Laboratory Activity No. 6

Inheritance, Encapsulation, and Abstraction

Course Code: CPE103	**Program:** BSCPE
Course Title: Object-Oriented Programming	**Date Performed:** 02/15/25
Section: 1-A	**Date Submitted:** 02/15/25
Name: Filjohn B. Delinia	**Instructor:** Engr. Maria Rizette Sayo

1. Objective(s):

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

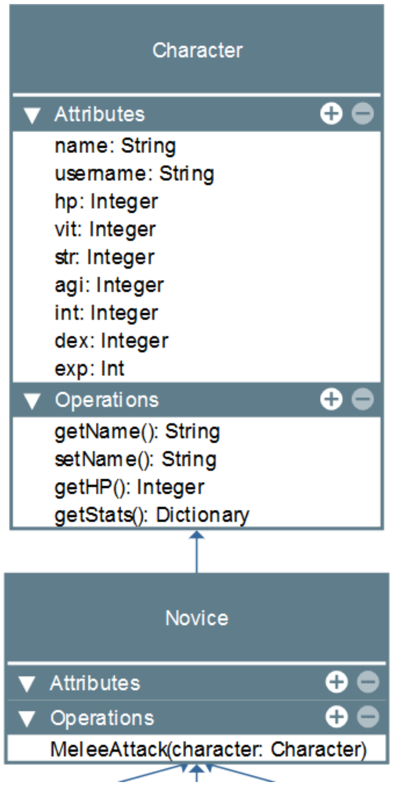
2. Intended Learning Outcomes (ILOs):

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.

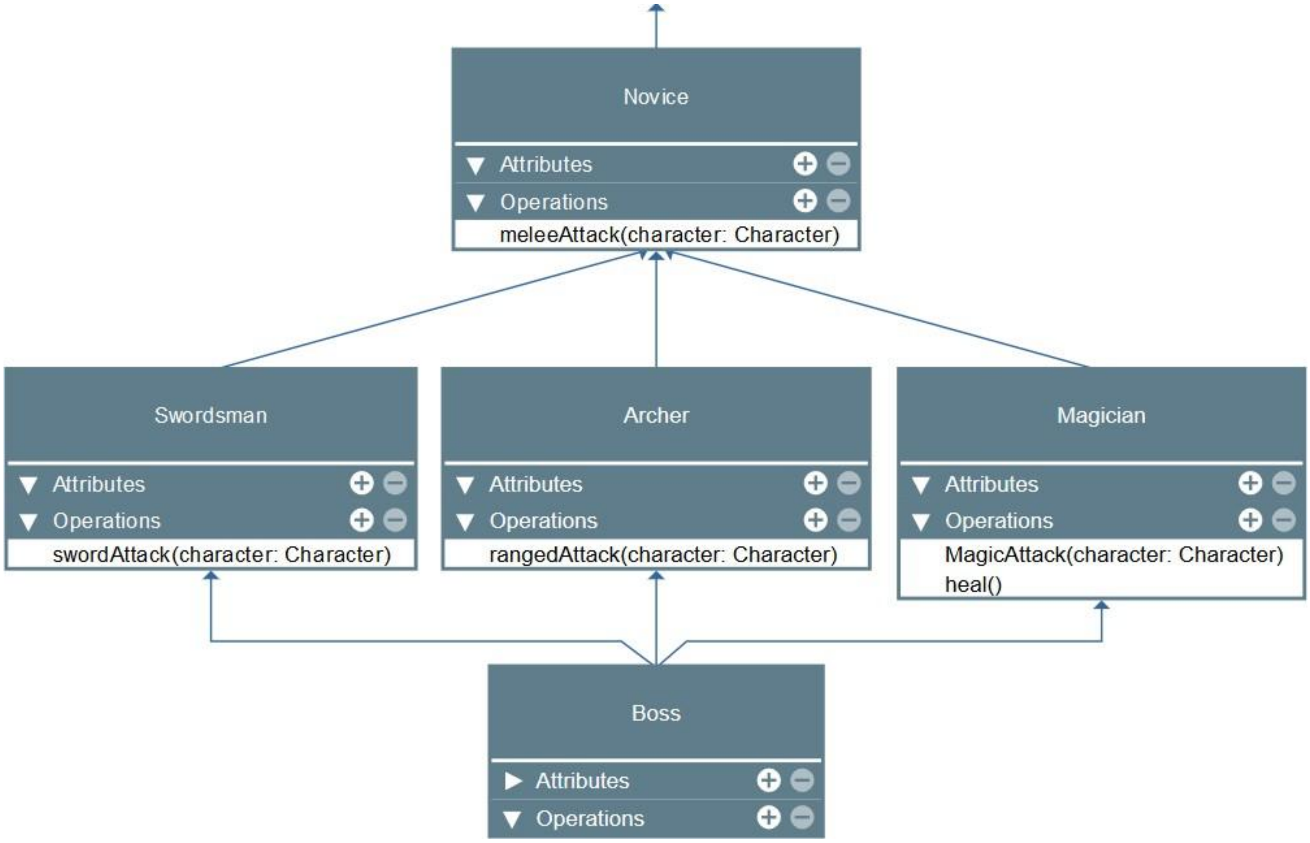
3. Discussion:

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.

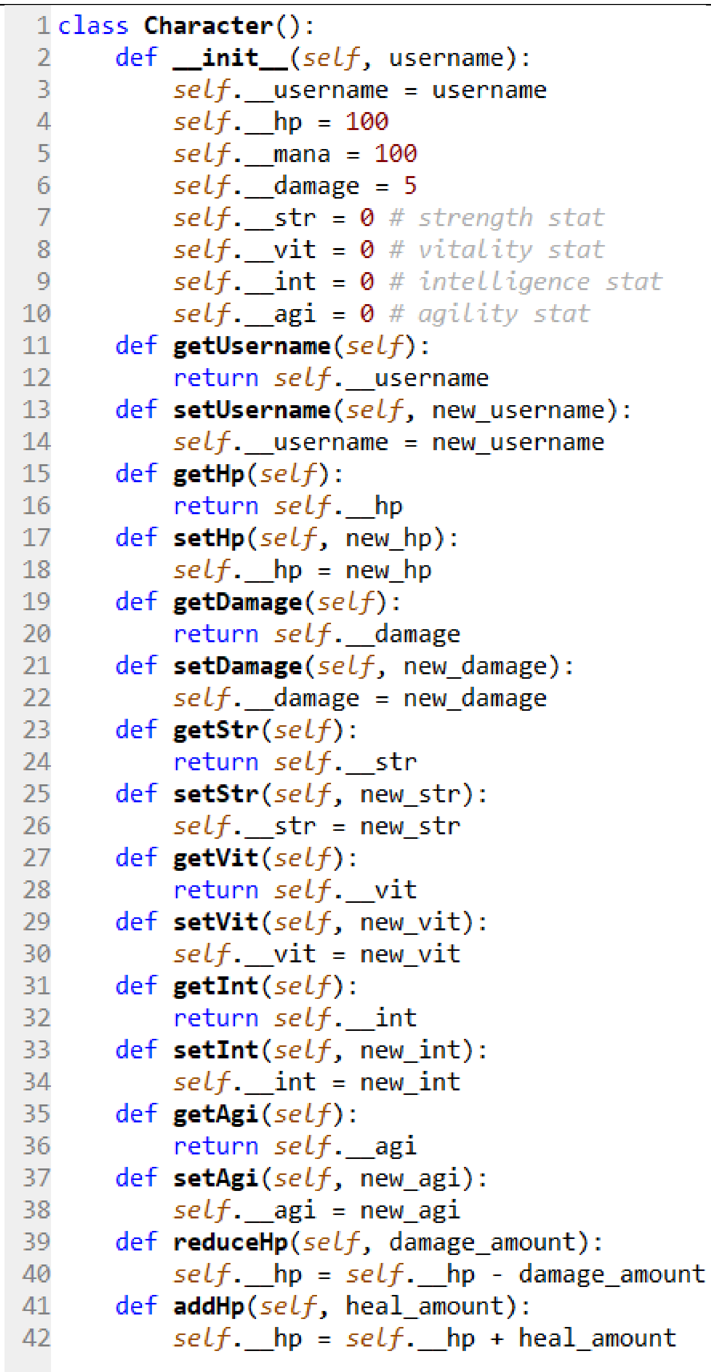
4. Materials and Equipment:

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

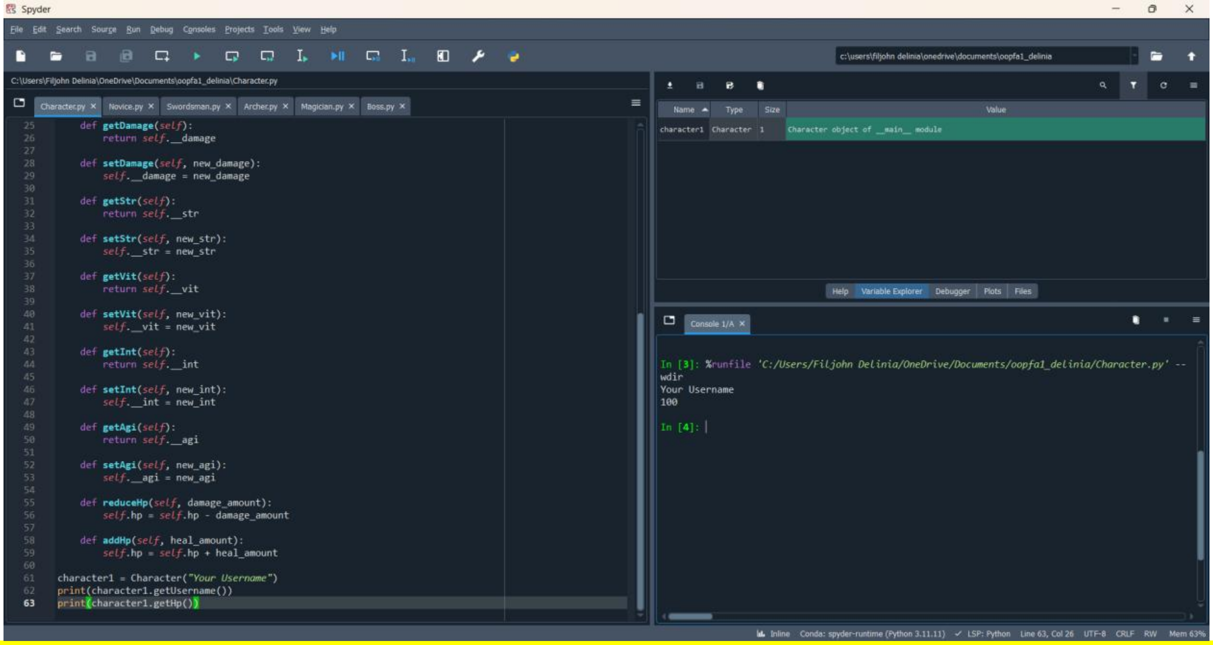


```python
class Character():
    def __init__(self, username):
        self.__username = username
        self.__hp = 100
        self.__mana = 100
        self.__damage = 5
        self.__str = 0 # strength stat
        self.__vit = 0 # vitality stat
        self.__int = 0 # intelligence stat
        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.

OUTPUT AND OBSERVATION: Creating the Classes



C:\Users\Filjohn Delinia\OneDrive\Documents\oopfa1_delinia\Character.py

```python
    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

character1 = Character("Your Username")
print(character1.getUsername())
print(character1.getHp())
```

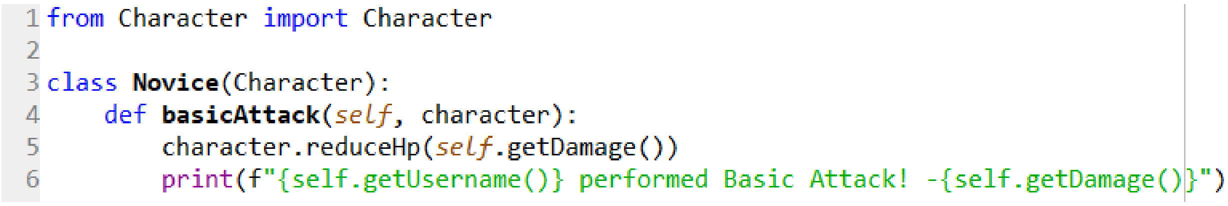
Name	Type	Size	Value
character1	Character	1	Character object of __main__ module

```
In [3]: %runfile 'C:/Users/Filjohn Delinia/OneDrive/Documents/oopfa1_delinia/Character.py' --
wdir
Your Username
100

In [4]: 
```

The added code creates a Character instance with the username "Your Username" and displays the username and default HP of 100. The Character class is well-structured, with getter and setter methods for secure attribute management and functions to modify HP dynamically.

Single Inheritance

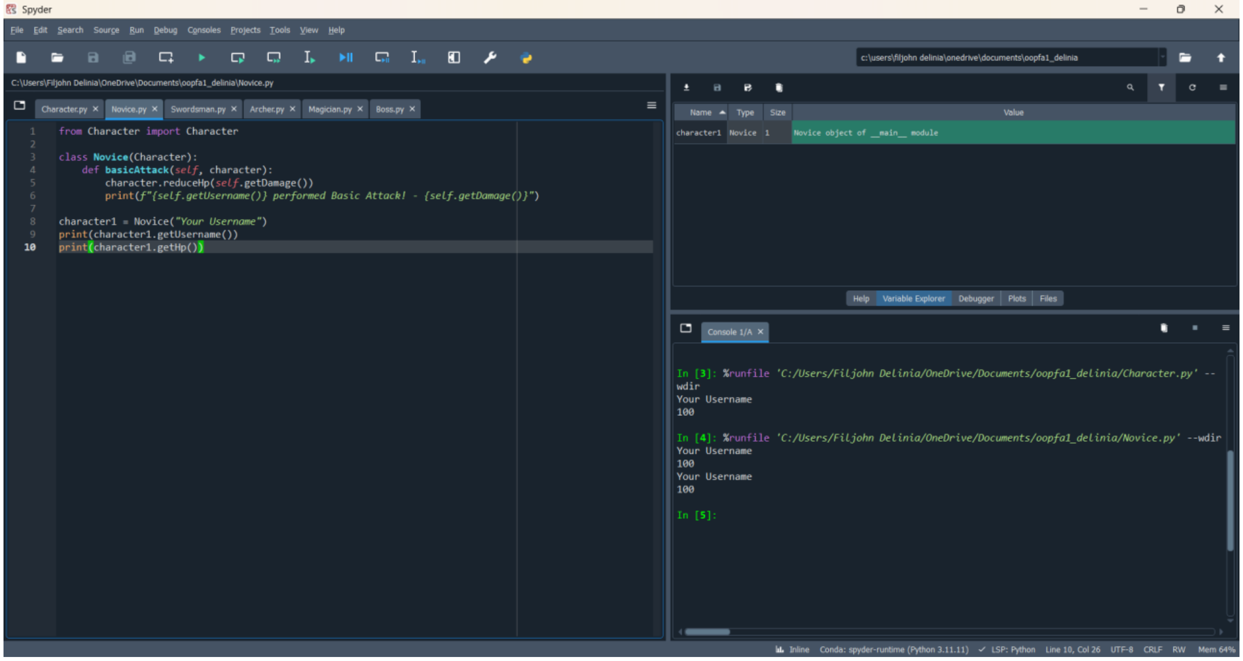
1. In the Novice.py class, copy the following code.



```
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.

OUTPUT AND OBSERVATION: Single Inheritance



Spyder
File Edit Search Source Run Debug Consoles Projects Tools View Help
c:\users\filjohn delinia\onedrive\documents\oopfa1_delinia
C:\Users\Filjohn Delinia\OneDrive\Documents\oopfa1_delinia\Novice.py
Character.py Novice.py Swordsman.py Archer.py Magician.py Boss.py
Help Variable Explorer Debugger Plots Files
Console 1/A
Inline Conda: spyder-runtime (Python 3.11.11) LSP: Python Line 10, Col 26 UTF-8 CRLF RW Mem 64%

```
from Character import Character

class Novice(Character):
    def basicAttack(self, character):
        character.reduceHp(self.getDamage())
        print(f"{self.getUsername()} performed Basic Attack! - {self.getDamage()}")

character1 = Novice("Your Username")
print(character1.getUsername())
print(character1.getHp())
```

Name	Type	Size	Value
character1	Novice	1	Novice object of __main__ module

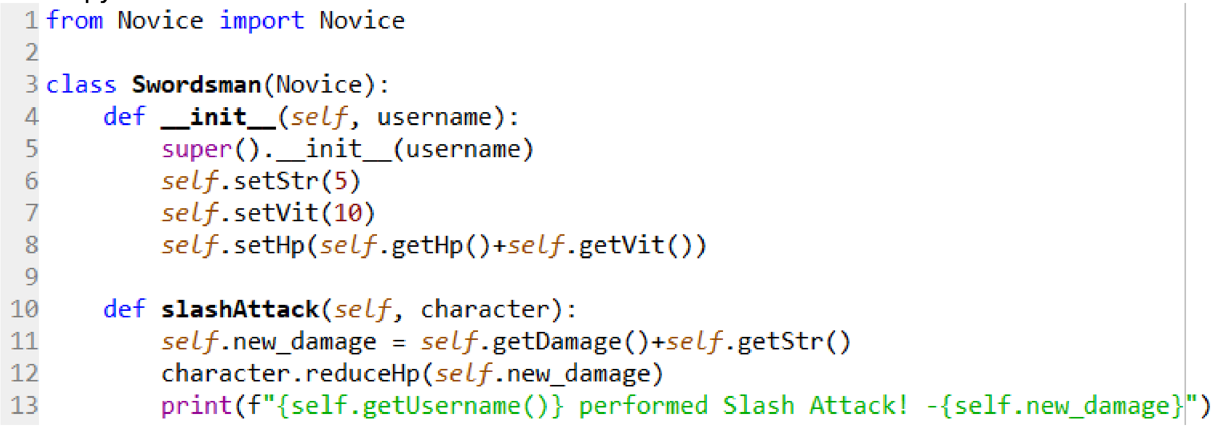
```
In [3]: %runfile 'C:/Users/Filjohn Delinia/OneDrive/Documents/oopfa1_delinia/Character.py' --
wdir
Your Username
100

In [4]: %runfile 'C:/Users/Filjohn Delinia/OneDrive/Documents/oopfa1_delinia/Novice.py' --wdir
Your Username
100
Your Username
100

In [5]:
```

The added code creates a Novice character, inheriting attributes from the Character class, including the default HP of 100. The output confirms the correct assignment of the username and HP. The Novice class introduces a basicAttack() method to reduce another character's HP, but since it's not called in this code, its effect is not shown.

Multi-level Inheritance

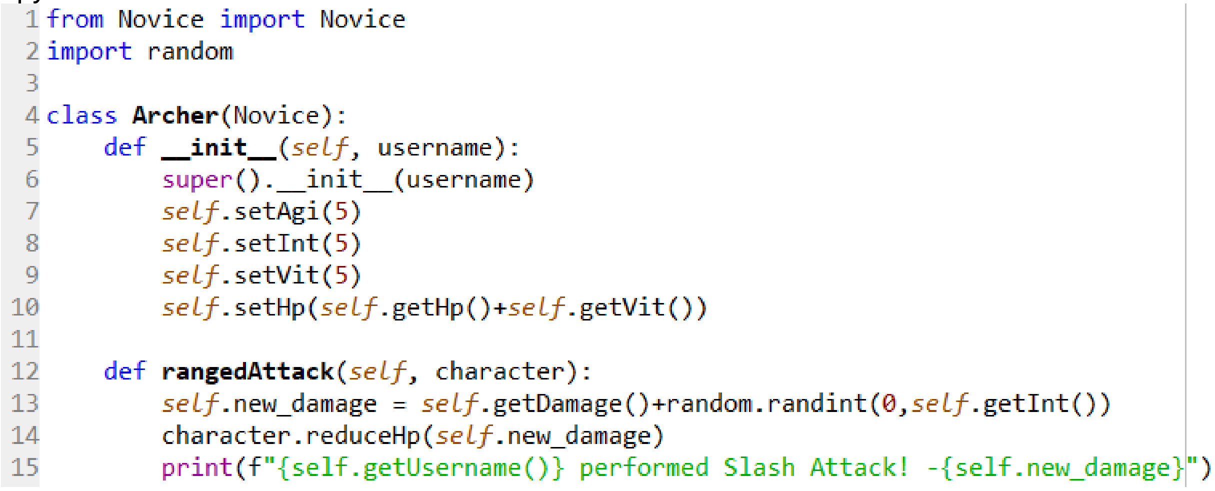
1. In the Swordsman, Archer, and Magician .py files copy the following codes for each file:

Swordsman.py



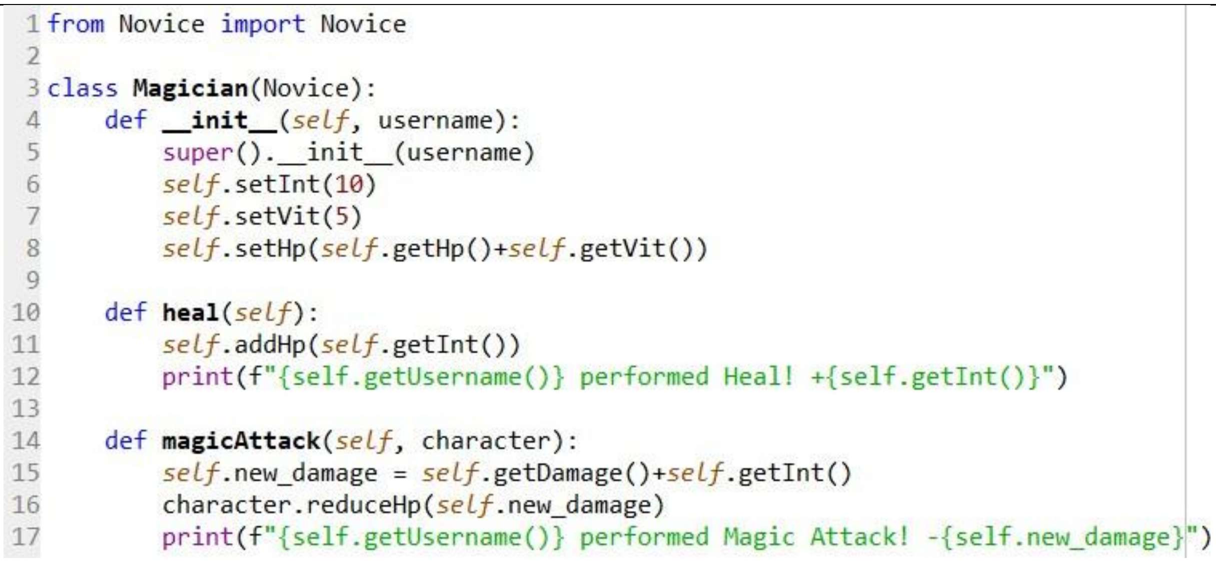
```python
from Novice import Novice

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



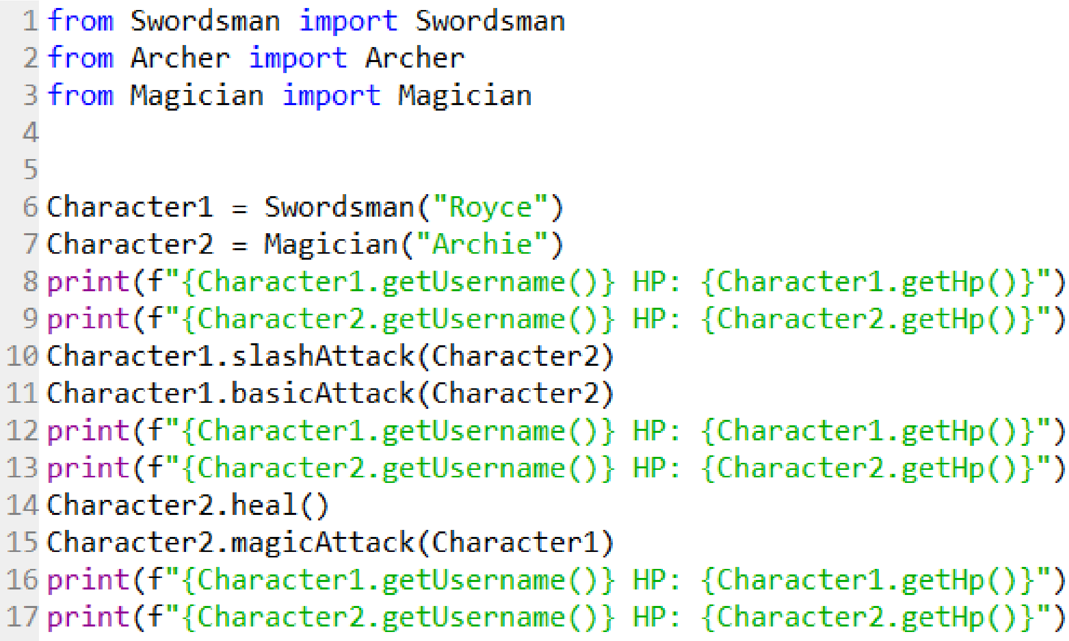
```python
from Novice import Novice
import random

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



```
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}")
```

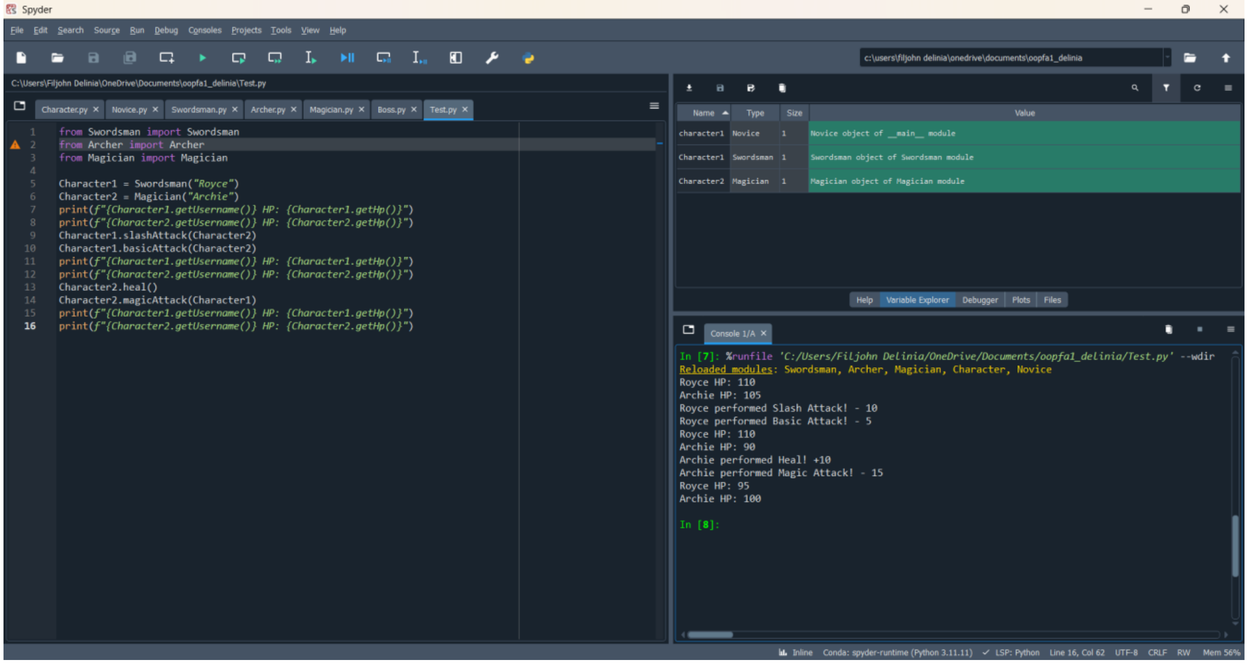
2. Create a new file called Test.py and copy the codes below:



```
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.

OUTPUT AND OBSERVATION: Multi-level Inheritance

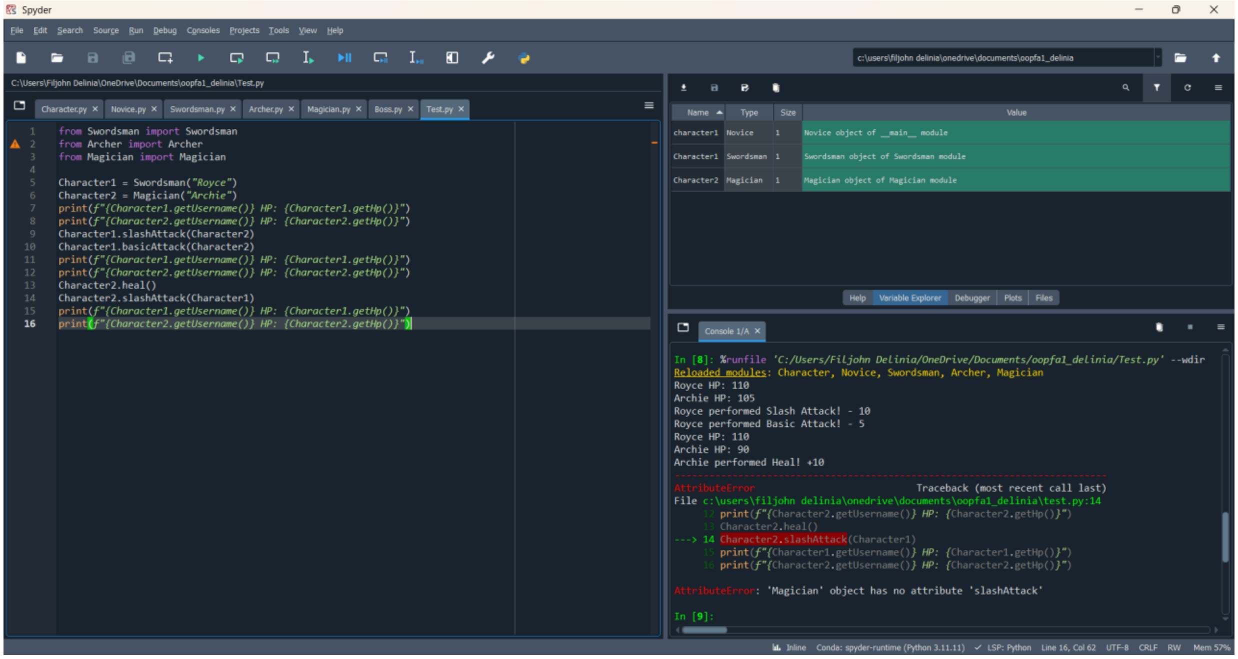

Spyder
File Edit Search Source Run Debug Consoles Projects Tools View Help
c:\users\filjohn delinia\onedrive\documents\oopfa1_delinia
C:\Users\Filjohn Delinia\OneDrive\Documents\oopfa1_delinia\Test.py
Character.py
Novice.py
Swordsman.py
Archer.py
Magician.py
Boss.py
Test.py
Help
Variable Explorer
Debugger
Plots
Files
Console 1/A
Inline Conda: spyder-runtime (Python 3.11.11) ✓ LSP: Python Line 16, Col 62 UTF-8 CRLF RW Mem 56%

```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()}")
```

Name	Type	Size	Value
character1	Novice	1	Novice object of __main__ module
Character1	Swordsman	1	Swordsman object of Swordsman module
Character2	Magician	1	Magician object of Magician module

```
In [7]: %runfile 'C:/Users/Filjohn Delinia/OneDrive/Documents/oopfa1_delinia/Test.py' --wdir
Reloaded modules: Swordsman, Archer, Magician, Character, Novice
Royce HP: 110
Archie HP: 105
Royce performed Slash Attack! - 10
Royce performed Basic Attack! - 5
Royce HP: 110
Archie HP: 90
Archie performed Heal! +10
Archie performed Magic Attack! - 15
Royce HP: 95
Archie HP: 100

In [8]:
```

The program successfully simulates a turn-based battle between Royce, a Swordsman, and Archie, a Magician. Royce starts with 110 HP, while Archie has 105 HP. Royce performs a Slash Attack and a Basic Attack, reducing Archie's HP to 90. Archie heals himself back to 100 HP before counterattacking with a Magic Attack, lowering Royce's HP to 95. The program accurately implements stat-based attacks and healing, effectively demonstrating the class-based system.


Spyder
File Edit Search Source Run Debug Consoles Projects Tools View Help
c:\users\filjohn delinia\onedrive\documents\oopfa1_delinia
C:\Users\Filjohn Delinia\OneDrive\Documents\oopfa1_delinia\Test.py
Character.py
Novice.py
Swordsman.py
Archer.py
Magician.py
Boss.py
Test.py
Help
Variable Explorer
Debugger
Plots
Files
Console 1/A
Inline Conda: spyder-runtime (Python 3.11.11) ✓ LSP: Python Line 16, Col 62 UTF-8 CRLF RW Mem 57%

```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.slashAttack(Character1)
print(f"{Character1.getUsername()} HP: {Character1.getHp()}")
print(f"{Character2.getUsername()} HP: {Character2.getHp()}")
```

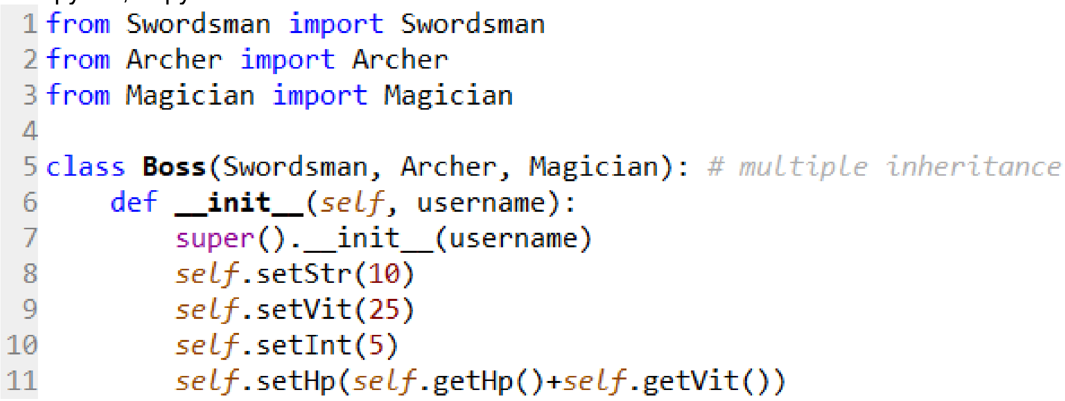
Name	Type	Size	Value
character1	Novice	1	Novice object of __main__ module
Character1	Swordsman	1	Swordsman object of Swordsman module
Character2	Magician	1	Magician object of Magician module

```
In [8]: %runfile 'C:/Users/Filjohn Delinia/OneDrive/Documents/oopfa1_delinia/Test.py' --wdir
Reloaded modules: Character, Novice, Swordsman, Archer, Magician
Royce HP: 110
Archie HP: 105
Royce performed Slash Attack! - 10
Royce performed Basic Attack! - 5
Royce HP: 110
Archie HP: 90
Archie performed Heal! +10
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
File c:\users\filjohn delinia\onedrive\documents\oopfa1_delinia\test.py:14
     12 print(f"{Character2.getUsername()} HP: {Character2.getHp()}")
     13 Character2.heal()
---> 14 Character2.slashAttack(Character1)
     15 print(f"{Character1.getUsername()} HP: {Character1.getHp()}")
     16 print(f"{Character2.getUsername()} HP: {Character2.getHp()}")

AttributeError: 'Magician' object has no attribute 'slashAttack'

In [9]:
```

Running Test.py initializes Royce (Swordsman) and Archie (Magician). Royce attacks with Slash and Basic Attacks, reducing Archie's HP. Archie heals but encounters an AttributeError when attempting slashAttack, which is not defined in the Magician class.

Multiple Inheritance

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



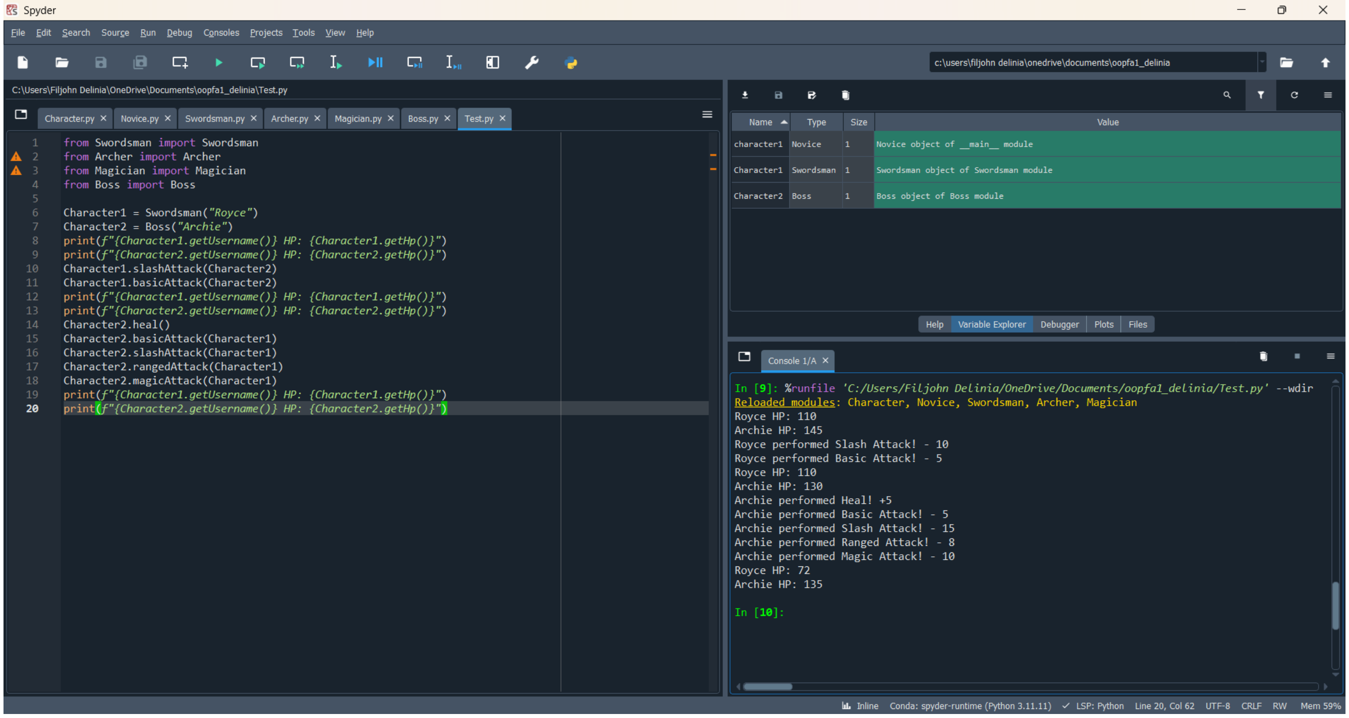
```python
from Swordsman import Swordsman
from Archer import Archer
from Magician import Magician

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.

OUTPUT AND OBSERVATION:


Spyder
File Edit Search Source Run Debug Consoles Projects Tools View Help
c:\users\filjohn delinia\onedrive\documents\oopfa1_delinia
C:\Users\Filjohn Delinia\OneDrive\Documents\oopfa1_delinia\Test.py
Character.py
Novice.py
Swordsman.py
Archer.py
Magician.py
Boss.py
Test.py
Help
Variable Explorer
Debugger
Plots
Files
Console 1/A
Inline
Conda: spyder-runtime (Python 3.11.11)
LSP: Python
Line 20, Col 62
UTF-8
CRLF
RW
Mem 59%

```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()}")
```

Name	Type	Size	Value
character1	Novice	1	Novice object of __main__ module
Character1	Swordsman	1	Swordsman object of Swordsman module
Character2	Boss	1	Boss object of Boss module

```
In [9]: %runfile 'C:/Users/Filjohn Delinia/OneDrive/Documents/oopfa1_delinia/Test.py' --wdir
Reloaded modules: Character, Novice, Swordsman, Archer, Magician
Royce HP: 110
Archie HP: 145
Royce performed Slash Attack! - 10
Royce performed Basic Attack! - 5
Royce HP: 110
Archie HP: 130
Archie performed Heal! +5
Archie performed Basic Attack! - 5
Archie performed Slash Attack! - 15
Archie performed Ranged Attack! - 8
Archie performed Magic Attack! - 10
Royce HP: 72
Archie HP: 135

In [10]:
```

Royce starts with 110 HP, while Archie, the Boss, has 145 HP due to his higher vitality. Royce attacks first, dealing 15 damage, lowering Archie’s HP to 130. Archie then heals 5 HP and counters with multiple attacks, dealing a total of 38 damage, reducing Royce’s HP to 72. By the end, Archie remains strong at 135 HP, demonstrating his superior durability and attack power.

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.

Questions

1. Why is Inheritance important?

Inheritance is a key concept in Object-Oriented Programming (OOP) that allows one class to inherit properties and behaviors from another. It's important because it promotes **code reusability**, meaning you don't have to rewrite the same code over and over.

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

Inheritance has several **advantages**: it reduces code duplication, makes programs easier to maintain, and allows you to extend functionality without rewriting existing code. For instance, if you have a Vehicle class, you can create a Car class that inherits basic features like speed and fuel while adding car-specific details. However, there are **disadvantages** too. Inheritance can create tight coupling between classes, meaning changes in the parent class might break child classes. It can also make the code harder to understand if the inheritance hierarchy becomes too deep or complex.

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

Inheritance comes in different forms. **Single inheritance** is when a class inherits from just one parent class, like a Bird class inheriting from an Animal class. **Multiple inheritance** allows a class to inherit from more than one parent, like a Flying Fish class inheriting from both Bird and Fish. While this can be useful, it can also lead to confusion if the parent classes have conflicting methods. **Multi-level inheritance** involves a chain of inheritance, like a Grandparent class, a Parent class inheriting from Grandparent, and a Child class inheriting from Parent.

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

It is used to call the constructor of the parent class. This ensures that the parent class's initialization logic is executed before the child class adds its own features. For example, if you have a Player class with a username attribute, and a Swordsman class that inherits from Player, using super().__init__(username) ensures the username is properly set up in the Player class before the Swordsman class adds its specific details.

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

Encapsulation bundles data and methods together, controlling access to protect data integrity—like hiding a bank balance and allowing changes only through `deposit()` or `withdraw()`. **Abstraction** hides complex details, exposing only what's necessary—like using a coffee machine without knowing how it works. Together, they make programs easier to use, maintain, and extend by keeping internal details hidden and providing a simple, clear interface.

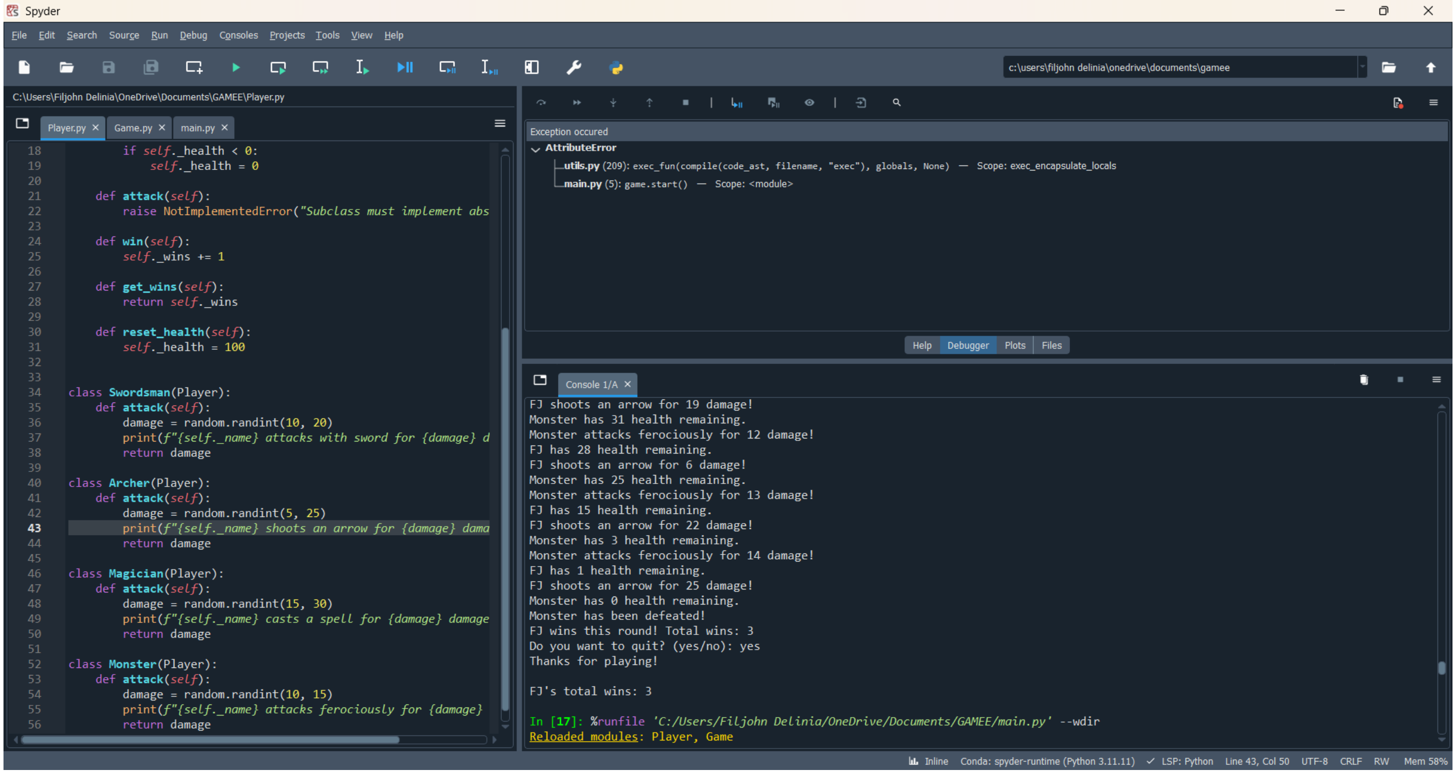
7. Conclusion:

In conclusion, inheritance, encapsulation, and abstraction are the building blocks of good Object-Oriented Programming (OOP). Inheritance lets you reuse and organize code by allowing classes to share common traits, making it easier to build on existing work. Encapsulation acts like a protective shield, keeping data safe and ensuring it's only accessed or modified in controlled ways. Abstraction simplifies things by hiding the messy details and showing only what's necessary, much like how you don't need to know how a car engine works to drive a car. Together, these principles help create software that's not only powerful and flexible but also easier to understand, maintain, and grow over time. While they come with challenges, like avoiding overly complex designs, using them thoughtfully can lead to cleaner, more intuitive, and future-proof programs.

8. Assessment Rubric:

SUPPLEMENTARY ACTIVITY OUTPUTS:

SINGLE PLAYER

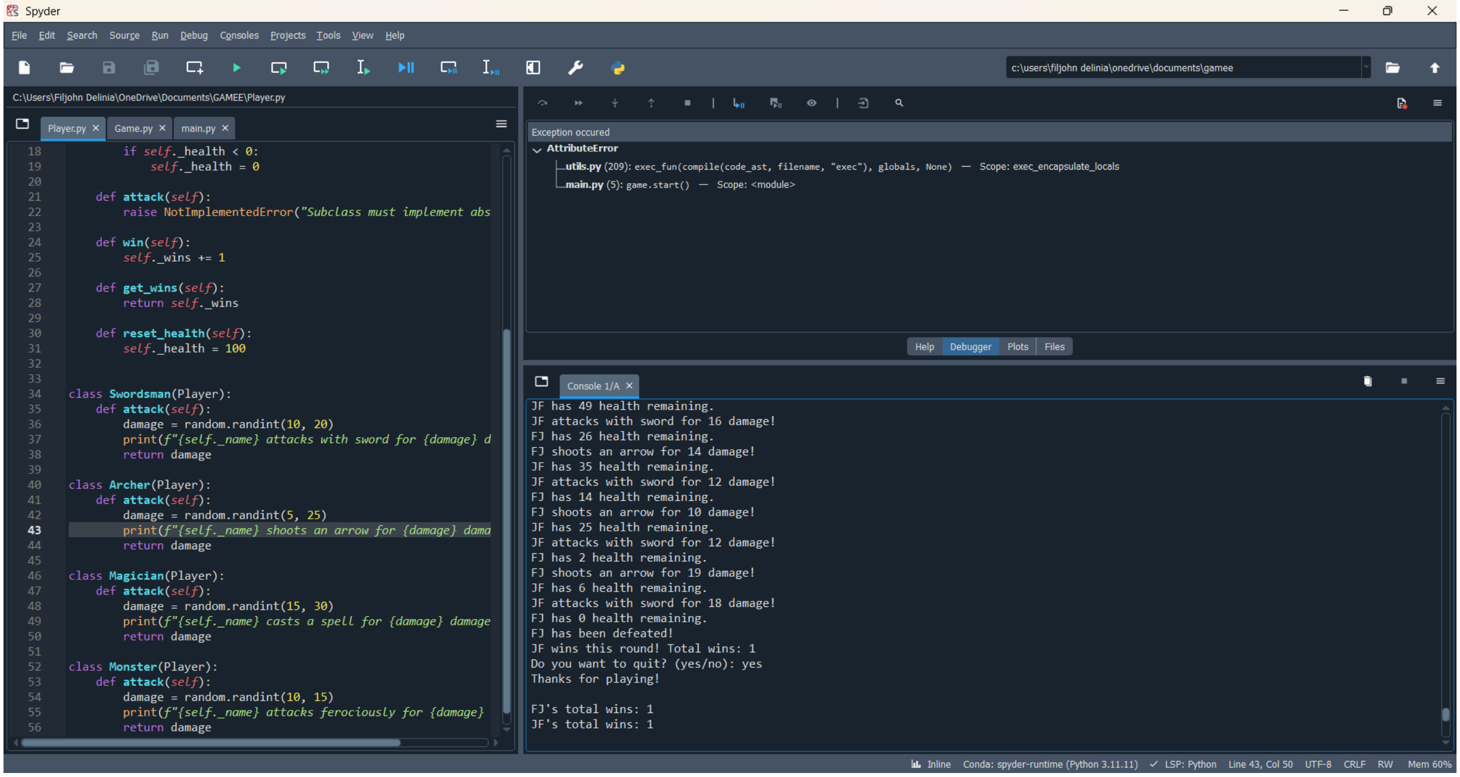

Spyder
File Edit Search Source Run Debug Consoles Projects Tools View Help
c:\users\filjohn delinia\onedrive\documents\gamee
C:\Users\Filjohn Delinia\OneDrive\Documents\GAMEE\Player.py
Player.py
Game.py
main.py
Help
Debugger
Plots
Files
Console 1/A
Inline
Conda: spyder-runtime (Python 3.11.11)
LSP: Python
Line 43, Col 50
UTF-8
CRLF
RW
Mem 58%

```python
        if self._health < 0:
            self._health = 0

    def attack(self):
        raise NotImplementedError("Subclass must implement abs

    def win(self):
        self._wins += 1

    def get_wins(self):
        return self._wins

    def reset_health(self):
        self._health = 100


class Swordsman(Player):
    def attack(self):
        damage = random.randint(10, 20)
        print(f"{self._name} attacks with sword for {damage} d
        return damage

class Archer(Player):
    def attack(self):
        damage = random.randint(5, 25)
        print(f"{self._name} shoots an arrow for {damage} dama
        return damage

class Magician(Player):
    def attack(self):
        damage = random.randint(15, 30)
        print(f"{self._name} casts a spell for {damage} damage
        return damage

class Monster(Player):
    def attack(self):
        damage = random.randint(10, 15)
        print(f"{self._name} attacks ferociously for {damage}
        return damage
```

```
Exception occured
AttributeError
  utils.py (209): exec_fun(compile(code_ast, filename, "exec"), globals, None)  —  Scope: exec_encapsulate_locals
  main.py (5): game.start()  —  Scope: <module>
```

```
FJ shoots an arrow for 19 damage!
Monster has 31 health remaining.
Monster attacks ferociously for 12 damage!
FJ has 28 health remaining.
FJ shoots an arrow for 6 damage!
Monster has 25 health remaining.
Monster attacks ferociously for 13 damage!
FJ has 15 health remaining.
FJ shoots an arrow for 22 damage!
Monster has 3 health remaining.
Monster attacks ferociously for 14 damage!
FJ has 1 health remaining.
FJ shoots an arrow for 25 damage!
Monster has 0 health remaining.
Monster has been defeated!
FJ wins this round! Total wins: 3
Do you want to quit? (yes/no): yes
Thanks for playing!

FJ's total wins: 3

In [17]: %runfile 'C:/Users/Filjohn Delinia/OneDrive/Documents/GAMEE/main.py' --wdir
Reloaded modules: Player, Game
```

PLAYER VS PLAYER



```
        if self._health < 0:
            self._health = 0

    def attack(self):
        raise NotImplementedError("Subclass must implement abs

    def win(self):
        self._wins += 1

    def get_wins(self):
        return self._wins

    def reset_health(self):
        self._health = 100


class Swordsman(Player):
    def attack(self):
        damage = random.randint(10, 20)
        print(f"{self._name} attacks with sword for {damage} d
        return damage

class Archer(Player):
    def attack(self):
        damage = random.randint(5, 25)
        print(f"{self._name} shoots an arrow for {damage} dama
        return damage

class Magician(Player):
    def attack(self):
        damage = random.randint(15, 30)
        print(f"{self._name} casts a spell for {damage} damage
        return damage

class Monster(Player):
    def attack(self):
        damage = random.randint(10, 15)
        print(f"{self._name} attacks ferociously for {damage}
        return damage
```

```
Exception occured
AttributeError
  utils.py (209): exec_fun(compile(code_ast, filename, "exec"), globals, None) — Scope: exec_encapsulate_locals
  main.py (5): game.start() — Scope: <module>
```

```
JF has 49 health remaining.
JF attacks with sword for 16 damage!
FJ has 26 health remaining.
FJ shoots an arrow for 14 damage!
JF has 35 health remaining.
JF attacks with sword for 12 damage!
FJ has 14 health remaining.
FJ shoots an arrow for 10 damage!
JF has 25 health remaining.
JF attacks with sword for 12 damage!
FJ has 2 health remaining.
FJ shoots an arrow for 19 damage!
JF has 6 health remaining.
JF attacks with sword for 18 damage!
FJ has 0 health remaining.
FJ has been defeated!
JF wins this round! Total wins: 1
Do you want to quit? (yes/no): yes
Thanks for playing!

FJ's total wins: 1
JF's total wins: 1
```