ASSIGNMENT 5

COMP-202, Winter 2019, All Sections

Due: Friday, April 12th, 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. 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.

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 an 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 ofn 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: Tamagotchi (100 points)

For this question, you will write a number of classes that you can use to implement a game that simulates a Tamagotchi. 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* `Toy` *class and the* `Tamagotchi` *class should be* `static` *or not.* `Playground` *and* `FileIO` *are utilities classes, therefore all their methods should be* `static`.

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

The general idea of the assignment is to implement a new data type `Tamagotchi`. An object of type `Tamagotchi` should be able to perform some basic tasks such as play, eat, and sleep. As a consequence to these actions the Tamagotchi's energy and experience will change. In addition to this, you will implement two utility classes that will allow you to use a Tamagotchi and to save the progress done each time you close the program.

(a) (25 points) Toy Class

`Toy.java` represents a toy that a Tamagotchi will be able to play with. You are provided with some starter code for this class. Note that this code already contains four `private static` attributes as well as three `private static` methods. You should use them when you implement the rest of the class.

The *Toy* class should contain the following `private` attributes:

- A `String` name
- A `String` color
- A `String` type
- A `int` representing the experience gained by playing with this Toy.

The class also contains the following `public` methods.

- Two constructors:
  - One constructor that takes as input three `String`s, and an `int`. These parameters represent the name, color, type, and experience awarded by the Toy, in that order. The constructor should use its inputs to initialize the attributes accordingly.
  - A second constructor that takes no inputs and initializes the attributes with values generated at random. Use `getRandomName()`, `getRandomColor()`, and `getRandomType()` to initialize the corresponding attributes. Generate a random integer between 10 (included) and 25 (excluded) to initialize the attribute representing the experience awarded by the toy.
- A `getName()`, `getColor()`, `getType()`, and `getXp()` to retrieve the corresponding attributes' values.
- The `toString()` method
  This method returns a `String` consisting of the toy's name, color, and type. For example, the method might return the following: `"Trooper the yellow stuffed cat"`, where "Trooper" is the name of the toy, "yellow" is its color, and "stuffed cat" is its type. This method will be very handy for debugging your code, and you can use it whenever you need to display information about a toy.

- `createToy()` method
  This method takes a `String` as input containing all the information needed to represent a Toy. The method returns a Toy created using such information. A valid input String will contain the name, the color, the type, and the experience to be awarded. Each piece of information will be separated by a tab character (`\t`). You can use the method `split()` from the `String` class to help you isolate each piece. If the input does not contain 4 pieces of information your method should throw an `IllegalArgumentException`. Otherwise the method will proceed to create the corresponding `Toy` and return it.
- `findBestToy()` method
  This method takes an `ArrayList` of `Toy`s as input and returns the Toy that awards the greatest experience for playing with it. If the list is empty the method should return `null`. If there's more than one Toy awarding the highest experience, then the first one in the list should be returned.

(b) (50 points) Tamagotchi Class

Note that you are provided with some starter code for this class too. The code already contains two `private static` attributes.

The *Tamagotchi* class should contain the following `private` attributes:

- A `String` representing its name
- An `int` indicating its level
- A `double` indicating its energy
- An `int` representing its experience points
- An `int` indicating how many meals the tamagotchi has consumed.
- An `ArrayList` of toys containing all its toys.

The class also contains the following `public` methods.

- Two constructors:
  - One constructor that takes as input a `String`, an `int`, a `double`, two `int`s, and an `ArrayList` of `Toy`s. These parameters represent the name, level, energy, experience, meals consumed, and toys of a tamagotchi, in that order. The constructor should use its inputs to initialize the attributes accordingly. Remember how you should initialize attributes that are mutable reference types.
  - A second constructor that takes as input only one `String` representing the name of the tamagotchi. The constructor uses the input to initialize the attribute name, and then it initializes the remaining attributes as follow:
    - level with 1
    - energy with `MAX_ENERGY` (10.0)
    - experience with 0
    - meals consumed with 0
    - the list of toys with a list with one element containing a Toy generated at random.
- A `getName()`, `getLevel()`, `getEnergy()`, `getXp()`, `getNumOfMeals()`, and `getToys()` to retrieve the corresponding attributes' values. Remember how you should write a get method when dealing with attributes that are mutable reference types.

- A `private` method called `levelUp()`
  This method does not receive any input. It checks whether the tamagotchi has enough experience to level up. If its experience is greater than or equal to $50 * level * (level + 1)/2$, then the tamagotchi should level up. This is obtained by increasing its level by one, resetting the meals consumed to 0, and creating a brand new toy (randomly generated) to be added to its list of toys. The method should also print a message letting the user know that the tamagotchi has levelled up, its new level, and which new toy was added to the list. If the experience of the tamagotchi is not enough to level up, then the method does not do anything. An example of the method output could be the following:

  ```
  *** YAY, time to level up!! ***
  Mimitchi is now level 2
  Your new toy is Mary the blue ball
  ```

- The `play()` method
  This method takes as input an integer indicating a playing mode. You can assume that this integer is going to be either equal to 1 or to 2. The method should do the following:

  – If the Tamagotchi has no toys or it has energy less than 2, then the method throws an `IllegalStateException` with a message indicating that the tamagotchi is not able to play.

  – Otherwise, if the playing mode is 1, the method selects the best toy out of this tamagotchi's toys, if the playing mode is equal to 2, it selects a random toy.

  – The method updates the experience and the energy of the tamagotchi. Let the experience gained be equal to x, then the energy used by the tamagotchi to play is a random double between $x/20.0$ (included) and $x/20.0 + 0.5$ (excluded).

  – The method displays messages indicating which toy the tamagotchi played with and what are its energy and experience as a consequence of this.

  – Finally, the method should call `levelUp()` in case the experience gained was enough for the tamagotchi to change level.

  For example, when called, the method might display the following:

  ```
  Mimitchi played with Trooper the yellow stuffed cat and earned 17 xp.
  Mimitchi has now 135 xp, and 8.78 energy.
  ```

- The `feed()` method
  This method takes no inputs. If the energy of the tamagotchi is less than 1 or if the tamagotchi has already consumed a number of meals greater than or equal to twice its current level, then the method throws an `IllegalStateException` indicating that the tamagotchi cannot eat. Otherwise, the method generates the following:

  – a random double between 0 (included) and 0.5 (excluded) representing the energy gained by eating.

  – a random integer between 1 (included) and 4 (excluded) representing the experience gained from eating.

  The method uses the two random numbers to update the corresponding attributes. It then increases the number of meals consumed by 1, and it displays a message indicating the updated energy and experience values of the tamagotchi. For example, when called, the method might display the following:

  ```
  Nom nom nom
  Mimitchi has now 137 xp, and 8.91 energy.
  ```

- The `sleep()` method
  This method takes no inputs. It simply displays that the tamagotchi is now going to sleep and it resets its energy back to `MAX_ENERGY`.
- The `toString()` method
  This method returns a `String` consisting of all the information regarding a Tamagotchi. For instance, if you display what this method return, you might see the following:

```
Name:       Mimitchi
Level:      2
Energy:     10.00
XP:         137
Meals:      3
Toys:   [Trooper the yellow stuffed cat, Mary the blue ball]
```

(c) (25 points) FileIO

`FileIO.java` must contain the following `private static` methods:

- `loadToys()` method.
  This method takes as input a filename as a `String` parameter, and returns an `ArrayList` of `Toys`.

  The `loadToys` method must use a `FileReader` and a `BufferedReader` in order to open the file specified by the filename. Declare the `IOException` in the header of the method using the `throws` keyword.

  You can assume that the files that `loadToys` receives as input have all the same format: each line consists of a toy name, color, type, and experience points, each separated by a tab character (`\t`).

  An example of a file containing toys is found in the provided files as *mimitchiToys.txt*.

  Using the information on each line and the method `createToy()` from the Toy class, create all the necessary `Toys`, and store them all in the `ArrayList` that will then be returned by the method.

  In this method you should catch the `IllegalArgumentException` that might be raised by the method `createToy`. In the catch block simply display a message stating that the format of the file is incorrect. In such a case, the method should return an *empty* `ArrayList`. (NOTE that empty does not mean `null`!)

- `saveToys()` method.
  This method takes as input an `ArrayList` of `Toys` as well as a `String` representing a filename.

  The `saveToys()` method must use a `FileWriter` and a `BufferedWriter` in order write to the file specified by the filename. Make sure to handle the `IOException` using a try-catch block and print a meaningful message if necessary.

  The format of the file created by `saveToys()` should match the format that is expected by `loadToys()`. (See the description given above)

  This method should return `true` if the file was successfully written, `false` otherwise.

`FileIO.java` must also contain the following `public static` methods:

- `loadTamagotchi()` method.
  This method takes as input a filename as a `String` parameter, and returns a `Tamagotchi`.

The method must use a `FileReader` and a `BufferedReader` in order to open the file specified by the filename. Declare the `IOException` in the header of the method using the `throws` keyword.

You can assume that the files that `loadTamagotchi` receives as input have all the same format: they contain 6 lines, with the following information about a tamagotchi

  - Name
  - Level
  - Energy
  - Experience
  - Meals consumed
  - The name of a file containing all its toys

An example of such files is the *mimitchi.txt* which is 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 `Tamagotchi`.

- `saveTamagotchi()` method.
  This method takes as input a `Tamagotchi`, a `String` representing the filename in which the tamagotchi should saved, and a `String` representing the filename in which the tamagotchi's toys should be saved.

  The `saveTamagotchi()` method must use a `FileWriter` and a `BufferedWriter` in order write to the file specified by the filename. Make sure to handle the `IOException` using a try-catch block and print a meaningful message if necessary.

  The format of the file created by `saveTamagotchi()` should match the format that is expected by `loadTamagotchi()`. (See the description given above) Note that method should use its third input to write the last line of the file. More over, the method should also make sure that all the toys belonging to the specified Tamagotchi will be saved to the appropriate file.

  This method should return `true` if the files was successfully written, `false` otherwise.

(d) (0 points) Playground (Optional)

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

The `play()` method takes no inputs. The method should do the following:

- Create a `Scanner` object to take input from the user.
- Welcome to user to the game and ask for the name of the tamagotchi they'd like to play with.
- Assume that the file containing the information regarding the tamagotchi is named as follows: "name.txt", where "name" is what the user provided as input and the file name contains only lower case letters of the alphabet. For instance, if the user provides you with the name "Mimitchi", then the program should be looking for a file named "mimitchi.txt".
- The program uses the name of the file to load the correct tamagotchi. If something goes wrong while doing so, then the program creates a new tamagotchi with the specified name using directly one of the tamagotchi constructors.
- The program then displays the info regarding the tamagotchi and asks the user to enter a command between play, feed, and sleep.
- Until the user enters the command "sleep", do the following
  - If the user entered *play* then ask the user to also enter the playing mode (an integer between 1 and 2) and call the appropriate method from the Tamagotchi class.

  - If the user enters *feed* call the appropriate method from the Tamagotchi class.
  - If the user enters any other command print a message that the input was not recognized.
  - If an `IllegalStateException` was raised by any of the methods, then the program should not crash and instead display the message from this exception. (Note, that you can use the method `getMessage()` on an object of type Exception to retrieve the appropriate message)
  - If an `InputMismatchException` was raised (because the user entered something that was not a int when asked for an int, for instance), then the program should not crash and instead display an appropriate message.
  - Ask the user for the next command.
- If the program gets out of the loop it's because the command `sleep` was entered. The program should then call the appropriate method and save the current state of the Tamagotchi to a file named "name.txt" (for example "mimitchi.txt") and its toys to a file named "nameToys.txt" (for example "mimitchiToys.txt")

On the next page you can find some sample output produced by running the `play()` method from the main method of the `Playground` class with no seeds in the constructors of the Random objects. The first output was generated when I did not have any text file containing information about a Tamagotchi. The second one was ran right after the first one. The txt files provided to you with this assignments are the text files created by my program after the second run.

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.

```
Welcome to our last game of the semester!

Please enter the name of the Tamagotchi you'd like to play with: Mimitchi

Your Tamagotchi has been created

      oooooo      oooooo
     o      o    o      o
     o      o    o      o
     o      o    o      o
     o ooooo oooo ooooo o
    o                    o
   o     ^          ^     o
  o     / \        / \     o
  o                        o
  o                        o
   o       ________       o
    oo                  oo
      ooo            ooo
         oo oooooo oo


Tamagotchi card
Name:           Mimitchi
Level:          1
Energy:         10.00
XP:             0
Meals:          0
Toys:   [Fisher the purple kite]

Enter a command (play, feed, or sleep):
play

Choose one of the following options:
1) Mimitchi plays with their best toy
2) Mimitchi plays with a random toy
1

Mimitchi played with Fisher the purple kite and earned 24 xp.
Mimitchi has now 24 xp, and 8.78 energy.

Please enter the next command:
feed

Nom nom nom
Mimitchi has now 25 xp, and 9.18 energy.

Please enter the next command:
feed

Nom nom nom
Mimitchi has now 28 xp, and 9.32 energy.

Please enter the next command:
feed

This tamagotchi cannot eat anymore!

Please enter the next command:
attack

Command not recognized.

Please enter the next command:
sleep

Mimitchi is going to take a nap! (-.-)Zzz...
```

Second output:

```
Welcome to our last game of the semester!

Please enter the name of the Tamagotchi you'd like to play with: Mimitchi

Your Tamagotchi has been loaded.

      oooooo      oooooo
     o      o    o      o
     o      o    o      o
     o      o    o      o
     o ooooo oooo ooooo o
    o                    o
   o     ^          ^     o
  o     / \        / \     o
  o                        o
  o                        o
   o       ________       o
    oo                  oo
      ooo            ooo
         oo oooooo oo


Tamagotchi card
Name:           Mimitchi
Level:          1
Energy:         10.00
XP:             28
Meals:          2
Toys:   [Fisher the purple kite]

Enter a command (play, feed, or sleep):
play

Choose one of the following options:
1) Mimitchi plays with their best toy
2) Mimitchi plays with a random toy
2

Mimitchi played with Fisher the purple kite and earned 24 xp.
Mimitchi has now 52 xp, and 8.52 energy.

*** YAY, time to level up!! ***
Mimitchi is now level 2
Your new toy is Garfield the orange teddy bear

Please enter the next command:
play

Choose one of the following options:
1) Mimitchi plays with their best toy
2) Mimitchi plays with a random toy
1

Mimitchi played with Fisher the purple kite and earned 24 xp.
Mimitchi has now 76 xp, and 6.99 energy.

Please enter the next command:
play
```

```
Choose one of the following options:
1) Mimitchi plays with their best toy
2) Mimitchi plays with a random toy
2

Mimitchi played with Fisher the purple kite and earned 24 xp.
Mimitchi has now 100 xp, and 5.54 energy.

Please enter the next command:
play

Choose one of the following options:
1) Mimitchi plays with their best toy
2) Mimitchi plays with a random toy
2

Mimitchi played with Fisher the purple kite and earned 24 xp.
Mimitchi has now 124 xp, and 4.01 energy.

Please enter the next command:
play

Choose one of the following options:
1) Mimitchi plays with their best toy
2) Mimitchi plays with a random toy
2

Mimitchi played with Fisher the purple kite and earned 24 xp.
Mimitchi has now 148 xp, and 2.75 energy.

Please enter the next command:
play

Choose one of the following options:
1) Mimitchi plays with their best toy
2) Mimitchi plays with a random toy
2

Mimitchi played with Garfield the orange teddy bear and earned 23 xp.
Mimitchi has now 171 xp, and 1.51 energy.

*** YAY, time to level up!! ***
Mimitchi is now level 3
Your new toy is Chuchu the purple car

Please enter the next command:
play

Choose one of the following options:
1) Mimitchi plays with their best toy
2) Mimitchi plays with a random toy
1

This tamagotchi cannot play

Please enter the next command:
sleep

Mimitchi is going to take a nap! (-.-)Zzz...
```

What To Submit

Please put all your files in a folder called Assignment5. Zip the folder (DO NOT RAR it) and submit it in MyCourses. 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.

`Toy.java`
`Tamagotchi.java`
`FileIO.java`
`Playground.java` (optional)
`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.