Lab 4: Objects and Inheritance

INFSCI 0201
Intermediate Programming (Spring 2025)

In this week's lab, we are going to put the concept of inheritance in object-oriented programming into practice in a playful way. You are going to be a game developer this time and develop a RPG game. However, game development is a complex job, so you will only be doing a small part of the job: implementing an item system for the characters.
You will get some hands-on experience in creating a hierarchy of class.

Submission Details

You are to write a Python program that meets the requirements outlined in the Lab 4 Tasks section.

- As with all programming assignments in this course, you must submit Lab 4 through GitHub in the git repository the instructor created for you. You must place your Lab 4 eclipse project in the folder **/labs/lab04/**
- If you did this all correctly, this project will be in the directory **/labs/lab04/**

Testing

There will be four main tests for this lab, which cover the following operations:

1. There is are a correct inheritance hierarchy between item, weapon, shield, and potion class.
2. The item, weapon, shield, and potion classes are correctly implemented
3. An alternative constructor for the potion class is implemented
4. Naming conventions are followed.

Note: Be sure you're strictly adhering to the Python coding style guide on Canvas.

Lab 4 task

In this lab, you'll create a basic RPG item system. The systems include all the items that a player character can carry in their backpack (inventory) or equip on the character themselves. Here is the detailed list from the game design team:

- **All of the items should have a name, a description, a rarity, and an ownership**.
- All of the items should be able to be picked up, thrown away and use.
- The rarity of items has four different levels: common, uncommon, epic, and legendary.
- When an item is picked up by a character, the ownership of the item changes to the character's name.
- When an item is thrown away, the item will lose its ownership. The ownership should indicate False in logic operations or boolean checks.
- All items fall into one of the 3 categories in the game:
    - Weapon
    - Shield
    - Potion
- **Weapons can be equipped**. When equipped, the weapon is considered active.
- Weapons have their **damage** numbers, a **hidden** passive **attack modifier** with a default value of 1.0 for common, uncommon, and epic rarity and 1.15 for legendary rarity. Weapons also have a type (i.e., ‘bow’, ‘sword’, ‘hammer’, etc.)
- Only equipped weapons can be used to attack others
- **Shields can also be equipped**. When equipped, the shield is considered active.
- Shields have **defense** numbers, a **hidden** passive **defense modifier** with a default value of 1.0 for common, uncommon, and epic rarity and 1.10 for legendary rarity.
- Shields also have an indicator for whether they are broken; broken shields have a hidden modifier of 0.5 for their defense power.
- Only equipped shields can be used to block damages
- **A potion can only be consumed but cannot be equipped**.
- The description of potions should describe what the potion does.
- Potions are destroyed after usage.
- Potions have **value** attributes indicating the changes in numbers they apply.
- There are three types of potions for now: attack, defense, and HP.
    - **Attack** Potion will add (value) attack powers when used
    - **Defence** Potion will add (value) defense powers when used
    - **HP** potion will restore (value) health when used.
- Potions also have an effective time (seconds), which indicates how long the effects will last. If the effective time is 0, that means the potion is a used-and-done type.
    - Most HP potions are use-and-done, while all the attack and defense potions have a positive effective time.
- Potions can also be created from player abilities: these ability-generated potions are all in common rarity, with fixed 50 in value, 30s in effective time, and owned by the player who generated them.

Coding Tasks

- You will design and implement a group of classes with a hierarchy that fits what is described by the game design team.
- Here is the UML Diagram for the item class for reference:

item
+ name + description = ‘’ (default to an empty string) + rarity = ‘common’ - _ownership = ’’ (default to an empty string)
+ pick_up(self, character: str) -> str + throw_away(self) -> str + use(self) -> str (return a string saying the item is used)

- Weapon, shield, and potion objects should all have “is a” relationship to the item class.
- Weapon/shield should have a method named `equip()`
    - Calling this method will print out a sentence saying the weapon (name) is equipped.
- When a weapon is used, its attacking power is calculated as **damage * attack modifier.**
- When a shield is used, its defense power is calculated as **defense * defense modifier (* broken modifier).**
- After a Potion is used (use() method called)
    - Calling this method should print out a sentence saying the potion (name) is consumed.
    - The Potion should be marked for destruction; you can set a dedicated flag for the used Potion waiting for destruction.
- All items should generate a user-friendly string when being printed out.
- If an item has no owner, the use of that item should yield no result.

Example:

```
belthronding = Weapon(name='Belthronding', rarity='legendary', damage
= 5000, type = 'bow')
long_bow.pick_up('Beleg') # Belthronding is now owned by Beleg
long_bow.equip() # Belthronding is equipped by Beleg
long_bow.use() # Belthronding is used, dealing 5750 damage
```

```
broken_pot_lid = Shield(name='wooden lid', description='A lid made of
wood, useful in cooking. No one will choose it willingly for a
shield', defense = 5, broken = True)
long_bow.pick_up('Beleg') # wooden lid is now owned by Beleg
broken_pot_lid.equip() # wooden lid is equiped by Beleg
broken_pot_lid.use() # wooden lid is used, blocking 2.5 damage
broken_pot_lid.throw_away() # wooden lid is thrown away
broken_pot_lid.use() # NO OUTPUT

attack_potion = Potion.from_ability(name='atk potion temp', owner =
'Beleg', type='attack')
attack_potion.use() # Beleg used atk potion temp, and attack increase
50 for 30s
attack_potion.use() # NO OUTPUT

isinstance(long_bow, Item) # True
isinstance(broken_pot_lid, Shield) # True
isinstance(attack_potion, Weapon) # False
```