This project is to be done on your own.

This project consists of creating three games: WordGame, NumberGame, and MyGame.

Use the best practices that we have learned in this course. A perfectly-running game with extremely bad style can lose up to 50% of the marks!!!

Create at least these four classes: Main, WordGame, NumberGame, and MyGame.

Main

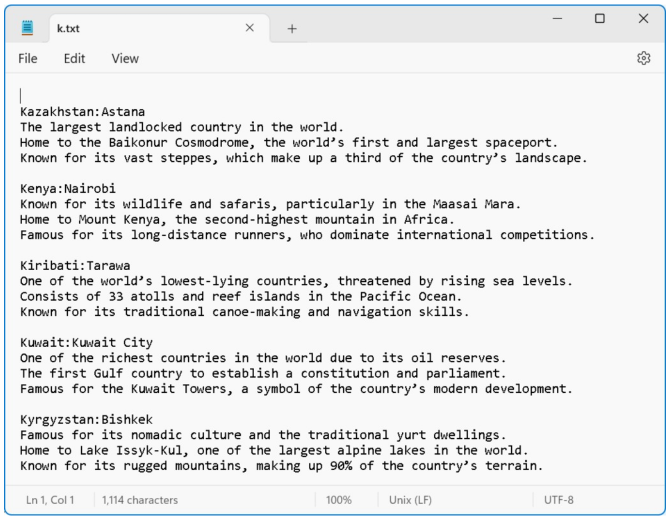
Offers a menu (in the terminal; i.e. no GUI) in an infinite loop until the user presses Q (or q):

Press W to play the Word game.
Press N to play the Number game.
Press M to play the <your game's name> game.
Press Q to quit.

If the user enters wrong data (not w, W, n, N, M, m, q, or Q), give an error message and re-prompt.

WordGame

CAUTION: Make sure your scoring passes the provided ScoreTest unit tests. This game uses the files provided to you: a.txt, b.txt, …, z.txt. Do not change the data in these files. Example:



```
Kazakhstan:Astana
The largest landlocked country in the world.
Home to the Baikonur Cosmodrome, the world’s first and largest spaceport.
Known for its vast steppes, which make up a third of the country’s landscape.

Kenya:Nairobi
Known for its wildlife and safaris, particularly in the Maasai Mara.
Home to Mount Kenya, the second-highest mountain in Africa.
Famous for its long-distance runners, who dominate international competitions.

Kiribati:Tarawa
One of the world’s lowest-lying countries, threatened by rising sea levels.
Consists of 33 atolls and reef islands in the Pacific Ocean.
Known for its traditional canoe-making and navigation skills.

Kuwait:Kuwait City
One of the richest countries in the world due to its oil reserves.
The first Gulf country to establish a constitution and parliament.
Famous for the Kuwait Towers, a symbol of the country’s modern development.

Kyrgyzstan:Bishkek
Famous for its nomadic culture and the traditional yurt dwellings.
Home to Lake Issyk-Kul, one of the largest alpine lakes in the world.
Known for its rugged mountains, making up 90% of the country’s terrain.
```

The Word Game does not use a GUI, so your prompts, menus, and user input requirements must be very clear and easy to follow.

At the start of the game, load all of the files into data structures as follows:

Create a **Country** class with instance variables, constructor arguments, and accessor methods for:

String name (e.g. Canada)
String capitalCityName (e.g. Ottawa)
Array (not ArrayList) facts; for example:
0: Home to the longest coastline in the world.

1: Famous for its maple syrup production, accounting for 71% of the world's supply.
2: One of the most multicultural nations in the world, with more than 200 ethnic origins represented.

Create a **World** class with a HashMap of all the countries. The key is the country name and the value is the Country reference (e.g. "Canada" maps to the Country whose data was just explained in the previous paragraph).

This is a geography trivia game, with the rules as follows:

The user will be asked ten questions at random, one at a time. Keep track of their score. The question will vary randomly between three types:

a) The program will print a capital city, and ask the user what country it is the capital of
b) The program will print the country name, and ask the user what is its capital city
c) The program will print one of the three facts, and ask the user which country is being described

The user will be given one of the questions above (a-c) and asked to guess which country (or city in the case of b). Then the game will report "CORRECT" or "INCORRECT". The user gets a second guess if they were INCORRECT. Keep track of their score (did they get it correct in one guess or two? Or did they get two incorrect guesses?). If they do not get it right in either guess, tell them the answer in the format of "The correct answer was Ottawa").

After ten questions have been asked, report the score in the format of:

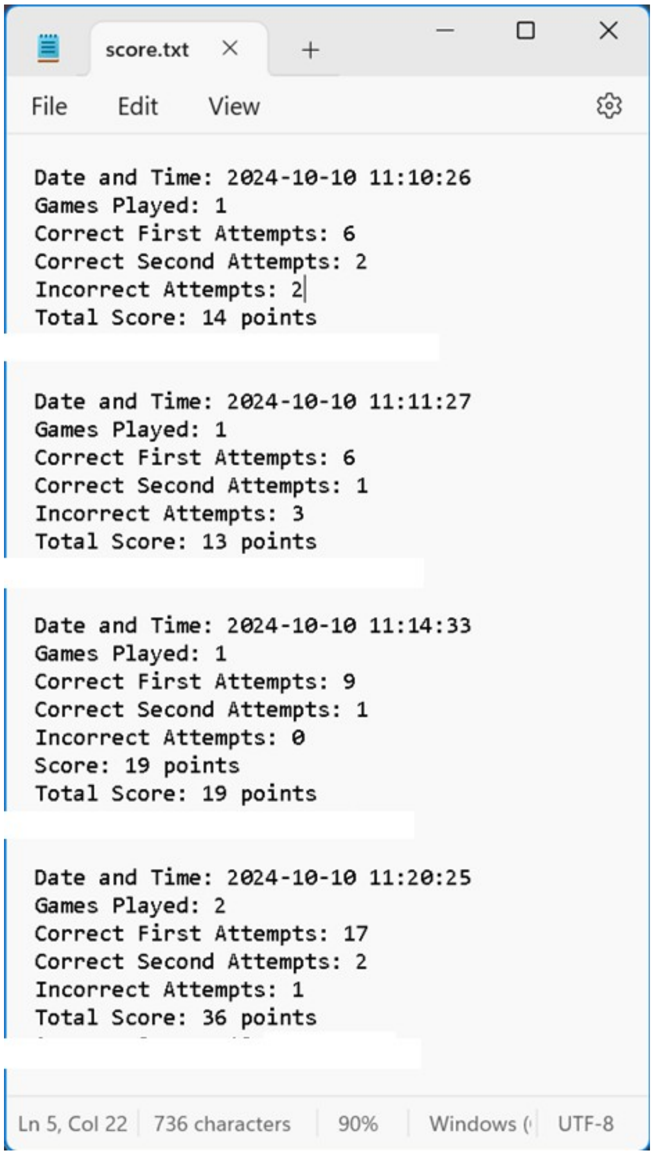
- 1 word game played
- 3 correct answers on the first attempt
- 2 correct answers on the second attempt
- 5 incorrect answers on two attempts each

Or

- 8 word games played
- 22 correct answers on the first attempt
- 10 correct answers on the second attempt
- 48 incorrect answers on two attempts each

Ask the user if they want to play again. If they say "Yes" (in any letter case), repeat the game. Track the total score for all the games they play. If they say "No" (in any letter case), quit this game so you return to the main menu. Other entries (not "Yes", not "No"), give a clear error message and repeat the prompt.

ALSO: when the user chooses to NOT play again, append their results (or create the file if it does not yet exist) in a text file named "score.txt" in this format:



```
Date and Time: 2024-10-10 11:10:26
Games Played: 1
Correct First Attempts: 6
Correct Second Attempts: 2
Incorrect Attempts: 2
Total Score: 14 points

Date and Time: 2024-10-10 11:11:27
Games Played: 1
Correct First Attempts: 6
Correct Second Attempts: 1
Incorrect Attempts: 3
Total Score: 13 points

Date and Time: 2024-10-10 11:14:33
Games Played: 1
Correct First Attempts: 9
Correct Second Attempts: 1
Incorrect Attempts: 0
Score: 19 points
Total Score: 19 points

Date and Time: 2024-10-10 11:20:25
Games Played: 2
Correct First Attempts: 17
Correct Second Attempts: 2
Incorrect Attempts: 1
Total Score: 36 points
```

Create a **Score** class with instance data for dateTimePlayed, numGamesPlayed, numCorrectFirstAttempt, numCorrectSecondAttempt, numIncorrectTwoAttempts.

NOTE: here is a good, simple way to capture and print the DateTime as displayed in the score.txt screenshot above:

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

final LocalDateTime currentTime;
final DateTimeFormatter formatter;
final String formattedDateTime;

currentTime = LocalDateTime.now();
formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
formattedDateTime = currentTime.format(formatter);
```

System.out.println(formattedDateTime);

When the user quits the WordGame, if they have the new "high score" (higher than any other score in the score.txt file), tell them a congratulations message in this format: "CONGRATULATIONS! You are the new high score with an average of 6.75 points per game; the previous record was 6.44 points per game on 2023-10-31 at 14:30:01". Here is how to calculate the percentage:

- First-attempt correct guesses get 2 points
- Second-attempt correct guesses get 1 point
- Both-attempt incorrect guesses get 0 points

Examples:

8 word games played	
22 correct answers on the first attempt	44 points
10 correct answers on the second attempt	10 points
48 incorrect answers on two attempts each	0 points

Total is 54 points in 8 games, for an average score of 6.75 points per game.

When the user quits the WordGame, if they do not have the new "high score" (higher than any other score in the score.txt file), tell them the highest score in a sentence like this: "You did not beat the high score of 6.44 points per game from 2023-10-31 at 14:30:01".

HINT: keep a List<Score> of Score references.

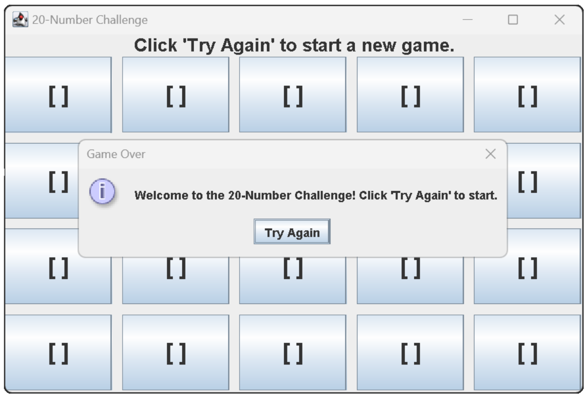
Pass all of the ScoreTest.java JUnit test cases. Use the provided **ScoreTest.java** file unit tests to show whether your Score class, score.txt, and scoring algorithms are all functioning correctly.

NumberGame

This game uses a GUI. The GUI can look exactly as in the examples below, or "different and better"…but not worse. Make as many interfaces, classes and methods as you feel are necessary; make at least one interface, one abstract class, and one concrete class. The game shows a board similar to that below: a grid of four rows of five buttons each.

The game (pseudo-) randomly selects 20 integers between 1 and 1000, inclusive, one at a time. Each time the game picks a number (invisible to the user), the user must click on a square into which that number is placed. Once placed, a number cannot be moved. If a square is clicked after a number has already been placed in it, the click is ignored. The goal is to place all 20 numbers into the grid such that the numbers are in ascending order. If the numbers are not in ascending order, you lose. For example, if the game picks number 20, and the user puts it into the first (lowest) square, but then the next number is lower than 20, there is nowhere for that new number to be placed. The game is over; the user lost. Advise the user the game is over and offer a re-try option and quit option; if they quit, they return to the main menu. Your code must use arrays.

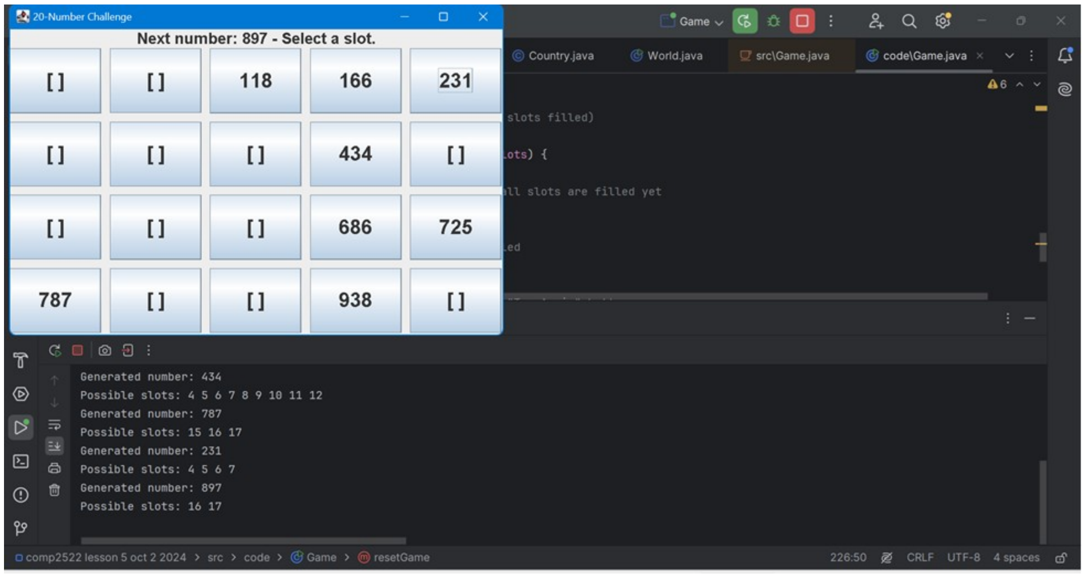
Game start:



Click 'Try Again' to start a new game.

Welcome to the 20-Number Challenge! Click 'Try Again' to start.

Your program will pick the numbers randomly from 1-1000 and each time offer the user the chance to place it in the grid of 20 squares, **always keeping the grid in order from lowest to highest:**

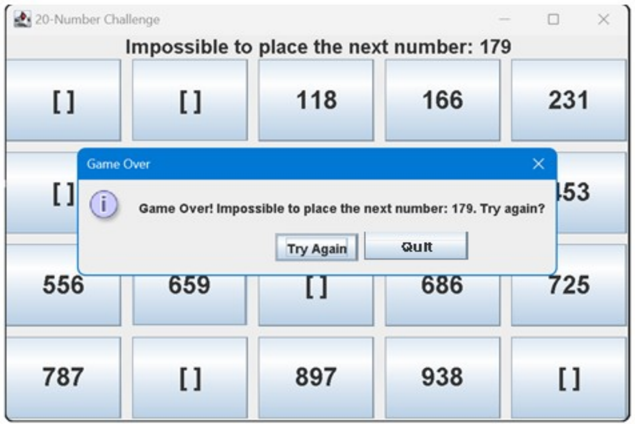


Next number: 897 - Select a slot.

```
Generated number: 434
Possible slots: 4 5 6 7 8 9 10 11 12
Generated number: 787
Possible slots: 15 16 17
Generated number: 231
Possible slots: 4 5 6 7
Generated number: 897
Possible slots: 16 17
```

Note the helpful debugging print() statements I used when creating my game.

The game continues until the user wins or runs out of squares in which to place the next number (this is a loss). In the next screen there is no space to put 179 so the user lost:



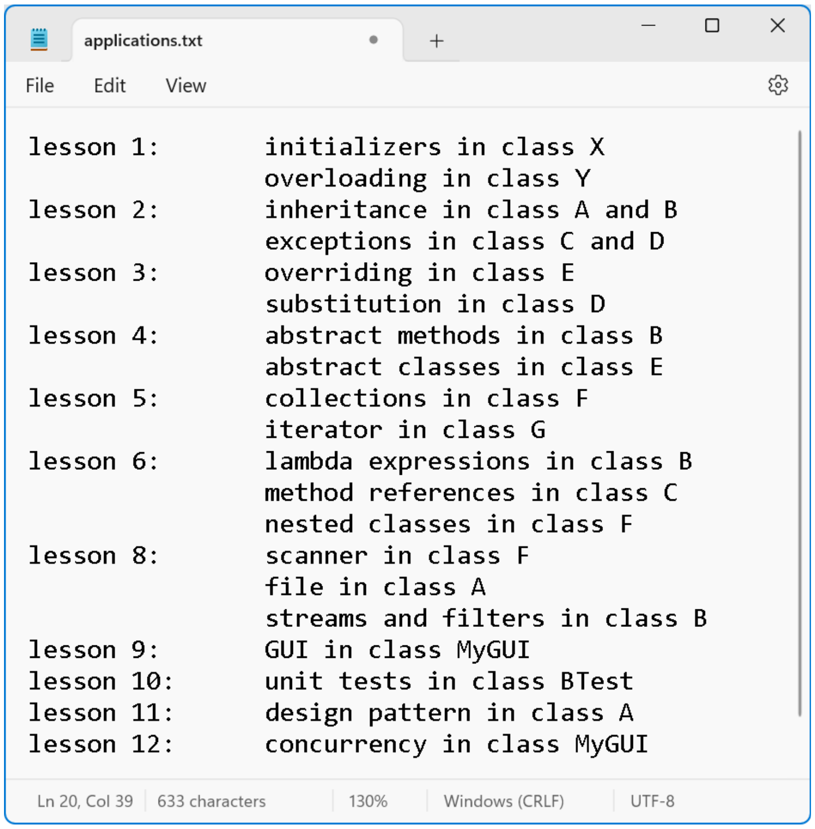
Impossible to place the next number: 179

Game Over! Impossible to place the next number: 179. Try again?

If the user clicks Try Again, keep track of the score (e.g. "You lost 1 out of 1 game, with 12 successful placements, an average of 12 per game", or "You lost 4 out of 4 games, with 57 successful placements, an average of 14.25 per game", or "You won 1 out of 2 games and you lost 1 out of 2 games, with 24 successful placements, an average of 12 per game", etc…. When the user quits, show their score status again (as in the previous sentence) and close the game window, and re-show the main menu again.

MyGame

Your code must apply code from every lesson of our course (with the possible exception of a GUI, which has already been tested in the NumberGame). Include a text file named **applications.txt** which shows where in your own personal game code you applied the lesson. If you do not submit this file you will lose 10% of the overall grade. Example:



```
lesson 1:       initializers in class X
                overloading in class Y
lesson 2:       inheritance in class A and B
                exceptions in class C and D
lesson 3:       overriding in class E
                substitution in class D
lesson 4:       abstract methods in class B
                abstract classes in class E
lesson 5:       collections in class F
                iterator in class G
lesson 6:       lambda expressions in class B
                method references in class C
                nested classes in class F
lesson 8:       scanner in class F
                file in class A
                streams and filters in class B
lesson 9:       GUI in class MyGUI
lesson 10:      unit tests in class BTest
lesson 11:      design pattern in class A
lesson 12:      concurrency in class MyGUI
```

Just show *something* important from each lesson is used in at least one place somewhere. The list does NOT have to show *everything*.

This game must be a UNIQUE game that you create yourself **with the help of an AI tool** such as ChatGPT. 100% of the score can be deducted if the style is not consistent with our course's best practices, or if the game uses non-COMP2522 practices, or if the game is not written by "you plus an AI" (e.g. the complete code is findable online). Examples of starting prompts to the AI could be "make me a Java game that is a combination of sudoku and wordle". Also include a **prompts.txt** text file that explains the best and worst parts of the experience you had trying to communicate with the AI bot and get its help. The more unique and fun the game is, the higher the score. You may use ONLY what we have learned in this course.

SUBMISSION:

Submit the following files to the learning hub before December 1 in the Activities/Term Project dropbox. Include these files in a single .zip file:

Main.java
WordGame.java
Country.java
Score.java
NumberGame.java
MyGame.java
score.txt
applications.txt
prompts.txt

…and any/all other Interface and Class files that you created. Do not upload the a.txt, b.txt, etc… files.

Scoring Rubric

Requirement	**Max Score**
WordGame functionality	40
High score functionality (pass the ScoreTest tests)	15
NumberGame functionality	30
applications.txt has data for each lesson	10
MyGame is fun and unique and clear how to play	30
prompts.txt is clear and useful; it should be about two pages long; be detailed	10
Video (details below) NOTE: your work will earn a grade of 0 if no videos are uploaded	50
Total	185

NOTE: you can lose all your marks if your coding style is unsafe, inconsistent, etc….

NOTE: your AI game must be unique; if the entire game (or almost all of the game) is findable online as-is, then you can score zero for the entire project; put time into MyGame to ensure it is substantially YOURS (yours plus the AI's).

Video submission details:

Submit two videos:

Video 1: worth 25 of 50 marks for the video portion

Name it “<firstname-lastname-video1.mp4>” (e.g. tiger-woods-video1.mp4)

90-150 seconds long; not shorter; not longer

This is a technical video; show your file system with all the project’s files; show the code in your IDE; show all the settings etc…. In other words, show that you understand all the code setup, IDE settings, etc….

Your video must start with the exact sentence “My name is <first name last name> in set <X> and I deserve xx% on this project because …”

For example: “My name is Tiger Woods in set E and I deserve 82% on this project because I have a perfect, fun personal game deserving 100%. I have the numbers game running perfectly. And the word game runs about 50%; it doesn’t work in the following way…. I used best practice everywhere.”

Video 2: worth 25 of 50 marks for the video portion

Name it “<firstname-lastname-video2.mp4>” (e.g. tiger-woods-video2.mp4)

90-120 seconds long; not shorter; not longer

This is a marketing video; show off your games running (show off each game’s best features) and “sell them” to the user. For very good marketing examples, check these out:

- https://www.youtube.com/watch?v=keCwRdbwNQY
- https://www.youtube.com/watch?v=xhrAGJviQJA
- https://www.youtube.com/watch?v=ewEk6l3WGVE