Hoang Linh Bui 100017631

Principles of Object-oriented Programming

I. Encapsulation:

One of the four principles of Object-oriented programming, Encapsulation, is about hiding the inner workings of an object, which are its methods and fields from the users, as well as containing them all within a single object, opposite to procedural programming, where variables(data) are stored within the scope of each function or procedure. For a method in a class, we know what it does, but that is all we need. We do not need to know how it works.

Not only does this reduces the complexity of learning how to use an object by removing the need to understand its internal content, by concealing the data inside from the users as well as forcing the users to modify or access it with defined functions only, it also restricts users from tinkering too much with the data and maintains the integrity of the data within the object.

There are three levels of access Encapsulation can determine for the elements in a class.

- **Public:** The lowest protection level. Public data and methods can be accessed from functions of all classes.
- **Protected:** Protected data can only be accessed by members of inheritance classes, which are either the same class or its derived/child classes.
- **Private:** The highest protection level. Private data can only be accessed by member functions of the same class

II. Inheritance:

The concept of Inheritance in object-oriented programming is when a new class, called a child class or derived class, is created from an existing class (or classes, in the case of certain programming languages, i.e: C++), called the base class or parent class.

This new class will possess all of the attributes and methods of its parent class (Although what it can access and utilize still depends on the level of protection of Encapsulation and which methods chosen to be passed down (either.

The advantages of Inheritance are:

- We can now define new attributes and/or methods in the subclass, which can still be applied to the attributes and methods it inherited from the base class.

- It allows us to create more specialized classes without starting from scratch. Combined with Polymorphism, we can also re-define the inherited methods

III. Polymorphism:

Meaning "having more than one different forms".

Although similar to Inheritance, while inheritance involves classes and their hierarchy, Polymorphism is more about the methods.in those classes

There are different types of Polymorphism, including *coercion, overloading, generics, and subtype*. However, only *subtype polymorphism* is specific to object-oriented programming, so we will only discuss on that type.

In object-oriented programming, as mentioned in the Inheritance section above. Using Polymorphism in conjunction with Inheritance, in a child class, you can re-define methods inherited from its parent classes into (or in C# language, override it with) a more specialized "form".

For example, we have a Shape class, which is a generic, or even abstract class that could represent any kind of shape, with a virtual, or abstract Draw method passed down to its children classes, Circle and Square. Because the way to draw a square and a triangle are different, in those children classes the Draw methods will have to act differently, even if they both come from the same method in the same parent class.

IV. Abstraction:

Abstract is generally quite similar to Encapsulation, as they both simplify an object and prevent the users from seeing the content of methods within a class. However, unlike Encapsulation, which conceals details, the process of abstraction removes the details when they are not needed, making the object even simpler. It is a step further from Encapsulation. We know a generic object has a method, but we do not need to know what it does.

As opposed to Polymorphism, in which more specialized objects are created, Abstraction generalizes objects.

For example, we have an animal class, with height and weight attributes hidden by Encapsulation, and will need methods to "measure" those. We know that an animal can move, so the Animal class has a Move method, but we do not need to be concerned with how it moves, so details about the Move method can be entirely omitted, and instead implemented in a more specialized class, such as the derived Dog class.