FA21 CSE 8B Homework 4: Hero vs. Monsters
Due Date: Wednesday, October 27th, 11:59 pm PDT

Learning goals:

● Understand object-oriented programming (OOP) concepts through:
  ○ Objects (and object-oriented thinking)
  ○ Classes

NOTE: This assignment should be completed INDIVIDUALLY. Pair programming is NOT allowed for this assignment.

IMPORTANT: You should NOT have to import any additional packages to complete this assignment. Any unnecessary imports may result in failure to compile on Gradescope.

THIS IS A LONG ASSIGNMENT, SO PLEASE START EARLY!!!

Coding Style (10 points)

For this programming assignment, we will be enforcing the CSE 8B Coding Style Guidelines. These guidelines can also be found on Canvas. Please ensure to have COMPLETE file headers, class headers, and method headers, to use descriptive variable names and proper indentation, and to avoid using magic numbers.

Part 0: Getting started with the starter code (0 points)

1. Make sure there is no problem with your Java coding environment. If there are any problems, then review Assignment 1, or come to the office/lab hours before you start Assignment 4.
2. Download the starter code.
   a. If you are working on your local machine, then you can download the starter code from Piazza → Resources → Homework → Assignment4.zip. Download the starter code to a directory of your choice, then extract the zip file. It should contain 5 Java files: Assignment4.java, Hero.java, Item.java, Monster.java, and Tower.java. Afterwards, open your terminal or command prompt, then navigate to the directory that contains those 5 Java files.
b. If you are working via UCSD Linux Cloud through your CSE 8B account, then use the commands below to copy the starter code to a new directory called PA4, to change your current directory to PA4/starter, and to print the files in the starter directory:

```
$ cp -r ~/../public/assignments/PA4 ~
$ cd ~/PA4/starter
$ ls
```

It should print out 5 Java files: Assignment4.java, Hero.java, Item.java, Monster.java, and Tower.java

3. Try to compile and run the starter code, and you should expect the following output:

```
PS C:\Users\Benson\Downloads\assignment4starter> javac *.java
PS C:\Users\Benson\Downloads\assignment4starter> java Assignment4
All unit tests passed.
Tower.java constructor not yet implemented.
Monster.java constructor not yet implemented.
Monster.java constructor not yet implemented.
Monster.java constructor not yet implemented.
Monster.java constructor not yet implemented.
Monster.java constructor not yet implemented.
Item.java constructor not yet implemented.
Tower.java setOneLevel not yet implemented.
Item.java constructor not yet implemented.
Tower.java setOneLevel not yet implemented.
Item.java constructor not yet implemented.
Tower.java setOneLevel not yet implemented.
Item.java constructor not yet implemented.
Tower.java setOneLevel not yet implemented.
Item.java constructor not yet implemented.
Tower.java setOneLevel not yet implemented.
Item.java constructor not yet implemented.
Tower.java setOneLevel not yet implemented.
Item.java constructor not yet implemented.
Tower.java setOneLevel not yet implemented.
Item.java constructor not yet implemented.
Hero.java constructor not yet implemented.
Assignment4.java `playGame` not yet implemented.
```

You will have to implement several methods required to play Hero vs. Monsters.

---

### Overview

In this assignment, you will implement the gameplay demonstration of a simple role-playing game (RPG). In the game, there is the hero, our main character, and the tower(s). In each level of the tower, there is a monster that is guarding an item. The hero will go from the first level to the top level of the tower, fighting a monster at each level. Once a monster is defeated, the hero will pick up an item on the same level to enhance their attributes. Both the hero and the monsters have four attributes: health, attack, defense, and speed. The hero and the monsters will take turns attacking each other. The attack indicates the maximum amount of
damage the hero or the monster will cause to their opponent (i.e., drop their opponents health) in each turn, but the attack can be weakened by their opponent's defense. 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. As such, please 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 four attributes of the hero once equipped.

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

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

**Notice how each member field is declared private.** This means that the member is only visible within the class, not from any other class. In other words, you will need to use accessors (i.e., getter methods) and mutators (i.e., setter methods) to access and modify, respectively, these private members. You must also use this keyword to modify and access member variables hidden by local variables.

You can assume that the attack will always be positive (meaning that opponents would not "heal" each other).

The Item object should contain the following member methods:

1. public Item(String name):
   This is the constructor of Item class. First, the constructor needs to set the name field and then iterate through the itemList to find the index of that item name. After that, the constructor should find the corresponding int array of size 4 in itemStats. The
number, from left to right, means health, attack, defense, and speed in that order. Finally, the constructor should set the fields according to the numbers. NOTE: If the input name is not in the itemList, then set all four attributes (health, attack, defense, and speed) to 0.

EXAMPLES:
When the input is "Knight Sword", health is set to 0, attack to 6, defense to 0, speed to -1, and name to "Knight Sword". In other words, the Knight Sword would increase the hero's attack at the cost of the hero's speed.
When the input is "something else", health is set to 0, attack to 0, defense to 0, speed to 0, and name to "something else". In other words, if the item does not exist in the itemList, then the item does not provide any attribute changes.

2. Five accessors: int getHealth(), int getAttack(), int getDefense(), int getSpeed(), and String getName(): Each getter method 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 where their defense can weaken the damage taken. Hero can also equip items. Hero gets an initial Item when constructed.

The both Hero and Monster objects should contain the following fields:
1. private String name: the name of the hero (or the monster)
2. private int health: the health attribute of the hero (or the monster)
3. private int attack: the attack attribute of the hero (or the monster)
4. private int defense: the defense attribute of the hero (or the monster)
5. private int speed: the speed attribute of the hero (or the monster)

You can assume that the attack will always be positive (meaning that opponents would not "heal" each other).

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, then return true. Otherwise, print out "XXX is defeated" and return false.
EXAMPLE:
If the Hero named "Bob" has -1 health, then calling this method should print "Bob is defeated" and return false.

2. Five accessors: getHealth(), getAttack(), getDefense(), getSpeed(), and getName():
   Getter methods 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 attributes of the Hero or the Monster object. This method is fully implemented in the starter code, so please do NOT change anything about this method.

Methods only for Hero:

1. public Hero(String name, int health, int attack, int defense, int speed, Item initialItem):
   The constructor of Hero class. First, you should set the private fields (name, health, attack, defense, and speed) to their respective parameters. Then, you need to update the attributes provided by the initialItem. HINT: equipItem(Item item) might be useful to update the Hero's attributes.

2. public void attack(Monster monster):
   This method attacks the input monster. This method causes the input monster to lose health that is equal to the hero's attack minus the monster's defense. However, if the monster's defense is greater than or equal to the hero's attack, then the monster should take 1 damage (the monster should ALWAYS take some damage from an attack). Then, this method should print out the message "XXX attacks XX, causing X damage" where XXX is the hero's name, XX is the monster's name, and X is the damage amount. Finally, the method should print the monster's attributes (HINT: recall that the printStats() method is already implemented for you).

   Example:
   If Bob's (the hero) attack is 4, the zombie's (the monster) health is 10, and the zombie's defense is 2, then after calling Bob.attack(zombie), the monster should have 8 health and the printed output is:

   ```
   Bob attacks zombie, causing 2 damage
   zombie - health: 8, attack: 1, defense: 2, speed: 1
   ```

3. public void equipItem(Item item):
   Simply add the four attributes (health, attack, defense, and speed) of the input item to the corresponding attributes of the hero. Then, print out "XXX received XXXX" where XXX is the hero's name and XXXX is the item's name.
Example:
If there is an Item object named thunderHammer with the attributes for a "Thunder Hammer", then calling Bob.equipItem(thunderHammer) should print the following 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, and return false otherwise.

Methods only for Monster:
1. public Monster(String name, int health, int attack, int defense, int speed):
   The constructor of Monster class. You should set the private fields respectively - name, health, attack, defense, and speed.
2. public void attack(Hero hero):
   Similar to the attack() method from the hero class, this method causes the input hero to lose health that is equal to the monster's attack minus the hero's defense. However, if the hero's defense is greater than or equal to the monster's attack, then the hero should take 1 damage (the hero should ALWAYS take some damage from an attack). Then, this method should print out the message "XXX attacks XX, causing X damage" and print the hero's attributes (HINT: recall that the printStats() method is already implemented for you).

Example:
If the zombie's attack is 1, Bob's health is 10, and Bob's defense is 4, then after calling zombie.attack(Bob), the hero Bob should have 9 health and the printed output is:

```
zombie attacks Bob, causing 1 damage
Bob - health: 9, attack: 16, defense: 4, speed: 5
```

---

**Part 4: Tower.java**

Before you can start demoing your RPG, you first need to implement the object class called Tower, which has several levels. 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. private int height: the number of levels the tower has
2. private Monster[] monsterEachLevel: an array of Monsters that corresponds to the monster at each level
3. private Item[] itemEachLevel: an array of Item that corresponds to the item at each level

The Tower object should contain the following member methods:

1. public Tower(int height):
   The constructor of Tower class. You should set the height field to input height. If the input height is smaller than 1, then set the height field to 1. Then, you should create empty arrays for monsterEachLevel and itemEachLevel where both should have lengths that are 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 method 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: unitTests() in Assignment4.java

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, please create at least five test cases that test all your different methods from Monster.java, Hero.java, Tower.java, and Item.java. 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: playGame() in Assignment4.java

Finally, besides main() and unitTests(), there are two other methods in Assignment4.java: setUpTower() and playGame(). setUpTower() is already fully implemented for you in the starter code, so you do NOT need to change anything in this method. setUpTower() 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>Agility Armor</td>
</tr>
<tr>
<td>Level 1</td>
<td>mummy</td>
<td>HP Potion</td>
</tr>
</tbody>
</table>
In `main()`, this tower along with a newly created `hero` called "CSE_8B_Hero" are passed to the arguments of method `playGame()`.

```java
public void playGame(Hero hero, Tower tower):
```
This method should simulate the game play described in the Overview and print the demonstration to the command line. To prepare for this method, please review the Overview and use the classes that you already implemented to complete this method.

Before giving away some implementation guidelines for `playGame()`, below is the expected output when running `java Assignment4` after correctly implementing `playGame()` and all 4 classes (`Hero`, `Monster`, `Item`, and `Tower`):
CSE_BB_Hero received Small Knife

Level 0: CSE_BB_Hero encounters slime
   slime - health: 2, attack: 2, defense: 0, speed: 2
CSE_BB_Hero attacks slime, causing 6 damage
   slime - health: -4, attack: 2, defense: 0, speed: 2
slime is defeated
CSE_BB_Hero received Knight Sword

Level 1: CSE_BB_Hero encounters mummy
   mummy - health: 5, attack: 3, defense: 1, speed: 1
CSE_BB_Hero attacks mummy, causing 11 damage
   mummy - health: -6, attack: 3, defense: 1, speed: 1
mummy is defeated
CSE_BB_Hero received HP Elixir

Level 2: CSE_BB_Hero encounters ghost
   ghost - health: 9, attack: 5, defense: 2, speed: 7
ghost attacks CSE_BB_Hero, causing 2 damage
   CSE_BB_Hero - health: 18, attack: 12, defense: 3, speed: 4
CSE_BB_Hero attacks ghost, causing 10 damage
   ghost - health: -1, attack: 5, defense: 2, speed: 7
ghost is defeated
CSE_BB_Hero received Platinum Shield

Level 3: CSE_BB_Hero encounters yeti
   yeti - health: 15, attack: 5, defense: 4, speed: 2
yeti attacks CSE_BB_Hero, causing 1 damage
   CSE_BB_Hero - health: 17, attack: 12, defense: 7, speed: 2
CSE_BB_Hero attacks yeti, causing 8 damage
   yeti - health: 7, attack: 5, defense: 4, speed: 2
yeti attacks CSE_BB_Hero, causing 1 damage
   CSE_BB_Hero - health: 16, attack: 12, defense: 7, speed: 2
CSE_BB_Hero attacks yeti, causing 8 damage
   yeti - health: -1, attack: 5, defense: 4, speed: 2
yeti is defeated
CSE_BB_Hero received Thunder Hammer

Level 4: CSE_BB_Hero encounters wyvern
   wyvern - health: 20, attack: 7, defense: 6, speed: 8
wyvern attacks CSE_BB_Hero, causing 1 damage
   CSE_BB_Hero - health: 15, attack: 24, defense: 7, speed: 1
CSE_BB_Hero attacks wyvern, causing 18 damage
   wyvern - health: 2, attack: 7, defense: 6, speed: 8
wyvern attacks CSE_BB_Hero, causing 1 damage
   CSE_BB_Hero - health: 14, attack: 24, defense: 7, speed: 1
CSE_BB_Hero attacks wyvern, causing 18 damage
   wyvern - health: -16, attack: 7, defense: 6, speed: 8
wyvern is defeated
CSE_BB_Hero received Treasure
The Hero Wins!
The two screenshots are consecutive and are only separated into two pictures for typographic convenience. Your code should be able to print the exact same output.

If the hero is defeated at any point throughout the simulation, print out "Game Over!" and return from the playGame() method. For example, see the screenshot below:

```
PS C:\Users\Benson\Documents\GitHub\FA21_CSE8B_PA4> java Ref_Assignment4
All unit tests passed.
CSE_8B_Hero received Small Knife
Level 0: CSE_8B_Hero encounters really strong slime
really strong slime - health: 50, attack: 30, defense: 0, speed: 1
CSE_8B_Hero attacks really strong slime, causing 6 damage
really strong slime - health: 44, attack: 30, defense: 0, speed: 1
really strong slime attacks CSE_8B_Hero, causing 27 damage
CSE_8B_Hero - health: -17, attack: 6, defense: 3, speed: 5
CSE_8B_Hero is defeated
Game Over!
```

Your code should also be able to print the exact same output.

Here are some key things to remember when implementing playGame():

1. The hero starts from Level 0 and attempts to make their way towards the top level. In the case of our example, our top level is Level 4. As such, this means that the hero will go from Level 0 → Level 1 → Level 2 → Level 3 → Level 4. How will you go through each level?
2. At each level, the hero will encounter the monster. You should print out "Level X: XXX encounters Xx" as well as the monster's attributes.
3. At each level, the hero and the monster take turns attacking each other.
   a. Speed determines who attacks first. If the hero has a higher speed than the monster, then the hero attacks first. Otherwise, the monster attacks first.
4. The hero and the monster keep fighting until one of their health drops to 0 (or below). In other words, they keep fighting until one of them is no longer alive.
   a. If the hero defeats the current level's monster (or the monster is no longer alive), then the hero can pick up the item that belongs to that level. The hero then continues on to the next level.
      i. If the hero defeats the last level of the tower (meaning that you went through all the levels), then the hero wins. You should print out "The Hero Wins!" if this is the case.
   b. If the current level's monster defeats the hero, then it's game over. At this point, you should print "Game Over!" and stop the simulation.

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. with different tower heights, different monsters,
different items, etc. Afterwards, change the main method to create a new tower object, then pass that new tower into playGame(). We will not grade on the main method - you just need to make sure your playGame() implementation is correct.

Submission

VERY IMPORTANT: Please follow the instructions below carefully and make the exact submission format.

1. Go to Gradescope via Canvas and click on PA4.
2. Click the DRAG & DROP section and directly select the required files (Assignment4.java, Hero.java, Item.java, Monster.java, Tower.java). Drag & drop is fine. Please make sure you don't submit a zip. Make sure the filenames are correct.
3. You can resubmit an unlimited number of 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 grading your uploaded files automatically. Make sure your code can compile on Gradescope.

NOTE: The Gradescope Autograder you see is a minimal autograder. For this particular assignment, it will only show the compilation results and the results of a few testers. After the assignment deadline, a thorough Autograder will be used to determine the final grade of the assignment. Thus, to ensure that you would receive full points from the thorough Autograder, it is your job to extensively test your code for correctness via unitTests and main.