



# 4 tính chất OOP

## Tính trừu tượng : —→ dùng Abstract và Interface

Thể hiện rõ nhất ở Abstract và Interface :

**Abstract** : sẽ gồm có phần xử lý và và ko xử lý trong **phương thức dùng trong kế thừa**

- Sẽ gọi tới bằng extends
- Ko hỗ trợ tính đa hình
- Gồm nhiều phương thức truy cập : public , private , protected

**Interface** : Sẽ không có phần xử lý trong **phương thức dùng trong DI**

- Sẽ gọi tới bằng Implement
- Có hỗ trợ tính đa hình
- sẽ chỉ có phương thức cập là Public

## Tính kế thừa (inheritance): —→ dùng extends

Chỉ đơn giản là chúng ta tái sử dụng lại các thuộc tính, phương thức ở class khác mà không cần phải xây dựng lại từ đầu.

```
public class TaiKhoan{
    protected String ID;
    protected String name;
    protected String userName;
    protected String passWord;

    protected void input(){
        // Triển khai nhập dữ liệu
    }

    protected void output(){
        // Xuất dữ liệu
    }
}

// Định nghĩa lớp tài khoản giáo viên kế thừa lớp tài khoản
public class TKGiaoVien extends TaiKhoan {
    private String specialized; // Chuyên ngành
    private String qualification; // trình độ chuyên môn

    // Tái sử dụng phương thức đã có sẵn trong class TaiKhoan
    @Override
    public void input(){
        super.input();
        // Triển khai tiếp nhập dữ liệu cho các thuộc tính
    }

    @Override
    public void output(){
        super.output();
        // Triển khai tiếp xuất dữ liệu cho các thuộc tính
    }
}
```

## Tính đóng gói (encapsulation).

Ẩn các thông tin và cung cấp các phương thức để truy cập đến chúng

Ta sẽ phân tích mức độ truy cập của  
public , private, protected :

## Tính đa hình (*Polymorphism*)

Một đối tượng thực hiện nhiều hành vi khác nhau dựa trên cùng một phương thức

### ▼ Overloading(Nạp chồng )

là việc 1 class có nhiều phương thức nhưng **khác tham số hoặc khác kiểu dữ liệu** của tham số, nó thể hiện tính đa hình lúc Runtime

```
1 public class OverloadingExample {  
2     static int add(int a, int b) {  
3         return a + b;  
4     }  
5  
6     static int add(int a, int b, int c) {  
7         return a + b + c;  
8     }  
9 }
```

### ▼ Overriding(Ghi đè)

cài đặt lại các phương thức đã có ở lớp cha, các phương thức override ở lớp con **phải bắt buộc giống hệt lớp cha**

```
class Animal {  
    void eat() {  
        System.out.println("eating...");  
    }  
}  
  
class Dog extends Animal {  
    void eat() {  
        System.out.println("eating bread...");  
    }  
}
```

## CẢ HAI ĐỀU ĐA HÌNH LÚC RUNTIME

>>>>>> phương thức lớp con được triệu hồi tại runtime.

# Keywords to Remember in C#

**sealed** : Prevents a class from being inherited.

**abstract**: Defines a class that cannot be instantiated and must be inherited

**override**: Provides a new implementation for an inherited method in a derived class.

**static**: Declares a member that belongs to the type itself rather than to a specific object

**virtual**: Allows a method or property to be overridden in a derived class.

**const**: Defines a compile-time constant.

**readonly**: Declares a field/property that can only be assigned during initialization or in a constructor.

**async**: Marks a method as asynchronous and allows the use of the await keyword within it.

Repost 



@MuhammadWaseem

**var**: Infers the type of a variable from the expression on the right-hand side.

## Override thông qua :

### Virtual

- Cho phép lớp dẫn xuất có lựa chọn ghi đè hoặc không ghi đè phương thức.

```

public class Employee
{
    public string Name { get; set; }
    public decimal BaseSalary { get; set; }

    // Sử dụng virtual thay vì abstract
    public virtual decimal CalculateSalary()
    {
        return BaseSalary; // Triển khai mặc định
    }
}

// Nhân viên bình thường, không cần ghi đè phương thức CalculateSalary
public class RegularEmployee : Employee
{
    // Không cần ghi đè CalculateSalary vì cách tính lương giống lẫn cơ sở
}

// Nhân viên bán hàng, cần ghi đè phương thức CalculateSalary
public class SalesEmployee : Employee
{
    public decimal CommissionRate { get; set; }
    public decimal SalesAmount { get; set; }

    public override decimal CalculateSalary()
    {
        return BaseSalary + (CommissionRate * SalesAmount);
    }
}

```

*Handwritten notes in red:*

- Arrow from `CalculateSalary` in `RegularEmployee` to `CalculateSalary` in `Employee`: "không định override" (don't override)
- Arrow from `CalculateSalary` in `SalesEmployee` to `CalculateSalary` in `Employee`: "override" (override)

## Abstract

- Bắt buộc phải đc thừa kế
- Bắt buộc phải override

```

public abstract class Shape
{
    // Thuộc tính chung cho tất cả các hình
    public string Color { get; set; }

    // Phương thức abstract - bắt buộc phải được triển khai bởi lớp con
    public abstract double CalculateArea();
    public abstract double CalculatePerimeter();

    // Phương thức không abstract - có thể được sử dụng bởi tất cả các lớp con
    public virtual void DisplayInfo()
    {
        Console.WriteLine($"This is a {this.GetType().Name} and its color is {Color}");
    }
}

public class Circle : Shape
{
    public double Radius { get; set; }

    public override double CalculateArea()
    {
        return Math.PI * Radius * Radius;
    }

    public override double CalculatePerimeter()
    {
        return 2 * Math.PI * Radius;
    }
}

public class Rectangle : Shape
{
    public double Width { get; set; }
    public double Height { get; set; }

    public override double CalculateArea()
    {
        return Width * Height;
    }

    public override double CalculatePerimeter()
    {
        return 2 * (Width + Height);
    }
}

```

Bắt buộc override

- Không thể được khởi tạo trực tiếp.

Ví dụ, hãy xem xét abstract class `Employee` mà chúng ta đã định nghĩa trước đó:

```
csharp Copy

public abstract class Employee
{
    public string Name { get; set; }
    public int Id { get; set; }

    public Employee(string name, int id)
    {
        Name = name;
        Id = id;
    }

    public abstract double CalculateSalary();

    public virtual void DisplayInfo()
    {
        Console.WriteLine($"Employee ID: {Id}, Name: {Name}");
    }
}
```

Nếu bạn cố gắng tạo một đối tượng trực tiếp từ lớp `Employee`, bạn sẽ gặp lỗi biên dịch:

```
csharp Copy

// Điều này sẽ gây ra lỗi biên dịch
Employee emp = new Employee("John Doe", 1001); // Lỗi!
```

**static** cho phép thay đổi giá trị, trong khi **const** và **static readonly** không cho phép thay đổi sau khi đã được khởi tạo



```

// Const - không thể thay đổi sau khi khai báo
public const string AppName = "My ASP.NET Application";

// Static - có thể thay đổi
public static int UserCount = 0;

// Static readonly - không thể thay đổi sau khi khởi tạo
public static readonly string DatabaseConnectionString;

0 references
static ConfigurationManager()
{
    DatabaseConnectionString = "Server=myServerAddress;Database=myDataBase;User Id=myUsername;Password=myPassword";
}

1 reference
public static void IncrementUserCount()
{
    UserCount++; // Có thể thay đổi giá trị của biến static

    // Các dòng sau sẽ gây ra lỗi biên dịch:
    AppName = "New Name"; // Không thể thay đổi const
    DatabaseConnectionString = "New Connection String"; // Không thể thay đổi static readonly
}

1 reference
public static void ChangeUserCount(int newCount)
{
    UserCount = newCount; // Có thể gán giá trị mới cho biến static
}

```

*Lỗi*

## Ví dụ về **readonly** với field

## Ví dụ về `readonly` với field

csharp

Sao chép mã

```
using System;

public class MyClass
{
    // Khai báo một field readonly
    private readonly int myField;

    // Khởi tạo giá trị cho field readonly trong constructor
    public MyClass(int initialValue)
    {
        myField = initialValue;
    }

    public void PrintField()
    {
        Console.WriteLine(myField);
    }
}

class Program
{
    static void Main()
    {
        MyClass obj = new MyClass(10);
        obj.PrintField(); // Output: 10

        // Không thể gán giá trị cho field readonly ngoài constructor hoặc
        // obj.myField = 20; // Lỗi biên dịch
    }
}
```