Author : Diogo Rangel Dos Santos

For each of the 4 principles of Programming with Classes:

1. **Abstraction**
2. **Encapsulation**
3. **Inheritance**
4. **Polymorphism**

Answer the following:

1. **Briefly define the principle.**
2. **How did you use that principle in one of your programs.**
3. **How did using that principle help that program become more flexible for future changes?**

Answers:

1. Abstraction

Definition:
Abstraction is the principle of hiding the complex implementation details of a system and exposing only the essential features. It allows a programmer to focus on high-level functionality without worrying about the inner workings.

Application in Program:
In one of my programs, I used abstraction by creating an abstract class Shape with abstract methods like Area() and Perimeter(). The specific implementations of these methods were provided by derived classes like Circle and Rectangle. The user interacts with the Shape class interface without needing to know how each shape calculates its area or perimeter.

Flexibility for Future Changes:
Using abstraction allows the program to easily accommodate future changes. For instance, if I decide to add a new shape (e.g., Triangle), I only need to implement the Area() and Perimeter() methods in the Triangle class. The rest of the code that uses Shape remains unaffected. This decouples the user interface from specific shape calculations, improving maintainability.

2. Encapsulation

Definition:
Encapsulation refers to the bundling of data (properties) and methods (functions) that operate on the data into a single unit or class. It also involves restricting access to certain components of the class, typically by using access modifiers like private, public, and protected.

Application in Program:
In a banking system program I developed, I encapsulated the Account class. The account balance was stored in a private variable, and I provided public methods like Deposit() and Withdraw() to modify the balance. These methods ensured that the balance couldn't be directly altered, preventing illegal operations (like withdrawing more money than available).

Flexibility for Future Changes:
Encapsulation makes it easier to change the internal workings of the class without affecting other parts of the program. If I wanted to modify how the balance is stored or introduce new validation rules for transactions, I could do so without changing how the class is used by other components of the system. This prevents external code from depending on the internal implementation details and promotes code safety.

3. Inheritance

Definition:
Inheritance is the process by which one class (derived class) acquires the properties and methods of another class (base class). This promotes code reuse and allows for the creation of hierarchies of classes.

Application in Program:
In a program where I modeled different types of employees in a company, I created a base class Employee with common properties such as Name, ID, and a method GetDetails(). Then, I derived specialized classes like Manager and Intern from Employee. These subclasses inherited common functionality from Employee but also had their specific features, like Bonus for Manager.

Flexibility for Future Changes:
Inheritance made the program more flexible by enabling the addition of new employee types without modifying the base class. For instance, if I needed to add a new type of employee (e.g., Contractor), I could simply create a new subclass and inherit the shared functionality from Employee. This reduces redundancy and makes it easier to extend the program with minimal changes to the existing code.

4. Polymorphism

Definition:
Polymorphism allows objects of different classes to be treated as objects of a common base class. It enables a single method or function to work in different ways depending on the object it is operating on, typically through method overriding or overloading.

Application in Program:
In the employee management system, I implemented polymorphism by

overriding the GetDetails() method in each subclass. For example, the Manager class's GetDetails() method returned additional information about the department, while the Intern class returned information about the internship period. When I called GetDetails() on a list of Employee objects (which could be of any type), the correct version of the method was invoked based on the actual object type, not the reference type.

Flexibility for Future Changes:
Polymorphism allows the system to easily support new behaviors without changing existing code. For instance, if a new employee type is added (e.g., Consultant), I can just override the GetDetails() method in that class. The rest of the system remains unchanged, ensuring that the application can evolve without breaking existing functionality, which makes it highly extensible and adaptable.