COMPILING AND RUNNING THE CODE:

All the java source code files are stored in the */coursework/project* folder. To compile the program, navigate to this folder by typing into Terminal the commands:

**cd project**

**javac Main.java**

This produces class files in the same folder. To then run the program, type in to terminal the command:

**java Main**

JAVA STYLE GUIDE:

We chose the Google Java Style Guide for the project. The Integrated Development Environment we used for the project (IntelliJ IDEA) used this style guide by default, so it made sense to choose this for simplicity.

NOTE ON FILE I/O STRUCTURE:

In our project, we decided that rather than creating an array of Question objects for each question in the file at the start, we would only create the few objects required for an individual game, when a user selects "Play the Game" from the menu. This is because we attempted the approach suggested in the Design Brief first, but found that there was a long wait time at the start of the game. Rather than sacrifice the number of questions available to players, we decided to adjust the structure so that the user experience is better. Our approach is also more efficient in terms of computer memory usage. (We discussed this briefly with Rae Harbird, who seemed to think it was an acceptable approach).

CHANGES:

* There is a new FileManagement class, which has methods that deal with file I/O, fulfilling the new requirement for data storage and retrieval in deliverable 2
* To make the login/register process more secure, we made use of a Java System method to mask the keyboard input when asking for passwords
* In UserManagement class, login() and register() methods altered to improve efficiency/stability, and to take inputs for first and last names from the user
* Taking on board feedback from deliverable 1, we changed to an ArrayList rather than an array to store the User objects, so that the number of registrations is not limited
* To avoid storing plain-text passwords, we used a one-way hashing algorithm to make password storage more secure
* The MenuAndAbout class used for deliverable 1 has been merged with the Main class, as we decided that it was unnecessary
* Removed the Game class, and merged its functionality with the GameManagement class, as we decided that it was not needed
* Created a Question class to fulfil deliverable 2 requirement to deal with questions
* MiscFunctions class has been extended with more methods

LIST OF CLASSES:

***Main***

The main class contains the *main()* method, which calls methods from the FileManagement class at the beginning and end of the program to read and write data to the userdata.txt file, and calls the menu() method.

This class also prints the menu text with the *menu()* method, which calls various methods depending on user input. It also contains, the *about()* method, which prints the game’s instructions.

***UserManagement***

This class handles the registration of new users and login, using several methods to do this. The class also informs other classes which is the last user to have logged in, using the *getUserLoggedIn()* method. This class contains the ArrayList of user objects, userObjects, which other classes access through getter and setter methods. The isUserOK() method is used by the User class constructor to check if the inputs are acceptable.

***User***

Currently, this class stores username and password variables for each user object. These variables can be changed, or obtained using *get/set* methods. This class is still relatively small, but now contains additional instance variables for firstName, lastName, numGames and totalScore. It also has a toString() method that returns the user's data in a form such that it can be written to the userdata.txt file.

***GameManagement***

This class has just two methods. The newGame() method is called from the menu, and, firstly, checks whether any user has logged in. If not, it returns to the menu. But if so, it generates an array of question objects using the other method in the class, and then prints out each question using the toStringRandomized() method of the Question class. This class will be extended later to include the game functionality.

***Question***

This class uses getters to return the question, correct answer, incorrect answers, and a String containing the new word, and a list of random answers. ArrayList is used to store answers—in particular, random answers are selected from the array for non-random answers and stored in their own array. Further down in the code, a method returns a String question is made of the new word, and the four potential answers in random order. A toString() method returns a String with the new word and gives the correct and incorrect answers.

***MiscFunctions***

This class is designed to store methods that perform miscellaneous functions in the app that are useful to multiple classes. It includes methods that perform functions useful to multiple classes.

***FileManagement***

This class interacts with user data and questions data in each file (userdata.txt and questions.txt). It has methods which, for example, based on the data saved in the question.txt, return question objects corresponding to a particular line. Also, the number of users and the number of questions are counted based on the number of lines in each text file. The last two methods in the code is used to read the files. The createUserArray() method is run at the start of the program, and adds all the users from the file to the userObjects array. There is also a method that writes the userObjects array to the file at the end.

TESTING THE APP:

To test the app, we made use of the IDE’s debugging functionality, including using breakpoints to stop the program at troublesome sections to see the values stored in variables etc. in order to work out the logical errors.

We tried to test the app systematically, for example testing the outcome when a user registration is attempted after the *userObjects* array is full, or when the return button is pressed in the menu without entering a character, to test whether the app functioned as expected, and whether any exceptions occurred.

CONTRIBUTION MARKS:

We wish to distribute the contribution marks evenly: 10 each.