ECE 422C Programming Assignment #2
Attack of the Aliens!

Due Date: Monday, February 10th, 11:59 pm CST

IMPORTANT: You may NOT import any additional packages to complete this assignment other than those already imported in the starter code. Any unnecessary imports may result in failure to compile on Gradescope.

PLEASE START EARLY!!!

Coding Style

For this programming assignment, we will be enforcing the ECE 422C Java Coding Guidelines as found on the Canvas homepage. Please ensure to have *COMPLETE* file headers, class headers, and method headers, to use descriptive variable names and proper indentation, and to avoid using magic numbers.

Getting started with the starter code

1. Make sure there is no problem with your Java coding environment. If there are any problems, then review Assignment 1, or come to the office/lab hours before you start Assignment 2.

2. Download the starter code.
    a. If you are working on your local machine, then you can download the starter code from Canvas. Download the starter code to a directory of your choice, then extract the zip file. It should contain 8 Java files: `Alien.java, Aliens.java, Game.java, GameCharacter.java, GameLoader.java, Item.java, Ripley.java, and Room.java`. There are also three input test files (input1.txt, input2.txt, and input3.txt) and 3 corresponding output files for you to match (output[123].txt). Afterwards, open your terminal or command prompt, then navigate to the directory that contains those Java files.

3. Compile the starter code with:

```
javac *.java
```

4. Run with one of the sample input files provided:

```
java Aliens input1.txt
```

5. You will get the message “Game not implemented yet!”. That’s your cue to get coding!

Overview

In this assignment, you will implement the gameplay demonstration of a simple role-playing game (RPG). In the game, which follows the plot of the classic 1979 horror movie Alien, we have the main character Ripley trying to navigate through the rooms of her spaceship. In each room, there is an alien that is guarding an item. Ripley traverses from the first room to the last room, fighting an alien at each level. Once an alien is defeated, Ripley will pick up the item in the room to enhance her attributes. Beyond their name, both Ripley and the aliens share three attributes: `health`, `attack`, and `speed`. Additionally, the aliens have a `speedDamage` attribute. Ripley and the aliens will take turns attacking each other. The `attack` attribute indicates the amount of damage Ripley or an alien will cause to their opponent (i.e., drop their opponent’s `health`) in each turn. One of them is defeated once their `health` drops to `0` (or less). `speed` determines which side attacks first (higher `speed` attacks first) when Ripley first enters a room. Ripley wins when the alien in the last room is defeated. Whenever Ripley gets killed, it’s game over.

IMPLEMENTATION TIP: You should NOT change any class field or method signatures in the starter code with the exception of the declaration of the `Alien class`, the `Ripley class`, and the `GameCharacter class`. These you may want to explore the relationship between them and figure out how to best apply the OO principles we have discussed. You should not add any files, remove any files, or alter the attributes of any field (e.g., you may not make something `public` that we have marked as `private`). Look through the starter code and search for the word “FILL” to see where you need to add code.

Format of Input File and Game Play

The code reads an initialization file to set up the game. The format of this file is as follows (see the provided “input[123].txt” file for an example):

1. The first line contains the `name` of our intrepid alien fighter, the beginning number of `health` points, the `attack` value for the fighter, and their `speed`. Assume `health`, `attack`, and `speed` will always be positive integers, and the `name` can be any valid string (we will stick with Ripley for this document but it should be able to be anything).
2.
    a. For instance: Ripley, 1000, 10, 20

3. Afterward there may be any number of lines describing rooms in the spaceship. The format for these lines is the room `name`; the `name` of the alien in the room; the alien's beginning `health`; the alien's `attack` value; the alien's `speed`; the alien's `speedDamage`; the `name` of the `item` in the room; and finally, three integers that describe the `health`, `attack`, and `speed` that the `item` adds to the fighter should the alien be defeated. Again, all integers are positive or 0.

    a. For instance: Hallway,Xenomorph,20,6,40,8,Blaster,0,4,5

4. The game goes through each room in the order specified in the file. Whoever has the higher `speed` value between Ripley and the alien in the room gets the first blow. When a blow lands on an alien, its `health` is reduced by the amount of the `attack` value of Ripley. When Ripley takes a blow, her `health` is reduced by the alien's `attack` value AND her `speed` is reduced by the alien's `speedDamage` (after all, you move slower the more injured you are!) Then the opposing fighter goes, and this cycles until either the alien or Ripley has died (e.g., their `health` value is less than or equal to 0).

5. You can assume an alien's `health` only goes down.

6. The game ends when Ripley has defeated all the aliens, or Ripley has been killed somewhere along the way.

7. We have provided 3 different input files for you to test your code against, as well as the corresponding output files that would indicate a correct answer. Your code must produce EXACTLY the same output as the test cases. We will be testing more than the three provided, so you are strongly encouraged to write your own test code to catch any corner cases.

Thing to Keep in Mind

1. You should not change either `Aliens.java` or `GameLoader.java`.

2. You may not `import` any additional packages beyond those we have already put at the top of each file.

3. Figure out your plan before you sit down to start coding.

a. Design your UML diagrams for each of the 8 classes corresponding to the java files we have provided and determine what methods each will required.
b. Be on the lookout for places where code can be reused through inheritance or polymorphism.
c. Write some test cases for your methods before you implement the methods

4. Remember you are not allowed to change the public/private attributes for any of the signatures we have provided in the starter code. All of the class level fields are marked `private` and must remain that way in your code to follow good OOP practices.

5. Ensure your output precisely matches ours for the three test cases provided.

Submission

VERY IMPORTANT: Please follow the instructions below carefully and make the exact submission format.

1. Go to Gradescope via Canvas and click on *Programming Assignment 2, Alien Attack!*
2. In a file named *pa2.zip*, include the following files to upload to Gradescope:
    a The 8 java files with your implementation: `Alien.java, Aliens.java, Game.java, GameCharacter.java, GameLoader.java, Item.java, Ripley.java, and Room.java`
3. Upload a file named *UML.pdf* that contains the class diagrams for each of the classes you have used on Canvas (5% of your grade)
4. Upload a ZIP file of a subdirectory you generate via *javadoc* the description of your work as generated by the *javadoc* tool. We should be able to navigate into the subdirectory after unzipping your submission and open the file *index.html* to read all about your work. (5% of your grade)
5. **You can resubmit an unlimited number of times before the due date.** Your score will depend on your final submission, even if your former submissions have a higher score.
6. The autograder is for grading your uploaded files automatically. Make sure your code can compile on Gradescope. Keep in mind your project **MUST WORK WITH JAVA VERSION 11** as that is the version in the provisioned Docker instance running the autograder will use.

    NOTE: The Gradescope Autograder you see is a minimal autograder. For this assignment, it will only show the compilation results and the results of a few testers. After the assignment deadline, a thorough Autograder will be used to determine the final

grade of the assignment. **Thus, to ensure that you would receive full points from the thorough Autograder, it is your job to extensively test your code for correctness and make sure you have followed the coding guidelines.**

Here is the breakdown for final points awarded for this assignment:

1. Following the coding style guidelines → 10 points
2. Correct game play → 80 points
3. Javadoc documentation → 5 points
4. Correct UML descriptions of all your classes → 5 points