**Đề ôn tập 2:Tính đóng gói, kế thừa, đa hình**

1. **Định nghĩa tính kế thừa trong lập trình hướng đối tượng và cho biết lợi ích của nó. Cho ví dụ minh họa.**

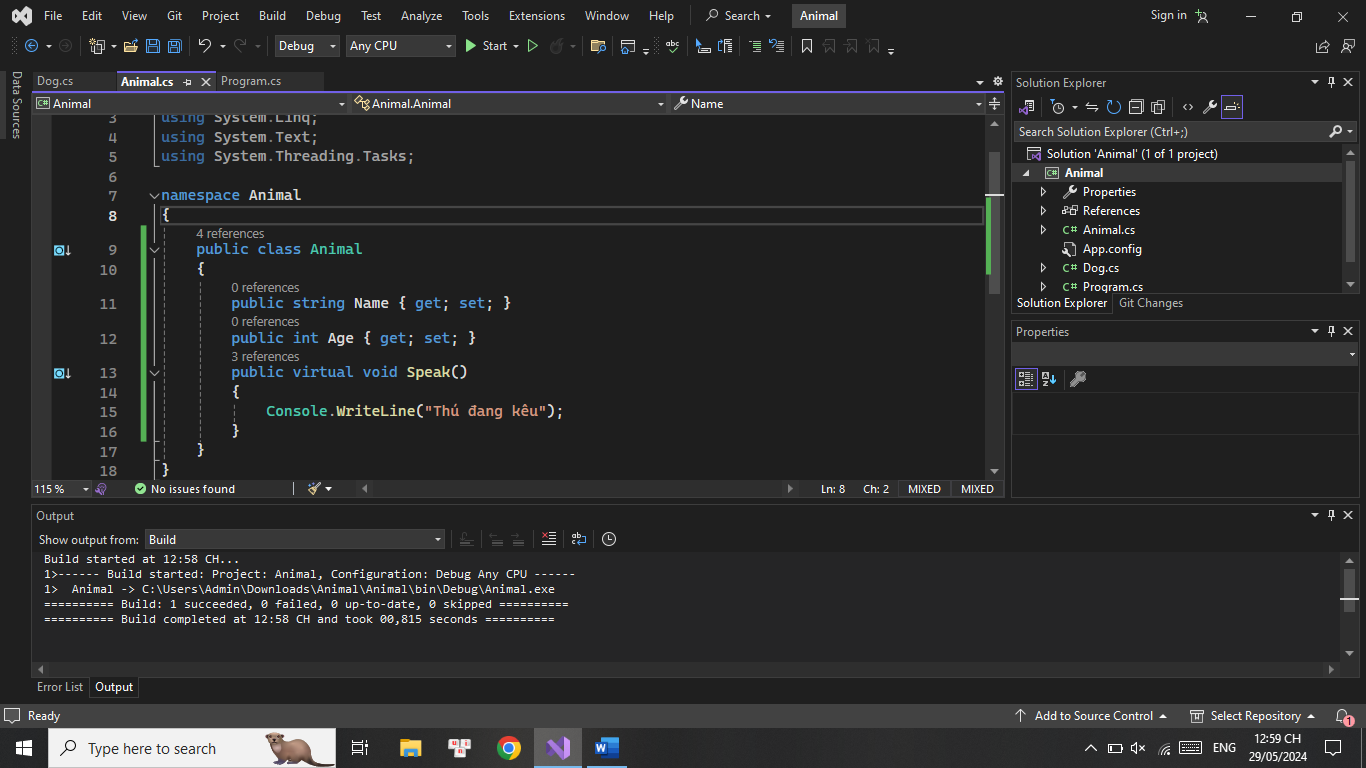
* Tính kế thừa cho phép tạo ra một lớp mới (lớp con) từ một lớp đã tồn tại (lớp cha). Lớp con sẽ thừa hưởng các thuộc tính và phương thức (trừ các thành phần private) của lớp cha, đồng thời có thể bổ sung thêm các thành phần riêng của mình.
* Mục đích:

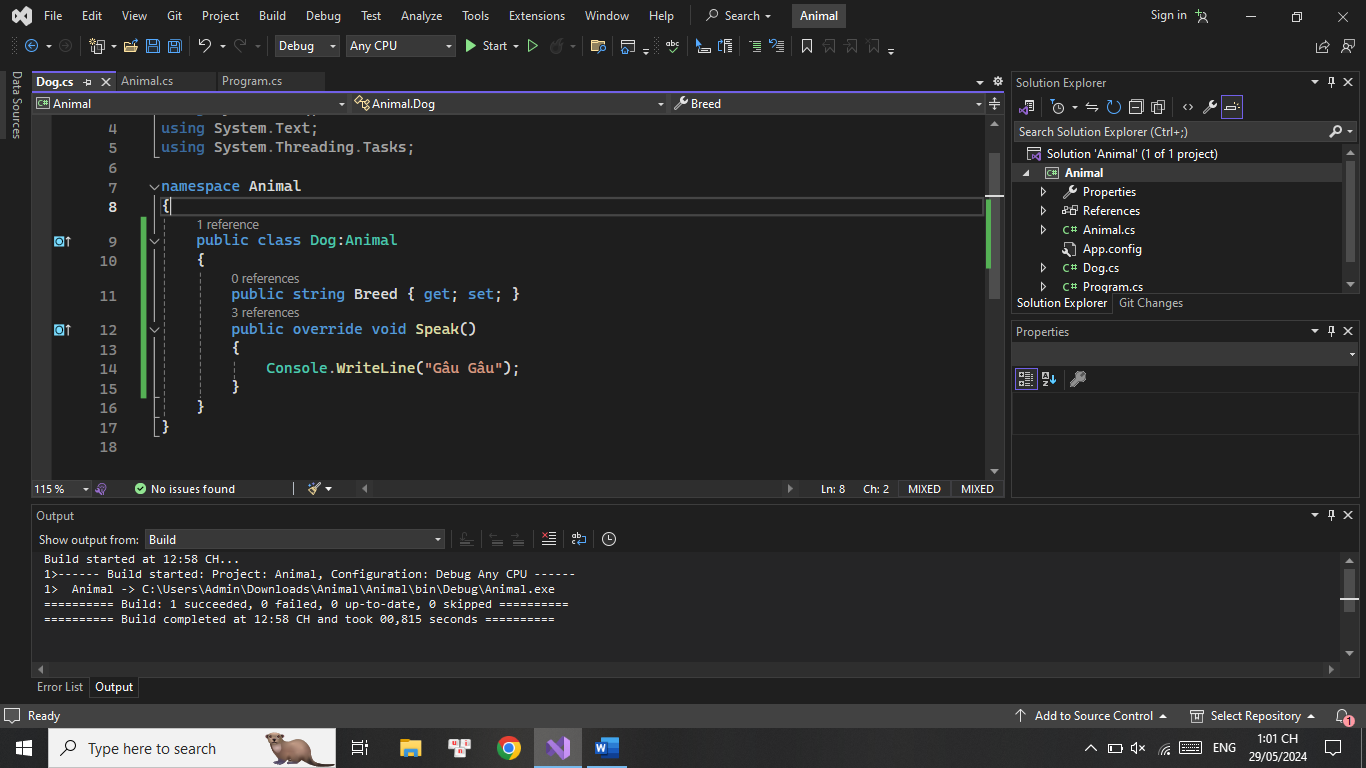
+ Tái sử dụng mã: giúp giảm thiểu việc viết mã lặp lại, tiết kiệm thời gian và công sức.

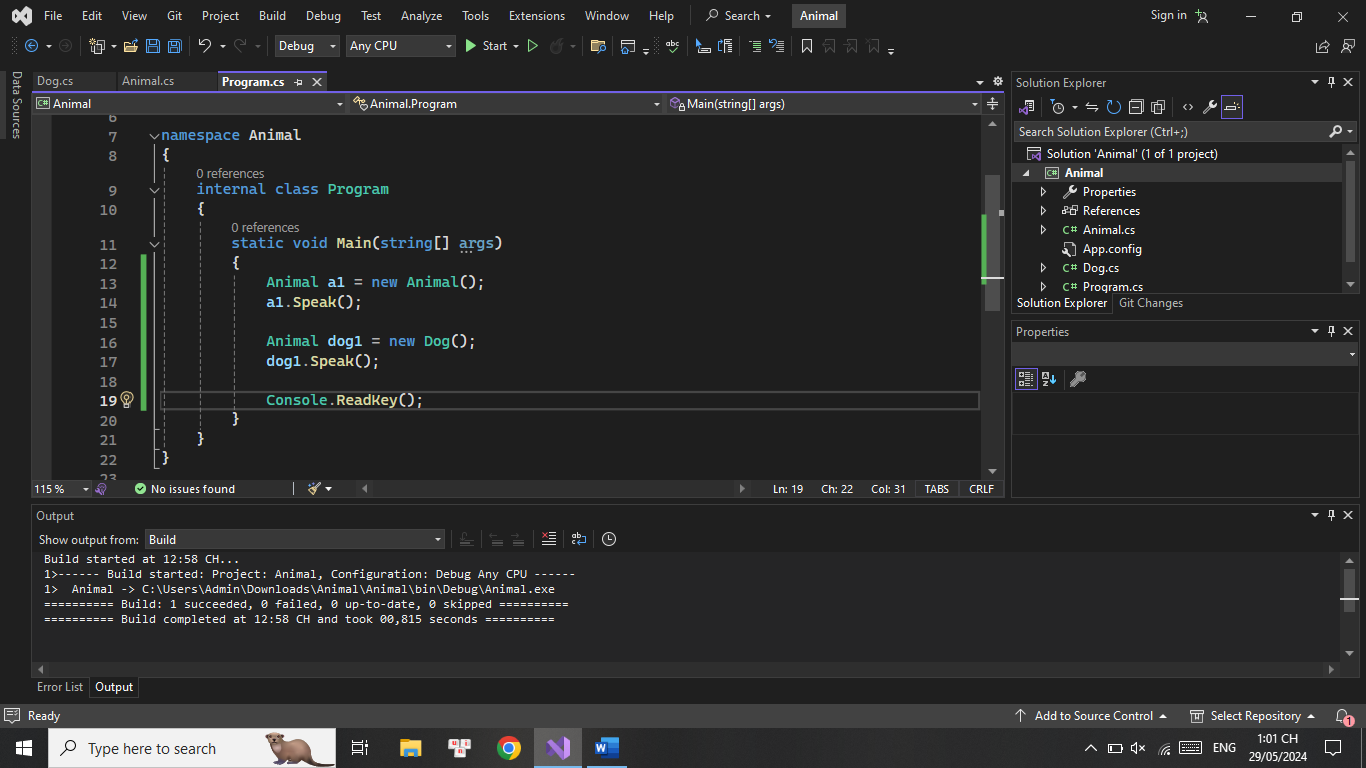
+ Tăng tính mở rộng: Cho phép dễ dàng mở rộng hệ thống bằng cách tạo ra các lớp con mới từ các lớp cha đã có.

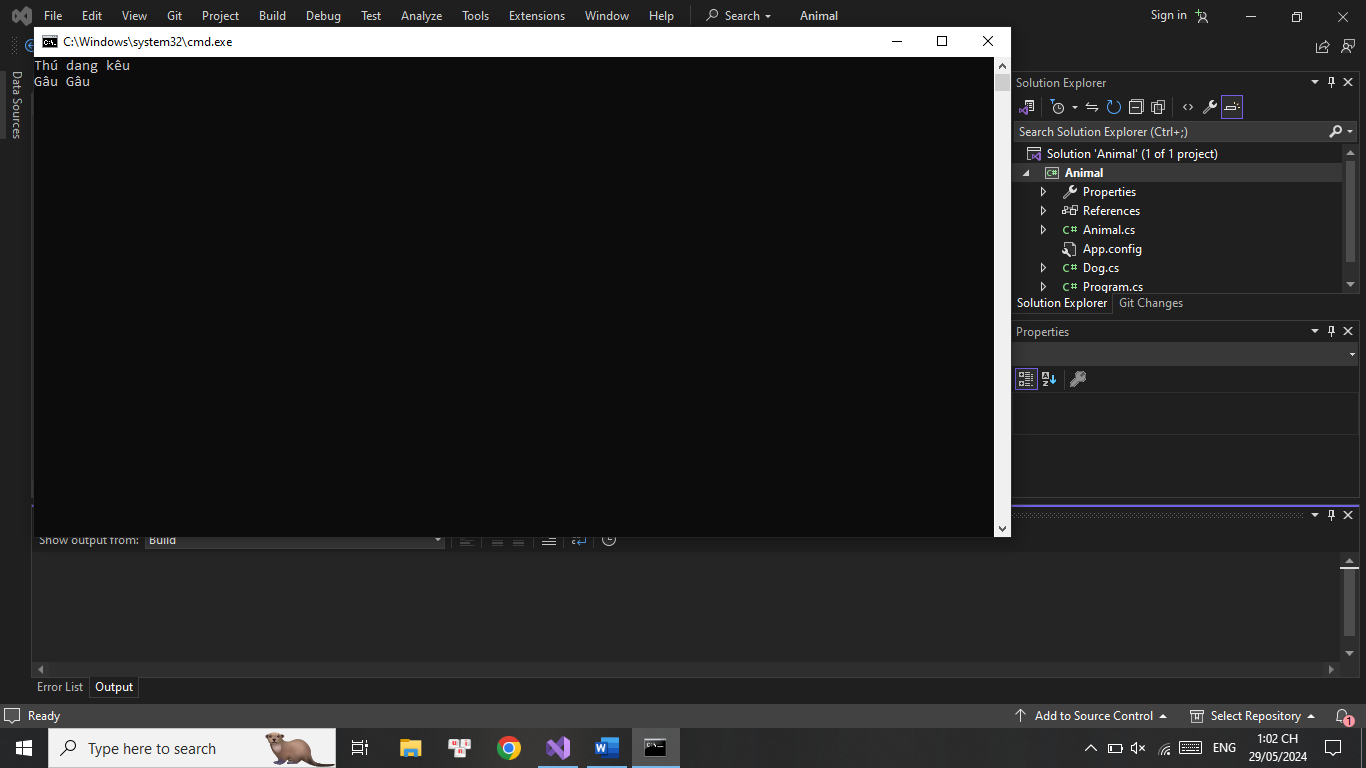
+ Tăng tính bảo trì: Giúp việc bảo trì hệ thống dễ dàng hơn do mã được tổ chức logic và hợp lý.

* Ví dụ minh họa:









1. **Mô tả các tính năng của interface và sự khác nhau giữa interface và abstract class. Cho ví dụ minh họa.**

Tính năng của interface:

* Không chứa các triển khai: phương thức của interface không chứa phần thân, mà chỉ định nghĩa phương thức.
* Hỗ trợ đa kế thừa: một lớp cố thể kế thừa nhiều interface.
* Không chứa dữ liệu: interface không thể chứa trường, dữ liệu thực sự, chỉ có thể khai báo các phương thức, thuộc tính.
* Không thể có hàm khai tạo.

Khác nhau giữa Interface và Abtractclass:

Interface:

* Không có phần thân phương thức: Interface chỉ chứa các phương thức mà các lớp triển khai nó phải cung cấp định nghĩa.
* Không chứa trường: Interface chỉ chứa các phương thức và các thuộc tính (properties) không có thân hàm.
* Hỗ trợ đa kế thừa: Một lớp có thể triển khai nhiều interface.

Abstract :

* Có thể có phần thân phương thức.
* Có chứa trường cụ thể.
* Chỉ hỗ trợ kế thừa đơn.

Ví dụ minh họa:

* Interface:

public interface Ianimal

{

void Speak();

}

public class Lion : Ianimal

{

public void Speak()

{

Console.WriteLine(“Grum…”);

}

}

class Program

{

static void Main(string[] args)

{

IAnimal lion = new Lion();

lion.Speak();

}

}

* Abstract:

public abstract class Animal

{

public abstract void Speak();

}

public class Dog : Animal

{

public override void Speak()

{

Console.WriteLine(“Gau Gau…”);

}

}

class Program

{

static void Main(string[] args)

{

Animal dog = new Dog();

dog.Speak();

}

}

1. **Giải thích từ khóa virtual, override, và abstract và cho ví dụ minh họa.**

Virtual:

* Virtual là một từ khóa được sử dụng để khai báo phương thức ảo (virtual method).
* Phương thức ảo là phương thức có thể được ghi đè (override) bởi phương thức con trong lớp kế thừa.
* Đặc điểm của phương thức ảo:
* Khai báo bằng từ khóa virtual.
* Có thể được ghi đè bởi phương thức con trong lớp kế thừa.
* Khi gọi phương thức ảo, kiểu của đối tượng tại thời điểm thực thi (run-time type) được kiểm tra và phương thức được ghi đè từ lớp con sẽ được gọi.
* Lợi ích của phương thức ảo:
* Tăng tính linh hoạt cho chương trình.
* Cho phép tạo ra các lớp kế thừa có thể thay đổi hành vi của phương thức được kế thừa.
* Hỗ trợ đa hình (polymorphism) trong lập trình hướng đối tượng.

Override:

* Override được sử dụng để ghi đè (override) một phương thức hoặc thuộc tính được đánh dấu virtual trong lớp con.
* Khi một phương thức hoặc thuộc tính được đánh dấu override, nó cung cấp một triển khai mới cho phương thức hoặc thuộc tính đó, thay thế triển khai của phương thức hoặc thuộc tính trong lớp cơ sở.
* Phương thức hoặc thuộc tính ghi đè sẽ có cùng chữ ký (signature) với phương thức hoặc thuộc tính trong lớp cơ sở.

Abstract:

* Abstract được sử dụng để chỉ định một phương thức hoặc một lớp là trừu tượng (abstract).
* Một phương thức trừu tượng không có thân hàm và phải được triển khai trong các lớp con.
* Một lớp trừu tượng không thể được khởi tạo, nhưng nó có thể chứa phương thức trừu tượng, thuộc tính, và phương thức thường (non-abstract).

Ví Dụ:

public abstract class Person

{

public abstract string Name { get; set; }

public abstract int Age { get; set; }

public abstract void Speak();

public virtual void Eat()

{

Console.WriteLine(“Đang ăn…”);

}

}

public class Student : Person

{

public override string Name { get; set; }

public override int Age { get; set; }

public abstract void Speak();

public override void Speak()

{

Console.WriteLine(“Tôi là sinh viên..”};

}

public override void Eat()

{

Console.WriteLine(“Sinh viên đang ăn…”);

}

}

1. **Làm thế nào để một lớp có thể triển khai nhiều interface? Cho ví dụ minh họa.**

interface IInterface1

{

void PhuongThuc1();

}

interface IInterface2

{

void PhuongThuc2();

}

class MyClass : IInterface1, IInterface2

{

public void PhuongThuc1()

{

Console.WriteLine("Hello");

}

public void PhuongThuc2()

{

Console.WriteLine("Hello World");

}

}

class Program

{

static void Main(string[] args)

{

MyClass kq = new MyClass();

kq.Method1();

kq.Method2();

}

}

1. **Tại sao một interface không thể chứa các phương thức với phần thân có định nghĩa thực thi?**

Vì mục đích chính của interface là định nghĩa các hành vi mà các lớp con sẽ phải cung cấp.

1. **Giải thích về tính đa hình (polymorphism). Cho ví dụ sử dụng interface để minh họa tính đa hình.**

Tính đa hình (polymorphism) là một khái niệm quan trọng trong lập trình hướng đối tượng, cho phép các đối tượng của các lớp khác nhau có thể được xử lý một cách đồng nhất thông qua một giao diện chung. Tính đa hình giúp tăng tính linh hoạt và tái sử dụng trong mã nguồn.

Ví Dụ:

interface IAnimal

{

void MakeSound();

}

class Dog : IAnimal

{

public void MakeSound()

{

Console.WriteLine("Woof");

}

}

class Cat : IAnimal {

public void MakeSound()

{

Console.WriteLine("Meow");

}

}

class Duck : IAnimal {

public void MakeSound()

{

Console.WriteLine("Quack");

}

}

class Program {

static void Main(string[] args)

{

IAnimal dog = new Dog();

IAnimal cat = new Cat();

IAnimal duck = new Duck();

dog.MakeSound(); // Output: Woof

cat.MakeSound(); // Output: Meow

duck.MakeSound(); // Output: Quack

}

}

1. **Làm thế nào để một lớp có thể triển khai hai interface có phương thức trùng tên? Cho ví dụ minh họa.**

Khi một lớp triển khai hai interface mà cả hai interface đều chứa phương thức có cùng tên, lớp đó phải cung cấp một triển khai cho cả hai phương thức đó. Cú pháp để triển khai các phương thức này sẽ giống nhau nhưng nội dung của phương thức có thể khác nhau tùy theo nhu cầu của lớp.

Ví dụ:

interface IShape

{

void Draw();

}

interface IColor

{

void Draw();

}

class Rectangle : IShape, IColor

{

public void Draw() {

Console.WriteLine("Drawing a rectangle...");

}

}

class Circle : IShape, IColor

{

public void Draw() {

Console.WriteLine("Drawing a circle...");

}

}

class Program

{

static void Main(string[] args)

{

Rectangle rect = new Rectangle();

Circle circle = new Circle();

rect.Draw();

circle.Draw();

}

}