ASSIGNMENT 4

CCCS-300, Winter 2020

Due: Friday, April 24^{th}, 11:59pm

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: Pokemon Master Cup (100 points)

For this question, you will write a number of classes that you can use to implement a game that simulates a Pokemon trainer battle. 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.

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

The general idea of the assignment is to mainly implement the following data types `Pokemon`, `SkillMove`, `PokemonZoo`, `PokemonTrainer`, and `MasterCup`. An object of type `PokemonTrainer` should have 5 Pokemons and each Pokemon should be able to perform skill moves. The number of skill moves that the Pokemon can perform depends on how strong the Pokemon is. For attributes, you need to initialize all your attributes in a constructor.

(a) SkillMove Class

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

- A `String` name
- A `String` type
- A `double` dmg
- A `double` missRate

name represents the name of the skill move, **type** represents the skill move type, **dmg** represents the damage the skill move can have, and finally **missRate** represents the miss-rate of the skill move.

The SkillMove class also contains the following public method:

- A `toString` method: The String which is returned must contain the name, and type. Below is an example of a SkillMove printing:

```
Solar-Beam (Gra)
```

Note that you will also have to add getters and setters for the instance attributes as needed.

(b) Pokemon Class

`Pokemon.java` represents a Pokemon in our battle game. The Pokemon class should contain the following (private) attributes:

- A `String` name
- A `double` maxHealth
- A `double` currentHealth
- A `String` type
- A `ArrayList<SkillMove>` moves

In this class, Pokemon should specify it's name, type, health, and most importantly, a Pokemon should have its own set of skill moves.

Here are the required public methods for this class. Again you will need getters and setters for the instance attributes as needed.

- An `attack` method: A method that takes a Pokemon as target. The target Pokemon should lose health if the current Pokemon successfully hits its skill move on the target. You should consider the skill move miss-rate here.
- A `toString` method: The String which is returned must contain the name, and list of skill moves. Below is an example of a Pokemon printing:

```
Bulbasaur, Moves:[Energy-Ball (Gra), Petal-Blizzard (Gra)]
```

(c) PokemonZoo class

Write a class `PokemonZoo.java` which has the following private attributes:

- A `ArrayList<SkillMove>` movesList
- A `ArrayList<Pokemon>` pokemonList

In this class, you should load all the skill moves and Pokemons into the attributes **movesList** and **pokemonList**, respectively.

The PokemonZoo class contains the **loadMoves()** method, and the **loadPokemon** method, though you are allowed to create as many other private methods as you want.

- `loadMoves` method: In this method you must use a FileReader and a BufferedReader to open the **skillMove.txt** file. Make sure to have two catch blocks to catch both FileNotFoundException and IOException when reading from the file. The file format is described as follows:

```
[type] [name of the move]   [dmg]   [miss rate]
 Gro    Precipice-Blades    76.5        0.65
 Ele      Thunder           41.7        0.5
```

  All the skill move data must be stored into the attribute **movesList** and you should print out the **moveList** when all the moves are loaded, Below is an example when the **moveList** is printed.

```
[Precipice-Blades (Gro), Origin-Pulse (Wat), Hydro-Cannon (Wat), Doom-Desire (Ste).....
```

- `loadPokemon` method: Similar to the previous method, in this method, you need to load all the Pokemons information from the file **pokemons.txt** to the attribute **pokemonList**. The file format is described as follows:

```
[name]    [health] [type]  [number of moves]
Pikachu     111     Ele            1
Charizard   186     Fir            3
```

Moreover, you should randomly assign skill moves for each Pokemon in this method. You should use the provided **getRandomInt()** method to select the random move from the **moveList**. Note that different Pokemons have different maximum number of moves and all the moves should match the type of the Pokemon.

