Laboratory Activity No. 2

Inheritance, Encapsulation, and Abstraction

Course Code: CPE009	**Program:** BSCPE
Course Title: Object-Oriented Programming	**Date Performed: 29/09/2024**
Section: CPE21S4	**Date Submitted: 29/09/2024**
Name: Kenn Jie Valleser	**Instructor: Engr. Maria Rizette Sayo**

1. Objective(s):

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

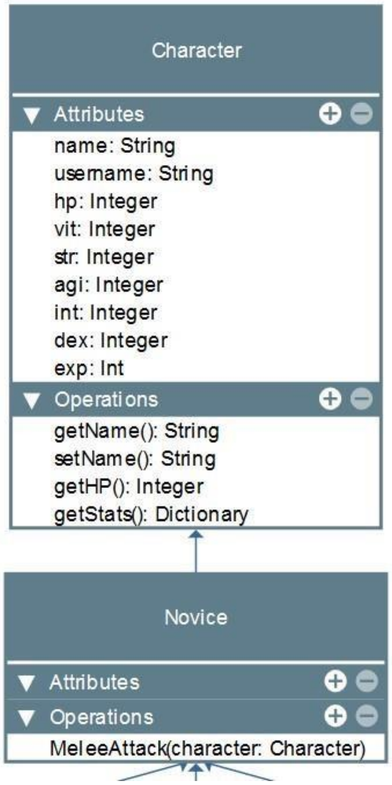
2. Intended Learning Outcomes (ILOs):

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:

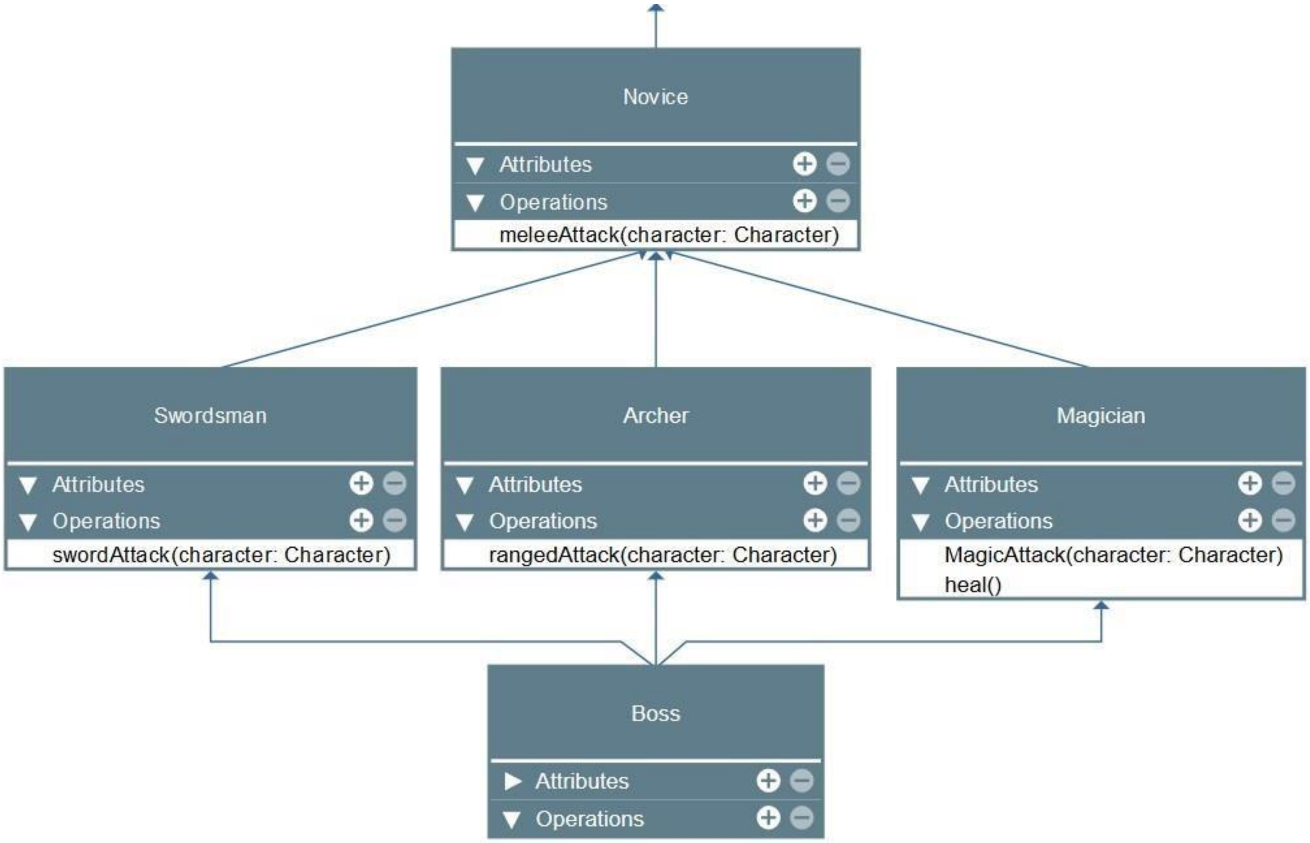
Object-Oriented Programming (OOP) has 4 core Principles: Inheritance, Polymorphism, Encapsulation, and Abstraction. The main goal of Object -Oriented Programming is code reusability and modularity meaning it can be reused for different purposes and integrated in other different programs. These 4 core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

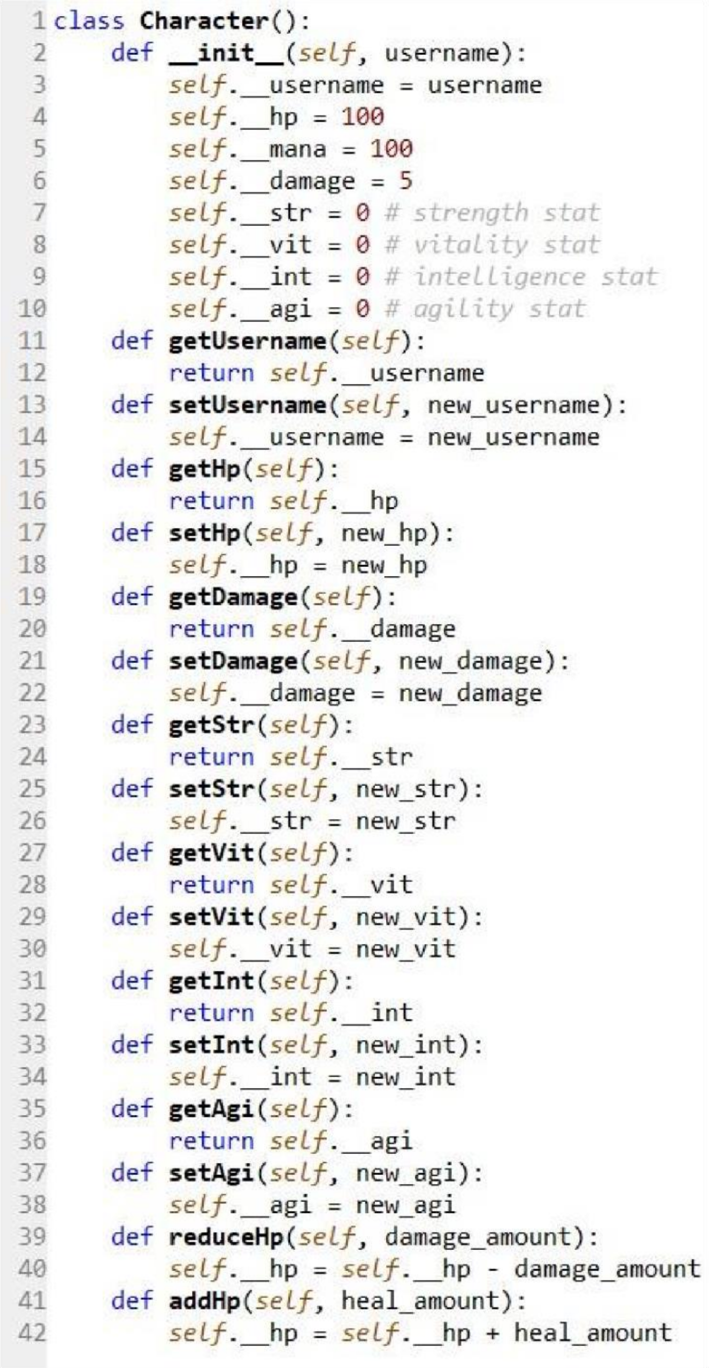
4. Materials and Equipment:

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



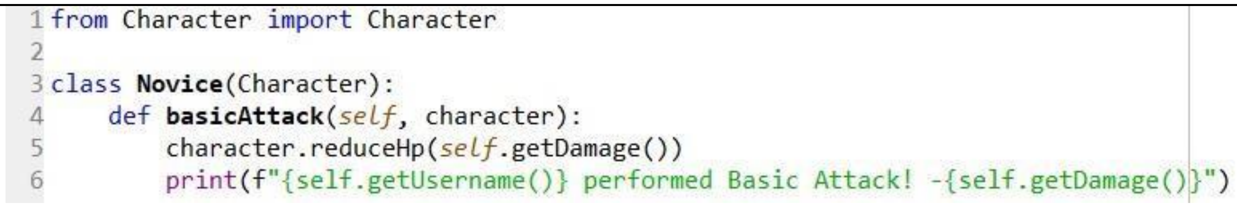
```python
class Character():
    def __init__(self, username):
        self.__username = username
        self.__hp = 100
        self.__mana = 100
        self.__damage = 5
        self.__str = 0 # strength stat
        self.__vit = 0 # vitality stat
        self.__int = 0 # intelligence stat
        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 Ex.
   character1 = Character(“YourUsername”)
   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.

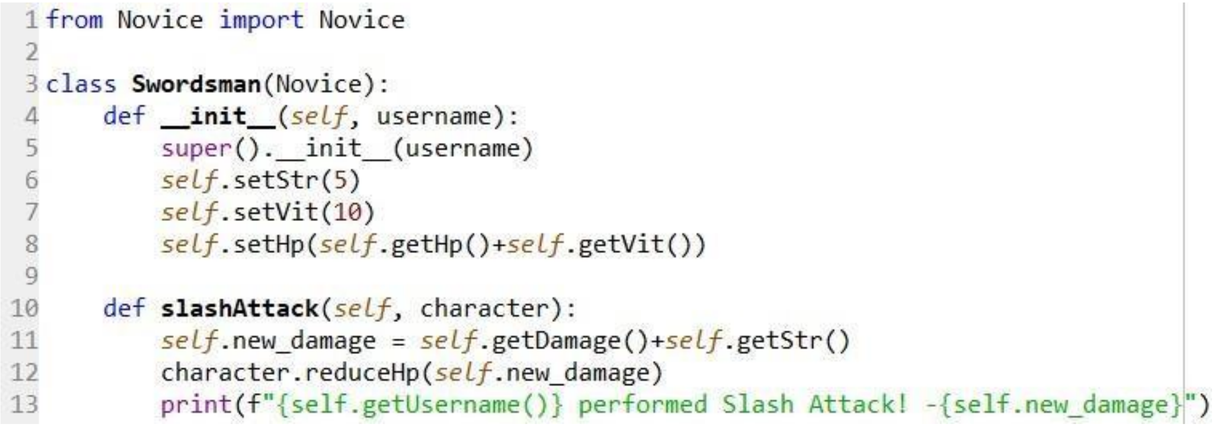
character1 = Novice(“Your Username”)
print(character1.getUsername())
print(character1.getHp())

3. Observe the output and analyze its meaning then comment the added code.

Multi-level Inheritance

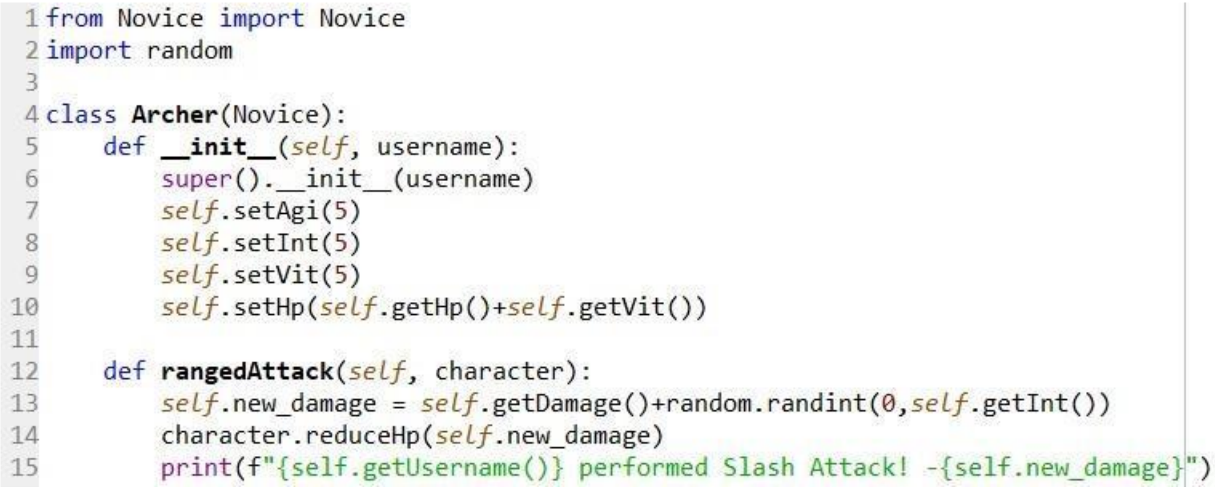
1. In the Swordsman, Archer, and Magician .py files copy the following codes for each file:

Swordsman.py



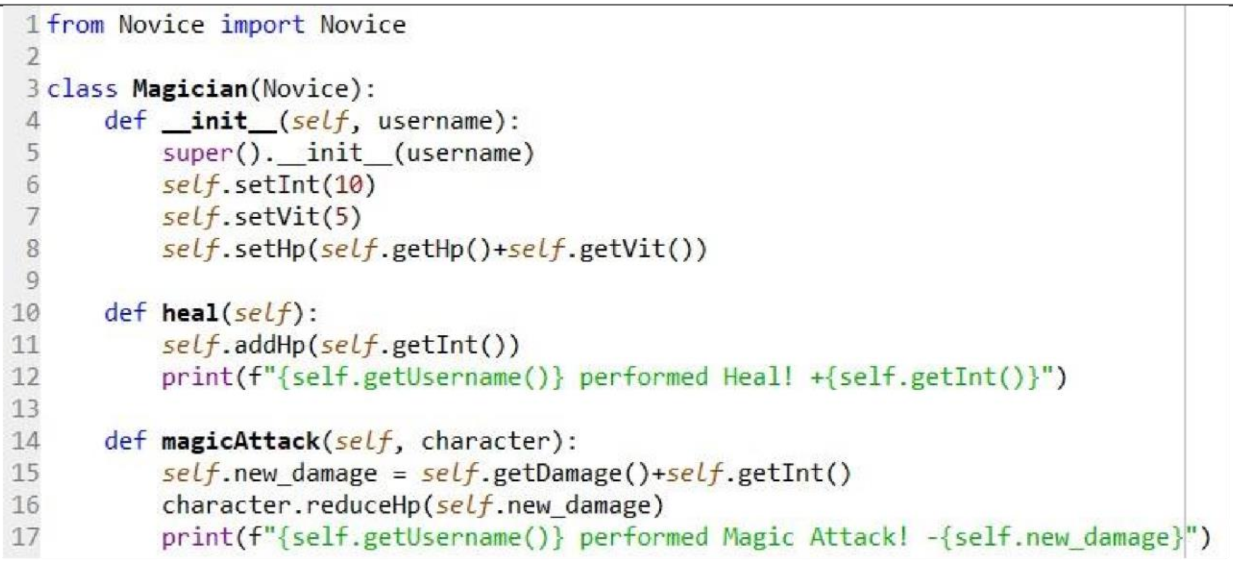
```
from Novice import Novice

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



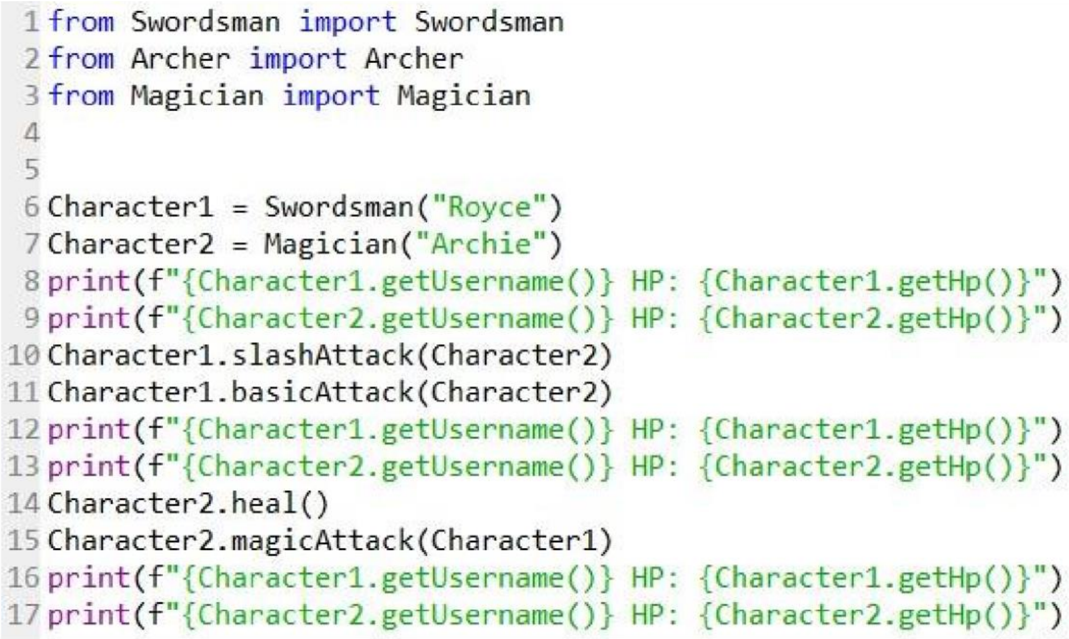
```
from Novice import Novice
import random

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

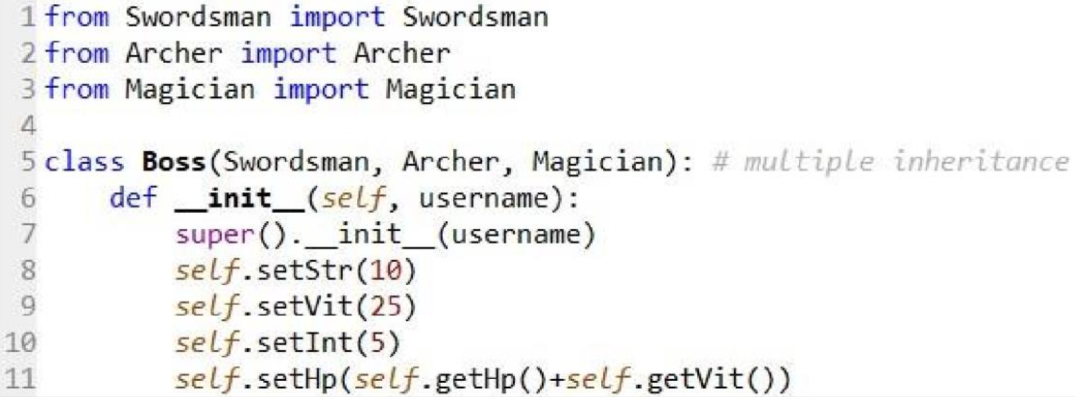


```python
from Swordsman import Swordsman
from Archer import Archer
from Magician import Magician


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 then run the program again and observe the output.

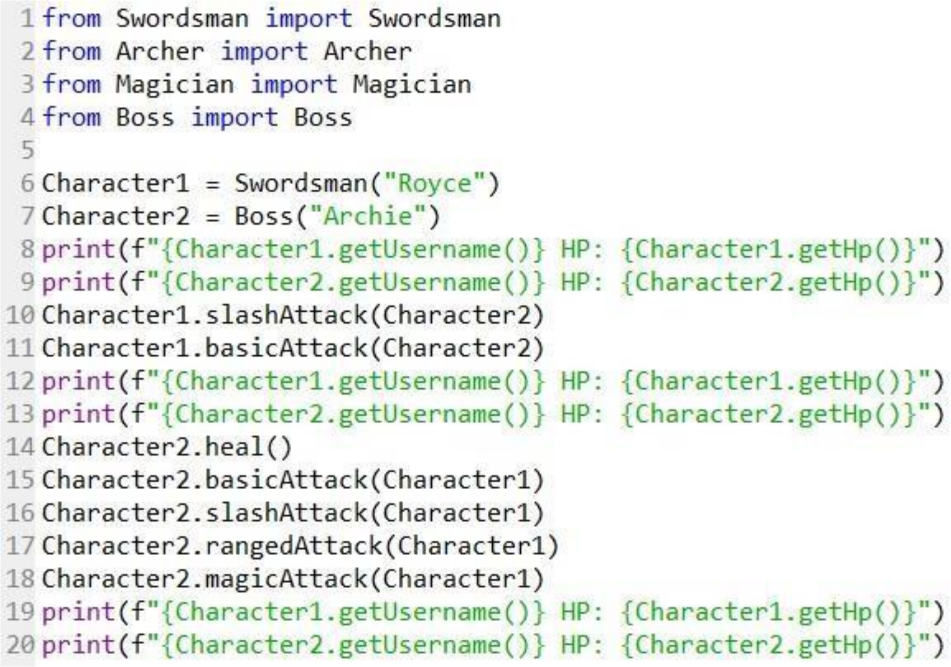
Multiple Inheritance

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.

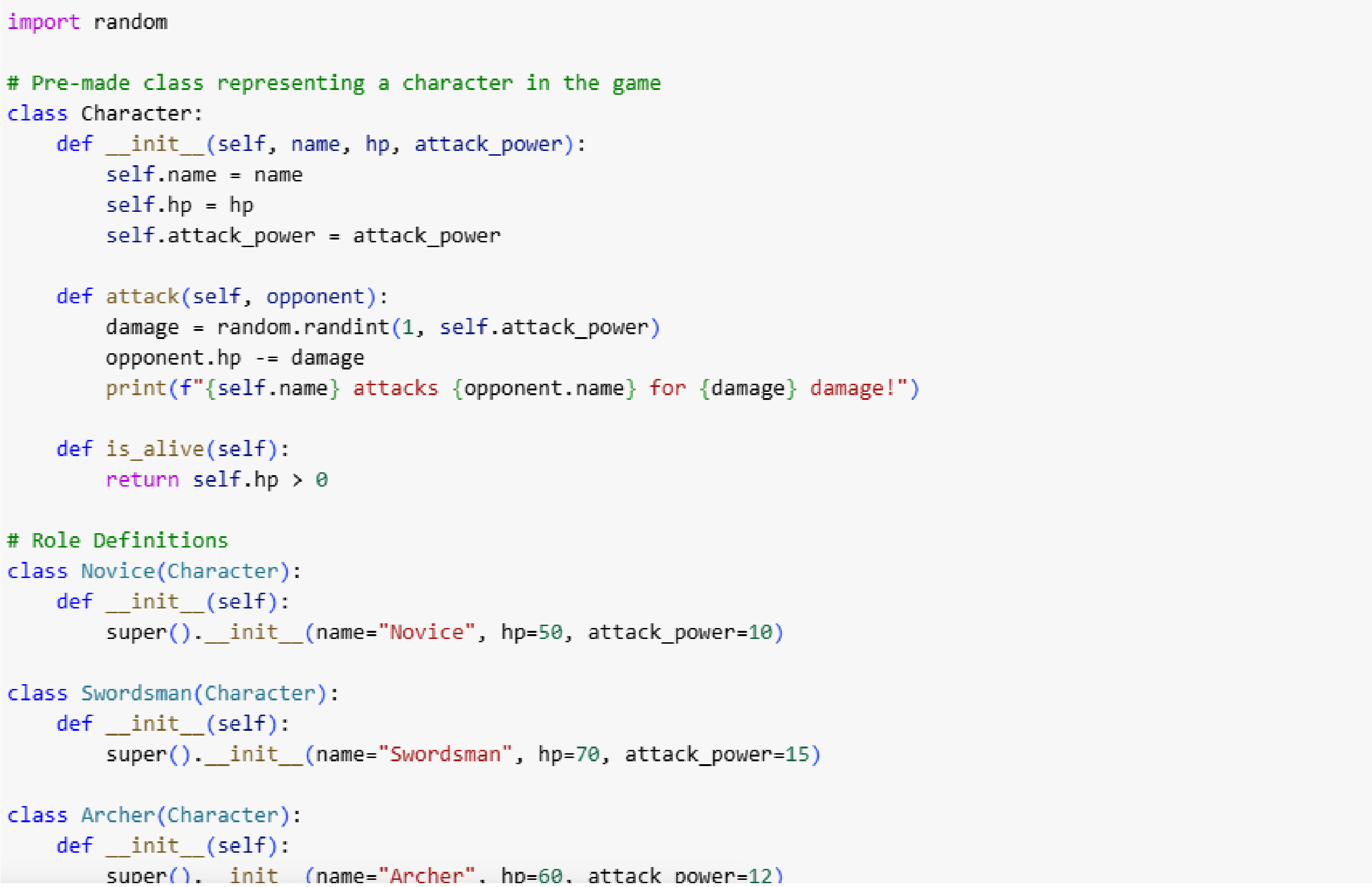
6. Supplementary Activity:

Task

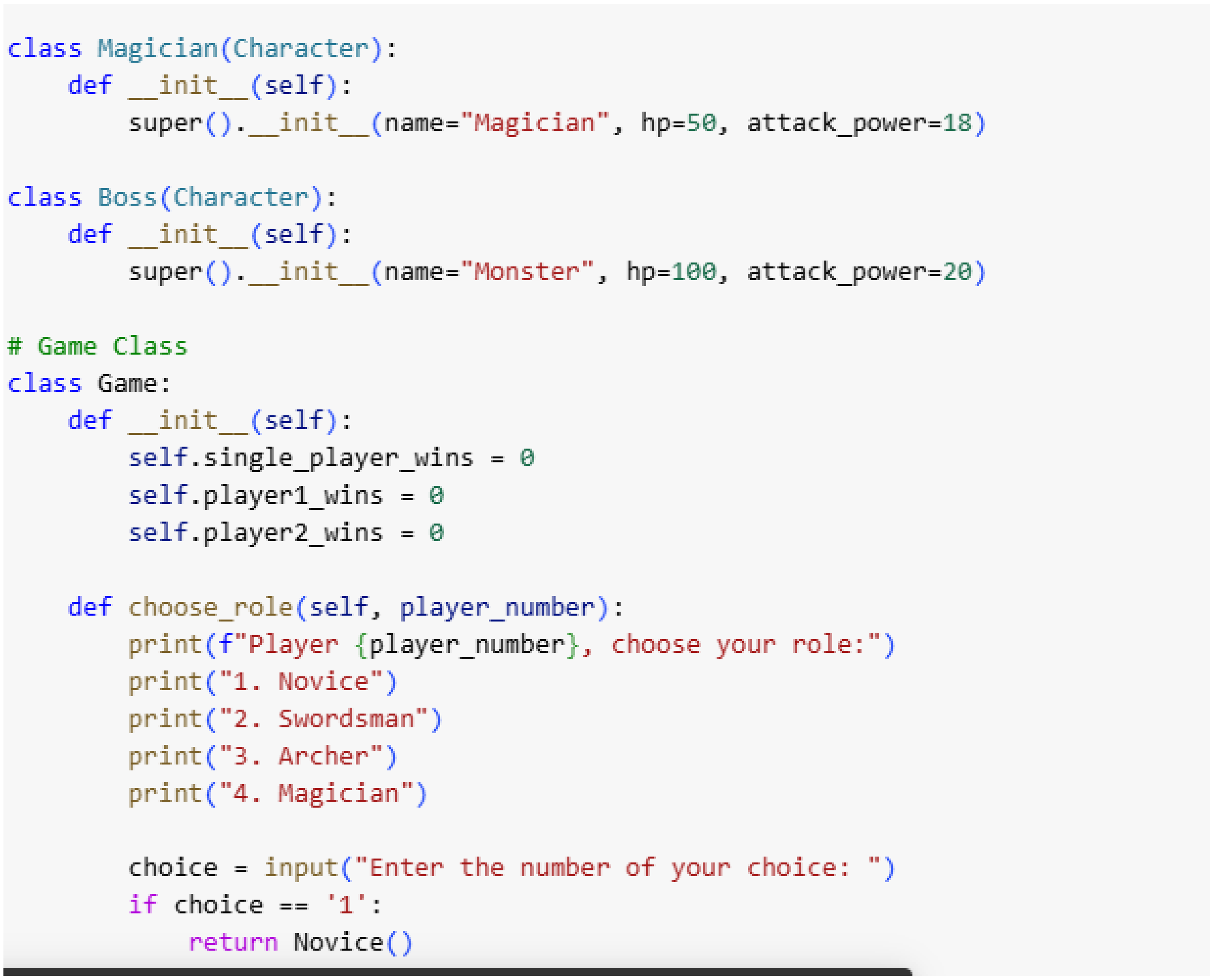
Create a new file Game.py inside the same folder use the pre-made classes to create a simple Game where two players or one player vs a computer will be able to reduce their opponent's hp to 0.

Requirements:

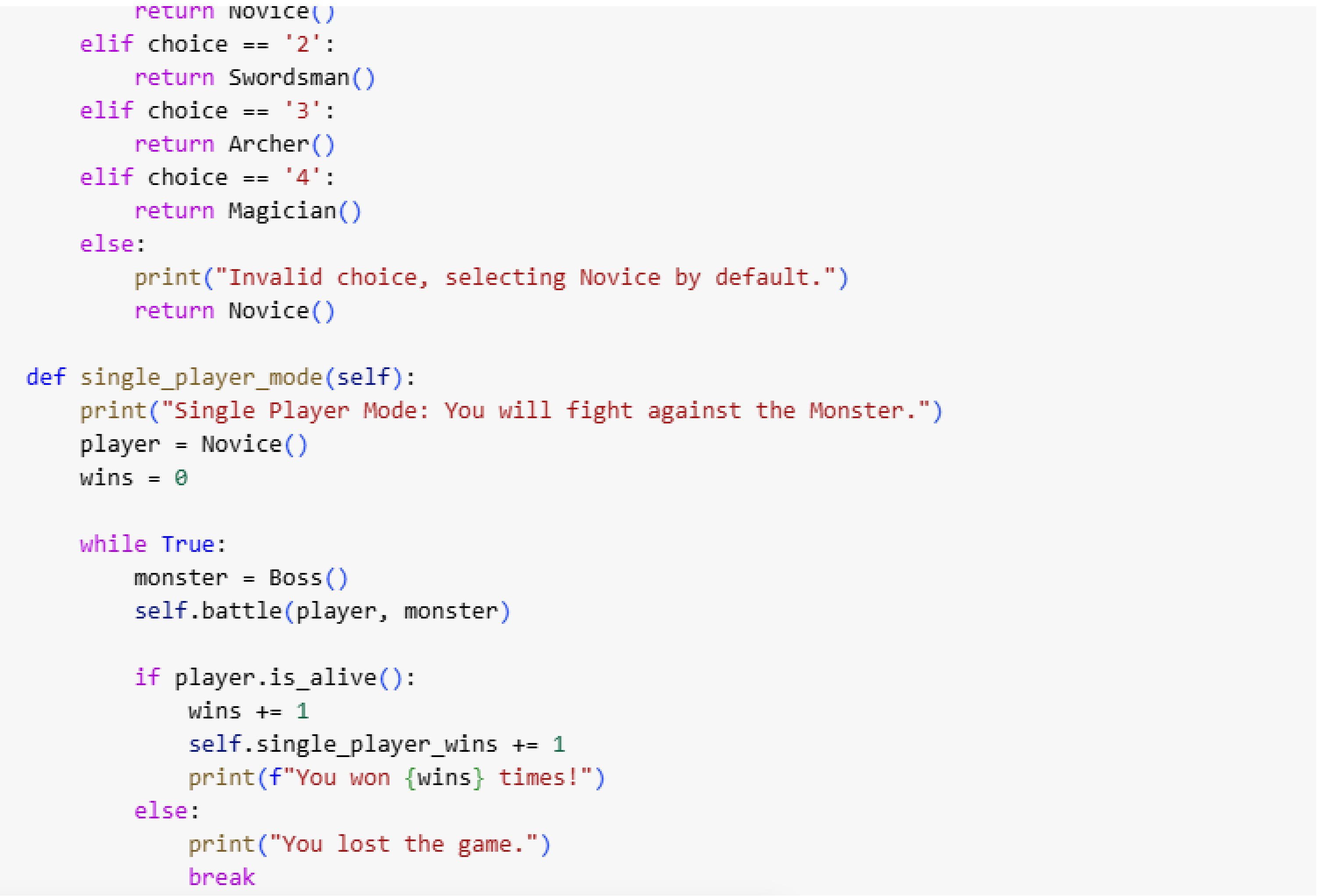
1. The game must be able to select between 2 modes: Single player and Player vs Player. The game can spawn multiple matches where single player or player vs player can take place.
2. In Single player:
   - the player must start as a Novice, then after 2 wins, the player should be able to select a new 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.



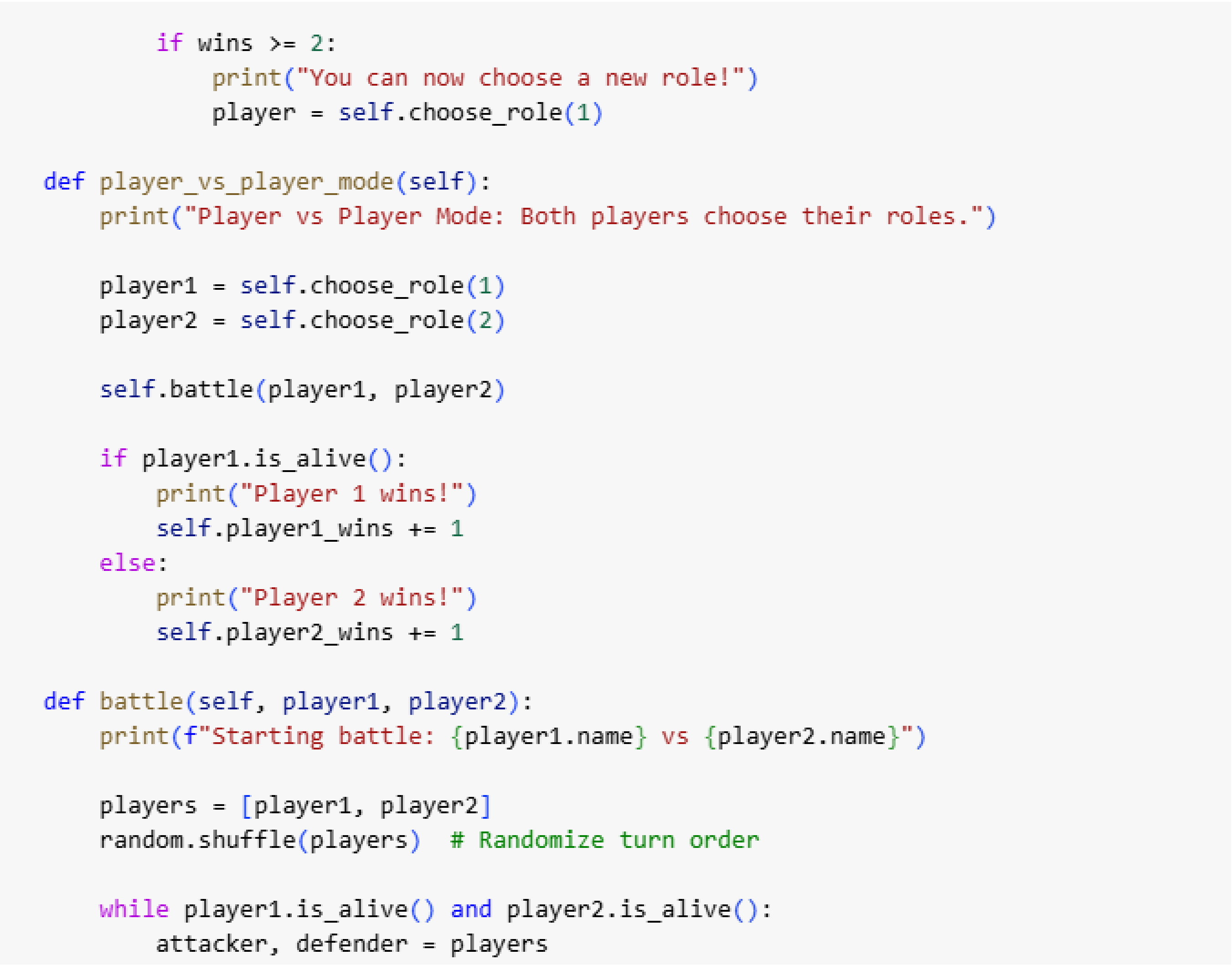
```
import random

# Pre-made class representing a character in the game
class Character:
    def __init__(self, name, hp, attack_power):
        self.name = name
        self.hp = hp
        self.attack_power = attack_power

    def attack(self, opponent):
        damage = random.randint(1, self.attack_power)
        opponent.hp -= damage
        print(f"{self.name} attacks {opponent.name} for {damage} damage!")

    def is_alive(self):
        return self.hp > 0

# Role Definitions
class Novice(Character):
    def __init__(self):
        super().__init__(name="Novice", hp=50, attack_power=10)

class Swordsman(Character):
    def __init__(self):
        super().__init__(name="Swordsman", hp=70, attack_power=15)

class Archer(Character):
    def __init__(self):
        super().__init__(name="Archer", hp=60, attack_power=12)
```



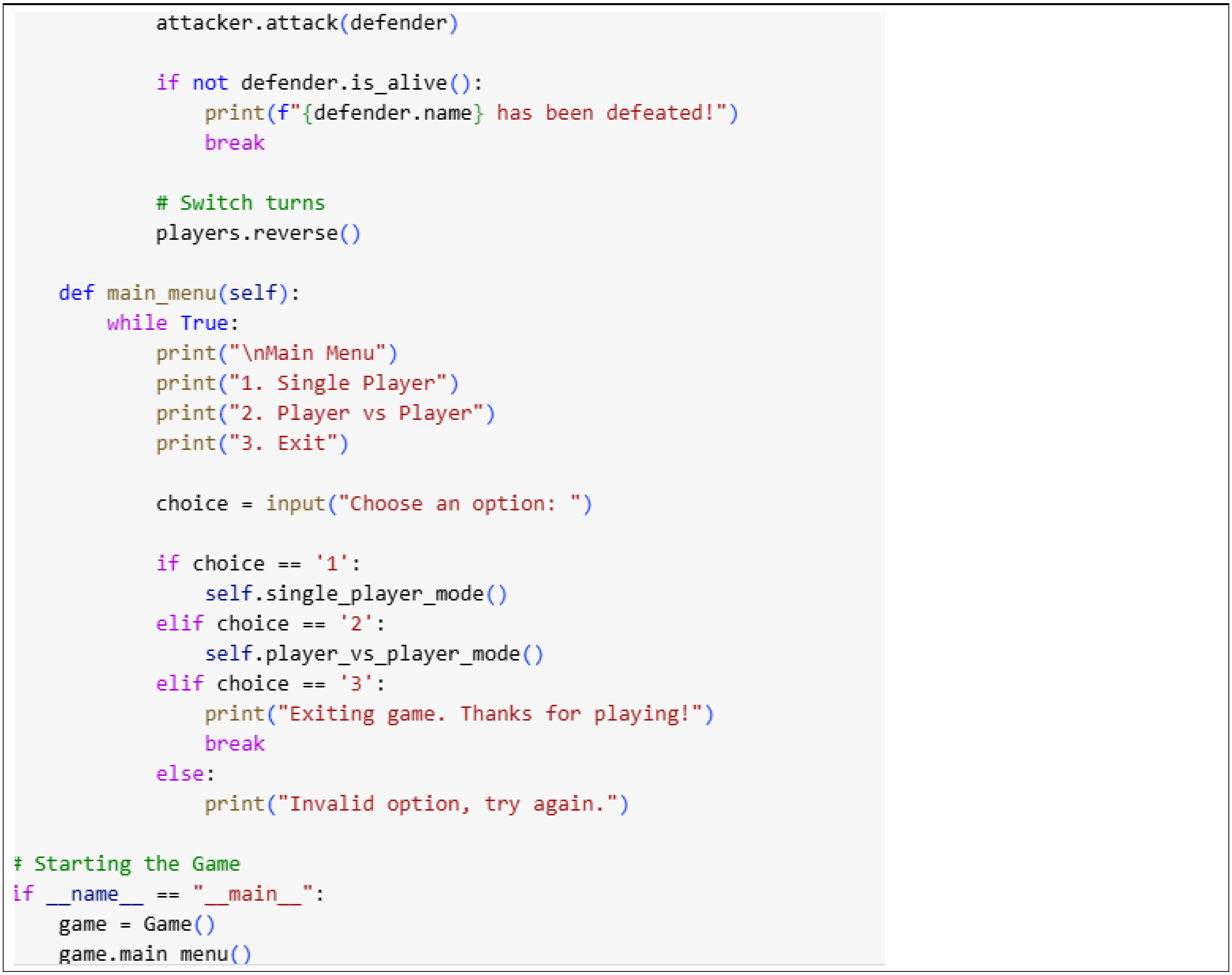
```python
class Magician(Character):
    def __init__(self):
        super().__init__(name="Magician", hp=50, attack_power=18)

class Boss(Character):
    def __init__(self):
        super().__init__(name="Monster", hp=100, attack_power=20)

# Game Class
class Game:
    def __init__(self):
        self.single_player_wins = 0
        self.player1_wins = 0
        self.player2_wins = 0

    def choose_role(self, player_number):
        print(f"Player {player_number}, choose your role:")
        print("1. Novice")
        print("2. Swordsman")
        print("3. Archer")
        print("4. Magician")

        choice = input("Enter the number of your choice: ")
        if choice == '1':
            return Novice()
```



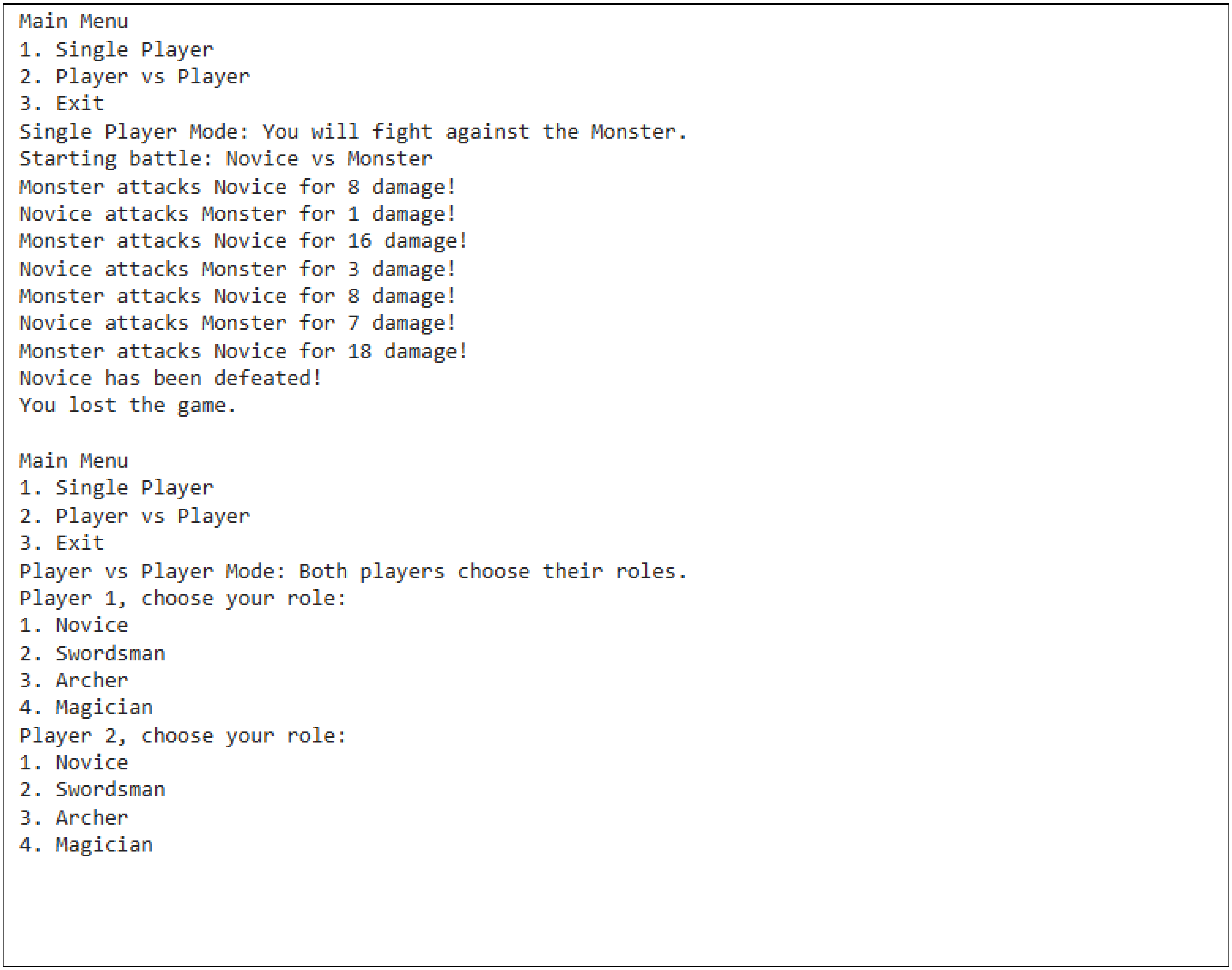
```
            return Novice()
        elif choice == '2':
            return Swordsman()
        elif choice == '3':
            return Archer()
        elif choice == '4':
            return Magician()
        else:
            print("Invalid choice, selecting Novice by default.")
            return Novice()

    def single_player_mode(self):
        print("Single Player Mode: You will fight against the Monster.")
        player = Novice()
        wins = 0

        while True:
            monster = Boss()
            self.battle(player, monster)

            if player.is_alive():
                wins += 1
                self.single_player_wins += 1
                print(f"You won {wins} times!")
            else:
                print("You lost the game.")
                break
```



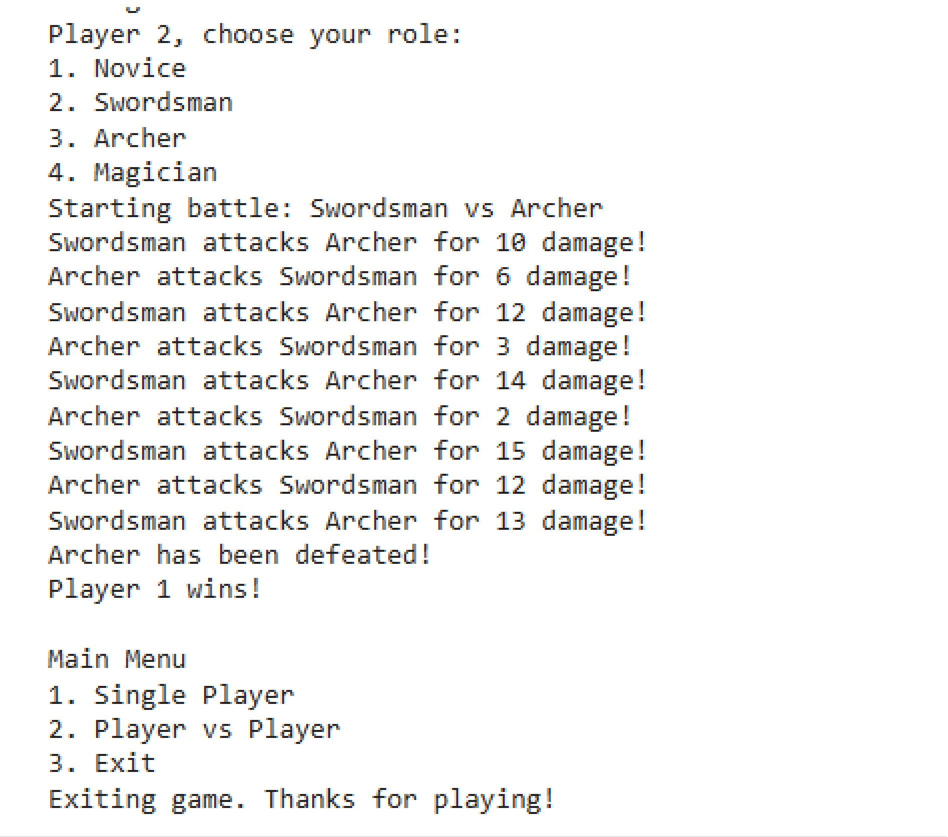
```python
            if wins >= 2:
                print("You can now choose a new role!")
                player = self.choose_role(1)

    def player_vs_player_mode(self):
        print("Player vs Player Mode: Both players choose their roles.")

        player1 = self.choose_role(1)
        player2 = self.choose_role(2)

        self.battle(player1, player2)

        if player1.is_alive():
            print("Player 1 wins!")
            self.player1_wins += 1
        else:
            print("Player 2 wins!")
            self.player2_wins += 1

    def battle(self, player1, player2):
        print(f"Starting battle: {player1.name} vs {player2.name}")

        players = [player1, player2]
        random.shuffle(players)  # Randomize turn order

        while player1.is_alive() and player2.is_alive():
            attacker, defender = players
```



```python
            attacker.attack(defender)

            if not defender.is_alive():
                print(f"{defender.name} has been defeated!")
                break

            # Switch turns
            players.reverse()

    def main_menu(self):
        while True:
            print("\nMain Menu")
            print("1. Single Player")
            print("2. Player vs Player")
            print("3. Exit")

            choice = input("Choose an option: ")

            if choice == '1':
                self.single_player_mode()
            elif choice == '2':
                self.player_vs_player_mode()
            elif choice == '3':
                print("Exiting game. Thanks for playing!")
                break
            else:
                print("Invalid option, try again.")

# Starting the Game
if __name__ == "__main__":
    game = Game()
    game.main_menu()
```



```
Main Menu
1. Single Player
2. Player vs Player
3. Exit
Single Player Mode: You will fight against the Monster.
Starting battle: Novice vs Monster
Monster attacks Novice for 8 damage!
Novice attacks Monster for 1 damage!
Monster attacks Novice for 16 damage!
Novice attacks Monster for 3 damage!
Monster attacks Novice for 8 damage!
Novice attacks Monster for 7 damage!
Monster attacks Novice for 18 damage!
Novice has been defeated!
You lost the game.

Main Menu
1. Single Player
2. Player vs Player
3. Exit
Player vs Player Mode: Both players choose their roles.
Player 1, choose your role:
1. Novice
2. Swordsman
3. Archer
4. Magician
Player 2, choose your role:
1. Novice
2. Swordsman
3. Archer
4. Magician
```



```
Player 2, choose your role:
1. Novice
2. Swordsman
3. Archer
4. Magician
Starting battle: Swordsman vs Archer
Swordsman attacks Archer for 10 damage!
Archer attacks Swordsman for 6 damage!
Swordsman attacks Archer for 12 damage!
Archer attacks Swordsman for 3 damage!
Swordsman attacks Archer for 14 damage!
Archer attacks Swordsman for 2 damage!
Swordsman attacks Archer for 15 damage!
Archer attacks Swordsman for 12 damage!
Swordsman attacks Archer for 13 damage!
Archer has been defeated!
Player 1 wins!

Main Menu
1. Single Player
2. Player vs Player
3. Exit
Exiting game. Thanks for playing!
```

Questions

1. Why is Inheritance important?
   To be able to reuse the codes on a class. With inheritance, a new class (child class) can use properties and behavior from an existing class (parent class), which reduces redundancy and makes the code more maintainable.

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

It will make it easier to use again the code you're looking for but the It will increase the chance of performance issues and complexity

3. Differentiate single inheritance, multiple inheritance, and multi-level inheritance.
   The single inheritance is one way code that a child class inherits from the one parent class.
   Multiple inheritance is when the child class inherits from two or more parent class.
   Multi-level inheritance is when the child class inherits from the another child class and this child class is inherited from the parent class.

4. Why is super().__init__(username) added in the codes of Swordsman, Archer, Magician, and Boss?
   It will ensures proper initialization of the superclass, passes necessary parameters, avoids code duplication. It ensures that the initialization logic from the parent class is executed in the child class. Specifically, it passes parameters like Name, HP and attack power to the parent class so the child class can inherit those properties.
5. How do you think Encapsulation and Abstraction helps in making good Object-Oriented Programs?
   It will help the programmers to make their code flexible and simple and be able to reuse it and modify it when needed. Encapsulation means bundling the data (attributes) and methods (functions) that operate on the data within a class, and restricting direct access to some of the class's components.

   And Abstraction means hiding the implementation details of a class and exposing only the needed features.

7. Conclusion:

Inheritance plays a big part on creating to make ur code to use it again and this activity helps us understand it better. Inheritance, encapsulation, and abstraction are foundational principles of Object-Oriented Programming that promote reusability and maintainability. Inheritance allows classes to inherit functionality, reducing redundancy and fostering code reuse.

8. Assessment Rubric: