Laboratory Activity No. 2	
Inheritance, Encapsulation, and Abstraction	
Course Code: CPE009	**Program:** BSCPE
Course Title: Object - Oriented Programming	**Date Performed:** September 29, 2024
Section: CPE21S4	**Date Submitted:** September 29, 2024
Name: Don Eleazar T. Fernandez	**Instructor:** Maria Rizette Sayo

1. Objective(s):

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

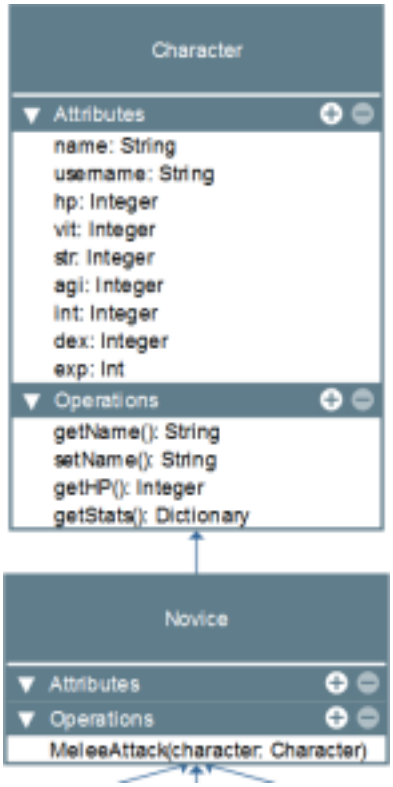
2. Intended Learning Outcomes (ILOs):

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:

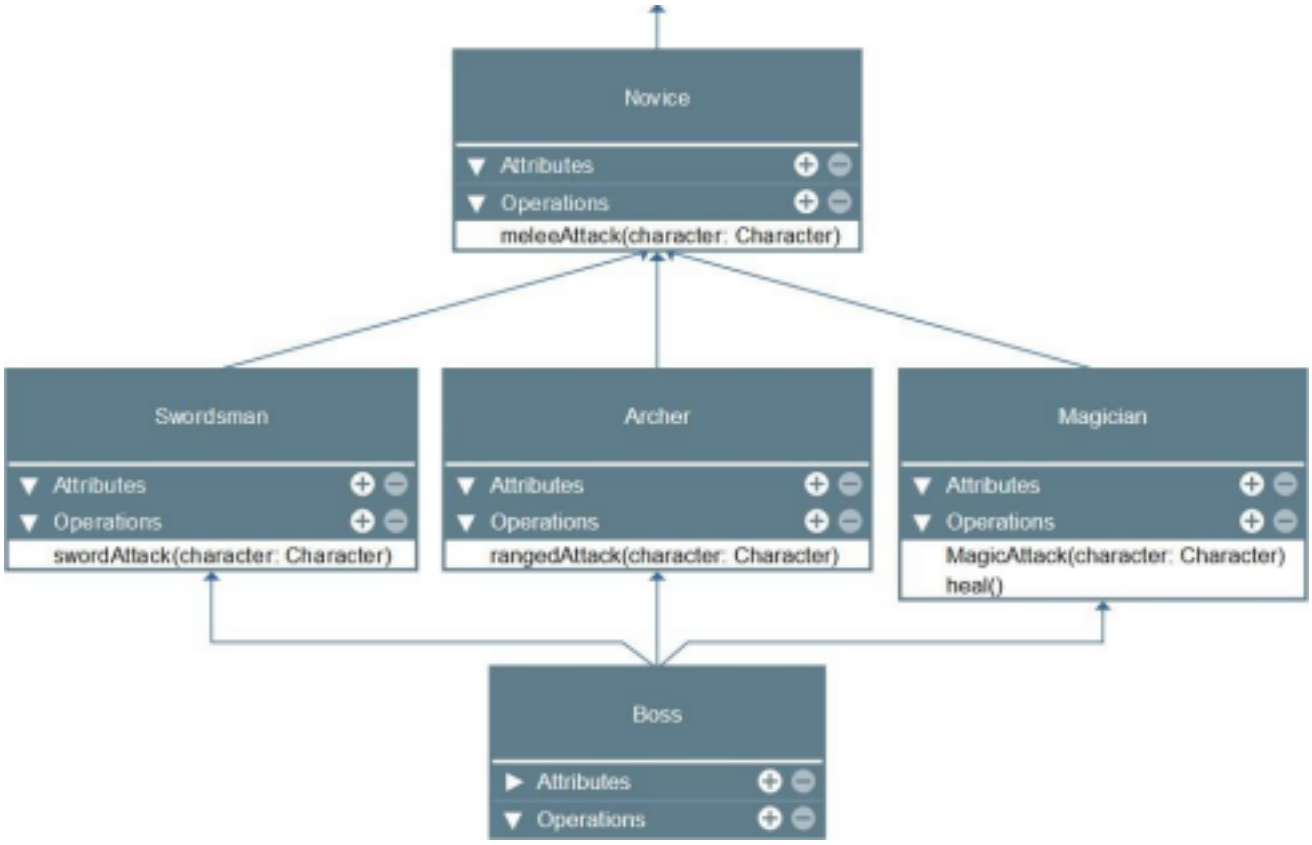
Object-Oriented Programming (OOP) has 4 core Principles: Inheritance, Polymorphism, Encapsulation, and Abstraction. The main goal of Object-Oriented Programming is code reusability and modularity meaning it can be reused for different purposes and integrated in other different programs. These 4 core 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

```python
class Character():
    def __init__(self, username):
        self.__username = username
        self.__hp = 100
        self.__mana = 100
        self.__damage = 5
        self.__str = 0 # strength stat
        self.__vit = 0 # vitality stat
        self.__int = 0 # intelligence stat
        self.__agi = 0 # agility stat
    def getUsername(self):
        return self.__username
    def setUsername(self, new_username):
        self.__username = new_username
    def getHp(self):
        return self.__hp
    def setHp(self, new_hp):
        self.__hp = new_hp
    def getDamage(self):
        return self.__damage
    def setDamage(self, new_damage):
        self.__damage = new_damage
    def getStr(self):
        return self.__str
    def setStr(self, new_str):
        self.__str = new_str
    def getVit(self):
        return self.__vit
    def setVit(self, new_vit):
        self.__vit = new_vit
    def getInt(self):
        return self.__int
    def setInt(self, new_int):
        self.__int = new_int
    def getAgi(self):
        return self.__agi
    def setAgi(self, new_agi):
        self.__agi = new_agi
    def reduceHp(self, damage_amount):
        self.__hp = self.__hp - damage_amount
    def addHp(self, heal_amount):
        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.

```python
from Character import Character

class Novice(Character):
    def basicAttack(self, character):
        character.reduceHp(self.getDamage())
        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

```python
from Novice import Novice

class Swordsman(Novice):
    def __init__(self, username):
        super().__init__(username)
        self.setStr(5)
        self.setVit(10)
        self.setHp(self.getHp()+self.getVit())

    def slashAttack(self, character):
        self.new_damage = self.getDamage()+self.getStr()
        character.reduceHp(self.new_damage)
        print(f"{self.getUsername()} performed Slash Attack! -{self.new_damage}")
```

Archer.py

```python
from Novice import Novice
import random

class Archer(Novice):
    def __init__(self, username):
        super().__init__(username)
        self.setAgi(5)
        self.setInt(5)
        self.setVit(5)
        self.setHp(self.getHp()+self.getVit())

    def rangedAttack(self, character):
        self.new_damage = self.getDamage()+random.randint(0,self.getInt())
        character.reduceHp(self.new_damage)
        print(f"{self.getUsername()} performed Slash Attack! -{self.new_damage}")
```

Magician.py

```python
from Novice import Novice

class Magician(Novice):
    def __init__(self, username):
        super().__init__(username)
        self.setInt(10)
        self.setVit(5)
        self.setHp(self.getHp()+self.getVit())

    def heal(self):
        self.addHp(self.getInt())
        print(f"{self.getUsername()} performed Heal! +{self.getInt()}")

    def magicAttack(self, character):
        self.new_damage = self.getDamage()+self.getInt()
        character.reduceHp(self.new_damage)
        print(f"{self.getUsername()} performed Magic Attack! -{self.new_damage}")
```

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

```python
from Swordsman import Swordsman
from Archer import Archer
from Magician import Magician


Character1 = Swordsman("Royce")
Character2 = Magician("Archie")
print(f"{Character1.getUsername()} HP: {Character1.getHp()}")
print(f"{Character2.getUsername()} HP: {Character2.getHp()}")
Character1.slashAttack(Character2)
Character1.basicAttack(Character2)
print(f"{Character1.getUsername()} HP: {Character1.getHp()}")
print(f"{Character2.getUsername()} HP: {Character2.getHp()}")
Character2.heal()
Character2.magicAttack(Character1)
print(f"{Character1.getUsername()} HP: {Character1.getHp()}")
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:

```python
from Swordsman import Swordsman
from Archer import Archer
from Magician import Magician

class Boss(Swordsman, Archer, Magician): # multiple inheritance
    def __init__(self, username):
        super().__init__(username)
        self.setStr(10)
        self.setVit(25)
        self.setInt(5)
        self.setHp(self.getHp()+self.getVit())
```

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

```
from Swordsman import Swordsman
from Archer import Archer
from Magician import Magician
from Boss import Boss

Character1 = Swordsman("Royce")
Character2 = Boss("Archie")
print(f"{Character1.getUsername()} HP: {Character1.getHp()}")
print(f"{Character2.getUsername()} HP: {Character2.getHp()}")
Character1.slashAttack(Character2)
Character1.basicAttack(Character2)
print(f"{Character1.getUsername()} HP: {Character1.getHp()}")
print(f"{Character2.getUsername()} HP: {Character2.getHp()}")
Character2.heal()
Character2.basicAttack(Character1)
Character2.slashAttack(Character1)
Character2.rangedAttack(Character1)
Character2.magicAttack(Character1)
print(f"{Character1.getUsername()} HP: {Character1.getHp()}")
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.

```python
from random import randint, shuffle

class Player:
    def __init__(self, name, role, hp, attack):
        self.name = name
        self.role = role
        self.hp = hp
        self.attack = attack

    def __str__(self):
        return f"{self.name} ({self.role}): HP: {self.hp}, Attack: {self.attack}"

    def attackOpponent(self, opponent):
        opponent.hp -= self.attack
        print(f"{self.name} attacks {opponent.name}! {opponent.name} now has {opponent.hp} HP.")

class Novice(Player):
    def __init__(self, name):
        super().__init__(name, "Novice", 100, 10)

class Swordsman(Player):
    def __init__(self, name):
        super().__init__(name, "Swordsman", 120, 15)

class Archer(Player):
    def __init__(self, name):
        super().__init__(name, "Archer", 100, 20)

class Magician(Player):
    def __init__(self, name):
        super().__init__(name, "Magician", 80, 25)

class Monster(Player):
    def __init__(self):
        super().__init__("Monster", "Boss", 150, 12)

class Game:
    def __init__(self):
        self.mode = None

    def startGame(self):
        print("\nWelcome to the Brawlhalla!\n")
        self.chooseMode()
        player1, player2 = self.createPlayer()
        self.playMatch(player1, player2)
```

```python
    def chooseMode(self):
        mode = input("Choose game mode:\n1. Single Player\n2. Player vs Player\nEnter your choice: ")
        self.mode = "Single Player" if mode == '1' else "Player vs Player"

    def createPlayer(self):
        if self.mode == "Single Player":
            return Novice(input("Enter your name: ")), Monster()
        else:
            return self.createPlayers(1), self.createPlayers(2)

    def createPlayers(self, playerNum):
        name = input(f"Enter Player {playerNum} name: ")
        roleMap = {'1': Swordsman, '2': Archer, '3': Magician}
        role = input("Choose role:\n1. Swordsman\n2. Archer\n3. Magician\nEnter choice: ")
        return roleMap.get(role, Novice)(name)

    def playMatch(self, player1, player2):
        print(f"\n{self.mode} Match:\n{player1}\n{player2}")
        turnOrder = [player1, player2]
        shuffle(turnOrder)

        while player1.hp > 0 and player2.hp > 0:
            for player in turnOrder:
                opponent = player1 if player == player2 else player2
                player.attackOpponent(opponent)
                if opponent.hp <= 0:
                    print (f"\n{player.name} wins the match!")
                    return

if __name__ == '__main__':
    Game().startGame()
```

```
Welcome to the Brawlhalla!

Choose game mode:
1. Single Player
2. Player vs Player
Enter your choice: 1
Enter your name: Mandy

Single Player Match:
Mandy (Novice): HP: 100, Attack: 10
Monster (Boss): HP: 150, Attack: 12
Monster attacks Mandy! Mandy now has 88 HP.
Mandy attacks Monster! Monster now has 140 HP.
Monster attacks Mandy! Mandy now has 76 HP.
Mandy attacks Monster! Monster now has 130 HP.
Monster attacks Mandy! Mandy now has 64 HP.
Mandy attacks Monster! Monster now has 120 HP.
Monster attacks Mandy! Mandy now has 52 HP.
Mandy attacks Monster! Monster now has 110 HP.
Monster attacks Mandy! Mandy now has 40 HP.
Mandy attacks Monster! Monster now has 100 HP.
Monster attacks Mandy! Mandy now has 28 HP.
Mandy attacks Monster! Monster now has 90 HP.
Monster attacks Mandy! Mandy now has 16 HP.
Mandy attacks Monster! Monster now has 80 HP.
Monster attacks Mandy! Mandy now has 4 HP.
Mandy attacks Monster! Monster now has 70 HP.
Monster attacks Mandy! Mandy now has -8 HP.

Monster wins the match!


...Program finished with exit code 0
```

```
Welcome to the Brawlhalla!

Choose game mode:
1. Single Player
2. Player vs Player
Enter your choice: 2
Enter Player 1 name: Mandy
Choose role:
1. Swordsman
2. Archer
3. Magician
Enter choice: 1
Enter Player 2 name: Rusian
Choose role:
1. Swordsman
2. Archer
3. Magician
Enter choice: 2

Player vs Player Match:
Mandy (Swordsman): HP: 120, Attack: 15
Rusian (Archer): HP: 100, Attack: 20
Rusian attacks Mandy! Mandy now has 100 HP.
Mandy attacks Rusian! Rusian now has 85 HP.
Rusian attacks Mandy! Mandy now has 80 HP.
Mandy attacks Rusian! Rusian now has 70 HP.
Rusian attacks Mandy! Mandy now has 60 HP.
Mandy attacks Rusian! Rusian now has 55 HP.
Rusian attacks Mandy! Mandy now has 40 HP.
Mandy attacks Rusian! Rusian now has 40 HP.
Rusian attacks Mandy! Mandy now has 20 HP.
Mandy attacks Rusian! Rusian now has 25 HP.
Rusian attacks Mandy! Mandy now has 0 HP.

Rusian wins the match!


...Program finished with exit code 0
Press ENTER to exit console.
```

