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**

The game’s formatting may not display correctly in Windows. It should be OK in Codio or macOS, however.

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.

CHANGES:

* Changed class name GameManagement to Game, and made some methods non-static, and added instance variables so that each time the player plays a game the program can easily keep track of that game’s statistics.
* Added a constructor to the Game class to initialize instance variables when a new game is played
* Added game functionality to Game class
* Extended project brief by adding code that times how long players take to answer questions. Players’ average time taken is stored in an instance variable in the User class, and as an entry in the userdata.csv file.
* Added additional checks on the csv files (e.g. checking that number of entries per line is correct, and that data type is correct)
* Created methods in the MiscFunctions class that take user inputs and validate the inputs. These are used by other classes.
* Split the FileManagement class into separate classes for each file, as suggested in feedback from deliverable 2
* Improved the appearance of the game using Unicode border symbols, and a method in MiscFunctions that formats text within a border
* Added a method to the User class that returns a User’s percentage of questions answered correctly
* Made User class implement Comparable, and added a compareTo() method, which is used to sort the User objects by their percentage of answers correct, and average time taken
* Added LeaderBoard class, which sorts the User objects, formats, and displays the leader board. The leader board’s size will adjust to fit the length of the data in the cells

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.

***Game***

This class contains much of the game functionality. When a player chooses to play the game from the menu, a new instance of the Game class is created, and the newGame() method is called. This checks whether there is a player logged in

***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.