Laboratory Activity No. 2	
Inheritance, Encapsulation, and Abstraction	
Course Code: CPE009	**Program:** BSCPE
Course Title: Object-Oriented Programming	**Date Performed: 09/29/24**
Section: CPE21S4	**Date Submitted: 09/30/24**
Name: Magistrado, Aira Pauleen M.	**Instructor:** Maria Rizette 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:
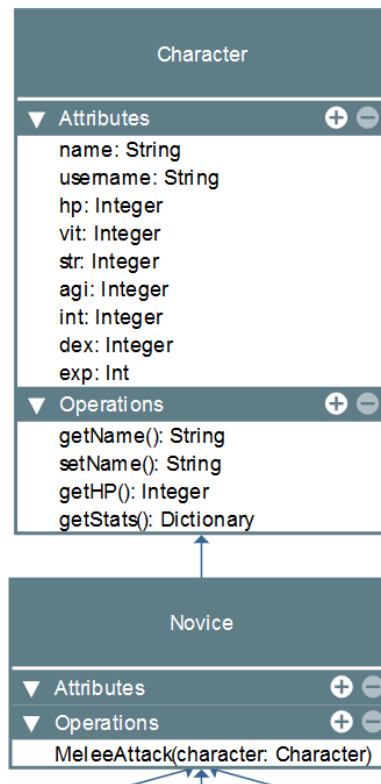
The Base Character class will contain the following attributes and methods and a Novice Class will become a child of Character. The OOP Principle of Inheritance will make Novice have all the attributes and methods of the Character class as well as other

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.
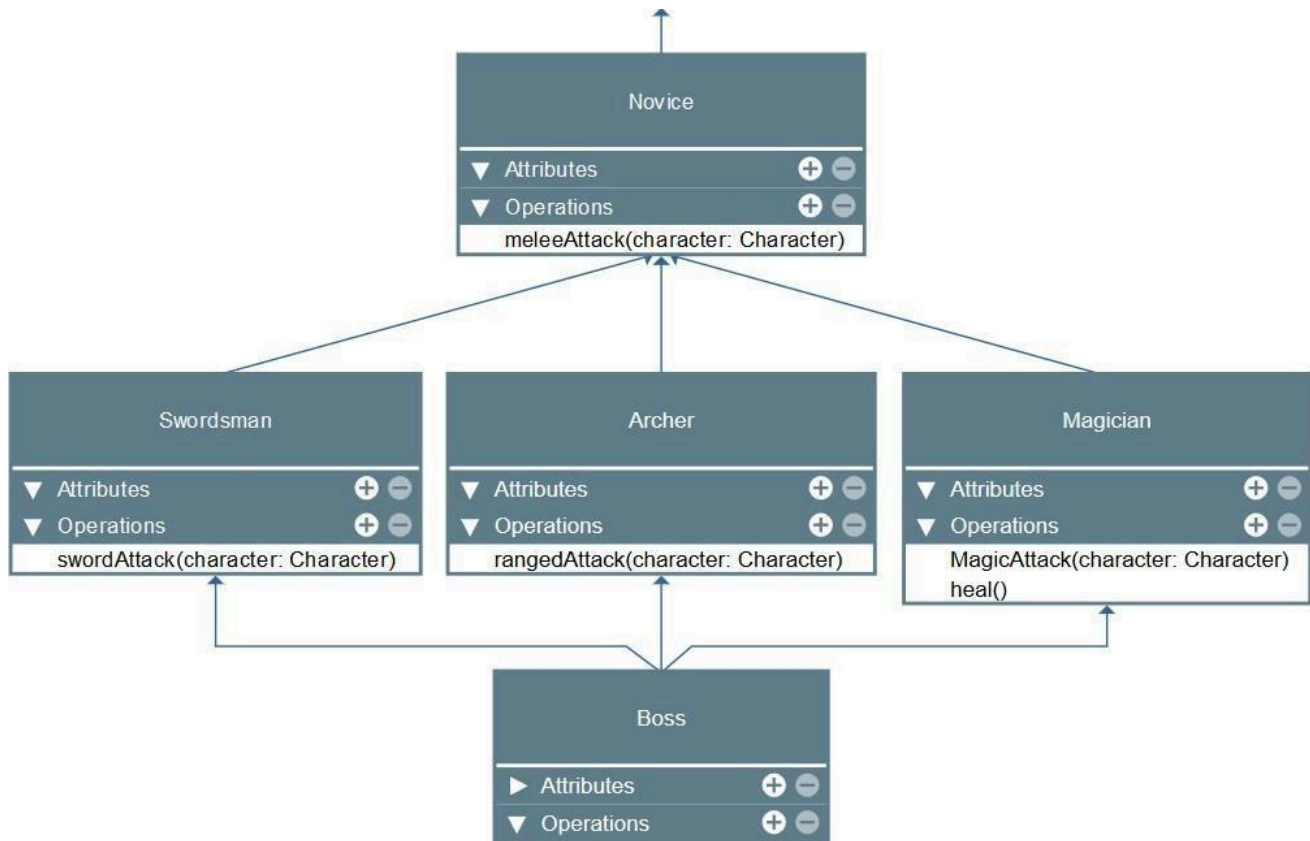
The last type of inheritance that will be explored is the Boss class which will inherit from the three classes under Novice. This Boss class will be able to use any abilities of the three Classes. This is referred to as Multiple inheritance.

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.

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.

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.

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:

```python
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.

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.

```
Select Game Mode:
1. Vs Computer
2. Vs Another Player
Enter 1 or 2: 1
Enter user: s
s's turn:
Computer's turn:
Computer performed Basic Attack! -5
s's turn:
s's turn:
Computer's turn:
Computer performed Basic Attack! -5
Computer's turn:
Computer performed Basic Attack! -5
Computer's turn:
Computer performed Basic Attack! -5
Computer's turn:
Computer performed Basic Attack! -5
Computer's turn:
Computer performed Basic Attack! -5
Computer's turn:
Computer performed Basic Attack! -5
Computer's turn:
Computer performed Basic Attack! -5
s's turn:
Computer's turn:
Computer performed Basic Attack! -5
s's turn:
Computer's turn:
```

```
Ai performed Basic Attack! -5
Computer's turn:
Computer performed Basic Attack! -5
Ai's turn:
Ai performed Basic Attack! -5
Computer's turn:
Computer performed Basic Attack! -5
Computer's turn:
Computer performed Basic Attack! -5
Ai's turn:
Computer's turn:
Computer performed Basic Attack! -5
Computer's turn:
Computer performed Basic Attack! -5
Computer's turn:
Computer performed Basic Attack! -5
Computer's turn:
Computer performed Basic Attack! -5
Ai's turn:
Ai performed Basic Attack! -5
Ai's turn:
Ai's turn:
Computer's turn:
Computer performed Basic Attack! -5
Computer's turn:
Computer performed Basic Attack! -5
Ai has been defeated!
Game Over! Your Score: 1, Computer Score: 0
```

```
Select Game Mode:
1. Vs Computer
2. Vs Another Player
Enter 1 or 2: 2
Player 1, Please enter your username: A
Select your role:
1. Novice
2. Swordsman
3. Archer
4. Magician
Enter 1, 2, 3, or 4: 1
Player 2, Please enter your username: B
Select your role:
1. Novice
2. Swordsman
3. Archer
4. Magician
Enter 1, 2, 3, or 4: 2
B's turn:
B's turn:
B performed Slash Attack! -10
B's turn:
A's turn:
B's turn:
A's turn:
B's turn:
B performed Basic Attack! -5
B's turn:
B performed Slash Attack! -10
```

```
A performed Basic Attack! -5
A's turn:
B's turn:
B performed Slash Attack! -10
A's turn:
B's turn:
A's turn:
B's turn:
B's turn:
B's turn:
B performed Slash Attack! -10
B's turn:
A's turn:
B's turn:
A's turn:
B's turn:
A's turn:
B's turn:
B performed Basic Attack! -5
A's turn:
A performed Basic Attack! -5
A's turn:
A's turn:
B's turn:
B performed Slash Attack! -10
A has been defeated!
Player 1 Wins: 1, Player 2 Wins: 0
Game Over!
```

```python
import random
from Swordsman import Swordsman
from Archer import Archer
from Magician import Magician
from Boss import Boss
from Novice import Novice

class Game:
    def __init__(self):
        self.player_wins = 0
        self.opponent_wins = 0
        self.mode = ""

    def select_mode(self):
```

```python
        print("Select Game Mode:")
        print("1. Vs Computer")
        print("2. Vs Another Player")
        choice = input("Enter 1 or 2: ")
        if choice == '1':
            self.mode = "Vs Computer"
            self.single_player_mode()
        elif choice == '2':
            self.mode = "Another Player"
            self.player_vs_player_mode()
        else:
            print("Please choose!")
            self.select_mode()

    def single_player_mode(self):
        player = Novice(input("Enter user: "))
        computer = Boss ("Computer")

        while True:
            self.battle(player, computer)
            if self.player_wins >= 2:
                print("Congratulations!!!")
                player = self.select_new_role(player.getUsername())
            else:
                break

        print(f"Game Over! Your Score: {self.player_wins}, Computer Score:
{self.opponent_wins}")

    def player_vs_player_mode(self):
        player1 = self.select_role(input("Player 1, Please enter your username: "))
        player2 = self.select_role(input("Player 2, Please enter your username: "))

        while True:
            self.battle(player1, player2)
            print(f"Player 1 Wins: {self.player_wins}, Player 2 Wins: {self.opponent_wins}")
            if player1.getHp() <= 0 or player2.getHp() <= 0:
                break

        print("Game Over!")

