ASSIGNMENT 5

COMP-202, Winter 2018, All Sections

Due: Monday, April 16^{th}, (23:59)

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

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, marks will be removed
- 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`. Print out the length of the `ArrayList`.

Warm-up Question 3 (0 points)

Create a new method from your program which takes your `ArrayList` of `Strings`, and writes it to a file. Use the `FileWriter` and `BufferedWriter` classes 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 4 (0 points)

Create a new method from your program which takes your `ArrayList` of `Strings`, and a `String` parameter. Search for the `String` in the `ArrayList`, ignoring the case of the `Strings`. If the `String` is in the `ArrayList`, return the index of the `String`. Otherwise, return -1.

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 its `toString` method, as well as a for-loop. Can you make the for-loop version look like the built-in `ArrayList toString` method?

[1]The documentation for 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

We *strongly recommend* that you complete all the warm-up questions before starting this problem.
The questions in this part of the assignment will be graded.

These three questions will guide you to create a space game where your ship travels through a solar system to search for artifacts left behind by aliens. Note that these steps refer to the same Java classes. That is, **you will only hand in your finished classes**, not the intermediate steps.

Question 1: Space Game (50 points)

For this first question, you will create the code for a spaceship, planets, and have a game loop to move around the solar system. Note that your code for this question 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 want.

(a) Planet

Write a class `Planet`. A `Planet` has the following *private* attributes:

- A `String` name
- A `double` chance of finding an alien artifact on this planet during a search
  - Note that this chance is a number between 0.0 and 1.0
- A `double` representing the damage possible when searching for an artifact
  - For example, Venus' acid rain can give up to 200 points of damage

The `Planet` class also contains the following *public methods*. You must decide if they are static or non-static:

- A constructor that takes as input the name, chance of success for artifact finding, and the potential amount of damage for artifact searching
  - This constructor must set the values for the class attributes using the parameter values
  - An `IllegalArgumentException` must be thrown if the chance of artifact success is less than zero or greater than one, or if the damage is less than 0.
- `getName` which returns the name of the planet
- A `toString` method. The `String` which is returned must contain the name, the success chance as a percentage from 0 to 100 (so a chance of 0.5 is reported as 50.0%), and the potential damage
  - For example: `Name: Venus Artifact Chance: 60.0% Possible Damage: 200.0`
- A `findPlanet` method. This method takes a `String` and an `ArrayList` of `Planets`. This method must search the `ArrayList` to find the planet with a name matching the `String` parameter (ignoring the case). The index of the `Planet` must be returned as an `int` if the `Planet` is found. Otherwise, return -1.
  - For example, if the "sol.txt" solar system is loaded into the `ArrayList planets`, then `findPlanet(''Venus'', planets)` must return 1.

(b) Spaceship Class

The `Spaceship` class represents a spaceship travelling through to different planets in the solar system.

The *Spaceship* class must contain the following (private) attributes:

- A `String` name
- A `double` current health value
- A `double` maximum health value
- An `int` for the number of artifacts this spaceship has discovered
- An `int` number of wins in the space game
- A `Planet` which is where the spaceship is (the current planet)
- A `static ArrayList` of `Planets`
  - This will store the list of planets in the solar system

Here are some of the public methods required in this class. You must decide if these methods are static or non-static. Note that you will also have to add getters and setters for attributes as needed.

1) A constructor

The constructor for the `Spaceship` class takes one `String`, one `double`, and one `int` as input. These parameters represent the name, maximum health, and number of wins for the spaceship, in that order.

Set the attributes of the spaceship using the parameters of the constructor. The current health of a new spaceship is the same as the maximum health.

2) The `toString` method

This method must return a `String` consisting of the spaceship's name, current health, and number of alien artifacts found. Format the `String` in any way you want. This method will be very handy for debugging your code, and will be used during the space game to keep track of the status of each spaceship.

3) The `setPlanets` method

This method sets the `planets` attribute. This method takes an `ArrayList` of planets as input.

Remember that `ArrayList` is a mutable reference type and as such you should make a copy of the input `ArrayList` when initializing the attribute. That is, you cannot just set the attribute to be the parameter, as they will both point to the same data in memory.

After the list is loaded, loop through the `ArrayList` and print out the details of every planet. Hint: Use the `toString` method of the `Planet` class

3) `moveTo` method

This method will move the spaceship to a different planet. This method has only a `String` parameter, which is the name of the planet to move to. This method does not return a value.

This method must call the `findPlanet` method from the `Planet` class, which will return the index of the planet to move to.

If the planet wasn't found, print a message that the spaceship could not move to that planet name. For example, *The FES Hawking tried to move to Krypton, but that planet isn't in this solar system!*.

Otherwise, move the spaceship by setting the `currPlanet` attribute. Also print a message about moving. For example, *The FES Hawking moved to Mars.*

4) `moveIn` method

The method will move the spaceship closer into the solar system (closer to the sun). For example, calling `moveIn` when a spaceship is at Earth will move it to Venus. This method takes no parameters and returns no value.

You must find the index of the current planet by calling the `findPlanet` method. After changing the index, call the `moveTo` method with the new planet's name.

If the spaceship is already at the first planet in the planet list, print a message. For example, *FES Hawking couldn't move in. No planet is closer in.*

5) `moveOut method`

This method is the opposite of the `moveIn` method. For example, calling `moveOut` when a spaceship is at Earth will move it to Mars.

The instructions for the `moveOut method` are the same as for the `moveIn` method. Make sure to reverse how you get the index of the new planet, and modify the error message of this method to say no planet is farther out.

6) `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 spaceship wins the space game.

(c) FileIO

`FileIO.java` contains the following two methods for the first question. You must figure out if the methods are static or non-static.

1) A method `loadSpaceship`

This method takes as input a filename as a `String` parameter, and returns a new `Spaceship`, using the constructor defined in the `Spaceship` class.

The `loadSpaceship` 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 error message that the file was not found, or that there was another error. Note that the catch block for the `FileNotFoundException` must be above the catch block for the `IOException` to this to work properly.

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

- Name of the spaceship
- Maximum health
- Number of wins so far in the space game

Use these values from the file to create and return a new `Spaceship`. Do not use the `Scanner` class.

2) The `loadPlanets` method.

This method takes a filename of planets to read, and returns an `ArrayList` of `Planets`. An example of a file containing planets is found in the provided files as `sol.txt`.

Read the file indicated by the method parameter `filename`, by using a `FileReader` and a `BufferedReader` to open the file specified by `filename`. Do not use the `Scanner` class. 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 error message that the file was not found, or that there was another error. Note that the catch block for the `FileNotFoundException` must be above the catch block for the `IOException` for this to work properly.

The planets are defined on each line, consisting of the planet name, the chance to find an artifact, and the possible amount of damage. Hint: Use the `split` method from the `String` class.

Use the three values on each line of the file to create new `Planet` instances, and then add these `Planet` instances to the new `ArrayList` of `Planets` which is returned from this method.

(d) SpaceGame

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

The *SpaceGame* class must contain the following attributes:

- A non-static `Scanner` attribute for getting input from the user
- A non-static `Spaceship` which is the player
- A `static final int NUM_ARTIFACTS_WIN` = 5
  - This is a *constant*, which stores the number of artifacts a ship must find to win

The `SpaceGame` class must contain the following methods. You are allowed to create as many other `private` methods as you want.

1) The `SpaceGame` constructor.

This constructor takes one parameter, which is the filename of the planets in the solar system.

This constructor must:

- Initialize the `Scanner` attribute to take input from the user
- Initialize the player `Spaceship`
  - An example player spaceship is given in "player.txt" in the provided files
- Use the method `loadPlanets` method from the `FileIO` class to load a file containing planets
- Use the `setPlanets` method from the `Spaceship` class to set the planets for all `Spaceships`
- Move the player to the first `Planet` in the list (index 0) by using the `moveTo` method
- Print out a welcome message to the game, which informs the player how many artifacts they must find

2) The `private checkForDestroyed` method

This method returns 1 if the player's health is equal to or below zero. Return 0 otherwise

3) The `private checkForWin` method

This method returns 1 if the player's number of artifacts found is equal to or greater than the `NUM_ARTIFACTS_WIN` constant. Return 0 otherwise

4) The `public playGame` method

- Loop while the player is still alive and has not found enough artifacts, using the `checkForDestroyed` and `checkForWin` methods
  - Ask the user for a command using the `nextLine` method of the `Scanner` instance
  - Choose a command based on the following choices, ignoring the case
  - If the input is *moveIn*:
    - Call the `moveIn` method on the player spaceship
  - If the input is *moveOut*:
    - Call the `moveOut` method on the player spaceship

  - If the input is *moveto*
    - Ask the user for a destination and get a line of input from the user
    - Call the `moveTo` method on the player's spaceship with the input line
  - If the input is anything else
    - Print a message that the input was not recognized
- When the above loop stops, print an appropriate message saying if their ship was destroyed, or they found enough artifacts.

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 attributes still appear.

```
Welcome to the SpaceGame!
Loaded solar system from sol.txt:
Name:  Mercury Artifact Chance:  20.0% Possible Damage:  100.0
Name:  Venus Artifact Chance:  60.0% Possible Damage:  200.0
Name:  Earth Artifact Chance:  0.0% Possible Damage:  0.0
Name:  Mars Artifact Chance:  20.0% Possible Damage:  200.0
Name:  Jupiter Artifact Chance:  90.0% Possible Damage:  400.0
Name:  Saturn Artifact Chance:  85.0% Possible Damage:  350.0
Name:  Uranus Artifact Chance:  70.0% Possible Damage:  200.0
Name:  Neptune Artifact Chance:  50.0% Possible Damage:  100.0
Name:  Pluto Artifact Chance:  1.0% Possible Damage:  10.0
Spaceship FES Hawking moved to Mercury

You must find 5 artifacts to win!

Enter a command:
moveout
Spaceship FES Hawking moved to Venus

Enter a command:
moveto
Enter the destination:
mars
Spaceship FES Hawking moved to Mars

Enter a command:
movein
Spaceship FES Hawking moved to Earth

Enter a command:
explode
Command not recognized:  explode
```

Question 2: Searching for Artifacts (20 points)

For this question, you will modify the above classes in order to add searching for alien artifacts.

Make sure that your code works for the first question before you try this question.

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

(a) Planet

Add two methods to the `Planet` class to handle searching for artifacts on a `Planet`.

- A `searchForArtifact` method which returns `true` or `false` whether an artifact was found. First, a random number from 0 to 1 is generated. Return true if the random number is below the chance to find an artifact, otherwise false. The chance to find an artifact is an attribute of the `Planet`.
  - For example, if the chance to find an artifact is 0, then it is impossible to generate a random number below this, and artifact searching will always fail
  - Use the `Random` class to generate the random numbers, and do not use a seed.
- A `getDamageTaken` method which returns the amount of damage taken by a spaceship when searching for an artifact (corrosive gases, running into terrain, etc.)
  - Generate a random number from 0 to 1, and multiply that number by the possible damage which is an attribute of the `Planet`. Return this value as a `double`
  - Use the `Random` class, and do not use a seed.
  - For example, the possible damage for searching Jupiter is from 0 to 400

(b) Spaceship

Modify the `Spaceship` class to add a method to search for an artifact.

This method must be named `doSearch`. It will take no parameters and return no value.

- Call the `searchForArtifact` method on the current planet and store the value
- If the value is true, print a message saying the spaceship found an artifact (including the spaceship's name), and increase the number of artifacts the spaceship has found
- If the value is false, print a message saying the spaceship did not find an artifact (including the spaceship's name)
- Call the `getDamageTaken` method on the current planet to find out how much damage the spaceship took while performing the search
- Print a message saying that the spaceship took that much damage
  - You can use this code to better format this message:
    ```
    String damageStr = String.format("%1$.2f", damageTaken);
    ```
- Subtract the damage from the current health of the spaceship
- If the current health is below zero, print a message that the spaceship explodes

(c) SpaceGame

Change the `playGame` method from the `SpaceGame` class to allow the player to search planets.

As well as the commands *moveto*, *movein*, and *moveout*, add the new command *search*. When the user uses this command, call the `doSearch` method on the player's spaceship

Question 3: Adding the Artificial Intelligence (15 points)

In this question, we'll add an artificial intelligence that makes random decisions.

We'll only need to change the `SpaceGame` class.

- Add an attribute of type `Spaceship` which represents the enemy spaceship to the `SpaceGame` class
- Initialize the enemy spaceship attribute in the `SpaceGame` constructor

  - Use the "enemy.txt" file as the filename
- Also in the constructor, call `moveTo` on the enemy spaceship, and move them to the last planet in the planets `ArrayList`

Change the game-ending conditions so that the player loses if the enemy spaceship collects enough artifacts, and the player wins if the enemy spaceship is destroyed.

- In the `checkForWin` method, return 2 if the enemy found enough artifacts
- In the `checkForDestroyed` method, return 2 if the enemy's current health is below zero

In the `playGame` method, make sure to detect in which cases the player wins and loses, and print appropriate messages. Also call the `increaseWins` method on the enemy spaceship if the enemy wins.

Finally, we'll create the random artificial intelligence to control the enemy spaceship. We'll add some code to the main loop in the `playGame` class, right underneath where the user's commands are executed.

- Generate a random number from 0 to 3 (not including)
- If the number is 0, call the `doSearch` method on the enemy spaceship
- If the number is 1, call the `moveIn` method on the enemy spaceship
- If the number is 2, call the `moveOut` method on the enemy spaceship

Question 4: 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 `saveSpaceship`. This method takes as input a `Spaceship` to write, and a `String` which is the filename to write to.

Within the `saveSpaceship` method, use `FileWriter` and `BufferedWriter` instances to write the spaceship's information back to a file. Do not catch the `IOException` that can occur when writing a file. Instead, add `throws IOException` to the header of the `saveSpaceship` method. You will have to catch this exception when the `saveSpaceship` method is called, and print an appropriate error message with the filename included.

The format of the file that is saved must match the format that is expected when a spaceship is loaded. That is, you should be able to save a spaceship to a file, and then load a spaceship from that same file.