(d) PokemonTrainer Class

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

- A `String` name
- An `int` win
- An `ArrayList<Pokemon>` team

name represents the name of the Pokemon Trainer. Each trainer should have 5 Pokemons in **team**. All the attributes must be initialized in the constructor. You should also provide a **toString** method here. The toString method should print using the following format:

```
Trainer: Jax, Wins: 0, team:[
Turtwig, Moves:[Giga-Drain (Gra), Seed-Bomb (Gra)],
Huntail, Moves:[Bubble-Beam (Wat), Aqua-Tail (Wat), Origin-Pulse (Wat)],
Sunflora, Moves:[Grass-Knot (Gra), Solar-Beam (Gra), Giga-Drain (Gra)],
Ninetales, Moves:[Heat-Wave (Fir), Blast-Burn (Fir), Blast-Burn (Fir)],
Slowpoke, Moves:[Bubble-Beam (Wat), Bubble-Beam (Wat)]]
```

(e) MasterCup Class

The code for this part will go in a file named `MasterCup.java`. This class is the main class that simulate all the matches between each Pokemon trainer. Therefore, you should have the following attribute in the class.

- `ArrayList<PokemonTrainer>` trainerList

Then, you need to have the following methods:

- `createTrainers` method: This method takes an integer as first parameter and an ArrayList as second parameter. The integer parameter represents the number of trainers in the master cup. For example, if the input integer is 5, you should generate 5 Pokemon trainers into the attribute **trainerList**. The name of the trainer should be obtained from the file **name.txt** in order.

  Moreover, each trainer should have 5 random Pokemons which are picked from the input ArrayList. Note that you need to use the provided random function to pick Pokemons. Finally, you should print out the **trainerList** when you complete it. Below is an example output of this method if we have 5 trainers.

  ```
  Trainer: Jax, Wins: 0, team:[
  Turtwig, Moves:[Giga-Drain (Gra), Seed-Bomb (Gra)],
  Huntail, Moves:[Bubble-Beam (Wat), Aqua-Tail (Wat), Origin-Pulse (Wat)],
  Sunflora, Moves:[Grass-Knot (Gra), Solar-Beam (Gra), Giga-Drain (Gra)],
  Ninetales, Moves:[Heat-Wave (Fir), Blast-Burn (Fir), Blast-Burn (Fir)],
  Slowpoke, Moves:[Bubble-Beam (Wat), Bubble-Beam (Wat)]],

  Trainer: Randall, Wins: 0, team:[
  Magmortar, Moves:[Weather-Ball(Fir), Fire-Punch(Fir), Flamethrower(Fir), Flame-Charge(Fir)],
  Lombre, Moves:[Water-Pulse (Wat), Hydro-Cannon (Wat)],
  Finneon, Moves:[Scald (Wat)],
  Shaymin, Moves:[Solar-Beam (Gra), Leaf-Blade (Gra), Leaf-Blade (Gra), Petal-Blizzard (Gra)],
  Exeggcute, Moves:[Leaf-Blade (Gra), Frenzy-Plant (Gra)]],

  Trainer: Minnie, Wins: 0, team:[
  Tentacruel, Moves:[Scald (Wat), Surf (Wat), Origin-Pulse (Wat)],
  Kingdra, Moves:[Origin-Pulse (Wat), Scald (Wat), Hydro-Cannon (Wat)],
  Snover, Moves:[Frenzy-Plant (Gra), Frenzy-Plant (Gra)],
  Pikachu, Moves:[Thunderbolt (Ele)],
  Manectric, Moves:[Thunder-Punch (Ele), Zap-Cannon (Ele), Parabolic-Charge (Ele)]],

  Trainer: Brianna, Wins: 0, team:[
  Bellossom, Moves:[Leaf-Blade (Gra), Seed-Bomb (Gra), Grass-Knot (Gra)],
  Cradily, Moves:[Grass-Knot (Gra), Solar-Beam (Gra), Power-Whip (Gra)],
  Pichu, Moves:[Discharge (Ele)],
  ```

```
Breloom, Moves:[Energy-Ball (Gra), Frenzy-Plant (Gra), Mega-Drain (Gra)],
Gloom, Moves:[Solar-Beam (Gra), Grass-Knot (Gra)]],

Trainer: Fredric, Wins: 0, team:[
Slowking, Moves:[Aqua-Tail (Wat), Surf (Wat), Surf (Wat)],
Magikarp, Moves:[Brine (Wat)],
Vileplume, Moves:[Grass-Knot (Gra), Energy-Ball (Gra), Solar-Beam (Gra)],
cherrim, Moves:[Power-Whip (Gra), Petal-Blizzard (Gra), Solar-Beam (Gra)],
Golduck, Moves:[Brine (Wat), Aqua-Tail (Wat), Weather-Ball (Wat)]]]
```

