Laboratory Activity No. 2	
Inheritance, Encapsulation, and Abstraction	
Course Code: CPE009	**Program:** BSCPE
Course Title: Object-Oriented Programming	**Date Performed:**
Section:	**Date Submitted:**
Name:	**Instructor:**

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

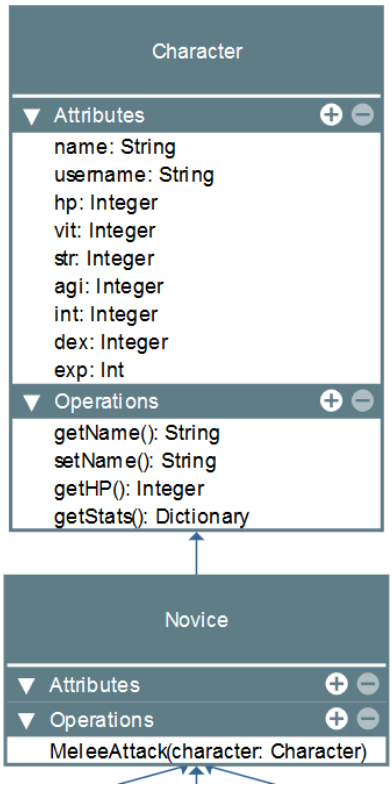
2.2 Create a class using the Python language

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.

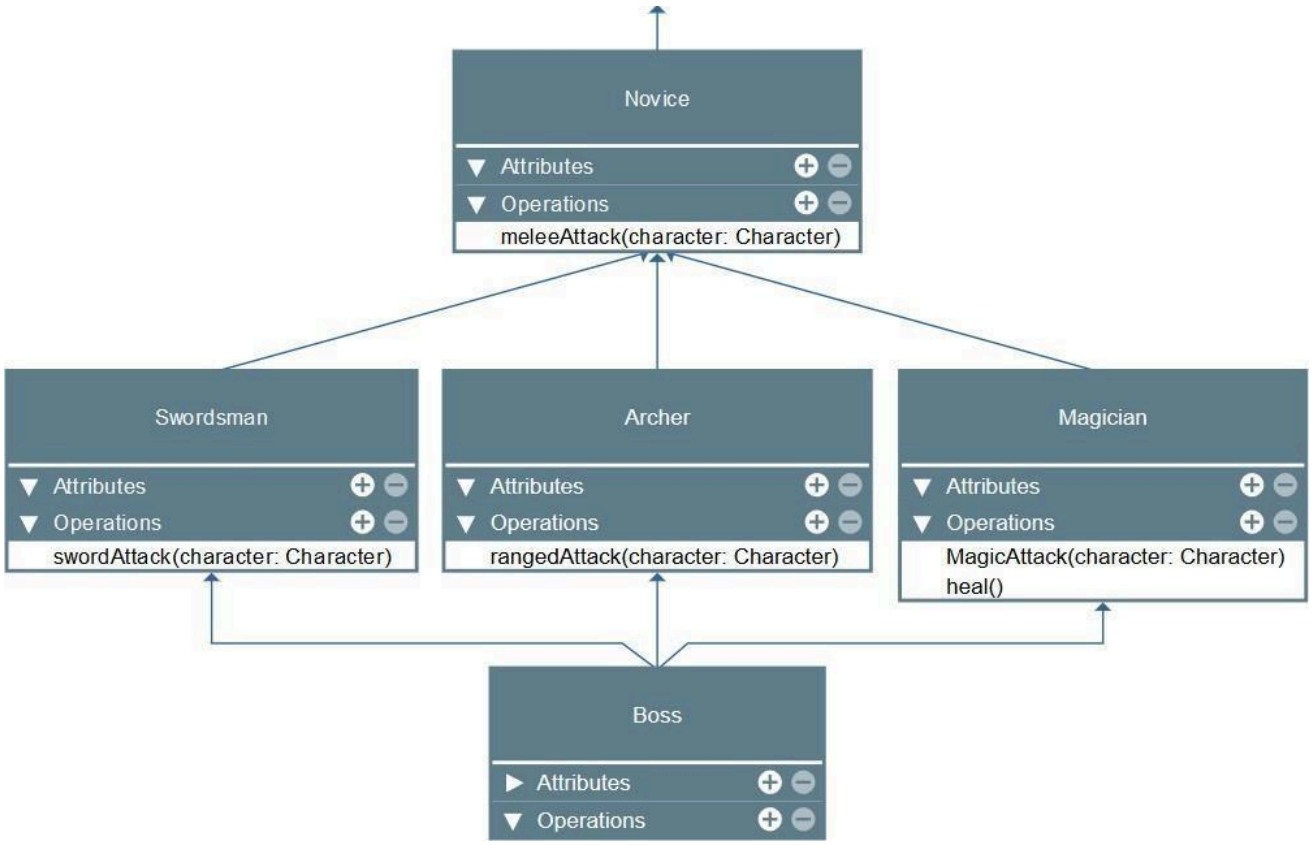
An example is given below considering a simple UML Class Diagram:



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:

```python
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 premade 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.

CODES ARE HERE IN THIS LINK:
https://drive.google.com/drive/folders/1sbl6cqezhtTvfWZ5L2-UvDdvX1QkTTgL?usp=sharing

```
Enter Player Name: dom
Starting Single Player match with dom against Monster.
dom performed Basic Attack! -5
dom HP: 100
Monster HP: 140
Monster performed Basic Attack! -5
Monster HP: 140
dom HP: 95
dom performed Basic Attack! -5
dom HP: 95
Monster HP: 135
Monster performed Basic Attack! -5
Monster HP: 135
dom HP: 90
dom performed Basic Attack! -5
dom HP: 90
Monster HP: 130
Monster perfomed Slash Attack! -5
Monster HP: 130
dom HP: 85
dom performed Basic Attack! -5
dom HP: 85
Monster HP: 125
Monster perfomed Slash Attack! -7
Monster HP: 125
dom HP: 78
dom performed Basic Attack! -5
dom HP: 78
Monster HP: 120
Monster perfomed Slash Attack! -8
Monster HP: 120
dom HP: 70
dom performed Basic Attack! -5
dom HP: 70
```

```
dom performed Basic Attack! -5
dom HP: 26
Monster HP: 80
Monster performed Basic Attack! -5
Monster HP: 80
dom HP: 21
dom performed Basic Attack! -5
dom HP: 21
Monster HP: 75
Monster perfomed Slash Attack! -9
Monster HP: 75
dom HP: 12
dom performed Basic Attack! -5
dom HP: 12
Monster HP: 70
Monster perfomed Slash Attack! -6
Monster HP: 70
dom HP: 6
dom performed Basic Attack! -5
dom HP: 6
Monster HP: 65
Monster performed Basic Attack! -5
Monster HP: 65
dom HP: 1
dom performed Basic Attack! -5
dom HP: 1
Monster HP: 60
Monster performed Basic Attack! -5
Monster HP: 60
dom HP: -4
dom has been defeated.
```

Questions

1. Why is Inheritance important?

- Inheritance is important because it allows for the reuse of code written in a class. By inheriting from a parent class, a child class gains access to all of the parent class's properties and functions, reducing redundancy and boosting code reuse.

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

- Inheritance supports code reuse, which results in more efficient and organized code. It also supports polymorphism, allowing objects to be seen as instances of their parent class, increasing program flexibility and scalability. However, inheritance has some drawbacks, particularly as hierarchies become increasingly complex. This can make understanding and maintaining code challenging. Furthermore, in some languages, inheritance may cause performance cost due to runtime lookups and method resolution.

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

- **Single inheritance**: A child class inherits from a single parent class, resulting in a visible hierarchy while limiting the child class's access to inherited properties and methods.
- **Multiple inheritance** occurs when a child class inherits from more than one parent class. This improves code reuse and flexibility, but it can also add difficulties, such as method or attribute conflicts across parent classes.
- **Multi-level inheritance** involves one child class inheriting from another, resulting in an inheritance hierarchy. This results in a more layered structure, but it requires careful design to avoid problems such as excessive coupling and dependencies.

4. Why is super(). init (username) added in the codes of Swordsman, Archer, Magician, and Boss?
- The super().__init__(username) method ensures that the superclass (parent class) is properly initialized. This call sends required parameters, like the username, to the parent class's constructor. By utilizing super(), we avoid repeating initialization code and verify that the inheritance hierarchy is properly configured.

5. How do you think Encapsulation and Abstraction helps in making good Object-Oriented Programs?
- Encapsulation aids in the creation of modular and maintainable code by grouping data and methods within a class and restricting access to internal information. Abstraction reduces complexity by exposing only essential features and hiding unneeded implementation details. Together, they promote cleaner, more organized code that is simpler to manage and scale.

7. Conclusion:

- I've discovered that encapsulation and abstraction are critical for developing effective Object-Oriented programs. Encapsulation helps to secure data security and maintainability by limiting access to internal information, whereas abstraction simplifies complicated systems by showing only what is required. Mastering these concepts allows me to write code that is clearer, more organized, more scalable. This insight is critical for honing my programming skills and developing reliable software.

8. Assessment Rubric: