In a Galaxy Far, Far Away...

1 Introduction

This assignment will cover inheritance and polymorphism. Make sure you have a solid understanding of the previous homework and the related concepts. Unlike the last homework, you will not get charts detailing each class, but rather descriptions of the required classes.

2 Problem Description

The year is 4 ABY and the galaxy is in the middle of a civil war. It's us, the Alliance to Restore the Republic (a.k.a. the Rebellion), against the Galactic Empire.

A couple years after the Battle at Yavin IV, we started building a simulator in order to better predict the outcomes of battles with the Empire. Unfortunately, it was being developed at our base on Hoth, and as I am sure you can guess, we lost pretty much everything last year when the Empire attacked and wiped us off the planet.

Luckily, we managed to save a couple parts of the program. Specifically, we saved the core `Simulation` code and the `Soldier` class, so you won't have to worry about writing them all over again. The `Soldier` class is the basic type for all units in the `Simulation`. The downside is we can't use this directly; we need you to make all of the concrete classes that the `Simulation` will use.

These are the requirements for all the different classes you need to make to get the `Simulation` up and running so we can defeat the Empire for good.

- As an overview, a `Soldier` is the basic type that will be used to make the other classes. `Soldiers` have the following attributes: health, attack, defense, and an identifier. Health can start within the range [30.0, 100.0] and cannot be raised above 100.0 but can be lowered to 0.0, at which point the Soldier is considered dead. Attack and defense can start within the range [0.0, 100.0] and cannot be raised or lowered to a value outside this range. The identifier is just a `String` generated by the `Simulation` that generally follows the pattern "XX-00".

- A `RebelSoldier` is a `Soldier` who is the common infantry of our army. Unlike the `Stormtroopers`, they have a 80% chance of dealing damage, but dont get any attack bonus. `RebelSoldiers` give their names as “Rebel Soldier <identifier>”.

- A `Stormtrooper` is a `Soldier` who is the basic grunt of the Empire. Luckily for us, their attacks have only a 60% change of dealing any damage. Unfortunately when they do deal damage, they get a 25% damage bonus. A `Stormtrooper` gives his name as “Stormtrooper <identifier>”.

- We need an interface, `ForceSensitive`, which has one method `useTheForce()` which takes no parameters and returns nothing for those who can wield the force. This method may be called on all `ForceSensitives` by the `Simulation` *before* attacks are made.

- A `Jedi` is a `ForceSensitive Soldier` who has a power value that is equal to a quarter of his defense value. When a `Jedi` uses the Force, he heals himself a quantity equal to $\frac{1}{4}$ of his power and boosts his defense by a quantity equal to his power. A `Jedi` will always succeed when attacking, does not receive a bonus to attack, and if the `Jedi` has used the Force since his last attack, his defense drops by a quantity equal to his power after he attacks. `Jedis` give their names as “Jedi <identifier>”.

- A `Sith` is a `ForceSensitive Soldier` who has a power value that is equal to $\frac{1}{6}$ the sum of his attack and defense values. When a `Sith` uses the Force, he hurts himself an amount equal to $\frac{1}{10}$ his power and boosts his attack by an amount equal to his power. Like a `Jedi`, a `Sith` will always succeed at dealing damage and receives no attack bonus. Additionally, if the `Sith` has used the Force since his last attack, his attack drops by a quantity equal to his power after he attacks. `Sith` give their names as “Sith <identifier>”.

- We need an additional `Rebel` who is a `RebelSoldier` that can attack however you like but has the following requirement: the constructor for this class must take in the same parameters in the same order as the `Soldiers` constructor. Your custom `RebelSoldier` must give his name in the same format as the `Soldiers` above.

- We also need an additional `Trooper` who is a `Stormtrooper` that can attack however you like but has the following requirement: the constructor for this class must take in the same parameters in the same order as the `Trooper`’s constructor. Your custom `Stormtrooper` must give his name in the same format as the `Soldiers` above.

- Finally, we need a class to run the `Simulation`. The runner should have a welcome prompt. This runner should prompt for the number of `RebelSoldiers`, `Jedi`, `Stormtroopers`, and `Sith`, the user would like to use in the `Simulation`. It should then make a simulation with those values and pass in an example object for each of your custom `Soldiers`. You should prompt the user to press enter to start the `Simulation` and wait until this is done before simulating a skirmish (that is a round of attacking). A skirmish is simulated by calling the `simulateSkirmish` method in `Simulation`. This method should always be called with a parameter of `false` (See the tips section for an explanation). You should pause between skirmishes, waiting for the user to press enter for another skirmish. If the `simulateSkirmish` method returns false, the simulation is over and you should determine which faction won the battle. In the case that one faction has no troops remaining, the other side is the clear victor, in the case that the simulation ends but both factions still have troops remaining, the faction with the most troops is considered to have won by “routing” the other faction. This should be printed to the screen. Additionally final troop counts should be displayed below the declaration of the winner. The name of this file must be `Battlefront.java`.

3 Tips

- In your main method in `Battlefront`, only instantiate one `Scanner` object.
- Use the `javadoc` command and read the Java Docs for the `Soldier` and `Simulation` classes and familiarize yourself with the various methods you will need to call.
- Be careful not to alter the `Soldier` and `Simulation` classes as doing so could have unintended consequences. Remember, we will be using the original versions to check your code!
- When you call `simulateSkirmish`, call it with a parameter of false. Calling it with true produces a large amount of cool extra output, but this is for debugging and entertainment only. Your final submission should have this extra output turned off!
- Han shot first!
- When you call the constructor for `Simulation`, be sure to pass one of your custom `RebelSoldiers` as the fifth parameter and one of your custom `Stormtroopers` as the sixth parameter where the values for each of these are the *average* values for each attribute (health, attack, and defense). The identifier given to these two objects does not matter.
- The constructors for all `Soldiers` you write should take no extra parameters over the ones taken in by `Soldier`.
- Remember `RebelSoldiers` and `Jedi` are part of the Rebellion while `Stormtroopers` and `Sith` are part of the Empire.
- Feel free to look around the `Simulate` class. A lot of what is used in that class will be introduced to you in the near future, though some is beyond the scope of this course.

4 Example Output

```
$ java Battlefront
+------------------------------------------+
| Welcome to the Battlefront1331 Simulator! |
+------------------------------------------+
How many Rebel Soldiers would you like to include?
100
How many Jedi would you like to include?
2
How many Stormtroopers would you like to include?
100
How many Sith would you like to include?
2
Press enter to begin the simulation

+---------------------------------------------------------------+
|                 Skirmish Complete                             |
+---------------------------------------------------------------+
| 17 Soldiers of the Rebellion died this skirmish! 85 remain. |
| 18 Soldiers of the Empire died this skirmish! 84 remain. |
+---------------------------------------------------------------+


Press enter to continue the simulation

+---------------------------------------------------------------+
|                 Skirmish Complete                             |
+---------------------------------------------------------------+
| 25 Soldiers of the Rebellion died this skirmish! 60 remain. |
| 23 Soldiers of the Empire died this skirmish! 61 remain. |
+---------------------------------------------------------------+


Press enter to continue the simulation

+---------------------------------------------------------------+
|                 Skirmish Complete                             |
+---------------------------------------------------------------+
| 13 Soldiers of the Rebellion died this skirmish! 47 remain. |
| 14 Soldiers of the Empire died this skirmish! 47 remain. |
+---------------------------------------------------------------+


Press enter to continue the simulation

+---------------------------------------------------------------+
|                 Skirmish Complete                             |
+---------------------------------------------------------------+
| 10 Soldiers of the Rebellion died this skirmish! 37 remain. |
| 10 Soldiers of the Empire died this skirmish! 37 remain. |
+---------------------------------------------------------------+


Press enter to continue the simulation

+-------------------------------------------------------------+
|                Skirmish Complete                            |
+-------------------------------------------------------------+
| 5 Soldiers of the Rebellion died this skirmish! 32 remain. |
| 8 Soldiers of the Empire died this skirmish! 29 remain. |
+-------------------------------------------------------------+


Press enter to continue the simulation
```

```
+-------------------------------------------------------------+
|                Skirmish Complete                            |
+-------------------------------------------------------------+
| 7 Soldiers of the Rebellion died this skirmish! 25 remain. |
| 10 Soldiers of the Empire died this skirmish! 19 remain. |
+-------------------------------------------------------------+


Press enter to continue the simulation

+-------------------------------------------------------------+
|                Skirmish Complete                            |
+-------------------------------------------------------------+
| 1 Soldiers of the Rebellion died this skirmish! 24 remain. |
| 8 Soldiers of the Empire died this skirmish! 11 remain. |
+-------------------------------------------------------------+


Press enter to continue the simulation

+-------------------------------------------------------------+
|                Skirmish Complete                            |
+-------------------------------------------------------------+
| 2 Soldiers of the Rebellion died this skirmish! 22 remain. |
| 9 Soldiers of the Empire died this skirmish! 2 remain. |
+-------------------------------------------------------------+


Press enter to continue the simulation

+-------------------------------------------------------------+
|                Skirmish Complete                            |
+-------------------------------------------------------------+
| 1 Soldiers of the Rebellion died this skirmish! 21 remain. |
| 2 Soldiers of the Empire died this skirmish! 0 remain. |
+-------------------------------------------------------------+


Simulation Complete!
The Rebels won!
19 of 100 Rebel Soldiers remain!
2 of 2 Jedi remain!
0 of 100 Stormtroopers remain!
0 of 2 Sith remain!
```

Note: In the above example output, the sections of output that are in table format and say ”Skirmish Complete” are generated by the `Simulation` class. You should **not** try to print these yourself.

5 Javadocs

For this assignment you will be commenting your code with Javadocs. Javadocs are a clean and useful way to document your code's functionality. For more information on what Javadocs are and why they are awesome, the online documentation for them is very detailed and helpful.

You can generate the javadocs for your code using the command below, which will put all the files into a folder called javadoc:

```
$ javadoc *.java -d javadoc
```

The relevant tags that you need to have are `@author`, `@version`, `@param`, and `@return`. Here is an example of a properly Javadoc'd class:

```java
import java.util.Scanner;

/**
 * This class represents a Dog object.
 * @author George P. Burdell
 * @version 13.31
 */
public class Dog {

   /**
    * Creates an awesome dog (NOT a dawg!)
    */
   public Dog(){
      ...
   }

   /**
    * This method takes in two ints and returns their sum
    * @param a first number
    * @param b second number
    * @return sum of a and b
    */
   public int add(int a, int b){
      ...
   }
}
```

Take note of a few things:

1. Javadocs are begun with `/**` and ended with `*/`.
2. Every class you write must be Javadoc'd and the `@author` and `@version` tag included. The comments for a class start with a brief description of the role of the class in your program.
3. Every non-private method you write must be Javadoc'd and the `@param` tag included for every method parameter. The format for an `@param` tag is `@param <name of parameter as written in method header> <description of parameter>`. If the method has a non-void return type, include the `@return` tag which should have a simple description of what the method returns, semantically.

5.1 Javadoc and Checkstyle

You can use the Checkstyle jar mentioned in the following section to test your javadocs for completeness. Simply add -j to the checkstyle command, like this:

```
$ java -jar checkstyle-6.2.1.jar -j *.java
Audit done. Errors (potential points off):
0
```

6 Checkstyle

You must run checkstyle on your submission. The checkstyle cap for this assignment is **60** points. Review the Style Guide and download the Checkstyle jar. Run Checkstyle on your code like so:

```
$ java -jar checkstyle-6.2.1.jar *.java
Audit done. Errors (potential points off):
0
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the points we would take off.

The Java source files we provide contain no Checkstyle errors. For this assignment, there will be a maximum of **60** points lost due to Checkstyle errors (1 point per error). In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

7 Turn-in Procedure

Submit all of the Java source files you modified and resources your program requires to run to T-Square. Do not submit any compiled bytecode (`.class` files) or the Checkstyle jar file. When you're ready, double-check that you have submitted and not just saved a draft.

Please remember to run your code through Checkstyle!

Verify the Success of Your Submission to T-Square
Practice safe submission! Verify that your HW files were truly submitted correctly, the upload was successful, and that the files compile and run. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

1. After uploading the files to T-Square you should receive an email from T-Square listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.
2. After submitting the files to T-Square, return to the Assignment menu option and this homework. It should show the submitted files.
3. Download copies of your submitted files from the T-Square Assignment page placing them in a new folder.
4. Recompile and test those exact files.
5. This helps guard against a few things.
   (a) It helps insure that you turn in the correct files.

(b) It helps you realize if you omit a file or files. [1] (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)

(c) Helps find last minute causes of files not compiling and/or running.

[1] Missing files will not be given any credit, and non-compiling homework solutions will receive few to zero points. Also recall that late homework will not be accepted regardless of excuse. Treat the due date with respect. The real due date is midnight. Do not wait until the last minute!