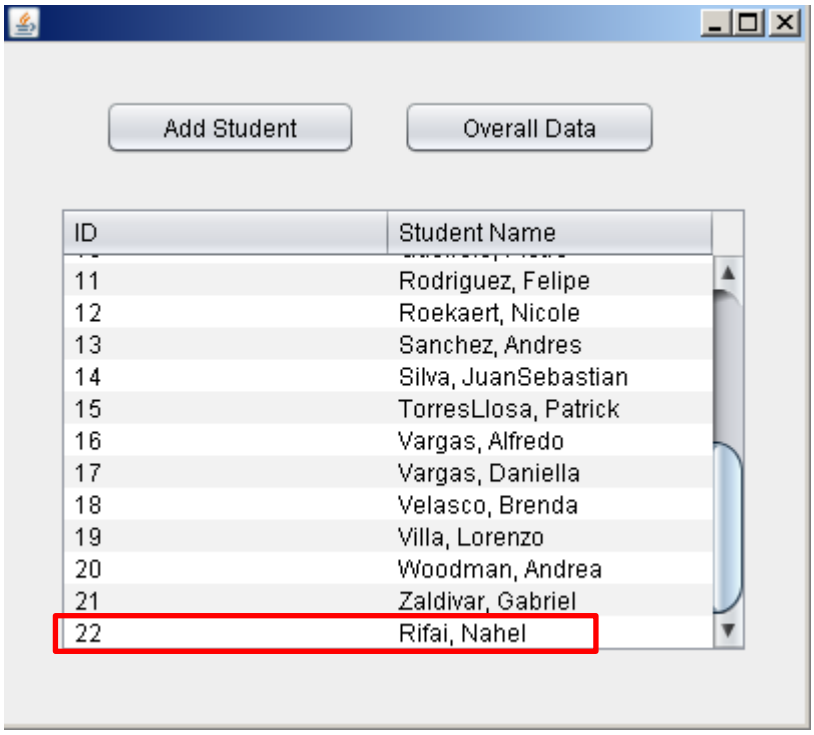


When created the new student should be assigned the ID 22. We see how the program successfully sets the correct ID.



We now see how the record was successfully stored in the random access file in the correct empty position



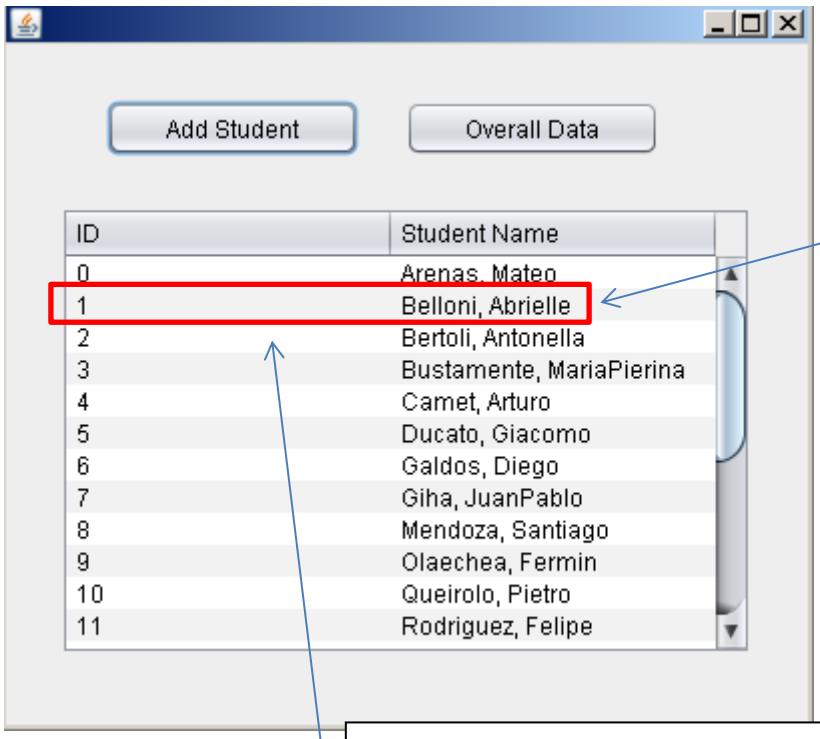


Now that we tested that the program successfully adds and displays students we will test how the admin may access the previous scores of each student by double clicking its record on the table displayed in the admin's application main menu.

