|  |  |
| --- | --- |
| **Laboratory Activity No. 6** | |
| **Inheritance, Encapsulation, and Abstraction** | |
| **Course Code:** CPE103 | **Program:** BSCPE |
| **Course Title:** Object-Oriented Programming | **Date Performed:** February 29, 2025 |
| **Section:** 1 – A | **Date Submitted:** March 16, 2025 |
| **Name:** Hermosura, Leigh B. | **Instructor:** Maria Rizette M. 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:   * 1. Identify the possible attributes and methods of a given object   2. Create a class using the Python language   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   1. In the Character.py copy the following codes |

|  |
| --- |
| Note: The double underscore signifies that the variables will be inaccessible outside of the class.   1. 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())   1. Observe the output and analyze its meaning then comment the added code.   **Single Inheritance**   1. In the Novice.py class, copy the following code. |

|  |
| --- |
| 1. 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())   1. 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     Archer.py    Magician.py |

|  |
| --- |
| 1. Create a new file called Test.py and copy the codes below:      1. Run the program Test.py and observe the output. 2. 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: |

|  |
| --- |
| 1. Modify the Test.py with the code shown below:      1. 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.   For the supplementary activity, please refer to this link <https://github.com/Leigh-Hermosura/CPE-103-OOP-1-A/tree/main/Laboratory-No.6/oopfa1_Hermosura/Supplementary%20Activity>  **Questions**   1. Why is Inheritance important?   Inheritance is important because it enables code reusability and makes code more organized rather than having to retype the same function over and over again for the same outcome.   1. Explain the advantages and disadvantages of using applying inheritance in an Object-Oriented Program.   The advantages of inheritance is reusability, organization of code, and reduces code redundancy while its disadvantages include code complexity, problems when changing the base class, and it’s limited flexibility. |

|  |
| --- |
| 3. Differentiate single inheritance, multiple inheritance, and multi-level inheritance.  Single inheritance is when an object inherits from a single parent class, multiple inheritance is when an object inherits from two or more parent class, and lastly, multi-level inheritance is when an object inherits from a parent class which is also a subclass from another parent class.   1. Why is super(). init (username) added in the codes of Swordsman, Archer, Magician, and Boss?   The super().\_\_init\_\_() function calls the constructor function of the parent class which is Character(). It is added in order to access the information from the user and utilize the functions written in Character().   1. How do you think Encapsulation and Abstraction helps in making good Object-Oriented Programs?   Encapsulation helps with providing security by hiding important data to prevent malicious tampering while abstraction help simplify code by hiding unnecessary details and focusing solely on how these classes work. |
| **7. Conclusion:** |
| Inheritance plays a crucial role in improving code efficiency by promoting reusability, reducing redundancy, and enhancing organization. However, it also introduces challenges such as increased complexity and limitations on flexibility, especially when dealing with base class modifications. The different types of inheritance—single, multiple, and multi-level—provide various approaches for reusing and extending functionality. The use of super().\_\_init\_\_() ensures that the child class can access and build upon the parent class's properties and methods. Additionally, encapsulation and abstraction work together to secure data and simplify code, respectively, contributing to cleaner, more maintainable software development. |
| **8. Assessment Rubric:** |