INTERMEDIATE PROGRAMMING ASSIGNMENT

Assignment 1 – Part 1 (W04)

Background

Imagine that you are working as a software developer in a very prestigious video games company. You have been assigned to a project for a client that wants a video game of Lord of the Rings. Your first task (but not the last, since there will be a second part of the assignment) is to create *orcs* characters.

Exercise

Code the class *Orc*, which represents the orcs of the video game. This class is defined by the following attributes:

1. The orc's name: *name* of type String (for example, *"Ogrorg"*)
2. The orc's strength score in the domain [0-5]: *strength* of type Float (for example, *4.3*)
3. Does the orc have a weapon? (some orcs own a weapon and others do not own it): *weapon* of type Boolean, which is within the domain [True, False]: (for example, *True*)

Implement the class using the principles of Object-Oriented Programming (marks depend on the right use of these principles). The class must have the following functionalities:

- A constructor for initialising instances
- Properties for accessing and modifying the values of all the attributes.
- Type and value error checking for the attributes of the class. You must check that the values' setting by external users of your module is correct and within the domain of the attributes. For numeric values, if the user introduces a greater value, it will be truncated to the max. value in the domain (for example, the max. *strength* value is 5). If the value is lower than the minimum value, then it will be truncated to the minimum value (for example, the min. *strength* value is 0). For type errors (for example, trying to fix *strength* to *"Ogrorg"*), the assignment will not be completed and the error message "type ERROR" will be printed on the screen.
- The __str__ method that returns the string representation of the instances in this format: *name strength weapon* (for example, *Ogrorg 4.3 True*). Don't forget the spaces between attributes!
- The overload of the ">" to determine if an *orc* is *greater* than another in terms of fighting. If an orc owns a weapon and the other does not own it, then the orc with a weapon is *greater* than the other one. If both orcs are in the same situation of ownership of weapons, then an orc is *greater* than another only if its strength is greater.
- The orc has a functionality, which is *fight* with another orc. If an orc is ">" than another orc (you must use the overloaded operator for checking such condition), then it wins and its *strength* increases in 1 point. Besides, the instance of the winner orc is printed on the screen (you must use the __str__ method). Otherwise, that is to say, if there does not exist a ">" orc, they both lose 0.5 points.

Furthermore, you have to implement a test module for checking all the functionalities of the *Orc* class. You have to import the module orc and tests each of the above-described functionalities (marks depend on the right functionality of your class).

Assignment 1 – Part 2 (W06-W07)

Background

Once you have your orc class coded and working perfectly, the client for which your company works tells you that more characters are needed! Of course, you apply the knowledge learned in your module about OOP and you use inheritance in order to re-use your code. Your task is to create *archers* and *knights* human characters.

Note that this assignment has been designed in this way on purpose in order to simulate real-life, where typically these scenarios happen, where the requirements of the software to be developed change over time due to changes in the needs of the clients (who typically do not about software development and therefore struggle to have a clear and specific definition of what they want). In these situations, OOP is extremely useful, as motivated in class.

Exercise

Code the classes *Archer* and *Knight*, which represents the human archers and knights of the video game. Both classes have as attributes 1.-*name* and 2.-*strength* (both with the same characteristics as described for orcs in part 1). Besides, they have some extra attributes:

3. Their kingdom: *kingdom* of type String (for example, *"Gondor"*)
4. Only the *knights* have an extra attribute: *archers_list* a list of objects of type *Archer*. This list represents the archers that are led by the knight. There is no limit in the number of archers that a knight can lead. However, they all must belong to the same kingdom as the leader *Knight.* If the list contains archer/s that belongs to another kingdom, they will be removed from the list and only the other archers who satisfy such constraint will be assigned to the knight. The error message "archers list ERROR" will be printed on the screen if the list contains any object that is not an *archer*.

Archers and k*nights* always have a weapon since the moment of their creation until the moment in which they are destroyed. For this reason, they do not have as attribute *weapon* (it would be redundant).

Archers and *knights* have the same functionalities as orcs: str, > and fight (see description in part1 of the assignment). However, there are some differences with the orcs' functionalities:

- All the characters (*archers, orcs* and *knights*) must be able to use the ">" operator with any other type of character. Note that humans do not have as attribute *weapon.*
- They can **only** *fight* with orcs. The error message "fight ERROR" will be printed on the screen if a human tries to fight against a human. Apart from this, the functionality *fight* works in the same way as the orcs.
- The __str__ method that returns the string representation of the instances is in this format: *name strength kingdom* (for example, *Aragorn 3.3 Gondor*). Besides, **only** for the *knights*, the list of archers that leads will be shown right after within square brackets (for example, *Aragorn 3.3 Gondor [archer1 2.1 Gondor, archer2 4.2 Gondor]*). Note: No spaces between elements!. Use the __str__ method of the class *Archer*.

Implement both classes using the principles of Object-Oriented Programming (marks depend on the right use of these principles). *Archer* and *Knight* classes must have the following functionalities:

- A constructor for initialising instances
- Properties for accessing and modifying the values of all the attributes.
- Type and value error checking for the attributes of the class
- The __str__, > and fight functionalities

You have to complete the following tasks:

1. First, design the UML class diagram (you can use word, a text editor, a painting software).

2. Then, implement the necessary classes for this project: *Orc, Archer* and *Knight*. You can also code more classes if it makes sense in terms of the principles of Object-Oriented Programming (marks depend on the right use of these principles). Remember that the objective is to re-use the code by using inheritance rather than copying and pasting code (marks depend on the right use of inheritance and code re-use). Note that for your implementations you will have to follow the descriptions of the whole assignment (part1 and part 2).

3. Furthermore, you have to extend your testing module from part 1 of the assignment. The test module has to import the classes *Orc, Archer* and *Knight* and it has to test each of the functionalities associated with each class (marks depend on the right functionality of your classes).

Submission Details:

Include all your classes in a **single** python module named: *YourSurname_YourName.py* (for example, in my case: Climent_Laura.py). Do not submit the testing module. Do not submit a zip. Then, upload your single file in Canvas in the link Assignment1_submission. In this submission link, you will find the deadline for the submission. Before this deadline, you must also upload your UML design (you can use word, a text editor, a painting software) in the link Assignment1_UML_submission.

Important: If you do not follow the above-mentioned instructions, you will get a reduction of 15 marks in your assignment. Also, note that it is not possible to submit after the deadline (submission after the deadline means that you will get 0 marks). Furthermore, consider that a software plagiarism detection tool will be applied to your assignment. Check the plagiarism slides for consequences of plagiarism (first lecture of the module).

Important: To evaluate your code, I will run my own test module that will import your module. It is therefore mandatory to use the same names of the classes, attributes, functionalities and prints on the screen mentioned in the assignment. Note that this also includes the error messages. You must also respect the upper and lower case. Note also that the properties must have the same names as the attributes. Besides, the order of the values for the attributes in the constructor must follow the specified order. For example, my test module will include instructions such as:

orc1=Orc("Ogrorg",4.3, True)

orc2=Orc("Grunch",3.3, False)

orc2.strenght=5.2

orc1.fight(orc2)

The same applies to the classes *Archer* and *Knight* and of course, the code above is just a small representation of my testing. However, it gives you a good example of the importance of using the correct names in your implementation.

If you do not follow the above-mentioned instructions, (for example, rather than *strength,* you call to your attribute *power)* you will get the corresponding marks reduction of the part that is implemented with the wrong name.

Evaluation Details:

The maximum marks of this assignment is 30 marks. Marks breakdown:

[up to 15 marks] Right functionalities of your classes. Including: access/modification of attributes, error and values range testing, operator overload, objects printing, specific functionality associated to each class.

Opt 1.- **[15 marks]** Software works perfectly, there are not mistakes.

Opt 2. -**[up to 10 marks]** Software works well but a/some functionalities do not work exactly how it should be.

Opt 3. -**[up to 5 marks]** Software works well but a/some functionalities do not work.

Opt 4. -**[0 marks]** Software with many mistakes.

[up to 15 marks] Right design of inheritance and or/composition for your classes. Including: code re-use and overriding of methods.

Opt 1.- **[15 marks]** Inheritance is perfectly done, there are not mistakes.

Opt 2. -**[up to 10 marks]** Something in the inheritance design is not exactly how it should be.

Opt 3. -**[up to 5 marks]** Several mistakes in the inheritance design and implementation.

Opt 4. -**[0 marks]** Inheritance design and implementation with many mistakes.