Laboratory Activity No. 6

Inheritance, Encapsulation, and Abstraction

Course Code: CPE103	**Program:** BSCPE
Course Title: Object-Oriented Programming	**Date Performed:** 2 / 21 / 2025
Section: 1-A	**Date Submitted:** 2 / 21 / 2025
Name: Asugas, Kenneth R.	**Instructor:** Engr. Maria Rizette H. Sayo

1. Objective(s):

This activity aims to familiarize students with the concepts of Object-Oriented Programming

2. Intended Learning Outcomes (ILOs):

The students should be able to:

2.1 Identify the possible attributes and methods of a given object

2.2 Create a class using the Python language

2.3 Create and modify the instances and the attributes in the instance.

3. Discussion:

Object-Oriented Programming (OOP) has 4 core Principles: Inheritance, Polymorphism, Encapsulation, and Abstraction. The main goal of Object-Oriented Programming is code reusability and modularity meaning it can be reused for different purposes and integrated in other different programs. These 4 core principles help guide programmers to fully implement Object-Oriented Programming. In this laboratory activity, we will be exploring Inheritance while incorporating other principles such as Encapsulation and Abstraction which are used to prevent access to certain attributes and methods inside a class and abstract or hide complex codes which do not need to be accessed by the user.

An example is given below considering a simple UML Class Diagram:
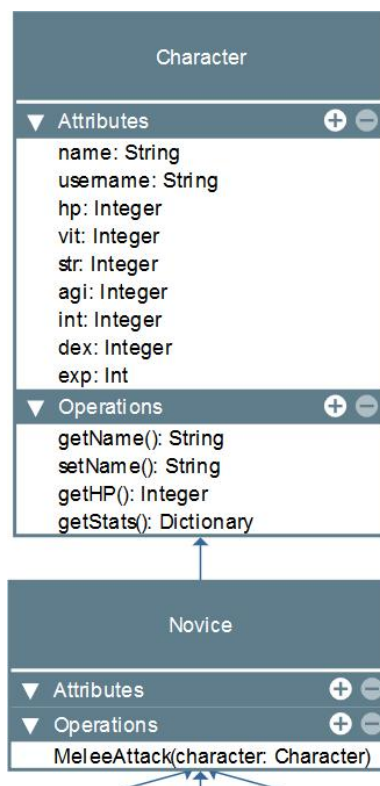
The Base Character class will contain the following attributes and methods and a Novice Class will become a child of Character. The OOP Principle of Inheritance will make Novice have all the attributes and methods of the Character class as well as other

unique attributes and methods it may have. This is referred to as Single-level Inheritance. In this activity, the Novice class will be made the parent of three other different classes Swordsman, Archer, and Magician. The three classes will now possess the attributes and methods of the Novice class which has the attributes and methods of the Base Character Class. This is referred to as Multi-level inheritance.
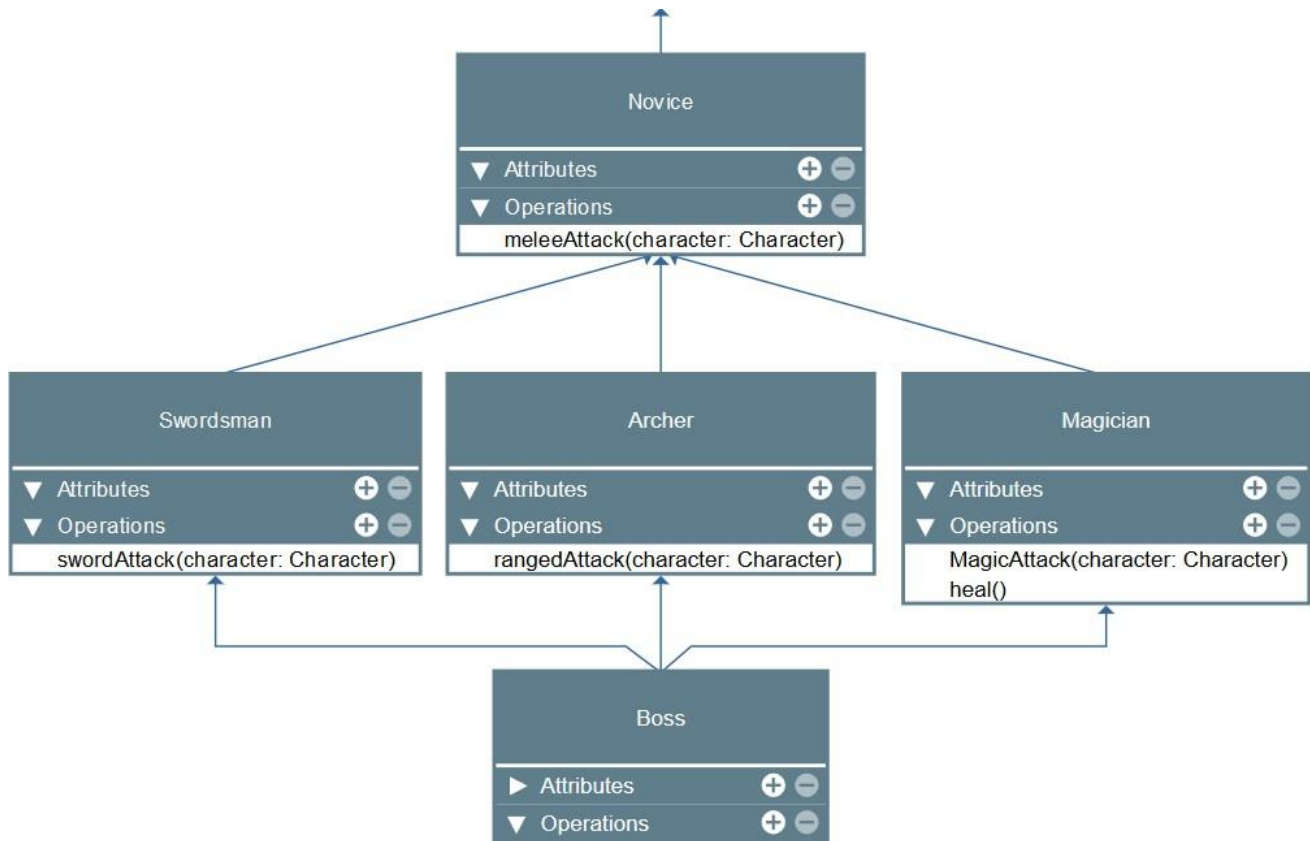
The last type of inheritance that will be explored is the Boss class which will inherit from the three classes under Novice. This Boss class will be able to use any abilities of the three Classes. This is referred to as Multiple inheritance.

4. Materials and Equipment:

Desktop Computer with Anaconda Python
Windows Operating System

5. Procedure:

Creating the Classes
1. Inside your folder **oopfa1_<lastname>**, create the following classes on separate .py files with the file names: Character, Novice, Swordsman, Archer, Magician, Boss.
2. Create the respective class for each .py files. Put a temporary pass under each class created except in Character.py
    Ex.
    class Novice():
        pass
3. In the Character.py copy the following codes

```
 1 class Character():
 2     def __init__(self, username):
 3         self.__username = username
 4         self.__hp = 100
 5         self.__mana = 100
 6         self.__damage = 5
 7         self.__str = 0 # strength stat
 8         self.__vit = 0 # vitality stat
 9         self.__int = 0 # intelligence stat
10         self.__agi = 0 # agility stat
11     def getUsername(self):
12         return self.__username
13     def setUsername(self, new_username):
14         self.__username = new_username
15     def getHp(self):
16         return self.__hp
17     def setHp(self, new_hp):
18         self.__hp = new_hp
19     def getDamage(self):
20         return self.__damage
21     def setDamage(self, new_damage):
22         self.__damage = new_damage
23     def getStr(self):
24         return self.__str
25     def setStr(self, new_str):
26         self.__str = new_str
27     def getVit(self):
28         return self.__vit
29     def setVit(self, new_vit):
30         self.__vit = new_vit
31     def getInt(self):
32         return self.__int
33     def setInt(self, new_int):
34         self.__int = new_int
35     def getAgi(self):
36         return self.__agi
37     def setAgi(self, new_agi):
38         self.__agi = new_agi
39     def reduceHp(self, damage_amount):
40         self.__hp = self.__hp - damage_amount
41     def addHp(self, heal_amount):
42         self.__hp = self.__hp + heal_amount
```

Note: The double underscore ___signifies that the variables will be inaccessible outside of the class.

4. In the same Character.py file, under the code try to create an instance of Character and try to print the username
   Ex.
   character1 = Character("Your Username")
   print(character1._username)
   print(character1.getUsername())
5. Observe the output and analyze its meaning then comment the added code.

   It worked on creating a instance of the Character class and printing username from there, however when trying to print username from it directly, the error message came up because the username attribute is private, but using the get Username() method worked just fine, and it shows an example of encapsulation, removing the internal details of a class.

Single Inheritance
1. In the Novice.py class, copy the following code.

```
1 from Character import Character
2
3 class Novice(Character):
4     def basicAttack(self, character):
5         character.reduceHp(self.getDamage())
6         print(f"{self.getUsername()} performed Basic Attack! -{self.getDamage()}")
```

2. In the same Novice.py file, under the code try to create an instance of Character and try to print the username
   Ex.
   character1 = Novice("Your Username")
   print(character1.getUsername())
   print(character1.getHp())
3. Observe the output and analyze its meaning then comment the added code.

   You could create an instance of Novice class, printing the username and HP, to see that Novice inherited attributes and methods from Character. That is the only way to demonstrate single inheritance, where a child class inherits from one parent class.

Multi-level Inheritance
1. In the Swordsman, Archer, and Magician .py files copy the following codes for each file:
Swordsman.py

```
1 from Novice import Novice
2
3 class Swordsman(Novice):
4     def __init__(self, username):
5         super().__init__(username)
6         self.setStr(5)
7         self.setVit(10)
8         self.setHp(self.getHp()+self.getVit())
9
10     def slashAttack(self, character):
11         self.new_damage = self.getDamage()+self.getStr()
12         character.reduceHp(self.new_damage)
13         print(f"{self.getUsername()} performed Slash Attack! -{self.new_damage}")
```

Archer.py

```
1 from Novice import Novice
2 import random
3
4 class Archer(Novice):
5     def __init__(self, username):
6         super().__init__(username)
7         self.setAgi(5)
8         self.setInt(5)
9         self.setVit(5)
10        self.setHp(self.getHp()+self.getVit())
11
12     def rangedAttack(self, character):
13         self.new_damage = self.getDamage()+random.randint(0,self.getInt())
14         character.reduceHp(self.new_damage)
15         print(f"{self.getUsername()} performed Slash Attack! -{self.new_damage}")
```

Magician.py

```
1 from Novice import Novice
2
3 class Magician(Novice):
4     def __init__(self, username):
5         super().__init__(username)
6         self.setInt(10)
7         self.setVit(5)
8         self.setHp(self.getHp()+self.getVit())
9
10    def heal(self):
11        self.addHp(self.getInt())
12        print(f"{self.getUsername()} performed Heal! +{self.getInt()}")
13
14    def magicAttack(self, character):
15        self.new_damage = self.getDamage()+self.getInt()
16        character.reduceHp(self.new_damage)
17        print(f"{self.getUsername()} performed Magic Attack! -{self.new_damage}")
```

2. Create a new file called Test.py and copy the codes below:

```
1 from Swordsman import Swordsman
2 from Archer import Archer
3 from Magician import Magician
4
5
6 Character1 = Swordsman("Royce")
7 Character2 = Magician("Archie")
8 print(f"{Character1.getUsername()} HP: {Character1.getHp()}")
9 print(f"{Character2.getUsername()} HP: {Character2.getHp()}")
10 Character1.slashAttack(Character2)
11 Character1.basicAttack(Character2)
12 print(f"{Character1.getUsername()} HP: {Character1.getHp()}")
13 print(f"{Character2.getUsername()} HP: {Character2.getHp()}")
14 Character2.heal()
15 Character2.magicAttack(Character1)
16 print(f"{Character1.getUsername()} HP: {Character1.getHp()}")
17 print(f"{Character2.getUsername()} HP: {Character2.getHp()}")
```

3. Run the program Test.py and observe the output.
4. Modify the program and try replacing Character2.magicAttack(Character1) with Character2.slashAttack(Character1) then run the program again and observe the output.

Instances of the classes Swordsman, Archer and Magician were able to inherit attributes and methods from both Novice and Character. By removing Character2. magicAttack(Character1) and replacing it with Character2. slashAttack(Character1) we showed that each subclass of the character could use its own methods which emphasizes multi-level inheritance.

Multiple Inheritance

1. In the Boss.py file, copy the codes as shown:

```
1 from Swordsman import Swordsman
2 from Archer import Archer
3 from Magician import Magician
4
5 class Boss(Swordsman, Archer, Magician): # multiple inheritance
6     def __init__(self, username):
7         super().__init__(username)
8         self.setStr(10)
9         self.setVit(25)
10        self.setInt(5)
11        self.setHp(self.getHp()+self.getVit())
```

2. Modify the Test.py with the code shown below:

```
1 from Swordsman import Swordsman
2 from Archer import Archer
3 from Magician import Magician
4 from Boss import Boss
5
6 Character1 = Swordsman("Royce")
7 Character2 = Boss("Archie")
8 print(f"{Character1.getUsername()} HP: {Character1.getHp()}")
9 print(f"{Character2.getUsername()} HP: {Character2.getHp()}")
10 Character1.slashAttack(Character2)
11 Character1.basicAttack(Character2)
12 print(f"{Character1.getUsername()} HP: {Character1.getHp()}")
13 print(f"{Character2.getUsername()} HP: {Character2.getHp()}")
14 Character2.heal()
15 Character2.basicAttack(Character1)
16 Character2.slashAttack(Character1)
17 Character2.rangedAttack(Character1)
18 Character2.magicAttack(Character1)
19 print(f"{Character1.getUsername()} HP: {Character1.getHp()}")
20 print(f"{Character2.getUsername()} HP: {Character2.getHp()}")
```

3. Run the program Test.py and observe the output.

To make things clear, the Boss class inherits attributes and methods from Swordsman, Archer and Magician. Run Test.py shows that the Boss class can use abilities from all three classes and represents multiple inheritance.

For the program please refer to this link: https://github.com/Kenneth-Asugas/CPE-103-OOP-1-A/blob/main/Lab_6.ipynb

6. Supplementary Activity:

Task

Create a new file Game.py inside the same folder use the pre-made classes to create a simple Game where two players or one player vs a computer will be able to reduce their opponent's hp to 0.

Requirements:
1. The game must be able to select between 2 modes: Single player and Player vs Player. The game can spawn multiple matches where single player or player vs player can take place.
2. In Single player:
   - the player must start as a Novice, then after 2 wins, the player should be able to select a new role between Swordsman, Archer, and Magician.
   - The opponent will always be a boss named Monster.
3. In Player vs Player, both players must be able to select among all the possible roles available except Boss.
4. Turns of each player for both modes should be randomized and the match should end when one of the players hp is zero.
5. Wins of each player in a game for both the modes should be counted.

   For the program please refer to this link: https://github.com/Kenneth-Asugas/CPE-103-OOP-1-A/blob/main/Lab_6.ipynb

Questions
1. Why is Inheritance important?

   Inheritance also allows for code reuse and modularity as child classes have attributes and methods inherited from their parent classes. This makes code more readable, more maintainable and easier to manage.

2. Explain the advantages and disadvantages of using applying inheritance in an Object-Oriented Program.

   Advantages: Inheritance supports code reuse, modularity, and scalability. Disadvantages: the classes become tightly coupled, you may have more complexity / deep hierarchies and also less flexibility.

3. Differentiate single inheritance, multiple inheritance, and multi-level inheritance.

   A class inherits from one parent. Multiple Inheritance: A class inherits from multiple parents Multi - level Inheritance: A class inherits from another class forming a hierarchy.

4. Why is super().__init__(username) added in the codes of Swordsman, Archer, Magician, and Boss?

   It initializes the parent class with the username given and check for proper implementation of inherited attributes / methods.

5. How do you think Encapsulation and Abstraction helps in making good Object-Oriented Programs?

   Encapsulation: Cover up internal details in order to maintain data integrity and improve security. Abstraction: Recommendation that all complex systems be pared down to their key aspects so that the code is easier to understand and maintain.

7. Conclusion:

In this activity we presented inheritance, encapsulation and abstraction. Inheritance can be used to encourage reuse of code, while encapsulation can help to protect data, while abstraction can help us overcome problems in software development. These principles can lead to efficient, organized, and maintainable code, improving software design and development.

8. Assessment Rubric: