CSCI-605 Advanced Object-Oriented Programming Concepts

Homework 4: Hero Storm

1 Introduction

You have been tasked by your company to develop the next big fantasy role playing game (RPG), Super Fantasy Hero Storm.

The game deals with the ongoing skirmish in the realm of The Feral Vales, between two ancient and opposing teams:

- **Dragons**: Giant monstrous flying reptiles who rule the skies and are characterized as ambitiously energetic leaders. The dragons' main weaknesses are arrogance and relentlessness. Their battle cry is, "Pride cometh before a fall!".
- **Lions**: Large gregarious predatory mammals who dominate on land and are characterized by their strength and power. The lions' main weaknesses are pride and a propensity for laziness. Their battle cry is, "Arrogance is knowledge minus wisdom!".

Each team is a party of three members, the heroes, each with their own unique characteristics and abilities:

- **Berserker**: A glass cannon character with high offensive power but weak defensive capabilities. When a berserker attacks an enemy, they inflict a large amount of damage to them. When attacked by an enemy, this hero must take all the damage inflicted by the enemy.
- **Healer**: A holy character and medical expert designed to restore allies health during combat. When a healer attacks an enemy, they first heal themselves and then all their non-defeated allies. After the healing operation, they inflict a small amount of damage to their enemy. When attacked by an enemy, this hero must take all the damage delivered.
- **Tank**: A heavily armored character with highly defensive capabilities, capable of absorbing large amounts of damage in combat. A tank attack inflicts a moderate amount of damage. However, when a tank is attacked they reduce the amount of actual damage they take by using their shield.

1.1 Goals

This homework helps students to gain experience working with:

- Inheritance
- Abstract Classes

- Factory Method Design Pattern
- Visibility modifiers: public vs protected vs private
- Method overriding
- Superclass method calling

1.2 Provided Files

1. `heroes` folder includes the starter code
2. `sample_run` folder includes sample runs showing the output of the program using different seeds.
3. The `test` folder includes JUnit tests that will help you develop and verify your heroes.
4. The `doc` folder includes the Javadoc documentation.

2 Game Play

The game is played by the two opposing teams of heroes, each of which is organized into a separate queue. Game play follows these steps until all heroes in one team have been defeated.

1. The two opposing heroes at the front of their respective queues engage in one on one combat.
2. The hero that attacks first is based on a toggle that flips each time there is a new engagement.
3. The first hero attacks their enemy based on their attacking characteristics.
4. The second hero takes the damage based on their defensive characteristics. If the damage kills the second hero, they fall and vanish from existence.
5. If the second hero has not fallen, they attack the first hero based on their attacking characteristics.
6. The first hero takes the damage based on their defensive characteristics.
7. If a hero does not fall, they re-enter the back of their team queue; otherwise they vanish from existence.

Using these rules, it is always guaranteed at the end that one team will be the victor and the other will be defeated.

3 Implementation

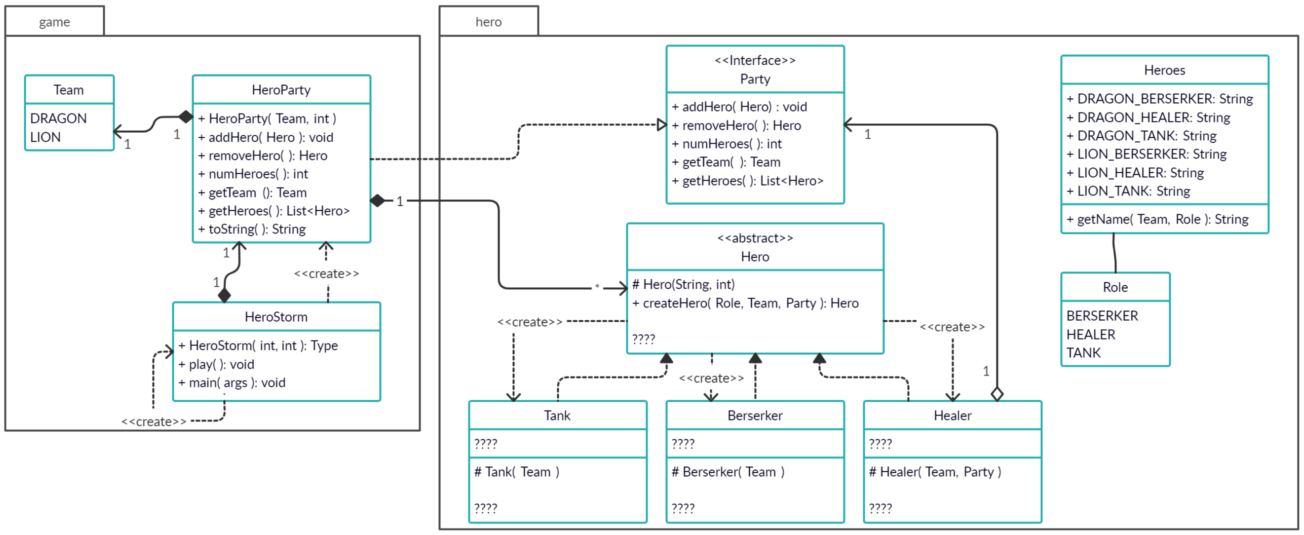
3.1 Design

We strongly suggest that you follow the supplied design for this homework. This class diagram details all the classes and their relationships.

The diagram and the documentation exclude the private state. They also omit the state and behaviors of the superclass Hero and all its subclasses. You are responsible for completing each of those classes using good inheritance design principles. For example, a data member shared by all subclasses should be defined in the shared superclass. While working on the design, try to answer the following questions:

1. What are the attributes common to all heroes?
2. What behaviors should all heroes be able to do?
3. What attributes and behaviors are unique to a particular hero?
4. Which heroes will need to customize the way they do something different from other heroes, and what are those customizations?

Note: You do not need to submit the answer to all the questions above. Those questions are meant to help you designing your program.



3.2 Heroes Characteristics

Here are the tables that list the constants that are used by the various heroes when attacking and taking damage.

Berserker		
Hit points	30	Initial hit points
Attack damage	20	Amount of damage this hero inflicts in an enemy

Healer		
Hit points	35	Initial hit points
Attack damage	10	Amount of damage this hero inflicts in an enemy
Heal amount	10	Amount to heal non-defeated allies (including self)

Tank		
Hit points	40	Initial hit points
Attack damage	15	Amount of damage this hero inflicts in an enemy
Defense	10%	Amount of damage this hero deflects with their shield when attacked

Assume the following:

- A hero may not exceed their maximum hit points, e.g. when being healed.
- When a hero's hit points drops to 0 or below, they are defeated and removed from the battle.

3.3 Standalone Output

The program runs on the command line like this:

```
$ java HeroStorm dragon_seed_# lion_seed_#
```

If the number of arguments are correct, they are guaranteed to be valid integers that will be used as separate seeds to the random number generator. The generators are used to initially shuffle the order of the heroes.

If the number of arguments is incorrect, display a usage message and exit the program, i.e.:

```
Usage: java HeroStorm dragon_seed_# lion_seed_#
```

After the seeds initialize the random number generators, the output of the game follows through each of the battles until eventually one team is victorious over the other. A typical battle will have the following output:

```
Battle #1
==========
DRAGON:
Dragon_Name1, ROLE, #/#
Dragon_Name2, ROLE, #/#
...

LION:
Lion_Name1, ROLE, #/#
Lion_Name2, ROLE, #/#
...

*** Hero_Name1 vs Hero_Name2

{... details of the individual battle...}
```

The `#/#` after each hero represents their current hit points followed by their maximum hit points.

Each battle you should display the current active party of heroes from Team Dragon, followed by Team Lion. Heroes who are defeated are no longer included in the party. The first and second hero for the battle are chosen by the toggle for the starting hero, which begins with Team Dragon.

If both heroes in combat are not healers, you will see a maximum of two messages that correlate to the attacks by each hero (only if the second hero survives the first heroes attack):

```
Hero_Name2 takes # damage
Hero_Name1 takes # damage
```

If the first hero defeats the second hero with their attack, the second hero falls and does not get to respond with an attack of their own. In this case the output would look like:

```
Hero_Name2 takes # damage
Hero_Name2 has fallen!
```

In the event a healer is the attacker, they first heal themselves, then their party, and finally they attack inflicting damage to their enemy, e.g.:

```
Healer_Name heals # points
Healer_Ally1 heals # points
Healer_Ally2 heals # points
Enemy_Name takes # damage
```

Keep in mind the healer will only heal the allies in their party that have not been defeated; the fallen are out of the game.

The game will end when one team has no more members in their party (they have all fallen). When this happens you should print the winning team and exit the program with a message like this:

```
Team {Dragon|Lion} wins!
```

For further detail, study the sample runs for complete outputs when running with different seeds, as well as the documentation for when and where the program should display the messages.

3.4 Seeding a Random Number Generator

When implementing HeroParty's constructor, use the static method `Collections.shuffle`. The first argument should be your collection of heroes and the second argument should be a new `Random` object that is seeded with the seed value for that party.

Please note, you must add the heroes to your collection in the order mentioned in the constructor or else your run will be different than the solutions:

1. Berserker
2. Healer
3. Tank

Use this table so that you can figure out which seed numbers map to which unique combinations of orderings of the three hero types after shuffling the collection.

Seed Value	Hero order (after shuffling)
0	Tank, Healer, Berserker
1	Healer, Tank, Berserker
2	Tank, Berserker, Healer
3	Berserker, Healer, Tank
5	Healer, Berserker, Tank
7	Berserker, Tank, Healer

In order to get the same results it is suggested you use either an ArrayList or a LinkedList for your collections of heroes in each party. For example:

```
Collections.shuffle(this.heroes, new Random(seed));
```

3.5 Factory Method Pattern

The `Hero` constructor is not accessible from the `game` package. A popular pattern in object-oriented programming for creating objects without worrying about the concrete class is the factory method. The `Hero` class provides a `public static` method, `create`. This allows a client to create a new Hero of any type without worrying about construction details, for example in HeroParty's constructor:

```
// create a berserker for this party
Hero hero = Hero.create(Role.BERSERKER, team, this);
```

From here, the `Hero.create` factory method would construct the berserker by calling the `Berserker` constructor to create an instance and return that as a `Hero` reference:

```
if (role == Role.BERSERKER) {
    return new Berserker(team);
}
```

3.6 Enum

When implementing the Java Enum, `game.Team` you do not need to implement the `valueOf` and `values` methods. Those are provided automatically when you implement an enum.

3.7 Sample Runs

Based on the seeds mentioned above, there are 36 distinct outcomes when playing the game using different combinations of heroes that take the hero orderings into effect.

You can find some of the sample outputs in the Provided files section. The format of the files indicates the seeds, for example the file 0-1.txt means Team Dragon has a seed of 0 and Team Lion has a seed of 1.

4 Submission

You will need to submit all of your code, including your tests, and your UML class diagram to the MyCourses assignment before the due date. You must submit your solution as a ZIP archive named "`hw4.zip`" (if you submit another format, such as 7-Zip, WinRAR, or Tar, you will not receive credit for this homework).

5 Grading

The following deductions will apply:

- Completeness: up to 100% if a part of the writeup is not observed.
- Correctness/Testing: up to 50% or affected part if the solution is not correct.
- Quality/Style: up to 50% if solution is poorly designed.
- Explanation: up to 100% if the student is not able to respond grader's questions.