Note:

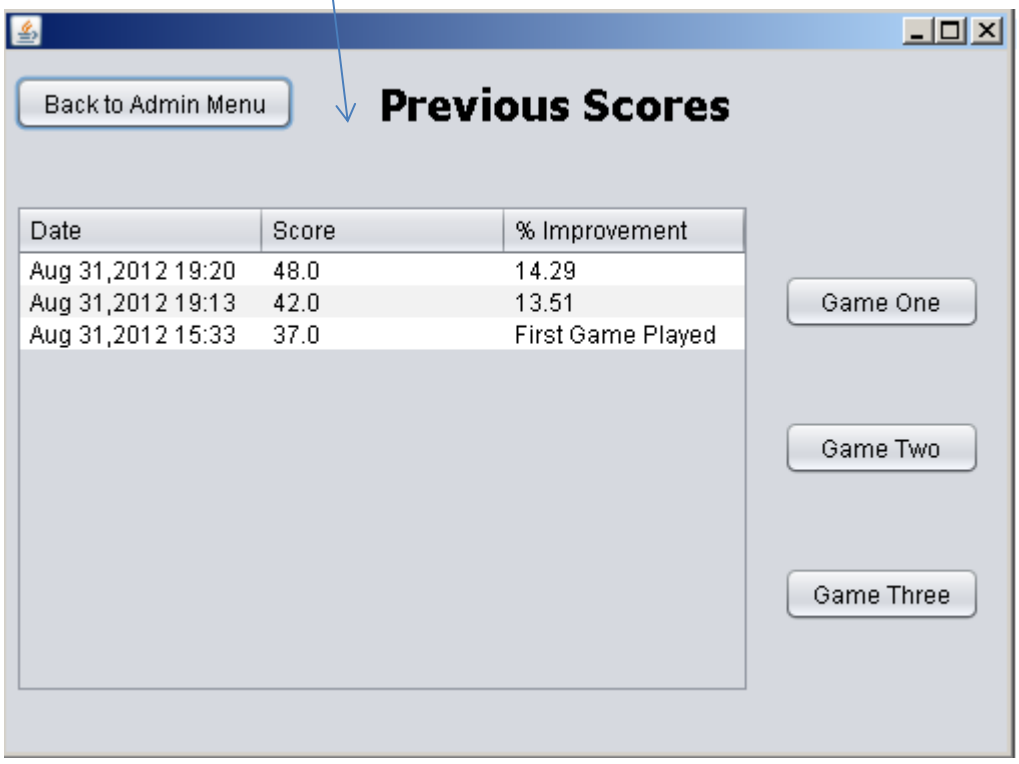
We will only test this and not all the other features of the previous scores panel because this class "AdminPreviousScoresPanel" has the same attributes as "PreviousScoresPanel" used in the student application and both use the same table generator "PreviousScoresTableModel" to display the correct data.

Displaying the student game records

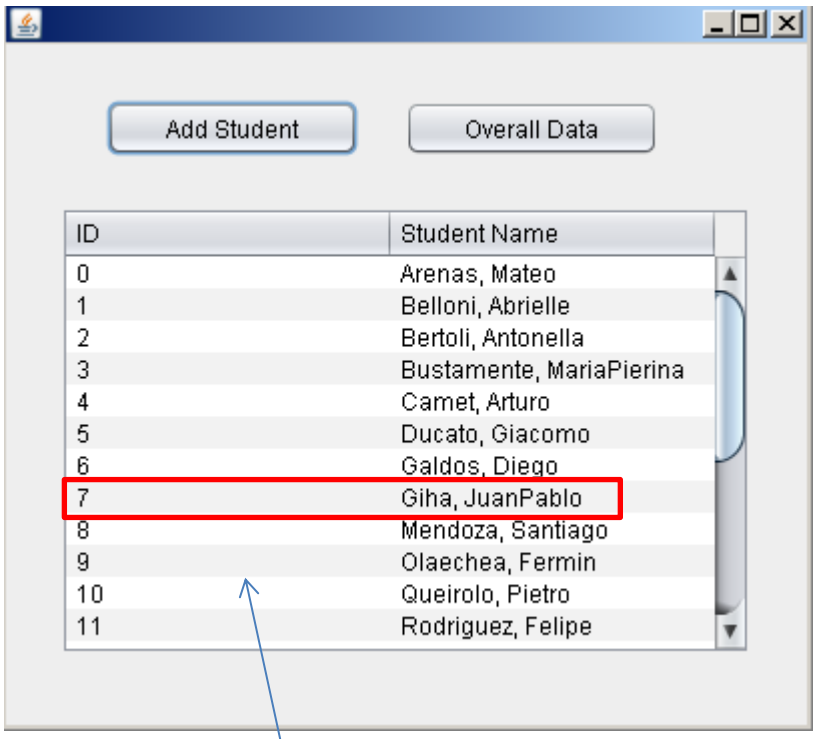


This record is double clicked and the previous scores of the student with ID "1" open. The previous records of Game 1 are automatically displayed but the admin may also see the previous records for other games

DOUBLE CLICK

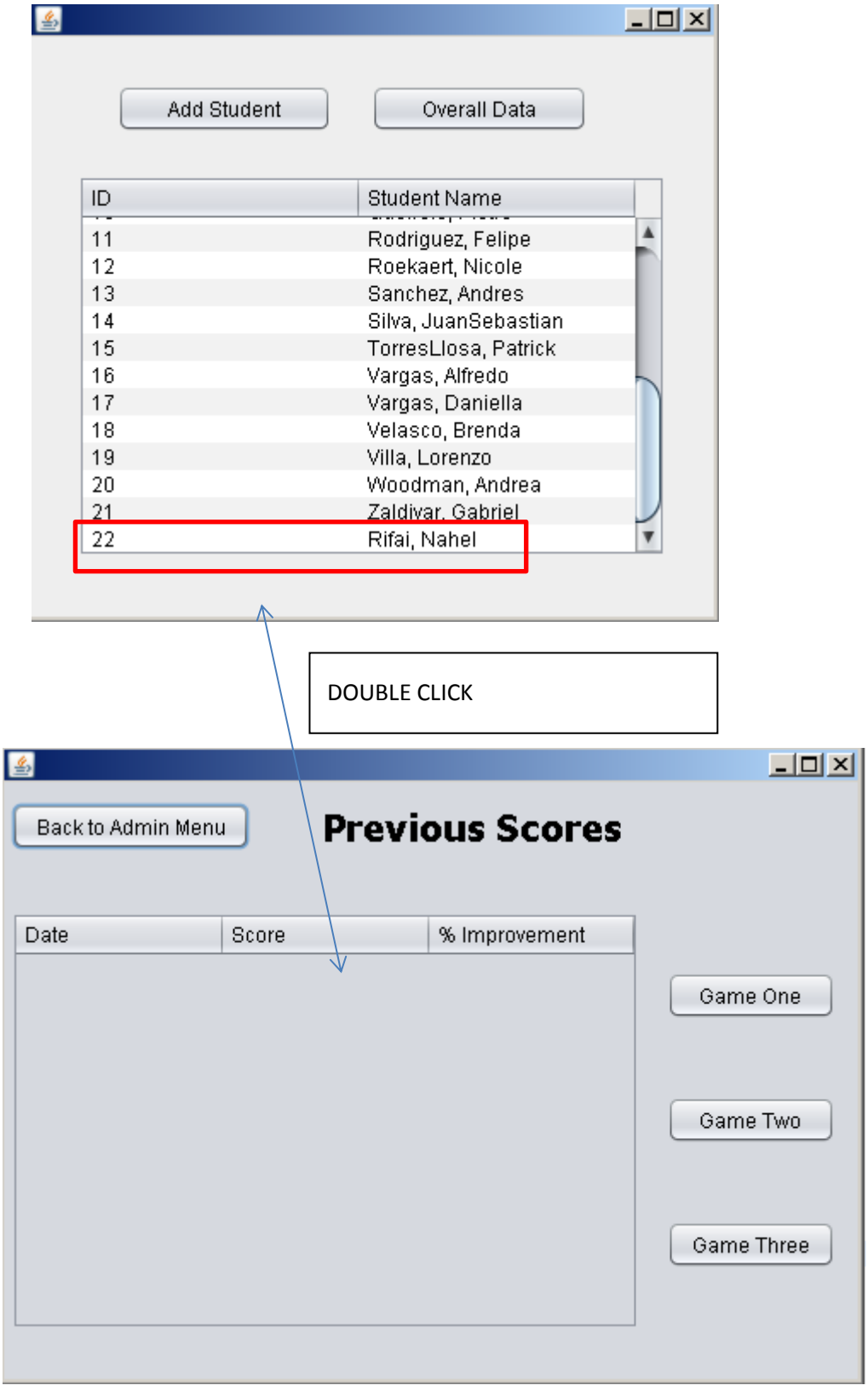


We will now test a different student, Juan Pablo Giha with ID “7”.



DOUBLE CLICK

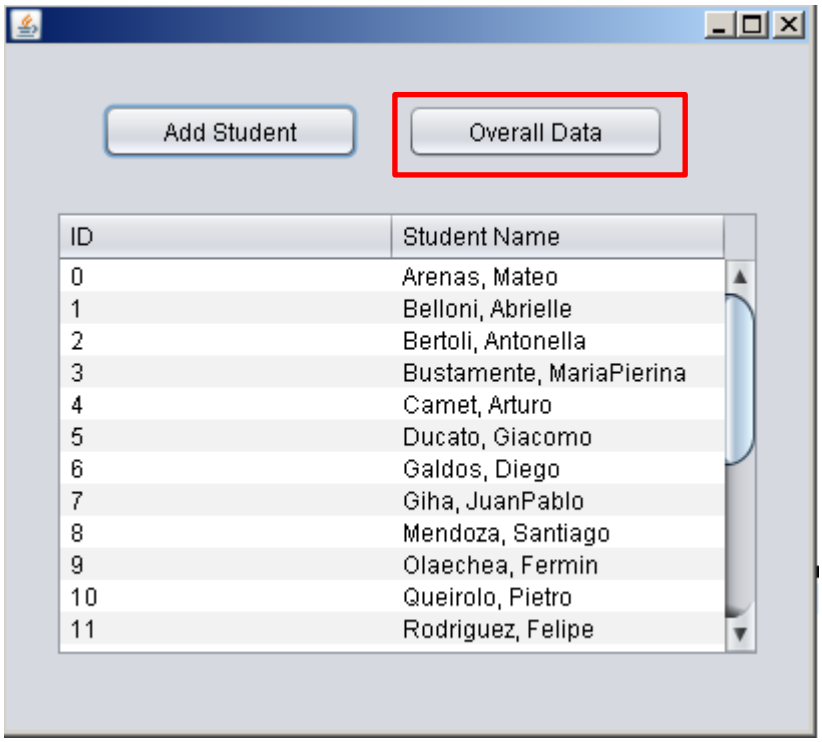
We see that if we click the newly created student, Nahel Rifai, with ID “22”, we will not see any game records for Game 1 because he hasn’t played any games of game 1 yet.



We are now going to test the display of high scores of the class to the admin. Same as before, we are not going to show all the features of the high scores panel because the class "AdminClassHighScoresPanel" shares the same attributes as "ClasshighScoresPanel" and both use the same table generator "HighScoresTableGenerator" class.

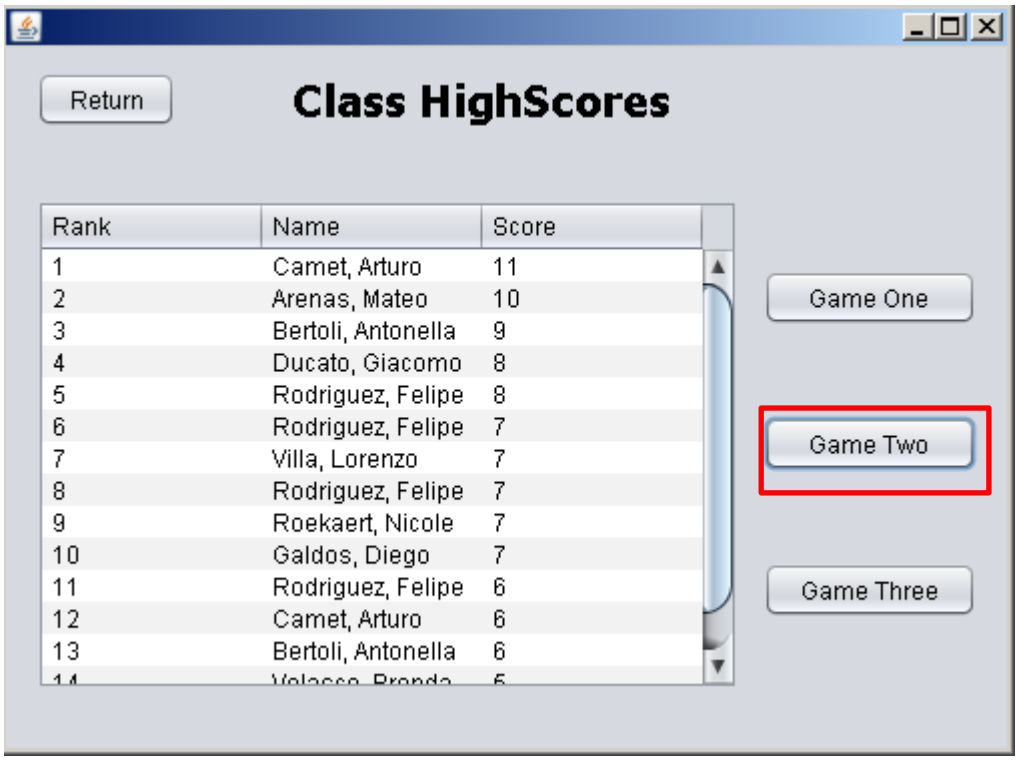
The Class High Scores in the Admin Application

The class high scores are displayed in the admin application by clicking the “Overall Data” button.

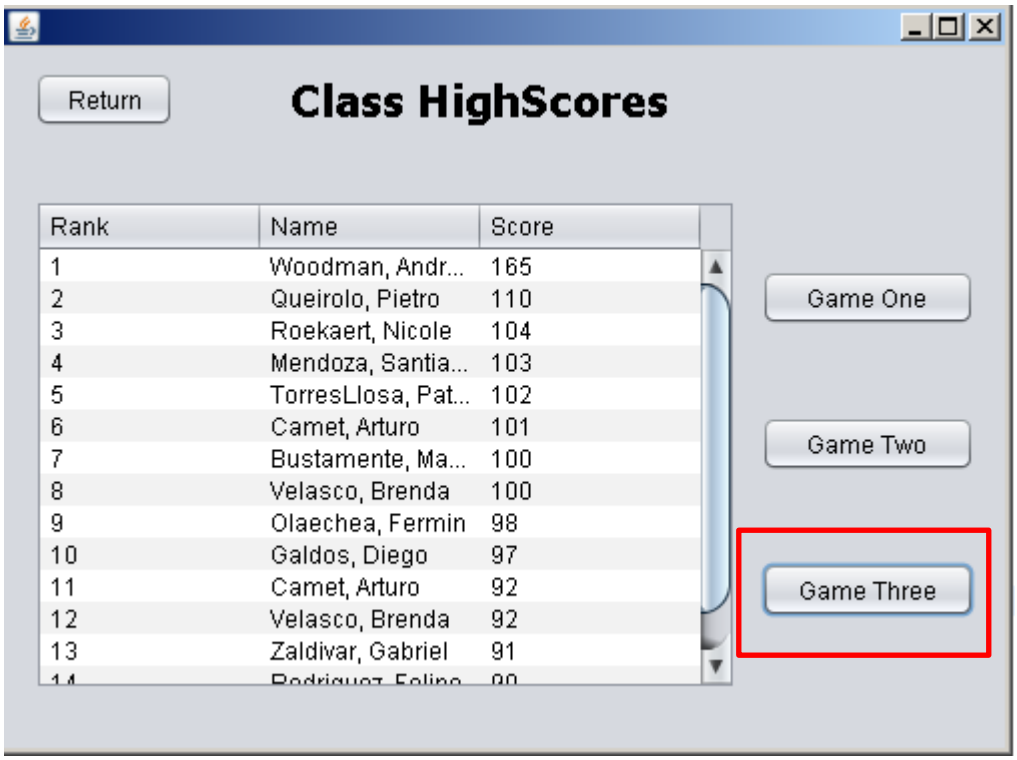


The admin is automatically presented with the high scores for Game One.

He can also access the Game Two by clicking the “Game Two” button.



And also game three high scores by clicking the “Game Three Button”.



D2 – Evaluating solutions

The constructed Math Speed Game application works as an effective solution for Sr. Salomon's problem.

Solution

As addressed in section A (refer to it for details), this teacher wanted to improve the mathematical solving speed of his youngest math class but he struggled to find a successful method in which he could achieve this. The program built acts as a solution for his problem because it has the stored records of all the students in his class. Furthermore, the children may access the program, play games, check their previous scores for each game and also see the class' high scores. Moreover, the second application, the admin's program, lets the teacher access the previous records of each student, the high scores of the class and even add new students to the system.

The method which Sr. Salomon has always addressed (without the program) involved a manual generation of the math problems, a process in which he had to address the sixth grade math syllabus when generating each type of question (multiplication, division, etc.) because the students require a specific difficulty that he finds hard to remember in complete detail. In addition, he had to make a manual correction of all the work of the students and calculate a score based on the number of correct questions and time taken to finish. Sr. Salomon found himself quite stressed in finding the time to do this, taking into account that he has a lot more math classes apart from the sixth grade class.

The solution not only generates unlimited and unrepeated questions based on the syllabus boundaries, but also makes an automatic correction and records all the results, which the students and Sr. Salomon may address at any time after the game has been played. Moreover, the concept of playing challenging games and competing with other students of the class in the high scores produces a positive perspective from the students towards the math exercises. This completely replaces the common and boring math homework's at home. With this, I may proudly say that the application itself fulfills all of the objectives presented in the analysis section, proving the successfulness of the built application in regard to the problem.

The solution was efficient in giving each student a unique ID (username) and account to log into the game. Their records were saved in a communal Random Access File where all the relevant student information was stored. The students had three different games available. The rules of all of the games were successfully set and different types of questions were generated, all with the adequate difficulty. The application was also efficient in correcting the questions and generating the right score for each game. The game records were saved in its corresponding sequential file and the high scores were successfully generated by constructing an ordered GameRecordNode list (from high to low). Furthermore, the students and teacher could see their previous scores in their account with the score they obtained on each game, the date it was played and the percentage improvement from the last game played. Finally, the administrator, the teacher, was able to add new students to the class and was handed an available ID (which didn't exist), so the new student could use access the student's application.

I think there are several ways the solution could be improved to help the students and Sr. Salomon even more. The student application could count with even more games which show figures such as shapes where they could calculate angles, the length of the side of a triangle or even the areas of squares. This would make the games more interactive towards the students as they would play

games that consist not only of a "2 operand and an operator" question, which after some time, may become quite dull. Furthermore, the timer used for the games was only updated after the next question was presented, it would be better if this timer was updated in real time so the students could be aware of the passed/remaining time at all times. Moreover, it would've also been quite useful for the student if the app generated a report, at the end of each game, where the student is described exactly which questions he did wrong, the answer he gave and the actual answer so the student could attempt the question again and realize his/her mistake. Finally, the administrator's app could improve by adding a delete option in order to delete a student record as well as all the game records stored with the ID of the deleted student.

Alternative solution

An alternative solution has already been described in section A. Sr. Salomon could generate, on paper, a bank of different questions and create a 20 question paper. The students would be handed this question paper on class and as soon they begin a timer would start. As the students finish the question paper they would quickly approach Sr. Salomon and he would record, on a table in Microsoft Excel, the time taken by each student. Later, he would correct the question paper and, based on the total amount of correct answers, he would calculate a final score using the time taken. This is a partial solution to the problem because it does improve the mathematical speed of the students but doesn't solve the problem for the teacher, he would still lose quite a lot of time in generating new questions, correcting the papers and calculating and storing the final score for each student.

Design

The design of the application was successful as it effectively dealt with GUI design, file structures and the data structures (the list of student records and the list of game records).

The GUI design was effective in solving the solution as it was quite user friendly and guided the students and administrator all along the program. It could be improved by adding pictures and other interactive elements such as flash images to attract the attention of the students.

The file structures successfully stored all the student records in a random access file, where they could be accessed directly, reducing the time of access. In regard to the game records, they were separated in three different sequential files, by game type (1, 2 and 3). Making the student record a random access file helped the speed of the system substantially. As there were only 30 students in the class it can be said that a random access file (with direct access) is not needed for processes such as the login because the 30 student accounts can be read extremely quickly. Nevertheless, for the high scores generator, each time a game record is output as a high score the program needs to check the student ID and directly access the student records file and look for the student's name. As there is a list of 15 top game records in the high scores it would take a considerable amount of time if the student records file would needed to be read sequentially for each of the 15 game records in order to find out the student's name. This is why the random access file, with direct access, was perfect for this situation and benefitted the application significantly in terms of speed and efficiency. The file structures could be improved by using random access files for the game records as well, this way they could've been accessed directly and therefore faster, improving the speed of processing and the generation of the previous scores and high scores of the class.

In concern to the data structures, the game records could easily be added to the head of the game record list after each game was finished. They were presented in the previous scores section and the list could be set in order, from high to low, to display the high scores. In regard to the student list, it was ordered by ID number; this way the students were successfully displayed in the administrator's application where the previous scores of each student could be checked. Furthermore, at the beginning, it was stated in the design, that a doubly linked list would've been needed to manage the

game records because a percentage improvement from the previous game record was needed. The doubly linked list wasn't actually required because, as the game records were stored in order (were always added to the bottom of the sequential file) a linked list was sufficient because it was constructed only with the game records with the corresponding ID of the student and it was possible to access the previous game record successfully as they were in order. Moreover, as an improvement, it could've been possible to create a Binary Tree to deal with the stored game records, because as time passes, and the game records for each student increase considerably (as they play and store more game records), the amount of node processing in the lists would be quite large. A binary tree to generate the high scores (total top 15 records in order by score from high to low) would be ideal for this situation as it would decrease the processing time significantly.

Additionally, polymorphism and inheritance was used for the MathQuestion class and the AdditionQuestion, SubstractionQuestion, DivisionQuestion and MultiplicationQuestion sub-classes. They were successful in generating different random questions for the math games and, for future development of the application, if more types of questions would be needed (as stated above in improvements to the solution), they could be added as sub-classes of the MathQuestion class. Nevertheless, there was one limitation when generating the questions prior to the game in the GamePlayingInterfacePanel, this was that the program only created an array of 20 MathQuestion objects. This was done because Game 1 only needs 20 questions and for Game 2 and 3 it is highly unlikely that a student completes more than 20 questions in the given time (e.g.: average of 3 seconds per question in Game 2). But, if a situation was presented in which a student managed to complete more than 20 questions for Game 2 or 3 the program would crash and there was no available error handling for this. A possible solution for this error would be to generate a condition that if a student is in Game 2 or 3 and arrives to question number 20 another batch of MathQuestion objects are created and the user may continue playing.

Finally, if the whole process had to be repeated, the one thing that would've been definitely done would be to conduct interviews with the students of the math class in order to address the solution that I wanted to build, the design that I had in mind and ask for any thoughts or ideas of their own that could've been implemented to the program. I think this would've provided a significant aid because in the end, this application not only serves as a great solution for the teacher but also for the students, who in reality, are the ones who benefit the most from it in the long run.