ASSIGNMENT 4

CCCS-300, Fall 2021

See MyCourses

Please read the entire PDF before starting. You must do this assignment individually.

Question 1:	100 points
	100 points total

It is very important that you follow the directions as closely as possible. The directions, while perhaps tedious, are designed to make it as easy as possible for the TAs to mark the assignments by letting them run your assignment, in some cases through automated tests. While these tests will never be used to determine your entire grade, they speed up the process significantly, which allows the TAs to provide better feedback and not waste time on administrative details.

Up to 30% can be removed for bad indentation of your code as well as omitting comments, or poor coding structure.

To get full marks, you must:

- Follow all directions below
  - In particular, make sure that all classes and method names are **spelled and capitalized exactly** as described in this document. Otherwise, you will receive a **50% penalty**.
- Make sure that your code compiles
  - **Non-compiling code will receive a 0**.
- Write your name and student ID as a comment in all .java files you hand in
- Indent your code properly
- Name your variables appropriately
  - The purpose of each variable should be obvious from the name
- Comment your work
  - A comment every line is not needed, but there should be enough comments to fully understand your program

Part 1 (0 points): Warm-up

Do **NOT** *submit this part, as it will not be graded. However, doing these exercises might help you to do the second part of the assignment, which will be graded. If you have difficulties with the questions of Part 1, then we suggest that you consult the TAs during their office hours; they can help you and work with you through the warm-up questions. You are responsible for knowing all of the material in these questions.*

Warm-up Question 1 (0 points)

Write a program that *opens* a **.txt**, *reads* the contents of the file line by line, and *prints* the content of each line. To do this, you should look up how to use the `BufferedReader` or `FileReader` class[1]. Remember to use the `try` and `catch` blocks to handle errors like trying to open an non-existent file. A sample file for testing file reading is found in the provided files as *dictionary.txt*.

Warm-up Question 2 (0 points)

Modify the previous program so that it stores every line in an `ArrayList` of `String` objects. You have to properly declare an `ArrayList` to store the results, and use `add` to store every line that your program reads in the `ArrayList`.

Warm-up Question 3 (0 points)

Modify your program so that, after reading all the content in the file, it prints how many words are inside the text file. To do this, you should use the `split` method of the `String` class. Assume the only character that separates words is whitespace `" "`.

Warm-up Question 4 (0 points)

Create a new method in your program which takes your `ArrayList` of `Strings`, and writes it to a file. Use the `FileWriter` and `BufferedWriter` classes in order to access the file and write the `Strings`. In the output file, there should be one `String` per line, just like the original file you loaded the `ArrayList` from.

Warm-up Question 5 (0 points)

Create a new method in your program which takes as input your `ArrayList` of `Strings`, and sort all the elements. The sorting criterion will be the length of the string. In other words, after calling this method, the shortest string must be located in the first position, the second shortest in the second position and so on.

Warm-up Question 6 (0 points)

Create a new method in your program which takes as input a sorted `ArrayList` (see the previous question for details about the sorting criterion) and two `ints`. The two `ints` will represent a range of values. This method should return an `ArrayList` with all the Strings whose length is inside that range. For example, if your original `ArrayList is equal to` {`"aa","aaa","aaaa","aaaaa"`} and the two `ints` are `3` and `4`, your method must return the `ArrayList` {`"aaa","aaaa"`} (because the length of the returned `Strings` is within `3` and `4`).

[1]The documentation of the `BufferedReader` class is available at `http://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html`. You can find an example on how to use it at `http://www.tutorialspoint.com/java/io/bufferedreader_readline.htm`

Warm-up Question 7 (0 points)

Write a class describing a `Cat` object. A cat has the following `attributes`: a name (String), a breed (String), an age (int) and a mood (String). The mood of a cat can be one of the following: `sleepy, hungry, angry, happy, crazy`. The cat `constructor` takes as input a String and sets that value to be the breed. The `Cat` class also contains a method called `talk()`. This method takes no input and returns nothing. Depending on the mood of the cat, it prints something different. If the cat's mood is `sleepy`, it prints *meow*. If the mood is `hungry`, it prints *RAWR!*. If the cat is `angry`, it prints *hssssss*. If the cat is `happy` it prints *purrrr*. If the cat is `crazy`, it prints a String of 10 gibberish characters (e.g. raseagafqa).

The cat `attributes` are all **private**. Each one has a corresponding **public** get method (ie: `getName(), getMood()`, etc.) which returns the value of the `attribute`. All but the `breed` also have a **public** set method (ie: `setName(), setMood()`, etc.) which takes as input a value of the type of the attribute and sets the attribute to that value. Be sure that only valid mood sets are permitted. (ie, a cat's mood can only be one of five things). There is no setBreed() method because the breed of a cat is set at birth and cannot change.

Test your class in another file which contains only a main method. Test all methods to make sure they work as expected.

Warm-up Question 8 (0 points)

Using the `Cat` type defined in the previous question, create a `Cat[]` of size 5. Create 5 `Cat` objects and put them all into the array. Then use a loop to have all the `Cat` objects `meow`.

Warm-up Question 9 (0 points)

Write a class `Vector`. A `Vector` should consist of three `private` properties of type double: x, y, and z. You should add to your class a constructor which takes as input 3 doubles. These doubles should be assigned to x, y, and z. You should then write methods `getX()`, `getY()`, `getZ()`, `setX()`, `setY()`, and `setZ()` which allow you to get and set the values of the vector. Should this method be static or non-static?

Warm-up Question 10 (0 points)

Add to your `Vector` class a method `calculateMagnitude` which returns a double representing the magnitude of the vector. Should this method be static or non-static? The magnitude can be computed by taking:

$$magnitude = \sqrt{x^2 + y^2 + z^2}$$

Part 2

The questions in this part of the assignment will be graded.

Question 1: Battle Game (50 points)

For this question, you will write a number of classes to create a battle game between a player and a monster. Your code for this assignment will go in multiple .java files. Note that in addition to the required methods below, you are free to add as many other private methods as you see fit.

`It is up to you to figure out if the methods from the Character class and the Spell class should be static or not. BattleGame and FileIO are utilities classes, therefore all their methods are static.`

We *strongly recommend* that you complete the warm-up questions before starting this problem.

(a) Character Class

Write a class `Character.java` that represent a character in our battle game. The monster and the player will both be characters, each with their own attributes.

The Character class should contain the following private attributes:

- A `String` name
- A `double` attack value
- A `double` maximum health value
- A `double` current health value
- A `int` number of wins in the battle game

The class also contains the following public methods.

- A `constructor`: The constructor for the Character class takes one `String`, two doubles, and one `int` as input. These parameters represent the name, attack value, maximum health, and number of wins in the battle game for the character, in that order. Note that the current health of a new character is the same as the maximum health.
- A `getName(), getAttackValue(), getMaxHealth(), getCurrHealth(), and getNumWins()` to retrieve the corresponding attributes' values.
- A `toString()` to returns a String consisting of the character's name and current health. Format the String in any way you want. This method will be very handy for debugging your code, and will be used during the battle game to keep track of the health of each character.
- A `getAttackDamage()`: This method takes an integer as input and calculates how much attack damage one character does when they attack. The method uses the input to generate a Random object with the given input as seed. The method then computes the damage as follows: `take the character's attack value and multiply it by a random value between 0.7 (inclusive) and 1.0 (exclusive).`
- A `takeDamage()`: This method takes the damage done to this character as an input of type double. It then subtract this value from the character's current health and returns a double indicating the current health of the character.
- A `increaseWins()`: This method will increase the number of wins by the character by one, and does not return anything. This method will be called when the character wins the battle game.

(b) FileIO Class

`FileIO.java` must contain the following public static method:

- A `readCharacter()`: This method takes as input a filename as a `String` parameter, and returns a new `Character`, using the constructor defined in the Character class. The **readCharacter** method must use a **FileReader** and a **BufferedReader** in order to open the file specified by the filename. Make sure to catch `FileNotFoundException` and `IOException`

  when reading from the file. If either exception is raised, print an appropriate error message and return null.

  You can assume that the files that readCharacter receives as input have all the same format:

  they contain 4 lines, with the following information

  - Name of the character
  - Attack value
  - Maximum health
  - Number of wins so far in the battle game

  Examples of such files are the `player.txt` and the `monster.txt` files are provided with the assignment. Use the readLine() method of the BufferedReader to retrieve the content of the file. Use the information retrieved to create and return the appropriate object of type Character.

(c) BattleGame class

The code for this part will go in a file named `BattleGame.java`. In this class, add a private static attribute of type Random and initialize it in place with a reference to a Random object created with no seed. (You can fix a seed for helping you debug your program if you want to)

The `BattleGame` class has only one public static method called `playGame()`. You are highly encouraged to add private helper methods that allow you to organize your code well.

The `playGame()` method takes two Strings as input containing the name of two files: the first containing the information about a player, the second containing the information about a monster. The method should do the following:

- Create the player and their enemy (the monster), using the readCharacter() method from the FileIO class and the inputs received. If the method does not return two valid references to objects of type Character, then print a message saying that the game cannot be played and terminate the method.
- Display the information of the two characters in the game including: the character's name, current health, attack value, and number of wins.
- Create a Scanner object to take input from the user.
- Until both the player and the monster have health above zero the method does the following:
  - Ask the user for a command. For the moment, the only options will be attack and quit.
  - If the user enters attack:
    - Generate a random integer (using the class variable and the method nextInt() with no input) and get the attack damage of the player.
    - Print out a statement with the player's name and how much damage they do. To make your output nicer, we suggest using a String formatting statement, though this is not

necessary. For example, you could write this to only show two decimal places of the attack damage:

```
String attackStr = String.format("%1$.2f", attack);,
where attack is a variable containing the damage value.
```

    * Apply the damage made by the player to the monster. If the current health of the monster after taking the damage is still above 0, print a message with their name and current health. Note that you can take advantage of having a toString() method in the Character class to do this. If on the other hand, the current health is less than or equal to 0, print a message saying that the player was knocked out and exit the loop.
    * Repeat the above steps swapping the roles of the player and the monster (i.e. the monster should now be attacking the player back).
  - If the user enters quit:
    * Print a goodbye message and terminate the method.
  - If the user enters any other command:
    * Print a message that the input was not recognized and suggest the attack or quit commands.
- If the loop stops because one of the character's health is zero or below, then that character is knocked out. Print an appropriate message either congratulating the player, or saying how they lost. Also make sure to increase the number of wins of either the player or the monster, depending on who won.

Here is some sample output produced by running the playGame method after Question 1 has been finished. Output for the finished assignment is found at the bottom of this document. Note that for this assignment, your output doesn't need to match exactly the samples provided. You are free to change these statements as you wish, as long as the required information still appear.

```
Name: Odin
Health: 30.00
Attack: 10.00
Number of Wins: 0

Name: Fenrir
Health: 30.00
Attack: 12:00
Number of Wins: 0
Enter a command:
attack

Odin attacks for 9.85 damage!
Fenrir current health is 20.15.
Fenrir attacks for 9.86 damage!
Odin current health is 20.14.
Enter a command:
attack

Odin attacks for 9.30 damage!
Fenrir current health is 10.85.
Fenrir attacks for 9.94 damage!
Odin current health is 10.20.

Enter a command:
quit

Goodbye!
```

Question 2: Extending the BattleGame (35 points)

For this question, you will modify the above classes and add one new class, in order to add magical spells for the player to use.

Note that you only hand in one set of files for this assignment.

(a) Spell Class

Write a class `Spell.java`. A Spell has the following private attributes:

- A `String` name
- A `double` minimum damage
- A `double` maximum damage
- A `double` chance of success for the spell (from 0 to 1)

The `Spell` class also contains the following public methods:

- A constructor that takes as input the name, minimum and maximum damage, and chance of success for the spell. Note that, an `IllegalArgumentException` must be thrown if the minimum damage is less than 0 or greater than the maximum damage, or if the chance of success is less than zero or greater than one.
- A `getName()` method which returns the name of the spell.
- A `getMagicDamage()` method that returns the damage produced by the spell. The method takes an integer as input and uses it to generate a Random object with the input as the seed. The method then computes the damage as follows: first, a random number double between 0 and 1 is generated. If the random number is above the chance of success, the spell fails, and 0 damage is returned. Otherwise, a random double between the minimum damage and the maximum damage is returned.
- A `toString()` method. The String which is returned must contain the name, minimum and maximum damage, and the success chance as a percentage from 0 to 100 (so a chance of 0.5 is reported as 50.0

(b) Character Class

Modify the Character class as follows:

- Add a private static attribute spells. This variable is an ArrayList which contains all the Spells that the characters can cast.
- Add a setter method setSpells() for the spells attribute. The method takes an ArrayList of Spells as input, and copies the Spells contained in the input parameter to a new ArrayList stored in the spells attribute.
- Add a new method displaySpells() which prints out one spell per line **(you should take advantage of the toString method from the Spell class)**. This method does not return a value.
- Add a new method castSpell(), which takes the name of a spell to cast as input as well as an integer. This method returns a double indicating the damage done by the specified spell. To do so, you will have to search through the list of spell for the spell with the name matching the input. Note that you should ignore the capitalization of the spell name when matching. If the spell cannot be found, the method should return -1. Otherwise, the method returns the damage done by casting the specified spell.

(c) FileIO Class

Add a new public static method called readSpells(). This method takes a filename as input and returns an ArrayList of Spells.

The method reads the file using `FileReader` and `BufferedReader`. Make sure to catch the `IOException` and `FileNotFoundException` exceptions. If either exception is raised print an appropriate error message and return null.

You can assume that the files that `readSpells()` receives as input have all the same format: each spell is on one line, consisting of the spell name, the min damage, the max damage, and the chance for success each separated by a tab character. You can use the `split()` method from the String class to split each line of the file.

An example of a file containing spells is found in the provided files as spells.txt.

Use the four values on each line of the file to create new Spell, and store all the spells in the ArrayList that will then be returned by the method.

(d) BattleGame Class Change the `playGame` method in the `BattleGame` class to allow the player to cast spells.

- Change the inputs to the `playGame()` method by adding a third input indicating the name of the file containing the information about the spells.
- At the beginning of the method, call the `readSpells()` method in the `FileIO` class with the filename of file containing spells. If the method returns null then print a message saying that the game will be played without spells. Use the `setSpells()` method from the Character class to initialize the attribute appropriately. You should then display all the available spells using the appropriate method from the Character class.
- In addition to the attack and quit commands, we will allow the user to cast spells. If the input is neither attack nor quit, then we assume that the player is trying to cast a spell. In this case, generate a random integer **(using the class variable and the method `nextInt()` with no input)** and use it, together with the name of the spell, to retrieve the damage done by the player when casting the specified spell.

  If the damage returned is less than or equal to 0, print a message saying that the player tried to cast a spell, but they failed. Otherwise, print a message saying that the spell was casted and produced a certain amount of damage. This is the damage that the monster receives. In both messages, the name of the player as well as the name of the spell should appear. See the example output at the end of this document. Note that, the monster is not allowed to cast spells. The monster will reply to the player's spell simply by attacking back (as before).

Question 3: Recording the Wins (15 points)

For this question, we will add code to write the characters back to a file, to save how many wins each character has.

Note that you only hand in one set of files for this assignment.

(a) FileIO Class

In the `FileIO` class add a public static method named `writeCharacter()`. This method takes as input a Character to write, and a String which indicates the filename to write to.

Within the `writeCharacter()` method, use `FileWriter` and `BufferedWriter` objects to write the character's information back to a file. Make sure to catch the `IOException` when writing a file and display an appropriate error message.

The format of the file must match the format that is expected when a character is read. That is, you should be able to write a character to a file, and then read a character from that same file.

Recall that the format of a character in a file is:

- Name of the character
- Attack value
- Maximum health
- Number of wins so far in the battle game

The above information appear on four different lines.

(b) BattleGame

In the playGame() method, after you have increased the wins for either the monster or the player, print how many wins each character has.

You must then write the winning character to the file you loaded it from. Remember that the playGame() method receives these filenames as input. For example, if playGame() was called as follows playGame("player.txt", "monster.txt", "spells.txt") and the player won the game, then you should write the player character into the "player.txt" file. This will save the number of wins for that character, so that after playing the battle game multiple times, you will know which character has won more often.

Here is some sample outputs for the finished program. The output is obtained by calling playGame("player.txt", "monster.txt", "spells.txt") from the main method of the BattleGame class. Again, your output doesn't need to exactly match, as long as the same information is presented. To make it easier for you to debug your program, in the samples below I used a seed to initialize the attribute of type Random from the BattleGame class.

Round 1 of the game with the Random attribute from the class `BattleGame` initialized using the seed 123. Note that if you use the same seed and you provide the same inputs, you should see the same output as the one displayed below. (The number of wins depend on how many times you ran the game)

```
Name: Odin
Health: 30.0
Attack: 10.0
Number of wins: 0

Name: Fenrir
Health: 30.0
Attack: 12.0
Number of wins: 0

Here are the available spells:
Name: fireball Damage: 5.0-10.0 Chance: 50.0%
Name: icestorm Damage: 1.0-7.0 Chance: 90.0%
Name: meteorstrike Damage: 10.0-10.0 Chance: 5.0%
Name: surge of frostfire Damage: 7.0-10.0 Chance: 30.0%

Enter a command:
fireball

Odin tried to cast fireball, but they failed.

Fenrir attacks for 10.23 damage!
Odin current health is 19.77.

Enter a command:
Surge OF Frostfire

Odin casts Surge OF Frostfire dealing 9.11 damage!
Fenrir current health is 20.89.

Fenrir attacks for 9.28 damage!
Odin current health is 10.49.

Enter a command:
attack

Odin attacks for 8.09 damage!
Fenrir current health is 12.81.

Fenrir attacks for 10.19 damage!
Odin current health is 0.30.

Enter a command:
ICESTORM

Odin casts ICESTORM dealing 1.19 damage!
Fenrir current health is 11.61.

Fenrir attacks for 9.30 damage!
Odin was knocked out!

Oh no! You lost!
Successfully wrote to file: monster.txt
Fenrir has won: 1 times
```

Round 2 of the game with the Random attribute from the class `BattleGame` initialized using the seed 456. Note that if you use the same seed and you provide the same inputs, you should see the same output as the one displayed below. (The number of wins depend on how many times you ran the game)

```
Name: Odin
Health: 30.0
Attack: 10.0
Number of wins: 0

Name: Fenrir
Health: 30.0
Attack: 12.0
Number of wins: 1

Here are the available spells:
Name: fireball Damage: 5.0-10.0 Chance: 50.0%
Name: icestorm Damage: 1.0-7.0 Chance: 90.0%
Name: meteorstrike Damage: 10.0-10.0 Chance: 5.0%
Name: surge of frostfire Damage: 7.0-10.0 Chance: 30.0%

Enter a command:
attack

Odin attacks for 8.01 damage!
Fenrir current health is 21.99.

Fenrir attacks for 9.05 damage!
Odin current health is 20.95.

Enter a command:
surge of frostfire

Odin casts surge of frostfire dealing 9.86 damage!
Fenrir current health is 12.13.

Fenrir attacks for 11.75 damage!
Odin current health is 9.20.

Enter a command:
surge of frostfire

Odin casts surge of frostfire dealing 7.35 damage!
Fenrir current health is 4.78.

Fenrir attacks for 9.02 damage!
Odin current health is 0.18.

Enter a command:
attack

Odin attacks for 7.75 damage!
Fenrir was knocked out!

Fantastic! You killed the monster!
Successfully wrote to file: player.txt
Odin has won: 1 times
```

Round 3 of the game with the Random attribute from the class `BattleGame` initialized using the seed 789. Note that if you use the same seed and you provide the same inputs, you should see the same output as the one displayed below. (The number of wins depend on how many times you ran the game)

```
Name: Odin
Health: 30.0
Attack: 10.0
Number of wins: 1

Name: Fenrir
Health: 30.0
Attack: 12.0
Number of wins: 1

Here are the available spells:
Name: fireball Damage: 5.0-10.0 Chance: 50.0%
Name: icestorm Damage: 1.0-7.0 Chance: 90.0%
Name: meteorstrike Damage: 10.0-10.0 Chance: 5.0%
Name: surge of frostfire Damage: 7.0-10.0 Chance: 30.0%

Enter a command:
Turn to Ash

Odin tried to cast Turn to Ash, but they don't know that spell.

Fenrir attacks for 8.52 damage!
Odin current health is 21.48.

Enter a command:
FIREBALL

Odin casts FIREBALL dealing 9.74 damage!
Fenrir current health is 20.26.

Fenrir attacks for 8.50 damage!
Odin current health is 12.98.

Enter a command:
attack

Odin attacks for 9.07 damage!
Fenrir current health is 11.19.

Fenrir attacks for 9.02 damage!
Odin current health is 3.96.

Enter a command:
meteorstrike

Odin tried to cast meteorstrike, but they failed.

Fenrir attacks for 8.74 damage!
Odin was knocked out!

Oh no! You lost!
Successfully wrote to file: monster.txt
Fenrir has won: 2 times
```

What To Submit

Please put all your files in a folder called *Assignment4*. Zip the folder (DO NOT RAR it or use other compression extension like .7z) and submit it on MyCourses. If you use other compression extension like .rar, .7z etc, you will lose marks. **Use only .zip**.

Inside your zipped folder, there must be the following files. **Do not submit any other files, especially .class files.** Any deviation from these requirements may lead to lost marks.

Note that you should submit only one set of files for this assignment. Do not submit your files from Question 1 or Question 2, but only the finished files from Question 3.

`BattleGame.java`
`Character.java`
`Spell.java`
`FileIO.java`
`Confession.txt` (optional) In this file, you can tell the TA about any issues you ran into doing this assignment. If you point out an error that you know occurs in your problem, it may lead the TA to give you more partial credit. On the other hand, it also may lead the TA to notice something that otherwise they would not.