    def battle(self, player, opponent):
        while player.getHp() > 0 and opponent.getHp() > 0:
            attacker = random.choice([player, opponent])
            if attacker == player:
                print(f"{player.getUsername()}'s turn:")
                attack = random.choice(['basicAttack', 'magicAttack', 'slashAttack',
'rangedAttack'])
                if attack == 'basicAttack':
                    attacker.basicAttack(opponent)
                elif isinstance(attacker, Magician) and action == 'magicAttack':
                    attacker.magicAttack(opponent)
                elif isinstance(attacker, Swordsman) and action == 'slashAttack':
                    attacker.slashAttack(opponent)
                elif isinstance(attacker, Archer) and action == 'rangedAttack':
```

```python
                    attacker.rangedAttack(opponent)
                self.check_winner(attacker, opponent)
            else:
                print(f"{opponent.getUsername()}'s turn:")
                action = random.choice(['basicAttack', 'magicAttack', 'slashAttack',
'rangedAttack'])
                if isinstance(opponent, Boss):
                    action = 'basicAttack'  # Boss will only use basic attack
                if action == 'basicAttack':
                    opponent.basicAttack(player)
                elif isinstance(opponent, Magician) and action == 'magicAttack':
                    opponent.magicAttack(player)
                elif isinstance(opponent, Swordsman) and action == 'slashAttack':
                    opponent.slashAttack(player)
                elif isinstance(opponent, Archer) and action == 'rangedAttack':
                    opponent.rangedAttack(player)
                self.check_winner(opponent, player)

    def check_winner(self, attacker, defender):
        if defender.getHp() <= 0:
            print(f"{defender.getUsername()} has been defeated!")
            if isinstance(attacker, Novice):
                self.player_wins += 1
            else:
                self.opponent_wins += 1

    def select_new_role(self, username):
        print("Select new role:")
        print("1. Swordsman")
        print("2. Archer")
        print("3. Magician")
        choice = input("Enter 1, 2, or 3: ")
        if choice == '1':
            return Swordsman(username)
        elif choice == '2':
            return Archer(username)
        elif choice == '3':
            return Magician(username)
        else:
            print("Please choose a role to start the game")
            return Novice(username)

    def select_role(self, username):
        print("Select your role:")
        print("1. Novice")
        print("2. Swordsman")
        print("3. Archer")
        print("4. Magician")
        choice = input("Enter 1, 2, 3, or 4: ")
        if choice == '1':
            return Novice(username)
        elif choice == '2':
            return Swordsman(username)
        elif choice == '3':
            return Archer(username)
```

```
        elif choice == '4':
            return Magician(username)
        else:
            print("Please choose a role to start the game")
            return Novice(username)

    if __name__ == "__main__":
        game = Game()
        game.select_mode()
```

Questions
   1. Why is Inheritance important?
        Inheritance is important so that you do not have to write the code again and again. Instead of starting from scratch, you can reuse an existing class to create a new one.

   2. Explain the advantages and disadvantages of using applying inheritance in an Object-Oriented Program.
The advantage of inheritance is that the code is reusable, which saves us from having to write the same code over and over, and using it makes code maintenance easier. Its disadvantage would be that it is complex and sometimes it's hard to follow the flow of the program, making it difficult to debug and track down errors.
   3. Differentiate single inheritance, multiple inheritance, and multi-level inheritance.
Single Inheritance is where the subclass has only one parent class. In multiple inheritance, the subclass has more than one parent class. In multi-level Inheritance, the subclass inherits from a single subclass then another subclass would inherit from the first subclass.

   4. Why is super()._init_(username) added in the codes of Swordsman, Archer, Magician, and Boss?
The super()._init_(username) is the constructor of the subclass to call the parent class. It was used in the Parent classes so that you don;t have to write the username again and again.
   5. How do you think Encapsulation and Abstraction helps in making good Object-Oriented Programs?
The usage of encapsulation and abstraction makes the code more simpler. The encapsulation restricts the access to certain variables. This ensures that the data is safe from any type of unnecessary modifications. Abstraction takes care of hiding the complexity of a program by making a much simpler interface. It makes the code easier to understand and maintain.

7. Conclusion:

In this lab activity, I learned about inheritance, encapsulation, and abstraction in object-oriented programming. I observed the use of inheritance in the supplementary activity by using the Character, Novice, Swordsman, Archer, Magician, and Boss classes to create a Game.py in which players could fight a computer or another player. The single inheritance inherits from a single parent, while multi-level inheritance operates hierarchically, and multiple inheritance, such as the class Boss, inherits from multiple classes. Encapsulation and abstraction are used in the code to simplify and improve the program's maintainability and understanding.

8. Assessment Rubric: