Lab 3 - World of Heroes

Objectives

- Students will read a problem description and design a solution
- Students will write a professional design document
- Students will compare and contrast possible techniques to describe why their solution is superior
- Students will utilize the Factory pattern
- Students will demonstrate an understanding of encapsulation, inheritance, polymorphism, and composition (delegation)

Your team is the newest hire on the production of World of Heroes.

The core design team has been hard at work creating the core of the game where heroes roam across different ecosystems and wreck them by killing off top level predators (Heroes kill monsters). The company is hoping to launch their game next year and have hired you to start working on player customization.

The core design team has given you the following constraints so that whatever you create will work well with the rest of the code.

All heroes should implement the `HeroInterface`, which must contain the following 6 methods (Note: STR is strength, DEX is dexterity, and INT is intelligence):

```
public int attack(int val)
public int getSTR()
public int getDEX()
public int getINT()
public String getRaceName()
public String getJobName()
```

You must also create a FACTORY called `HeroFactory` that must contain the following static method:

```
public static HeroInterface createHero(String race, String job)
```

You may of course include more methods as you need them, but these methods are what the core design team is using to build all of their stuff.

Particulars
Core designers want all heroes to start with values of 20 for each stat: STR, DEX, INT.

Players can pick from one of 3 races: ELF, DWARF, ROBOT.

Elves increase DEX by 5, Dwarves increase STR by 5, and Robots increase INT by 5.

Players can pick from one of 3 character jobs: WARRIOR, ARCHER, MAGE. At this point, jobs only affect the way attack values are calculated using player's stats, as well as a parameter, *val*.

Warrior attack is the hero's STR * val + DEX
Archer attack is the hero's DEX * 1.5 * val
MAGE attack is the hero's DEX + INT * val

For instance, if you use the factory to create a Hero:

HeroInterface hero = HeroFactory.createHero("DWARF","WARRIOR")

then hero would be a concrete class that implements the interface HeroInterface. Hero would be a dwarf warrior, with STR of 25, DEX of 20, INT of 20 and an attack of 25 * val + 20.

If either the race or job are not valid, the function should return null.

Part 0: Schedule a time to meet with your partner before Friday to complete the lab.

Part 1: Design

Design how you will implement the concrete classes necessary to complete this task. You do have some **constraints.**

1. Except for HeroFactory and any testing you develop, you **MAY NOT** use any if or switch statements. If statements slow down the code too much for the game's core designers.
2. There are rumors that the core design team has plans to have a total of 10 races and 10 jobs by the end the launch time. How does this information affect your design? How many classes would you have to create?
3. You may not store any numeric data as a data member of any class. (can't store ints). No realistic reason for this, except to make your design more interesting without giving you even more work.

Deliverables at the start of lab!

1. A printed paper with a UML diagram describing the classes that you will need to implement and how they all fit together. Use violet UML to create this document.

2. A printed paper describing how you plan to satisfy both of the consideration above (no if stmts, number of classes to implement when there are 10 races, 10 jobs, no numeric data).

Part 2: Create a class called TestHero.java

Be sure that asserts are enabled in jGrasp. (Build → enable assertions).

Inside of TestHero create a main method that will test all of the requirements of the HeroFactory. (Note that by extension this will test all of the other things as well.)

Keep in mind that your testing should aim to FIND problems, not verify that it works.

You will be able to build your code, but you will not be able to test it locally as the HeroFactory is only full of stubs.

Submit

Upload to Moodle. Keep in mind that you will not think of everything the first time. Read the error messages for clues to find all of the bad examples. You can also use System.out to get more information from moodle as you solve this problem.

Part 3: Implement your Heroes

Implement your factory and heroes and test them out.

Submit

All of your files. There will be several. Besides HeroFactory, you may name them anything you like.

Part 4: Write up your design document

Here is a link to the how and why of design documents

http://blog.slickedit.com/2007/05/how-to-write-an-effective-design-document/

0) Both of your names

1) Include how did you do your testing in an efficient manner (you didn't write all of those tests out by hand, did you?)

2) Include your UML for the design

3) Talk about why you chose this design over two other designs (you pick them out). Talk about the strengths of your approach and why your approach is superior to other approaches.

Hand in the paper version (one for each group)