Activity No. 2	
Inheritance, Encapsulation, and Abstraction	
Course Code: CPE 009B	**Program:** Computer Engineering
Course Title: Object Oriented Program	**Date Performed: 09/29/2024**
Section: CPE21s4	**Date Submitted:09/29/2024**
Name(s): Anna Marie Zolina	**Instructor: Ms. Maria Rizette Sayo**

5. Procedure:

Creating the Classes:

```
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 #agality 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

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

Output:

```
  File c:
\users\amzol\desktop\oopfa1_zolina\character.py:45
    print(character1.__username)

AttributeError: 'Character' object has no
attribute '__username'
```

without the “print(character1.__username)”

```
In [4]: runfile('C:/Users/amzol/Desktop/
oopfa1_zolina/Character.py', wdir='C:/Users/amzol/
Desktop/oopfa1_zolina')
Your Username
```

Single Inheritance:

```
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())
```

Output:

```
PS C:\Users\amzol> & C:/Users/amzol/AppData/Lo
Your Username
Your Username
100
PS C:\Users\amzol>
```

Multi-level Inheritance

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! -1{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 Arrack! -{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()} perfomed Heal! +{self.getInt()}")


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

Creating Test File:

```
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()}")
```

Output:

```
Archie HP: 105
Royce performed Slash Attack! -110
Royce performed Basic Attack! -5
Royce HP: 110
Archie HP: 90
Archie perfomed Heal! +10
Archie Perfprmed Magic Attack! -15
Royce HP: 95
Archie HP: 100
```

Changing the "magic attack" to basic attack.

```
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()}")
```

Output:

```
Archie HP: 90
Archie perfomed Heal! +10
Traceback (most recent call last):
  File "c:\Users\amzol\Desktop\oopfa1_zolina\Test.py", line 15, in <module>
    Character2.slashAttack(Character1)
    ^^^^^^^^^^^^^^^^^^^^^^
AttributeError: 'Magician' object has no attribute 'slashAttack'. Did you mean: 'basicAttack'?
```

- **Error because the magician does not use slash attack.**

```
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()}")
```

Output:

```
Royce HP: 110
Archie HP: 130
Archie perfomed Heal! +5
Archie performed Basic Attack! -5
Archie performed Slash Attack! -15
Archie performed Ranged Attack! -7
Archie Perfprmed Magic Attack! -10
Royce HP: 73
Archie HP: 135
```

6. Supplementary Activity

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

class Game:
    def __init__(self):
        self.single_player_wins = 0
        self.current_role = Novice
        self.roles = {
```

```
            0: Novice,      # 0 wins: Novice
            1: Swordsman,   # 1 win: Swordsman
            2: Archer,      # 2 wins: Archer
            3: Magician     # 3 wins: Magician
        }
        self.mode = None

    def start_game(self):
        print("Welcome to the game!")
        self.select_mode()  # Select mode at the start

    def select_mode(self):
        while self.mode not in ["1", "2"]:
            self.mode = input("Select game mode (1 for Single Player, 2 for Player
vs Player): ")
            if self.mode not in ["1", "2"]:
                print("Invalid mode. Please select a valid mode.")

        if self.mode == "1":
            self.single_player_mode()
        elif self.mode == "2":
            self.player_vs_player_mode()

    def single_player_mode(self):
        while True:
            player = self.current_role("Player")
            boss = Boss("Monster")
            self.set_boss_stats(boss)
            self.play_match(player, boss)
            if player.getHp() > 0:  # Only allow role change if player won
                self.single_player_wins += 1
                self.change_role()

            if not self.play_again():
                break  # Exit single player mode

        self.reset_game()

    def player_vs_player_mode(self):
        while True:
            player1 = self.select_role("Player 1")
            player2 = self.select_role("Player 2")
            self.play_match(player1, player2)
```

```
            if not self.play_again():
                break  # Exit player vs player mode

        self.reset_game()

    def change_role(self):
        if self.single_player_wins in self.roles:
            self.current_role = self.roles[self.single_player_wins]
            print(f"You have upgraded to {self.current_role.__name__}!")
        else:
            print("You are already at the highest level (Magician).")

    def select_role(self, player_name):
        print(f"Select a role for {player_name}:")
        for num, role in self.roles.items():
            print(f"{num}: {role.__name__}")
        role_choice = input("Enter the number of your chosen role: ")
        return self.roles.get(int(role_choice), Novice)(player_name)

    def play_match(self, player1, player2):
        print(f"\n{player1.getUsername()} vs {player2.getUsername()}")
        while player1.getHp() > 0 and player2.getHp() > 0:
            for player in [player1, player2]:
                print(f"\n{player.getUsername()}'s turn")
                action = input("Enter 'a' to attack or 'h' to heal:
").strip().lower()

                if action == 'a':
                    player.basicAttack(player2 if player == player1 else player1)
                elif action == 'h' and isinstance(player, Magician):
                    player.heal()
                else:
                    if action == 'h':
                        print("Only Magician can heal.")
                    else:
                        print("Invalid action. Please enter 'a' or 'h'.")
                    continue

                # Display HP after action
                print(f"{player1.getUsername()} HP: {player1.getHp()}")
                print(f"{player2.getUsername()} HP: {player2.getHp()}")

        if player1.getHp() <= 0:
            print(f"{player2.getUsername()} wins!")
```

```
        else:
            print(f"{player1.getUsername()} wins!")

    def set_boss_stats(self, boss):
        # Set Boss stats based on player's win count
        if self.single_player_wins < 2:
            boss.setHp(50)  # Easy level Boss HP
            boss.setDamage(5)  # Easy level Boss Damage
        elif self.single_player_wins < 5:
            boss.setHp(75)  # Medium level Boss HP
            boss.setDamage(10)  # Medium level Boss Damage
        else:
            boss.setHp(100)  # Hard level Boss HP
            boss.setDamage(15)  # Hard level Boss Damage

    def play_again(self):
        return input("Do you want to play again? (y/n): ").strip().lower() == 'y'

    def reset_game(self):
        # Reset for a new game session
        self.single_player_wins = 0
        self.current_role = Novice
        self.mode = None  # Reset mode selection

if __name__ == "__main__":
    game = Game()
    game.start_game()
```

Questions

1. Why is Inheritance important?

- As I began writing code, I noticed some errors in the classes I had created, because some functions from other classes were not added from that specific class. I believe that inheritance is important because it allows me to create new classes based on existing ones, enhancing code reuse and reducing redundancy. This not only keeps my code cleaner, but it's also easier to maintain and extend.

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

- The benefits of using inheritance include increased code reusability and a logical class hierarchy, which makes my program easier to understand. However, it has some drawbacks, such as a close connection between classes, which can make modifications difficult if the base class changes..

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

- In my code, single inheritance is obvious because the Swordsman class inherits solely from Novice. Multiple class from the Boss class inherits from Swordsman, Archer, and Magician, giving it access to all three classes' features. Multi-level inheritance occurs when a class inherits from a subclass, adding another layer to the hierarchy.

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

- In subclasses, I make sure to call the parent class's constructor and initialize inherited attributes appropriately by using super().__init__(username). For classes like Swordsman and Archer, this is essential because it establishes fundamental attributes like damage and HP while enabling me to add particular traits particular to each subclass.

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

- By keeping some attributes private, encapsulation helps in the protection of an object's internal state, preventing outside modification and preserving data integrity. Abstraction, in the meantime, enables me to create classes that highlight crucial behaviors without disclosing unneeded information. When combined, they allow me to create a more organized and effective code structure that facilitates working with my objects.

7. Conclusion

- To summarize, the utilization of inheritance, encapsulation, and abstraction in my object-oriented program greatly improves its maintainability and comprehensibility. My classes stay secure and neat because of the design principles, which also encourage code reuse. With the help of these ideas, I was able to make my game more manageable and extensible, as well as more adaptable and stable.

8. Assessment Rubric