Lab Test 2 - Specification

Lab Test 2 - 05/12/2022
Class Group: TU856 / TU858

Worth: 15% of the overall mark for the module

Time allowed: 1h 50m (It must be submitted on Brightspace by 17:55)

Attendance sheet: You **must sign** the attendance sheet before leaving the lab room. Submissions will only be corrected if you have signed it.

Open book test

This is an open book test but you are only allowed to use **your OWN previous lab examples and notes**. **You can't use the internet, email or your phone!**

Submission

Please submit your solution (Python file) via Brightspace. It is your responsibility to make sure you've uploaded the correct file as changes after the test has finished won't be accepted.

Marking Scheme

- The task will be evaluated based on functionality, use of classes, inheritance, composition/aggregation, layout and use of comments.
- Indentation and white space have to be used appropriately.
- Code should be easy to read, and naming conventions should be adopted.
- Ensure that variable names are informative while appropriate and meaningful comments are placed.
- The code should work without issue when run on the lecturer's machine.

Plagiarism

Plagiarism and answers built with chatgpt will result in a zero mark (0%). You should make yourself familiar with the plagiarism policy of Technological University Dublin. **You might be contacted by the lecturer to demo your code if the submission is believed to contain suspected plagiarism.**

Lab Test 2 - Description of the task

For tasks 1 to 4, implement the required classes and methods. For task 5 implement the main scope of the code.

Task 1 (15%)

Download the **“main.py”** file in the assignment section and inherit the “Hero” class to **implement your favorite archetype**. For example, the Barbarian could extend the Hero class. Create your own archetype class (that is not a Barbarian) using inheritance from Hero.

Add **at least one instance attribute and two methods to your class**. One method should represent an attack and return a float indicating the attack power, while the other should represent a form of defence and increase the hero's health points.

Task 2 (15%)

Create a **monster** class. A monster should have at least a **name (string), strength (float), health points (float) and roar (string)**. In the monster constructor also add a print command that outputs the monster roar. The monster class should also have an **“attack” method** that returns a value linearly related to its strength.

Task 3 (20%)

Using operator overloading, add the method **__add__** (for addition) to the Monster class. Adding two monsters should create a third monster (a kind of monstrosity) with concatenated names, multiplied strengths and summed health points. For the roar, concatenate the two roars and add three exclamation marks !!!.

Task 4 (30%)

Within your hero class inherited from the Hero class, **create a "combat" method** that takes a "monster" parameter and returns True/False if the hero wins/loses the combat. Check if the monster parameter is an actual instance of monster, and if not, print a message saying the hero only fights monsters, and leave the method.

In a combat, monster and hero attack each other until only one of them survives. Write a loop in your method to simulate that.The hero attacks first, and each of his attacks reduces the respective monster health points. The monster attacks second, and each of his attacks reduce the hero's health points.

Print whether the hero wins (monster with no health points) or loses (hero gets to zero health points).

Task 5 (20%)

Create an instance of your hero class. **Create two instances of monster** using the monster constructor. **Create a third instance of monster** by adding the two previous monsters. Make your hero combat one monster at a time while s/he has any health points. Print whether the hero emerges victorious or has been defeated after fighting all monsters.

Hints:

- Implement the tasks in the order they are given. Each task will help you to proceed to the next one.
- Start with the easy parts. Define only your constructors and add new functionalities one by one.
- Don’t forget to call the constructor of the parent class in the child class through an unbound method.
- Add docstrings to your classes and methods
- Add comments that are meaningful
- Use PyCharm formatting suggestions