Introduction

Hello folks! In assignment 4, you will get a sense of object oriented programming and why it is useful to implement complicated programs, such as a simple role-playing game (RPG) demo that you will develop in this assignment. You are also required to come up with your own tests to make sure your code has the correct behavior as described in this writeup. You should complete this assignment **individually.** This is a long assignment, so start early!

Part 1: Get started

1. Make sure there is no problem with your Java coding environment. If there is any, review assignment 1 or come to the office hour before you start this assignment.
2. Review lectures 7 and 8, and/or read chapters 9 and 10 (through section 10.6) of *Introduction to Java Programming and Data Structures.*
3. Download the starter code.
   
   If you work with your **local machine,** download it from Piazza -> Resources -> Homework -> Assignment4.java, Hero.java, Item.java, Monster.java, and Tower.java.
   
   Start terminal/command line and navigate to the directory that contains Assignment4.java, Hero.java, Item.java, Monster.java, and Tower.java.
   
   If you decide to use **Linux Cloud,** use the following command to copy the starter code to a new directory called HW4:

   ```
   $ cd ~
   $ mkdir HW4
   $ cp -a ../public/assignments/PA4 ./HW4
   ```

   Note: you will not be able to compile the start code because most of the methods are left unimplemented.

Overview:

In this assignment, you will implement a demo of gameplay of a simple RPG. In the game, there is the hero, our main character, and the tower. In each level of the tower, there is a monster and
an item. The hero will go from the first level to the top level of the tower, fighting a monster at each level along the way. Once a monster is defeated, the hero will pick up an item on the same level to enhance themselves. Both hero and monsters have three attributes: attackDamage, health, and speed. The hero and monster will take turns attacking each other. The attackDamage indicates the amount of damage the hero or monster will cause to their opponent (i.e., drop their 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). The hero wins when the monster on the top level is defeated. Whenever the hero is defeated, it’s game over. Implementation tip: you should not change any data field or method signature in the starter code. So, observe the starter code and read the instructions below to make sure you understand what each field means before you start to implement.

**Part 2: Item.java**

First, you need to implement the object class called Item, which can change the three attributes of the hero once equipped.

The Item object should contain the following fields (all are provided in the starter code):

1. **String name**: the name of the item
2. **int attackDamage**: the attackDamage to be added when hero equips the item
3. **int health**: the health to be added when hero equips the item
4. **int speed**: the speed to be added when hero equips the item
5. **String itemList**: a list of all allowed item names. Do not change any of these.
6. **Int[][] itemStats**: the [attackDamage, health, speed] attributes of each corresponding item. Do not change any of these.

The Item object should contain the following member methods:

1. **public Item(String name)**:
   
   This is the constructor of Item class. First it needs to set the name field and then iterate through the itemList to find the index of that item name. After that, it should find the corresponding int array of size 3 in itemStats. The number, from left to right, means attackDamage, health, and speed in that order. Finally, it should set the fields according to the numbers. If the input name is not in the itemList, set all three attributes (attackDamage, health, and speed) to 0.
Examples:
When the input is “Knight’s Sword”, attackDamage is set to 4, health to 0, speed to -1, and name to “Knight’s Sword”.
When the input is “Something else”, attackDamage is set to 0, health to 0, speed to 0, and name to “Something else”.

2. Four getters: `getAttackDamage()`, `getHealth()`, `getSpeed()`, and `getName()`:
Each getter should simply return the corresponding private field of this Item object.

Part 3: Hero.java and Monster.java

Class Hero and Monster share similar fields and methods as given in the starter code. They can both attack and receive damage. Hero can also equip items while a monster can roar. Hero gets an initial item when constructed.

The both Hero and Monster objects should contain the following fields:

1. String name: the name of hero (or monster)
2. int attackDamage: the attackDamage attribute of hero (or monster)
3. int health: the health attribute of hero (or monster)
4. int speed: the speed attribute of hero (or monster)

Field only for Monster:
1. String roarSound: The String that indicates roar sound of the monster

Both Hero and Monster objects should contain the following member methods. Make sure you implement the following first 3 methods first in both Hero.java and Monster.java as they will help you write later methods.

1. public boolean isStillAlive():
If health > 0, returns true. Otherwise, print out “XXX is defeated” and return false.

Example:
The hero named Bob has -1 health, then calling this method should print “Bob is defeated” and return false.

2. Two getters: `getSpeed()` and `getName()`:
Getter should simply return the corresponding private field.

3. public void receiveDamage(int damage):
Deduct health by the amount of input damage.
4. **Public void printStats():**
   This method prints out the statistics of Hero or Monster object. This method is fully implemented in the start code, do not change anything of this method.

Methods only for Hero:

1. **public Hero(String name, Item initialItem, int attackDamage, int health, int speed):**
   The constructor of Hero class. Should set the private fields - name, attackDamage, health, and speed. The attackDamage, health, and speed from the parameter are the default attributes for the hero. In addition, you need to update attributes provided by the initialItem.
   Hint: `equipItem(Item item)` might be useful.

2. **public void attack(Monster monster):**
   This method attacks the monster. It causes the input monster to lose health that is equal to the hero’s attackDamage. Then it should print out the message “XXX attacks XXX, causing X damage” and print the statistics (Hint: recall that printStats method is already implemented for you) of the attacking target after dualing the damage.
   **Example:**
   If Bob’s (the hero) attackDamage is 4 and the zombie’s health is 10 before calling this method.
   After calling `Bob.attack(zombie)`, the monster should have 6 health and the printed output is:
   ```
   Bob attacks zombie, causing 4 damage
   zombie - attack: 3, health: 6, speed: 2
   ```

3. **public void equipItem(Item item):**
   Simply add the three attributes (attackDamage, health, and speed) of the input item to the corresponding attributes of the hero. Then print out “XXX received XXXX”.
   **Example:**
   Calling Bob.equipItem(thunderHammer), following should be printed to the command line:
   ```
   Bob received Thunder Hammer
   ```

4. **public boolean isFasterThan(Monster monster):**
   Return true if the hero has a higher speed than the input monster.

Methods only for Monster:
1. **public Monster(String name, String roarSound, int attackDamage, int health, int speed):**
   The constructor of Monster class. Should set the private fields - name, attackDamage, health, speed, and roarSound.

2. **public void attack(Hero hero):**
   Causes the input hero to lose health that is equal to the monster’s attackDamage. Then similarly, it should print out the message “XXX attacks XXX, causing X damage” and print the statistics (Hint: recall that printStats method is already implemented for you) of the hero after dualing the damage.
   
   **Example:**
   If the zombie’s attackDamage is 3 and Bob’s health is 16 before calling this method. After calling zombie.attack(Bob), the monster should have 13 health and the printed output is:
   
   ```
   zombie attacks Bob, causing 3 damage.
   Bob - attack: 4, health: 13, speed: 4
   ```

3. **public void roar():**
   Print out the roarSound string to the command line as shown in example.
   
   **Example:**
   If the name of monster is “zombie” and roarSound is “Rueeeeee”, the printed output is:
   
   ```
   zombie: Rueeeeee
   ```

**Part 4: Tower.java**

First, you need to implement the object class called Tower, which has several levels, and each level should have one monster and one item.

The Tower object should contain the following fields (all are provided in the starter code):

1. **int height**: the number of levels the tower has
2. **Monster[] monsterEachLevel**: an array of Monster that corresponds to monsters at each level
3. **Item[] itemEachLevel**: an array of Item that corresponds to items at each level

The Tower object should contain the following member methods:
1. **public Tower(int height):**
   The constructor of Tower class. Set the height field to input height. If the input height is smaller than 1, set it to 1. Create empty arrays for monsterEachLevel and itemEachLevel, both should have lengths that equal to height.

2. **public void setOneLevel(int level, Monster monster, Item item):**
   Set the input monster and item to the input level of the tower.

3. **public int getHeight():**
   Getter of height field.

4. **public Monster getMonsterAtLevel(int level):**
   Return the Monster object at a certain level indicated by input level.

5. **public Item getItemAtLevel(int level):**
   Return the Item object at a certain level indicated by input level.

---

**Part 5: Unit tests**

As before, you are encouraged to create as many test cases as you think to be necessary to cover all the edge cases. However, since there are so many methods in this assignment, we will not ask you to create test cases for each method. To get full credit, create at least **six** test cases that test **different methods**. We suggest making some print messages in each of your test cases so that you will know which test case is failing. The unitTests method should return true only when all the test cases are passed, otherwise return false.

**Part 6: Complete playGame() in main**

Besides main and unitTests, there are two other methods in Assignment4.java: setUpTower() and playGame(). setUpTower() is already fully implemented for you in the start code (you don’t need to change anything in this method). It constructs a tower of 5 levels and sets the monsters and items as below:

<table>
<thead>
<tr>
<th>Level</th>
<th>Monster</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 0</td>
<td>slime</td>
<td>Ninja Suit</td>
</tr>
<tr>
<td>Level 1</td>
<td>nightbat</td>
<td>HP Potion</td>
</tr>
<tr>
<td>Level 2</td>
<td>ghost</td>
<td>Platinum Shield</td>
</tr>
<tr>
<td>Level 3</td>
<td>zombie</td>
<td>Thunder Hammer</td>
</tr>
</tbody>
</table>
In main, this tower along with a newly created hero called “CSE_8B_Ninja” are passed to the arguments of method `playGame()`.

```java
public static void playGame(Hero hero, Tower tower):
```

This method should simulate the game play described in overview and print the gaming process to the command line. Review the **Overview** part and use the classes that we already implemented to complete this method. Below is the expected output when running Assignment4 after correctly implementing `playGame()` and all 4 classes:

```
$ java Assignment4
All unit tests passed.
CSE_8B_Ninja received Knife
Level 0: CSE_8B_Ninja encounters slime
   slime: Zizizi
   slime - attack: 1, health: 4, speed: 1
   CSE_8B_Ninja attacks slime, causing 3 damage
   slime - attack: 1, health: 3, speed: 1
   slime attacks CSE_8B_Ninja, causing 1 damage.
   CSE_8B_Ninja - attack: 3, health: 9, speed: 3
   CSE_8B_Ninja attacks slime, causing 3 damage
   slime - attack: 1, health: -2, speed: 1
   slime is defeated
   CSE_8B_Ninja received Ninja Suit
Level 1: CSE_8B_Ninja encounters nightBat
   nightBat: Ummnnn
   nightBat - attack: 2, health: 7, speed: 5
   nightBat attacks CSE_8B_Ninja, causing 2 damage.
   CSE_8B_Ninja - attack: 4, health: 9, speed: 5
   CSE_8B_Ninja attacks nightBat, causing 4 damage
   nightBat - attack: 2, health: 3, speed: 5
   nightBat attacks CSE_8B_Ninja, causing 2 damage.
   CSE_8B_Ninja - attack: 4, health: 7, speed: 5
   CSE_8B_Ninja attacks nightBat, causing 4 damage
   nightBat - attack: 2, health: -1, speed: 5
   nightBat is defeated
   CSE_8B_Ninja received HP Potion
Level 2: CSE_8B_Ninja encounters ghost
   ghost: Whinwhin
   ghost - attack: 4, health: 6, speed: 3
   CSE_8B_Ninja attacks ghost, causing 4 damage
   ghost - attack: 4, health: 2, speed: 3
   ghost attacks CSE_8B_Ninja, causing 4 damage.
   CSE_8B_Ninja - attack: 4, health: 9, speed: 5
   CSE_8B_Ninja attacks ghost, causing 4 damage
   ghost - attack: 4, health: -2, speed: 3
   ghost is defeated
   CSE_8B_Ninja received Platinum Shield
```

Two screenshots are consecutive and are only separated into two pictures for typographic convenience. Your code should print the same output.
If the hero is defeated, print out “Game Over!” and return from the playGame method. As shown below:

```
$ java Assignment4
All unit tests passed.
CSE_8B_Ninja received Knife
Level 0: CSE_8B_Ninja encounters dragon
dragon: Ahsllhh
  dragon - attack: 6, health: 20, speed: 7
dragon attacks CSE_8B_Ninja, causing 6 damage.
  CSE_8B_Ninja - attack: 3, health: 4, speed: 3
CSE_8B_Ninja attacks dragon, causing 3 damage
dragon - attack: 6, health: 17, speed: 7
dragon attacks CSE_8B_Ninja, causing 6 damage.
  CSE_8B_Ninja - attack: 3, health: -2, speed: 3
CSE_8B_Ninja is defeated
Game Over!
```

If you still have any problem with how playGame should work, check the discussion video that will be released on Friday.

You are encouraged to create different tower and hero settings to validate your implementation or just for fun. To do so, just mimic the setUpTower method and create for example, setUpTower2(), setUpTower3(), etc. Then change the main and run. We will not grade on the main method, you just need to make sure you playGame implementation is correct.

**Part 6: Coding style**

When coding in Java, there are several style rules that people usually follow to make the code clean and readable. In this course, you are asked to follow rules specified in link below:

[https://cseweb.ucsd.edu/classes/fa20/cse8B-a/styleguide.html](https://cseweb.ucsd.edu/classes/fa20/cse8B-a/styleguide.html)

Read the coding style guide carefully and refine your code for this and all future assignments.
Submission

Very important! Please follow the instructions below carefully and make the exact submission format. This is important since we will use scripts to grade so if you don't follow the same submission format you probably will receive a zero.

1. Go to Gradescope and click on PA4.
2. Click the DRAG & DROP section and directly select the required file (Assignment4.java, Hero.java, Item.java, Monster.java, Tower.java). Drag & drop is fine. Please make sure you don't submit a zip. Just the file solely. Make sure the name of the file is correct.
3. You can resubmit unlimited times before the due date. Your score will depend on your final submission, even if your former submissions have a higher score.
4. The autograder is for the use of the instructional team. You won't see the result of the autograder. As long as you uploaded your file you're good to go.