CSE 216 – Homework II

Instructor: Dr. Ritwik Banerjee

This homework document consists of 5 pages. Carefully read the entire document before you start coding. You will notice that this homework may be a little more open-ended that the typical assignment in your previous coding semesters: here, you are being told *what* behavior is expected of your code, but *how* to implement it is not entirely specified. You are expected to use the concepts from the lectures and recitations to figure this out on your own.

1 Language & Development Environment

This assignment requires you to understand and make use of the object oriented principles in Python, together with its parameter passing and the type system details. You must Python `3.x`. The recommended (but not mandatory) development environment is **PyCharm**.
Note: You may have seen an environment called Jupyter. This is great for quick prototyping, writing short scripts, or demonstrating how a code snippet runs, but it is not recommended for a larger assignment. Do NOT use Jupyter for this assignment.

2 A Fictional World

You are taken to a fictional world where among the ordinary people reside three ferocious kinds: `Fighter`s, `Warrior`s, and `Knight-Errant`s. Your job is to implement a few classes to simulate this fictional world of these three classes of people in a way that honors the specified properties and attributes. These three classes of people are, of course, `Person`s. By law, they are also required to be `Adult`s ($\geq$ 18 years old).

2.1 A Person

A `Person` is required to have a `name: str` and `age: int`. They may optionally also have `wealth: int`. Depending on the age, each person is (or isn't) an `adult`. The default creation of a new person creates a non-adult with zero wealth. For this assignment, you may assume that the name-plus-age combination uniquely identifies a person.

2.2 A Fighter

A `Fighter` is an adult person who has some positive wealth and a set of skills. Of course, just having a skill doesn't mean that the fighter has perfected it, so a skill is associated with a `level`, which is a score between 0 and 10. The skill details of each fighter must be private.

2.2.1 Skills

There are four types of skills – `spear`, `unarmed_combat`, `mace`, and `broadsword`. A fighter's complete skillset may be represented as a set of mappings of skills to levels. A fighter who is an expert with

the spear but has no other skills could, for example, have their skills represented by {`spear: 10, unarmed combat: 0, mace: 0, broadsword: 0`}.

2.2.2 Fight Rules

A fighter `a` can invoke a `challenge(...)` to fight another fighter `b`. The challenge must also specify the skill to be used.

1. A fight can only happen between two fighters, and a fighter cannot fight themselves!
2. A fighter also cannot enter into a fight (as a challenger or as the recipient of a challenge) if their wealth is zero.
3. Only one skill may be used in a fight. If `b` is able to, s/he will accept the challenge immediately.
4. The outcome of the `challenge` is a winner, determined by which fighter has a higher level of the chosen skill. If both `a` and `b` have the same level, the winner is determined by luck (see **random.choice**). Otherwise, the fighter with higher skill level wins. The winner gains 10 wealth points, and the loser forfeits 10 wealth points (restricted to a minimum of zero points)[1].
5. Each fight also provides a random chance for each participating fighter to add 1 point to their skill level (for the skill chosen in that fight), subject to the maximum limit of 10.
6. Of course, if fighters openly knew each others skill levels, they would only fight weaker opponents to keep earning money, since the opponent can't decline a challenge. This is why these details were required to be made private in Sec. 2.2.

2.3 A Warrior

A warrior "is-a" fighter with some additional freedoms. A warrior can keep a list of challenge requests from other fighters, and can accept or decline each request. A warrior may also simply leave requests unanswered. However, if a fighter issues a challenge to a warrior, but then ends up fighting someone else before this warrior accepts the request, the fighter has the right to withdraw the request (e.g., the fighter may have suffered an injury in this fight, and cannot fight the warrior any more).

1. If an ordinary fighter wins against a warrior, they attain 25 wealth points. They also increase their skill level by one (with certainty, no randomness this time).
2. If a warrior loses against an ordinary fighter, they forfeit 25 wealth points (restricted to a minimum of zero points) and gain no additional skill from the fight they lost.
3. For any remaining scenarios, the property that a warrior "is-a" fighter should be used to determine the outcome. That is, if two warriors fight each other, you should use the properties of the base class `Fighter` and the fight rules in Sec. 2.2.2.

2.4 A Knight-Errant

A knight errant "is-a" warrior, but often has better things to do! For instance, they travel a lot. Therefore, at any given point of time, a knight errant may be `traveling` or not.

1. When a knight-errant is traveling, he cannot challenge another fighter. Nor can he accept a challenge. Others can, however, leave challenge requests during this time.
2. A knight-errant can sometimes find treasures when they are traveling. This treasure's worth (in terms of amount of wealth) gets added to the knight-errant's wealth when they return from a travel.

[1] This may not be a direct transfer of wealth from one fighter to another (e.g., there may be an arena where the crowd is paying to watch), so even if one fighter loses only, say, 6 wealth points, the winner still gains 10.

3. If an ordinary fighter wins against a knight-errant, s/he gains 40 wealth points and 2 skill levels.
4. If a warrior wins agains a knight-errant, s/he gains 20 wealth points and 1 skill level.
5. If a knight-errant loses against an ordinary fighter, he loses 40 wealth points. And similarly, if he loses against a warrior, he loses 20 wealth points. He also does not gain any skill.
6. For any remaining scenarios, you should use the fact that a knight-errant "is-a" warrior, and therefore, "is-a" fighter (just like point 3 in Sec. 2.3).

3 Tasks

In this assignment, you are required to implement three classes for the three types of fighters: `Fighter`, `Warrior`, and `KnightErrant`, and the base `Person` class. Further, a class `Fight` also must be implemented. The fictional world has already been described above, and your implementation must honor all those specifications. To provide some guidance, a partial API and class definition is provided below:

3.1 `Person`

- Attributes for name, age, wealth, and whether or not the person is an adult.
- Methods to check whether or not two person instances are equal.

3.2 `Fight`

- Two attributes for the two fighters in a fight.
- A method called `winner()` that returns the winner of the fight.

3.3 `Fighter`

- A method `challenge(...)` and returns `None`. This method must carry out the responsibility of ensuring that all the rules of a challenge are observed (e.g., wealth change, skill change, whether or not the challenge can even be issued and/or accepted, etc.). [*Note that you may have to override this method in some other classes.*]
- A method `withdraw(...)` to withdraw a challenge request s/he issued.

3.4 `Warrior`

- A method `accept_random(...)` that accepts a random challenge from the list of challenges.
- A method `decline_random(...)` that declines a random challenge from the list of challenges.
- Has similar methods `accept_first(...)` and `decline_first(...)`.

3.5 `KnightErrant`

- A method `travel(...)` to mark the beginning of a journey.
- A method `return_from_travel(...)` to mark the return from a journey. This method must implement any treasure-related activities.

3.6 A `test.py` script

In addition to the python module, you are also required to write a script that has test cases for various scenarios described in this document. You are required to have at least 10 test cases, with at least three tests for each type of fighter (`Fighter`, `Warrior`, and `KnightErrant`). You should also have at least one test case involving challenge requests[2].

NOTES

The partial API specifications have intentionally left '...' in the parameters to give you the freedom you may need to design this fictional world in Python. Some methods have a specific return type, but for others, you may choose to use any return type (as required by your design).

The methods in your code MUST contain type annotations!

The behavior of this fictional world is provided to you in quite a lot of detail. So you only have to submit the module(s). No main module is required.

You must provide a `str()` method in your classes to nicely print the state of each type of fighter or person. This method will be used to check if the fights actually change the state as required by the specifications.

You must also provide printed error statements for any situation that is invalid according to the rules of this fictional world. However, *your code should not exit*! After raising the error, handle it by simply printing it instead of exiting.

4 What to submit?

A single `.zip` file consisting of your Python code (all the modules as well as the test script).

5 FAQ

Are the methods mentioned in Sec. 3 mandatory?

Yes. In the few cases where the return type is mentioned, you must also follow the return type specification.

Can we add more attributes to our classes?

Yes. [In Python, methods are also attributes.]

If an API method gets too complicated, can we factor out some parts of the complex logic to other methods and call them from the method mentioned in the API?

Yes. In fact, this is the recommended approach.

[2] You can, of course, include more than 10 test cases. This is the minimum requirement. In general, conducting more tests is always a good idea.

I submitted something wrong by mistake . . . what should I do now?

You are allowed multiple attempts until the deadline (at which point, submissions will no longer be accepted). If you have made a mistake, you must resubmit ALL files again (i.e., create your `.zip` file again). Partial resubmissions are not possible on Blackboard.

My code compiles on my laptop but . . .

Chances are you have used a wrong version of Python. Use 3.x only! **If your code does not compile, it will not be graded.**

I missed the deadline by 2 minutes . . .

Do NOT continue coding until the last minute! Verifying the code, properly creating the .zip file, etc. . . . these things take some time, and you are responsible for managing all this within the deadline. **Late submissions will not be accepted under any circumstances**.

- If too many students are trying to submit at, say, 11:58 pm one minute before the deadline, Blackboard server may become very slow and you may end up missing the submission deadline! To be safe, always, ALWAYS, prepare to submit ahead of time, not exactly AT last moment!

6 Grading Scheme

`Person`	(10 points)
`Fighter` base class and its functionalities	(15 points)
`Warrior` class and its functionalities	(15 points)
`KnightErrant` class and its functionalities	(15 points)
`Fight` class and its implementation of a fight	(15 points)
Type annotations	(10 points)
Test cases in `Test.py`	(10 points)
Docstring documentation for the API methods	(10 points)
TOTAL	100 points

Submission Deadline: Apr 05, 2019, 11:59 pm