- `healAll` method: This method takes one PokemonTrainer as the input and heal all the Pokemons that the trainer has. This means that all Pokemons' current health should set back to the initial health.
- `pvp` method: This method should takes two PokemonTrainer as the input. In this method, you need to simulate a battle game between two trainers. Since each trainer has 5 Pokeomons, the trainer who defeats all the opponent's Pokemons win the game. A Pokemon is considered defeated when it's current health is less or equal to zero. Note that the first Pokemon trainer is the gym leader and the second trainer is the challenger, therefore, the first Pokemon trainer should make the move first. In addition, the trainer who wins the game should update its win attribute. Remember to heal all the Pokemons when the winner is found.
- `startMatch` method: This method simulates a group stage of the match. Each Pokemon trainer can only be a gym leader once, and the gym leader should play with all other trainers at least once. For example, suppose we have $\{a, b, c\}$ trainers. a should be the gym leader and play against b and c. Then, b will be the next gym leader and play against a and c. Finally, c will be the next gym leader and play against a and b.
- `writeScores` method: This method should write all the information to a file called **scores.txt**. Below is an example of scores that you need to write to the file.

```
[

Trainer: Jax, Wins: 3, team:[
Turtwig, Moves:[Giga-Drain (Gra), Seed-Bomb (Gra)],
Huntail, Moves:[Bubble-Beam (Wat), Aqua-Tail (Wat), Origin-Pulse (Wat)],
Sunflora, Moves:[Grass-Knot (Gra), Solar-Beam (Gra), Giga-Drain (Gra)],
Ninetales, Moves:[Heat-Wave (Fir), Blast-Burn (Fir), Blast-Burn (Fir)],
Slowpoke, Moves:[Bubble-Beam (Wat), Bubble-Beam (Wat)]],

Trainer: Randall, Wins: 6, team:[
Magmortar, Moves:[Weather-Ball (Fir), Fire-Punch (Fir), Flamethrower (Fir), Flame-Charge (Fi
Lombre, Moves:[Water-Pulse (Wat), Hydro-Cannon (Wat)],
Finneon, Moves:[Scald (Wat)],
Shaymin, Moves:[Solar-Beam (Gra), Leaf-Blade (Gra), Leaf-Blade (Gra), Petal-Blizzard (Gra)],
Exeggcute, Moves:[Leaf-Blade (Gra), Frenzy-Plant (Gra)]],

Trainer: Minnie, Wins: 5, team:[
Tentacruel, Moves:[Scald (Wat), Surf (Wat), Origin-Pulse (Wat)],
Kingdra, Moves:[Origin-Pulse (Wat), Scald (Wat), Hydro-Cannon (Wat)],
Snover, Moves:[Frenzy-Plant (Gra), Frenzy-Plant (Gra)],
Pikachu, Moves:[Thunderbolt (Ele)],
Manectric, Moves:[Thunder-Punch (Ele), Zap-Cannon (Ele), Parabolic-Charge (Ele)]],

Trainer: Brianna, Wins: 2, team:[
```

```
Bellossom, Moves:[Leaf-Blade (Gra), Seed-Bomb (Gra), Grass-Knot (Gra)],
Cradily, Moves:[Grass-Knot (Gra), Solar-Beam (Gra), Power-Whip (Gra)],
Pichu, Moves:[Discharge (Ele)],
Breloom, Moves:[Energy-Ball (Gra), Frenzy-Plant (Gra), Mega-Drain (Gra)],
Gloom, Moves:[Solar-Beam (Gra), Grass-Knot (Gra)]],

Trainer: Fredric, Wins: 4, team:[
Slowking, Moves:[Aqua-Tail (Wat), Surf (Wat), Surf (Wat)],
Magikarp, Moves:[Brine (Wat)],
Vileplume, Moves:[Grass-Knot (Gra), Energy-Ball (Gra), Solar-Beam (Gra)],
cherrim, Moves:[Power-Whip (Gra), Petal-Blizzard (Gra), Solar-Beam (Gra)],
Golduck, Moves:[Brine (Wat), Aqua-Tail (Wat), Weather-Ball (Wat)]]]
```

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.

`PokemonZoo.java`
`Pokemon.java`
`PokemonTrainer.java`
`SkillMove.java`
`MasterCup.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.