UniSA STEM

COMP 1046: Object-Oriented Programming

Assignment Part 2: Implementation

University of South Australia

Copyright

This material has been reproduced and communicated to you by or on behalf of the University of South Australia in accordance with section 113P of the Copyright Act 1968 (Act). The material in this communication may be subject to copyright under the Act. Any further reproduction or communication of this material by you may be the subject of copyright protection under the Act.

Marking Criteria

Task	Mark
Header: • Comment with student details (full name, student id, username) for main code file and test file	No assignment mark if not provided
Output: • Appropriate behaviour/ no errors occur.	10
Object-Oriented Principles: • All classes correctly identified, and mirror real world objects as defined in specification. • Relationships are correctly implemented (association, aggregation, composition, and inheritance). • Abstract classes are correctly implemented.	35
Docstrings: • Single line docstring per method (except initialiser, getters, or setters) • Multiline class docstring per class.	10
Code Consistency & Style: • Classes are correctly named. (Singular, TitleCase and meaningful) • Attributes and methods have a consistent naming convention (snake_case / camelCase) • Minimal to no use of code duplication. • Strings written with “ ” • Consistent use of tabs or spaces used for indentation	10
Properties: • Correctly implement a property for your values which accesses getters and setters (if needed). • Correct use of properties.	10
Git Version Control: • 10 commits provided. • Commits must be over (at least) over 5 different days. • Commits must include an appropriate message.	No assignment mark if not provided. 10
Testing: • Tests are written in a separate file in the same folder as main code • file correctly tests code using either pytest or unittest modules. • Each method is tested • Testing is appropriate to the given method.	15

Penalty for late submission:

See course outline section “Penalties for late submission”.

No mark for the assignment will be given if:

- File is not written in Python code
- OOP principles have not been applied at all
- Main code is not in a single file
- Test code is not in a separate file

Specifications

The assignment is intended to provide you with the opportunity to put into practice what you have learnt in the course by applying your object-oriented programming knowledge and skills. It will require you to apply object-oriented methods to design, implement, and test a text-based crafting application to create weapons and enchantments.

The programming assignment is worth for 20% of the assessments in this course.

Submission

- Submitted to learn online on the course website.
- This task is to be completed and submitted individually.
- Submissions are only accepted as a zipped file including:
    - Two Python (*.py) files containing all classes and code
        - One file for the main code
        - Second for test code
    - Git repository folder .git
- The folder must be named with your username (email ID): <email001>.zip

Due Date

Please see course website for details.

Note

This document is a specification of the required end product that will be generated by implementing the assignment. Like many software specifications, it is written in UML and plain English and hence will contain some imperfectly specified parts. Please make sure you seek clarification if you are not clear on any aspect of this assignment.

Course Objectives

By completing this assignment, you are expected to partially fulfill the following course objects:

- CO1. Convert a problem statement into an object oriented solution
- CO2. Use inheritance and polymorphism to solve problems
- CO3. Debug and fix code defects

- CO4. Apply international coding style standards
- CO5. Plan and conduct testing of software
- CO6. Analyse and explain the behaviour of object oriented programs

Learning Outcomes

This assignment is for assessing the following learning outcomes:

- Implement an object-oriented designed software in Python.
- Apply object-oriented principles including abstraction, data hiding, encapsulation, inheritance, and polymorphism to software implementation.
- Practice Python programming skills including commenting with docstrings, handling exceptions, conducting unit tests, and version control with git.

Extensions

See course outline and course introduction slides in O-Week.

Academic Misconduct

Note: This assignment is an individual task that will require an individual submission.
Academic integrity is the foundation of university life and is fundamental to the reputation of UniSA and its staff and students. Academic integrity means a commitment by all staff and students to act with honesty, trustworthiness, fairness, respect, and responsibility in all academic work.

Students are reminded that they should be aware of the academic misconduct guidelines available from the University of South Australia website. Deliberate academic misconduct, such as plagiarism, is subject to penalties. Information about Academic Integrity and what constitutes academic misconduct can be found in the Academic Integrity Policy and Procedure (https://i.unisa.edu.au/policies-and-procedures/university-policies/academic/ab-69). To learn more on academic integrity and how to avoid academic misconduct, please refer to the Academic Integrity Module: https://lo.unisa.edu.au/mod/book/view.php?id=252142

Work Plan

To complete of this assignment, you will need to perform the following steps:

1. Read the assignment documentation carefully.
2. Setup your project folder and add the necessary .py files.
3. Initialise your git repository and make your first commit with the empty project files.
4. Create placeholder code for all the classes specified in the provided UML diagram. Then do the same for any methods and attributes (using pass statement where necessary to avoid errors).
5. Commit every time when you have finished implementing a functionality or at latest by end of each day when you work on it.

6. Write placeholder docstrings for your classes and methods – You can change these as you go but it may help to further understand what must be done in these classes/methods. If you do this later, you may forget something, and it is more effort to insert documentation at a later stage.
7. Remember to use consistent coding style and make frequent commits to git repository.
8. Make sure your code adheres to the OOP design principles, including abstraction, data hiding, encapsulation, inheritance, and polymorphism.
9. Start testing the software early by writing tests for methods as you develop them.
10. Read and test your code thoroughly.
11. Submit your assignment.

Requirements

1. Correctly setup your Python/VSCode project. You must use Python 3.10+.
2. Your submission must implement all the features to correctly produce the behaviours and output as specified in the documentation.
3. Include a header file for all .py files.

```
'''
File: filename.py
Description: A brief description of this Python module.
Author: Billy Bizilis
StudentID: 110100110
EmailID: bizvy001
This is my own work as defined by the University's Academic Misconduct Policy.
'''
```

4. Doctstrings - Your code must be documented appropriately using docstrings. If you are not familiar with docstrings, please review week 1 tutorial. Each class and each method should have one docstring which briefly describes it.
5. Make sure your coding style is consistent throughout.
6. Properties – Make sure you use properties where specified.
7. Git - You must use git version control to keep track of your progress during implementation. You should perform regular commits as you implement features and fix errors (at least 10 commits must be made). Ensure each commit comment contains a short subject line. Your commit comments should reflect the context of the changes that were made in each commit—a fellow developer can always diff the contents to see exactly what changed but that does not provide the context. Your submission should comprise your entire version control repository specific to the assignment. For example, you must create the git repository in the directory for your assignment project. Do not commit your assignment to a repository created in a parent folder or any other location.
8. Testing - Your submission must include a separate Python module for testing. It is recommended that you test individual methods as you go, rather than trying to test the whole application. You can choose your preferred module for testing (ie. Pytest or Unittest).

Description

This assignment reflects what you might find in any common RPG game. You will implement the behaviour of an alchemist who is owned by a laboratory. In the laboratory an alchemist can mix and drink potions as well collect and refine reagents. Potions and reagents are stored in a laboratory and there different types of potions and reagents as shown in Figure 1.

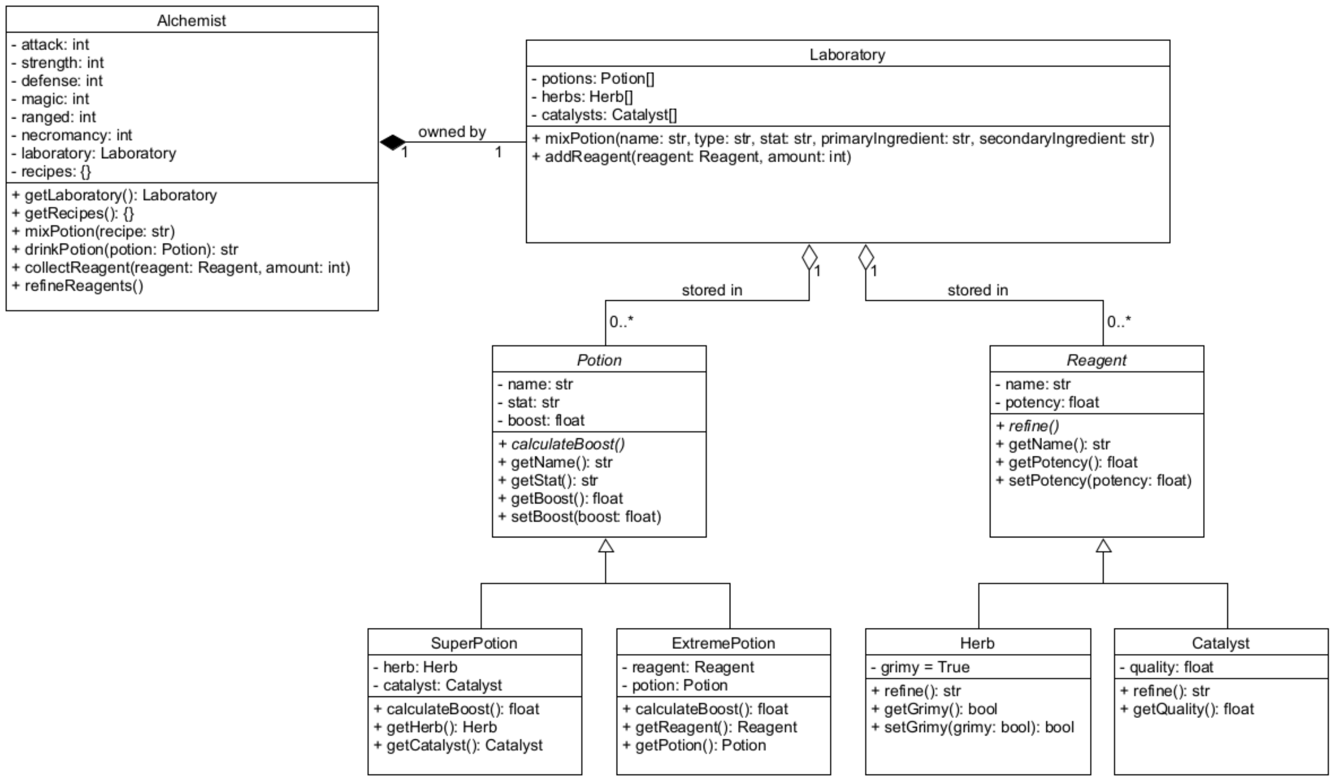

Figure 1: UML Class diagram of the Alchemist game.

Alchemist

The attributes in Alchemist from attack to necromancy are values between 0 and 100 which indicate the strength of that attribute. An alchemist knows the following recipes that can create two types of potions, super potions and extreme potions. The recipe of a super potions consists of a herb and a catalyst. The recipe of an extreme potion consists of a herb or catalyst and a super potion:

Super Attack: Irit, Eye of Newt
Super Strength: Kwuarm, Limpwurt Root
Super Defence: Cadantine, White Berries
Super Magic: Lantadyme, Potato Cactus
Super Ranging: Dwarf Weed, Wine of Zamorak
Super Necromancy: Arbuck, Blood of Orcus
Extreme Attack: Avantoe, Super Attack

Extreme Strength: Dwarf Weed, Super Strength
Extreme Defence: Lantadyme, Super Defence
Extreme Magic: Ground Mud Rune, Super Magic
Extreme Ranging: Grenwall Spike, Super Ranging
Extreme Necromancy: Ground Miasma Rune, Super Necromancy

Herbs with their potency value:
Arbuck 2.6
Avantoe 3.0
Cadantine 1.5
Dwarf Weed 2.5
Irit 1.0
Kwuarm 1.2
Lantadyme 2.0
Torstol 4.5

Catalysts with their potency and quality value:
Eye of Newt 4.3, 1.0
Limpwurt Root 3.6, 1.7
White Berries 1.2, 2.0
Potato Cactus 7.3, 0.1
Wine of Zamorak 1.7, 5.0
Blood of Orcus 4.5, 2.2
Ground Mud Rune 2.1, 6.7
Grenwall Spike 6.3, 4.9
Ground Miasma Rune 3.3, 5.2

An alchemist can mix a potion and will make use of the functionality in a laboratory for it.

If an alchemist drinks a potion then it will increase their attack, strength, defence, magic, range, or necromancy depending on what the potion's status is.

When an alchemist collects a reagent then they will add it to its laboratory.

Refining reagents means that an alchemist will refine all herbs and catalysts in the laboratory.

Laboratory
The laboratory stores the potions, herbs, and catalysts.

It provides functionality for an alchemist to mix potions: Each potion that is mixed has a name and its first ingredient is either a herb or a catalyst that is stored in the laboratory. The second ingredient is either a catalyst or another potion. Depending on the potion type, the proper potion must be created. Which attribute of an alchemist the potion is increasing is specified in the status of the potions.

Reagents can be added to the laboratory. To keep it simple, this functionality specifies the amount of reagents being available in the laboratory.

Potion
A potion has a name, what kind of attribute of an alchemist it can increase (i.e., stat attribute), and by how much the attribute is increased (i.e., boost attribute).

The boost is calculated differently depending on if it is a super potion or an extreme potion:

- Super potion: *potency of its herb + (potency of its catalyst * quality of its catalyst) * 1.5*. The result should be rounded by 2 decimals.
- Extreme potion: *(potency of its reagent * boost value of its super potion) * 3.0*. The result should be rounded by two decimals.

Reagent
A reagent has a name and a potency value. A reagent can be refined but it depends on which type of reagent it is and the refinement will result in something different:

- Herb: Refining leads to a herb that is not grimy and will multiply its existing potency by 2.5. It will print this information on the screen as well.
- Catalyst: If the quality of the catalyst is below 8.9 then its quality will be increased by 1.1 and this information is printed on the screen. If its quality is equal or above 8.9 then its quality is set to 10. It should print on the screen the quality and say that "it cannot be refined any further".

Testing & Outputs
You need to test your implementation and come up with test cases. This is an important task in software development. For details on testing, please revise content of Week 11.