PD 8 HW $49_{16}$ 2019-2-15 Jason Lam

**0. uses of polymorphism**
In an RPG, there are a lot of different characters (inherent in the name) that use the same general framework.
As a result, each character can use the same calculations (and follow the same rules in the game world) and use the same superclass as a framework.
With subclasses, one can specialize the different characters.
This can restrict what characters can go in what parameters, as to make sure a villain doesn't end up in a parameter meant for a hero, or vice versa.

**1. generalized uses of polymorphism**
Describe the types of programming problems that benefit from polymorphic solutions. (hw45#6)
Polymorphism is used to manage different classes as to develop a hierarchy between them. This means that as one goes up the classes to a superclass, the class gets more and more broad, while as one goes down the classes to a subclass, the class gets more and more specific.
This is especially useful in situations where the type of class that is being used in a situation is crucial. Polymorphism is essential for keeping track of groups that may or may not overlap with one another, and make sure that a role does not contradict with another role.
It is also useful for OOP in order to convey information about how a class operates with respect to another class, and inherit methods as to not have one massive file containing all methods.

2. Cite sources.
Credits go to:
Mr. Brown-Mykolyk for the framework
Piazza #280 for a baseline for Character.java (given by Kyra Gunluk)
Mohidul Abedin and Oliver Cai for suggestions and troubleshooting
Class discussion for ideas on the superclass variable and arrays

3. How much of the thinking did you do?
Other than the framework presented by Brown-Mykolyk, I felt that I did most of the work for this project. Most of the understanding was done when working on the code for homework, and the class discussion merely served to reinforce what I had done. However, I did use Piazza #280 as a base for Character.java, Protagonist.java, and Monster.java (I had already done the work, I was merely complying with the homework instructions). However, I cannot discount the role of discussion with my peers like Oliver Cai and Mohidul Abedin, who troubleshooted and gave suggestions on how code could be made more perspicuous. Overall, I would say that 80-90% of non-framework code was done by me, and I would say 95% of the thinking was done by me.

4. inheriting instance methods
table for Assassin.java

| method signature | class where this method is defined |
|---|---|
| private int getDefense() | Character |
| private void lowerHP(int dmg) | Character |

| | |
|---|---|
| public int attack(Character target) | Character |
| public String getName() | Protagonist |
| public void specialize() | Protagonist |
| public void normalize() | Protagonist |
| public void passive() | Assassin |
| public Assassin(String name) | Assassin |
| public void passive() | Assassin |

5. imaginative uses of polymorphism
The game feature added was a collection of monsters that the hero would have to kill in order to achieve victory. By creating a group of monsters, any type of monster could go into this collection, and be treated as a monster despite having different attributes. This created diversity between each battle and within the monsters in the game.

6. collection of super-class objects
As stated above, the collection of monsters named mob was implemented as an array of size MAX_ENCOUNTERS that held references to monsters. As such, any subclass of monster could be put in, and could be manipulated by the game as a monster regardless of its special properties. This collection of monsters was randomly generated using Math.random(), and the last monster in the array was much stronger. This array kept track of all of the monsters in the game, and the order in which to face them.

7. one super-class variable
The superclass variable called boss was a reference to the final element in the array mob. In other words, during the generation of mob, the last element in the array is given higher stats. Boss is assigned a reference to this element, and when a boss creature is in combat, the game does additional things, including attacking twice. It holds a reference to a monster, and as a result could be any one of the three monsters.

8. polymorphism in Java
A variable's object type and variable type are determined at different times. The variable type is determined at compile time by the compiler by the type preceding the variable, and the object type is determined at run time by the JVM with the assignment operator. In other words, what a variable can contain is designated by the compiler at compile time, but what the variable actually contains is determined by the JVM at run time.