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

- Name of the character
- Maximum health
- Number of wins so far in the space game

You may need to write getter methods for these attributes in the `Spaceship` class.

(b) SpaceGame

In the `playGame` method, after you have increased the wins for either the player or the enemy, print how many wins each spaceship has.

As well, you must save the spaceships to the files you loaded them from. For example, save the player spaceship to `player.txt` and the enemy character to `enemy.txt`.

To save these spaceships, call the `saveSpaceship` method from the `FileIO` class, and pass the spaceship and filename as parameters. This will save the number of wins for that spaceship, so that after playing the space game multiple times, you will know whether the player or the enemy 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**.

```
Welcome to the SpaceGame!
Loaded solar system from sol.txt:
Name:  Mercury Artifact Chance:  20.0% Possible Damage:  100.0
Name:  Venus Artifact Chance:  60.0% Possible Damage:  200.0
Name:  Earth Artifact Chance:  0.0% Possible Damage:  0.0
Name:  Mars Artifact Chance:  20.0% Possible Damage:  200.0
Name:  Jupiter Artifact Chance:  90.0% Possible Damage:  400.0
Name:  Saturn Artifact Chance:  85.0% Possible Damage:  350.0
Name:  Uranus Artifact Chance:  70.0% Possible Damage:  200.0
Name:  Neptune Artifact Chance:  50.0% Possible Damage:  100.0
Name:  Pluto Artifact Chance:  1.0% Possible Damage:  10.0
Spaceship FES Hawking moved to Mercury
Spaceship PSS Fenrir moved to Pluto

You must find 5 artifacts to win!

Enter a command:
search
FES Hawking failed to find an artifact!
FES Hawking took 64.87 damage while searching for an artifact on Mercury!
Name:  FES Hawking Health:  435.13 Num.  Artifacts:  0

PSS Fenrir failed to find an artifact!
PSS Fenrir took 6.92 damage while searching for an artifact on Pluto!
Name:  PSS Fenrir Health:  613.08 Num.  Artifacts:  0
Enter a command:
search
FES Hawking failed to find an artifact!
FES Hawking took 23.85 damage while searching for an artifact on Mercury!
Name:  FES Hawking Health:  411.27 Num.  Artifacts:  0

PSS Fenrir couldn't move.  No planets farther out. Enter a command:
moveout Spaceship FES Hawking moved to Venus

PSS Fenrir failed to find an artifact!
PSS Fenrir took 1.82 damage while searching for an artifact on Pluto!
Name:  PSS Fenrir Health:  611.26 Num.  Artifacts:  0
Enter a command:
search
FES Hawking failed to find an artifact!
FES Hawking took 146.11 damage while searching for an artifact on Venus!
Name:  FES Hawking Health:  265.17 Num.  Artifacts:  0

PSS Fenrir couldn't move.  No planets farther out.
Enter a command:
search
```

```
FES Hawking found an artifact!
FES Hawking took 174.45 damage while searching for an artifact on Venus!
Name:  FES Hawking Health:  90.72 Num.  Artifacts:  1

Spaceship PSS Fenrir moved to Neptune
Enter a command:
search
FES Hawking found an artifact!
FES Hawking took 73.30 damage while searching for an artifact on Venus!
Name:  FES Hawking Health:  17.42 Num.  Artifacts:  2

Spaceship PSS Fenrir moved to Pluto
Enter a command:
moveto
Enter the destination:
jupiter
Spaceship FES Hawking moved to Jupiter

PSS Fenrir couldn't move.  No planets farther out.
Enter a command:
search
FES Hawking found an artifact!
FES Hawking took 104.71 damage while searching for an artifact on Jupiter!
Name:  FES Hawking Health:  -87.29 Num.  Artifacts:  3
FES Hawking exploded!

Spaceship PSS Fenrir moved to Neptune
Oh no!  You lost!
PSS Fenrir has won:  5 times
Successfully wrote to file:  player.txt
Successfully wrote to file:  enemy.txt
```

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. If you do not know how to zip files, please ask any search engine or friends. Google will be your best friend with this, and a lot of different little problems as well.

Instead your zipped folder, there must be the following files. **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, 2, or 3, but only the finished files from Question 4.

`SpaceGame.java`
`Spaceship.java`
`Planet.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.

Question 1			
	Question 1a:	10	points
	Question 1b:	20	points
	Question 1c:	10	points
	Question 1d:	15	points
	Question 1 Total:	55	points
Question 2			
	Question 2a:	10	points
	Question 2b:	10	points
	Question 2c:	5	points
	Question 2 Total:	25	points
Question 3			
	Question 3 Total:	10	points
Question 4			
	Question 4a:	5	points
	Question 4b:	5	points
	Question 4 Total:	10	points
		100	**points**