ASSIGNMENT 5

COMP-202, Winter 2017, All Sections

Due: Tuesday, April 11^{th}, 11:59pm

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

Question 1:	50 points
Question 2:	35 points
Question 3:	15 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. Plus, if the TA is in a good mood while he or she is grading, then that increases the chance of them giving out partial marks. :)

Up to 30% can be removed for bad indentation of your code as well as omitting comments, or poor coding structure. Marks will be removed as well if the class and method names are not respected.

To get full marks, you must:

- Follow all directions below
- Make sure that your code compiles
  - Non-compiling code will receive a very low mark
- 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 `Student` class that has a private `String name` attribute. Create a getter for the `name` attribute, a constructor, and a `toString` method.

Warm-up Question 6 (0 points)

Create an `ArrayList` that stores `Students`, and add five `Students` to the `ArrayList`. Print out the `ArrayList` using the `toString` method., and a for-loop. Can you make the for-loop version look like the built-in `ArrayList toString` method?

[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`

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 question will go in multiple `.java` files.

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

Note that in addition to the required methods below, you are free to add as many other `private` methods as you want.

(a) Character Class

`Character.java` represents a character in our battle game. The monster and the player will both be instances of the `Character` class, 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

Here are the required public methods for this class. Note that you will also have to add getters and setters for the instance attributes as needed.

1) 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.

2) The `toString` method

This method 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.

3) `calcAttack` method

This method calculates how much attack damage one character does in a battle. The calculation is as follows: Take the character's attack value and multiply it by a random value between 0.3 (inclusive) and 0.7 (exclusive). Return this value as a double. Use the `Random` class to generate the random numbers, and do not use a seed.

4) `takeDamage` method

In this method, we take the damage done to this character as a `double` parameter, and then we subtract this value from the character's current health. This method does not return anything.

5) `increaseWins` method

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

`FileIO.java` must contain a static method `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` to open the file specified by `filename`. Make sure to have two catch blocks to catch both `FileNotFoundException` and `IOException` when reading from the file. In these cases, throw an `IllegalArgumentException` with an appropriate message that the file was not found, or that there was an IO exception. Note that the catch block for the `FileNotFoundException` must be directly above the catch block for the `IOException`.

If you prefer, you can also use the `throws` statement to throw the `FileNotFoundException` and `IOException` up to the calling `playGame` method. Then these `Exceptions` would be caught in a try/catch block there.

Example *player.txt* and *monster.txt* files are provided with the assignment. The format of these files is exactly this, one value per line:

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

Use the `readLine()` method of the `BufferedReader` to get the lines from the file. Do not use the `Scanner` class. Then, use the `Double.parseDouble()` and `Integer.parseInt()` methods to parse the lines. Finally, use those values to create and return a new `Character`.

(c) BattleGame

The code for this part will go in a file named `BattleGame.java`.

The `BattleGame` class only contains the `playGame` method, and the `doAttack` method, though you are allowed to create as many other `private` methods as you want.

1) `playGame` method must do the following things:

- Create the player and their enemy, using the `FileIO.readCharacter` method
  - After a character is created, print the character's name, current health, attack value, and number of wins.
  - We recommend you place this print statement in a public method inside the `Character` class
- Use a `Scanner` to take input from the user
- Loop while both the player and the monster have health above zero
  - Ask the user for a command
  - For this question, the only options will be *attack* and *quit*
  - If the input is *attack*:
    - Call the `doAttack` method (in the `BattleGame` class) with the **player** as first parameter and the **monster** as second parameter
    - Call the `doAttack` method with the **monster** as first parameter and the **player** as second parameter

- If the input is *quit*
  - Print a goodbye message and return from the method
- If the input is anything else
  - 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 call the `increaseWins` method on either the player or the monster, depending who won.

2) The `doAttack` method takes two `Characters` as input, and returns nothing as output. This method must:

- Get the damage from the first `Character` by calling the `calcAttack` method
- Print out a statement with the first character's name and how much damage they do.
- Apply the damage to the second `Character` with the `takeDamage` method
- If the second character's current health is still above zero, print a message with their name and current health. If their current health is zero or below, print out a message saying that the second character was knocked out.

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);`

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 this exactly. You are free to change these statements as you wish, as long as the required information still appears.

```
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 takes 9.85 damage!
Name:  Fenrir Health:  20.15

Fenrir attacks for 9.86 damage!
Odin takes 9.86 damage!
Name:  Odin Health:  20.14

Enter a command:
attack
Odin attacks for 9.30 damage!
Fenrir takes 9.30 damage!
Name:  Fenrir Health:  10.85
```

```
Fenrir attacks for 9.94 damage!
Odin takes 9.94 damage!
Name:  Odin Health:  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

Write a class `Spell`. 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
  - An `IllegalArgumentException` must be thrown if the minimum damage is less than 0 or greater than the maximum damage, or the chance of success is less than zero or greater than one.
- `getName` which returns the name of the spell
- A `getDamage` method that returns the damage for a spell casting. First, a random number from 0 to 1 is generated. If the random number is above the chance of success, the spell fails, and 0 damage is returned. Otherwise, a random number from the minimum damage to the maximum damage is returned. Use the `Random` class to generate the random numbers, and do not use a seed.
- 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

Modify the `Character` class.

1) Add a private static attribute `spells`. This variable will be an `ArrayList` which contains `Spells` that the character can cast. Note that this list of spells will be available to all `Characters`.

2) Add a setter method `setSpells` for the `spells` attribute. Recall that a setter method allows other classes to set values for private attributes. This method will take one parameter, and will copy the `Spells` contained in the input parameter into a new `ArrayList` stored in the `spells` attribute. This method does not return a value.

3) Add a new method `castSpell`, which takes the name of a spell to cast as a parameter `spellName`. This method will return the damage done by the spell as a double.

This method will iterate through the list of spells, searching for the spell with the name matching `spellName`. Note that you should ignore the capitalization of the spell name when matching.

If the spell cannot be found, print a message which says that the character tried to cast an unknown spell. Also return zero damage.

If the spell is found, the damage done by the spell is given by calling the `getDamage` method on the spell. If the damage returned is zero, then the spell failed to cast. Print a message that the character failed to cast the spell. If the damage was above zero, print a message that the spell was cast. Return the damage from the `castSpell` method.

In all messages printed in this `castSpell` method, the message should have the character's name, as well as the name of the spell. See the example output at the end of this document.

(c) FileIO

Add a new static method, the `readSpells` method. This method takes a filename of spells to read, and returns an `ArrayList` of `Spells`.

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

Read the file indicated by the method parameter `filename`, by using a `BufferedReader`. Catch the `IOException` and `FileNotFoundException` exceptions in two separate catch blocks, and throw an `IllegalArgumentException` with an appropriate error message if these occur. Note that the catch block for the `FileNotFoundException` must be above the catch block for the `IOException`.

As before, if you prefer you can also use the `throws` statement to throw the `FileNotFoundException` and `IOException` up to the calling `playGame` method. Then these `Exceptions` would be caught in a try/catch block there.

Each spell is on one line, consisting of the spell name, the min damage, the max damage, and the chance for success, separated by spaces. Use the `split()` method to split each line of the file by the space delimiter.

Use the four values on each line of the file to create new `Spell` instances, and then add these `Spell` instances to the `ArrayList` which will be returned from this method. Do not use the `Scanner` class to parse the file.

(d) playGame

Change the `playGame` method in the `BattleGame` class to allow the player to cast spells.

1) At the beginning of the `playGame` method, call the `readSpells` method in the `FileIO` class with the filename of the file containing spells. This method returns an `ArrayList` of spells.

Print out each spell (using the `toString` method of the `Spell` class), so that the player knows which spells they may cast.

Then call the `setSpells` method in the `Character` class with this list of spells.

2) In addition to the *attack* and *quit* commands, we will allow the user to cast spells. If the input is not *attack* or *quit*, then the input must be passed to the `castSpell` method on the player's character, as we assume it is the name of a spell to cast.

The damage returned from the `castSpell` method should then be applied to the monster, using the `takeDamage` method.

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

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

Within this `writeCharacter` method, use the `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 throw an `IllegalArgumentException` with an appropriate error message.

If you prefer, you can also use the `throws` statement to throw the `IOException` up to the calling `playGame` method. Then this `Exceptions` would be caught in a try/catch block there.

The format of the file that is written 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 each value on a separate line:

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

Note that this means you will have to write getter methods for the attributes in the `Character` class.

(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.

As well, you must write the characters to the files you loaded them from. For example, write the player character to `player.txt` or the monster character to `monster.txt`.

Call the `writeCharacter` method in the `FileIO` class, and pass the character who won and the file which stores that character as parameters. 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 output for the finished program. Again, your output doesn't need to exactly match, as long as the same information is presented.

```
Name:  Odin Health:  30.00
Attack:  10.00
Number of Wins:  8
Name:  Fenrir Health:  30.00
Attack:  12.00
Number of Wins:  12

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%

Enter a command:
fireball
Odin tried to cast fireball, but they failed
Fenrir attacks for 9.84 damage!
```

```
Odin takes 9.84 damage!
Name:  Odin Health:  20.16

Enter a command:
ICESTORM
Odin casted icestorm for damage of 6.50
Fenrir takes 6.50 damage!
Name:  Fenrir Health:  23.50

Fenrir attacks for 10.18 damage!
Odin takes 10.18 damage!
Name:  Odin Health:  9.98

Enter a command:
badbreath
Odin tried to cast badbreath, but they don't know that spell
Fenrir attacks for 10.09 damage!
Odin takes 10.09 damage!
Odin was knocked out!
Name:  Odin Health:  0.00

Oh no!  You lost!
Fenrir has won:  13 times
```

What To Submit

You have to submit one zip file that contains all your files to myCourses - Assignment 5. If you do not know how to zip files, please obtain that information from any search engine or friends. Google might be your best friend with this, and for a lot of different little problems as well.

These files should all be inside your zip. Do not submit any other files, especially .class files.

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.