Questions

1. Why is Inheritance important?
    - The inheritance allows to reuse a particular set of code.

2. Explain the advantages and disadvantages of using applying inheritance in an Object-Oriented Program.
    - The advantages of applying inheritance in an Object-Oriented Program are the capability to reuse and the better modularity of a particular set of code, while the disadvantage is the complexity of implementing and using a particular set of code as it becomes dependent on the main class.

3. Differentiate single inheritance, multiple inheritance, and multi-level inheritance.
    - A single inheritance is when a class inherits from one parent class, a multi inheritance is when a class inherits from one or two parent classes, and a multi level inheritance is when a class inherits from a class that inherits from a parent class.

4. Why is super(). init (username) added in the codes of Swordsman, Archer, Magician, and Boss?
    - The super().__init__(username) is added in the codes of Swordsman, Archer, Magician, and Boss to ensure proper initialization of the class.

5. How do you think Encapsulation and Abstraction helps in making good Object-Oriented Programs?
    - The encapsulation and abstraction help in making good Object-Oriented Programs by making a particular set of code capable of reuse and modification when necessary.

7. Conclusion:

- The inheritance allows the reuse and organization of a block of code under a main class for better modularity. Based on the situation, inheritance may or may not be appropriate. Software systems are more adjustable and manageable when inheritance is used properly.

8. Assessment Rubric: