Activity # 2 - Inheritance, Encapsulation, and Abstraction	
Bona, Andrei Nycole So	09/29/2024
CPE009B - CPE21S4	Prof. Maria Rizette Sayo

6. Supplementary Activity:

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

def choose_role(username, is_boss=False):
    if is_boss:
        return Boss(username)
    else:
        print("Choose a role:")
        print("1. Novice")
        print("2. Swordsman")
        print("3. Archer")
        print("4. Magician")
        choice = int(input("Enter the number of your choice: "))
        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("Invalid choice, defaulting to Novice.")
            return Novice(username)

def player_turn(attacker, defender):
    print(f"\n{attacker.getUsername()}'s turn!")
    print("Choose an action:")
    print("1. Basic Attack")
    if isinstance(attacker, Swordsman):
        print("2. Slash Attack")
```

```python
    elif isinstance(attacker, Archer):
        print("2. Ranged Attack")
    elif isinstance(attacker, Magician):
        print("2. Magic Attack")
        print("3. Heal")

    choice = int(input("Enter the number of your choice: "))
    if choice == 1:
        attacker.basicAttack(defender)
    elif choice == 2:
        if isinstance(attacker, Swordsman):
            attacker.slashAttack(defender)
        elif isinstance(attacker, Archer):
            attacker.rangedAttack(defender)
        elif isinstance(attacker, Magician):
            attacker.magicAttack(defender)
    elif choice == 3 and isinstance(attacker, Magician):
        attacker.heal()
    else:
        print("Invalid choice, performing Basic Attack.")
        attacker.basicAttack(defender)

def play_match(player1, player2):
    while player1.getHp() > 0 and player2.getHp() > 0:
        if random.choice([True, False]):
            player_turn(player1, player2)
        else:
            player_turn(player2, player1)

        print(f"\n{player1.getUsername()} HP: {player1.getHp()}")
        print(f"{player2.getUsername()} HP: {player2.getHp()}")

    if player1.getHp() <= 0:
        print(f"\n{player1.getUsername()} has been defeated!")
        return player2
    else:
        print(f"\n{player2.getUsername()} has been defeated!")
        return player1

def single_player_game():
    player = Novice("Player")
    wins = 0

    while True:
        opponent = Boss("Monster")
        winner = play_match(player, opponent)

        if winner == player:
            wins += 1
            print(f"Congratulations! You have {wins} win(s).")
            if wins == 2:
                print("\nYou can now choose a new role!")
                player = choose_role("Player")
        else:
            print("You lost! Better luck next time.")
            break

def player_vs_player_game():
    player1 = choose_role("Player 1")
    player2 = choose_role("Player 2")
    wins_p1 = 0
    wins_p2 = 0

    while True:
        winner = play_match(player1, player2)

        if winner == player1:
            wins_p1 += 1
            print(f"Player 1 wins! Total wins: {wins_p1}")
        else:
            wins_p2 += 1
            print(f"Player 2 wins! Total wins: {wins_p2}")

        play_again = input("Do you want to play another match? (yes/no):
").lower()
        if play_again != "yes":
            break

def main():
    while True:
        print("Welcome to the Game!")
        print("1. Single Player")
        print("2. Player vs Player")
        print("3. Exit")
        mode = int(input("Enter the number of your choice: "))

        if mode == 1:
            single_player_game()
        elif mode == 2:
            player_vs_player_game()
        elif mode == 3:
            print("Thanks for playing!")
            break
        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()
```

Questions
1. Why is Inheritance important?
Inheritance is a fundamental feature in object-oriented programming that lets classes reuse the code from other classes, so making the development process faster and simpler. It also helps in making a distinct flow thus reducing the replication of the code and also it becomes easy to maintain and develop additional features.

2. Explain the advantages and disadvantages of using applying inheritance in an Object-Oriented Program.
The main advantage of inheritance is code reuse, which avoids redundancy and simplifies maintenance, and better arranges code. On the other hand, overuse of inheritance makes code complex and harder to maintain when changes are made to a base class because they may multiply down to many derived classes. Therefore, it may have tight dependency and modifications become riskier.

3. Differentiate single inheritance, multiple inheritance, and multi-level inheritance.
Single Inheritance: A class inherits from one parent class.
Multiple Inheritance: A class inherits from multiple parent classes, combining their functionalities.
Multi-level inheritance: A class inherits from another derived class, creating a chain of inheritance.

4. Why is super(). init (username) added in the codes of Swordsman, Archer, Magician, and Boss?
super().__init__(username) calls the constructor of its parent so that all attributes declared in the base class can be properly initialized in the derived class. It doesn't repeat the code, and at the same time, it guarantees that the child class inherits all the properties that the parent class possesses.

5. How do you think Encapsulation and Abstraction helps in making good Object-Oriented Programs?
The concept of encapsulation bundles data and methods in one class and limits direct access to an internal state, hence helping in protection from data misuse. The concept of abstraction hides unnecessary details, focuses on important features, and makes the code easier to use and understand, hence together making code more secure, organized, and easier to maintain.

7. Conclusion:

Inheritance, encapsulation, and abstraction are key OOP concepts that help create reusable, organized, and maintainable code. They reduce redundancy, protect data, and simplify complex systems, leading to better software design and easier maintenance.

8. Assessment Rubric: