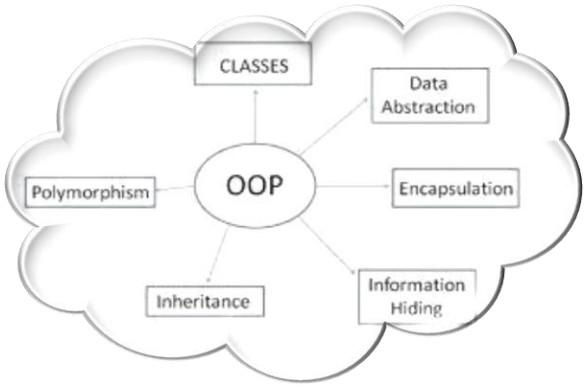
MISA®
TIN CẬY - TIỆN ÍCH - TẬN TÌNH

Lập trình hướng đối tượng với C#

Nguyễn Thanh Tùng – CTO

Nội dung

1. Nguyên lý cơ bản lập trình hướng đối tượng
2. Trừu tượng
3. Kế thừa
4. Đóng gói
5. Đa hình



Nguyên lý cơ bản OOP

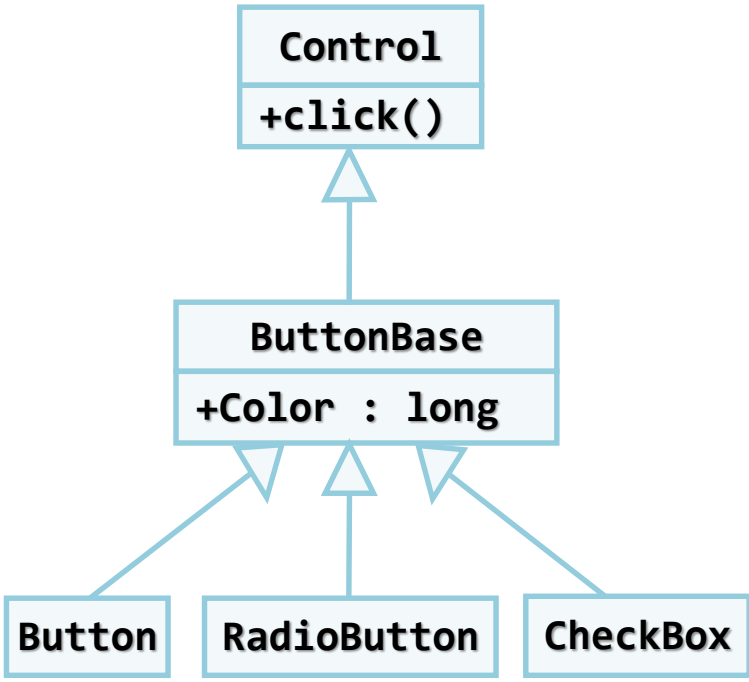
OOP

- Tính trừu tượng (Abstraction)
  - Định nghĩa và thực thi các hành động trừu tượng
- Tính đóng gói (Encapsulation)
  - Ẩn đi các chi tiết bên trong của một lớp
- Tính kế thừa (Inheritance)
  - Kế thừa phương thức, thuộc tính từ lớp cha
- Tính đa hình (Polymorphism)
  - Truy cập tới một lớp thông qua lớp cha

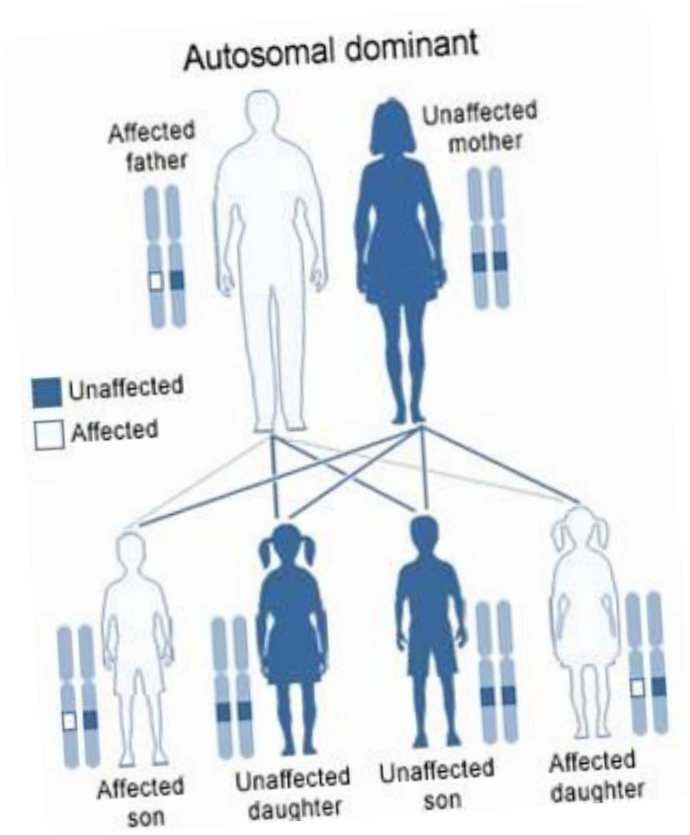
Tính trừu tượng (Abstraction)

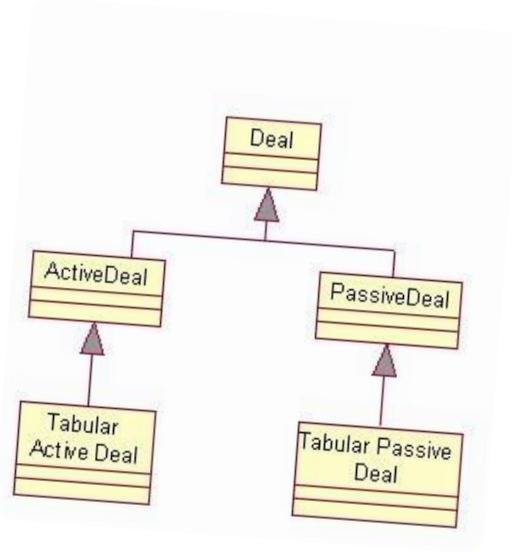
Tính trừu tượng

- Trừu tượng nghĩa là bỏ qua những thuộc tính, chức năng không liên quan mà chỉ tập trung vào những thứ mình đang cần của một đối tượng
- Việc này giúp giảm độ phức tạp của đối tượng đó trong bài toán đang giải quyết.
- Áp dụng trong việc mô hình hóa các đối tượng để giải quyết bài toán cụ thể



Tính kế thừa (Inheritance)





Tính kế thừa

- Kế thừa là khả năng một lớp (lớp con) dùng lại được các phương thức, thuộc tính… của một lớp khác (lớp cha). VD: lớp Con chó kế thừa từ lớp Động vật
- Kế thừa tăng khả năng reuse code, loại bỏ duplicate code, giúp thiết kế dễ mở rộng hơn.
- Kế thừa là mối quan hệ is-a. Hiểu đơn giản con chó **cũng là** một động vật
- .NET không cho phép một lớp kế thừa từ nhiều lớp, chỉ có giải pháp là implement nhiều interface để thực hiện đa kế thừa.

Cách kế thừa trong C#

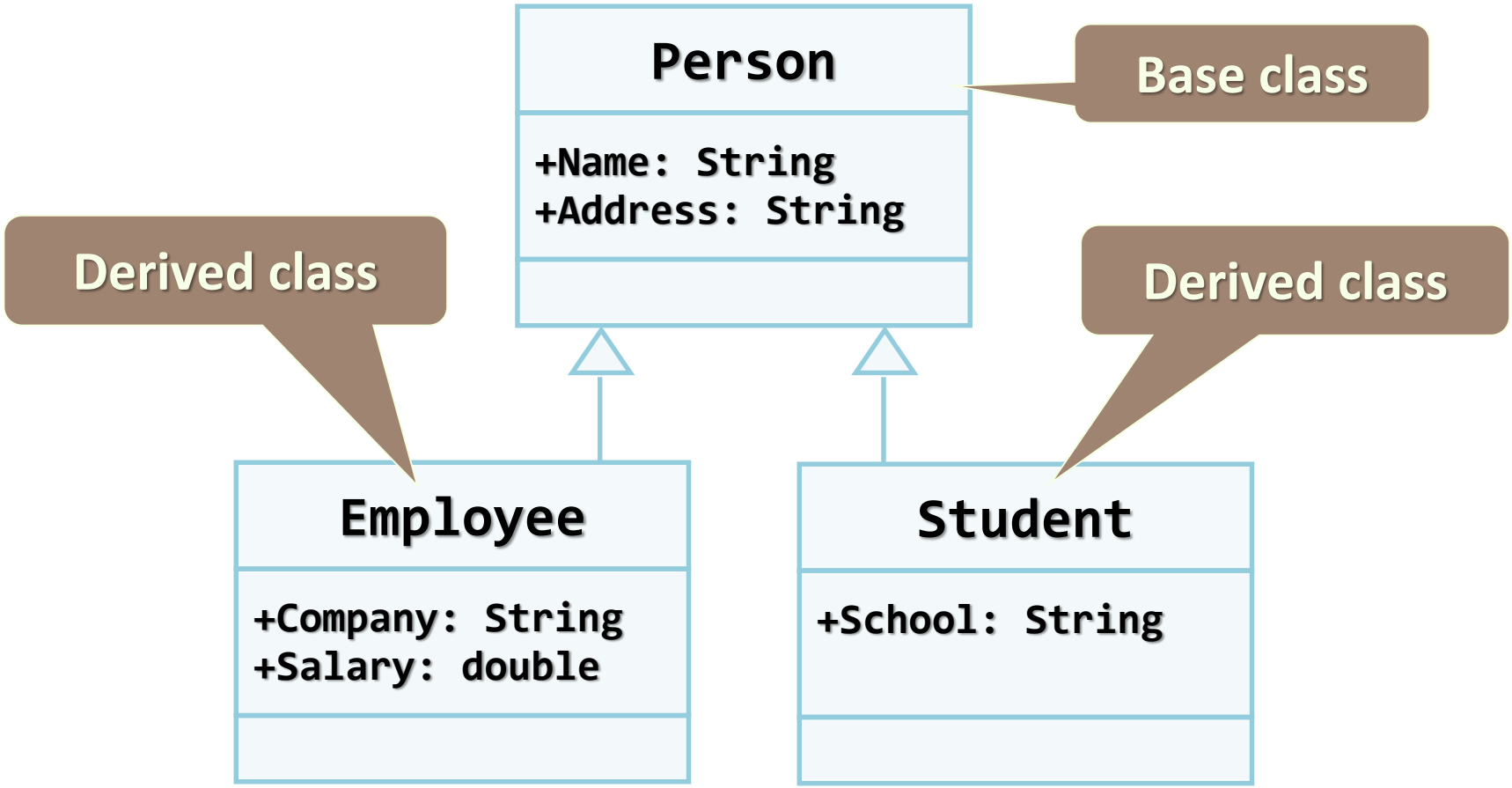
- Đặt base class sau dấu : như sau

```
public class Shape
{...}
public class Circle : Shape
{...}
```

- Constructor của lớp con sử dụng keyword base để gọi hàm khởi tạo ở lớp cha

```
public Circle (int x, int y) : base(x)
{...}
```

Cách kế thừa trong C#



VD Inheritance

```
public class Animal
{
    private int age;

    public Animal(int age)
    {
        this.age = age;
    }

    public int Age
    {
        get { return age; }
        set { age = value; }
    }

    public void Sleep()
    {
        Console.WriteLine("I'm sleeping!");
    }
}
```

VD Inheritance (2)

```
public class Dog : Animal
{
   private string breed;
   public Dog(int age, string breed): base(age)
   {
      this.breed = breed;
   }
   public string Breed
   {
      get { return breed; }
      set { breed = value; }
   }
   public void WagTail()
   {
      Console.WriteLine("Tail wagging...");
   }
}
```

VD Inheritance (3)

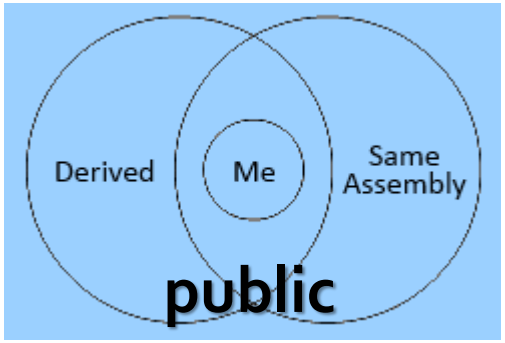
```
static void Main()
{
    // Create 5 years old animal
    Animal animal = new Animal(5);
    Console.WriteLine(animal.Age);
    animal.Sleep();

    // Create a bulldog, 3 years old
    Dog dog = new Dog(3,"Bulldog");
    dog.Sleep();
    dog.Age = 4;
    Console.WriteLine("Age: {0}", dog.Age);
    Console.WriteLine("Breed: {0}", dog.Breed);
    dog.WagTail();
}
```

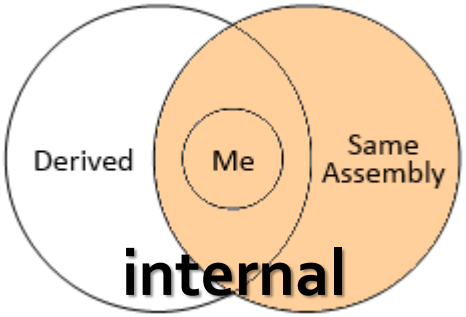
Tính đóng gói (Encapsulation)

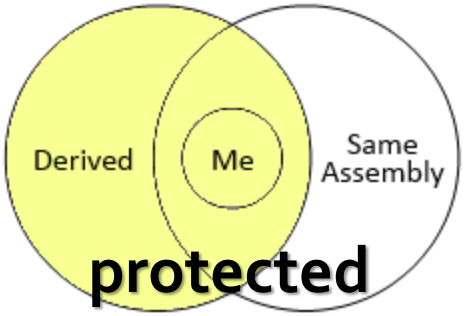
Tính đóng gói

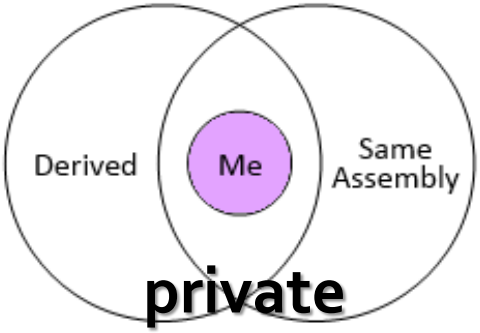
1. Ẩn đi chi tiết bên trong của Class
2. Chỉ cung cấp ra các public interface là các method và property cho các lớp khác sử dụng
3. Tất cả các data member đều nên ẩn đi và chỉ cho phép truy xuất qua property.

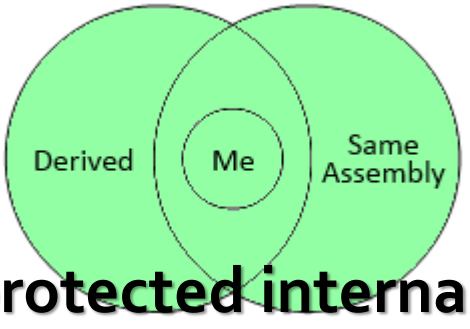
Cách thực hiện tính đóng gói











Demo Tính đóng gói

```
class Creature {
   protected string Name { get; private set; }

   protected void Walk()
   {
      Console.WriteLine("Walking ...");
   }

   private void Talk()
   {
      Console.WriteLine("I am creature ...");
   }

}

class Animal: Creature
{
   // base.Walk() can be invoked here
   // base.Talk() cannot be invoked here
   // this.Name can be read but cannot be modified here
}
```

Polymorphism

Tính đa hình (Polymorphism)

- Đa hình hiểu là:
  - Một lớp con có thể được dùng qua interface của lớp cha
  - Một lớp con có thể ghi đè (override) một vài hành vi của lớp cha
- Đa hình cho phép tạo ra các cấu trúc thiết kế linh hoạt và mềm dẻo hơn

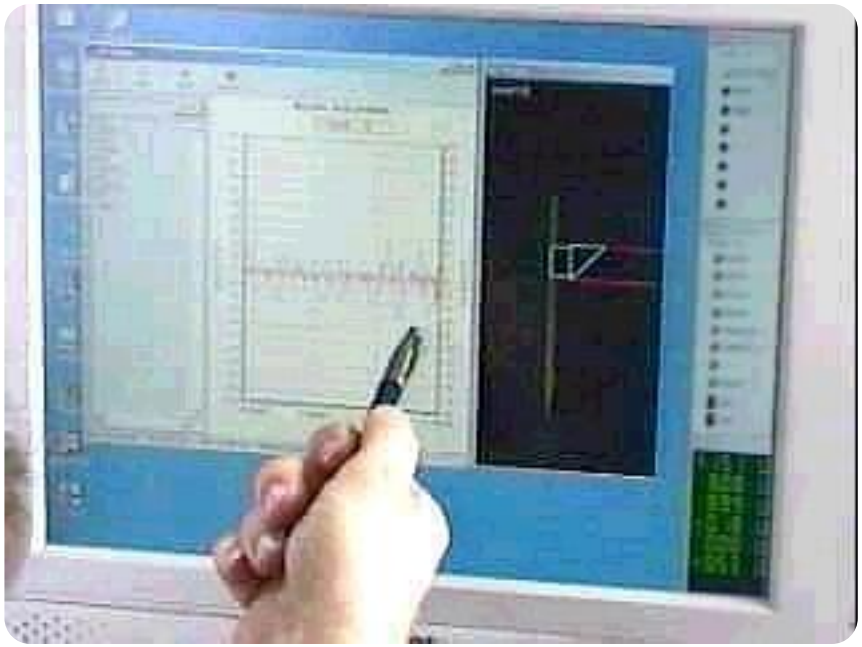
VD đa hình sử dụng Virtual

```
class Person
{
    public virtual void PrintName()
    {
        Console.WriteLine("I am a person.");
    }
}

class Trainer : Person
{
    public override void PrintName()
    {
        Console.WriteLine("I am a trainer.");
    }
}

class Student : Person
{
    public override void PrintName()
    {
        Console.WriteLine("I am a student.");
    }
}
```

VD đa hình sử dụng Virtual (2)

```
static void Main()
{
    Person[] persons =
    {
        new Person(),
        new Trainer(),
        new Student()
    };
    foreach (Person p in persons)
    {
        Console.WriteLine(p);
    }

    // I am a person.
    // I am a trainer.
    // I am a student.
}
```

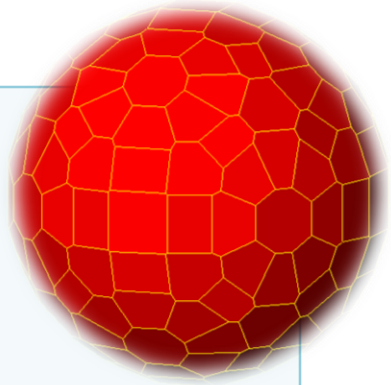
Interfaces and Abstract Classes



Tính đa hình (Polymorphism)

- Mô tả định nghĩa một nhóm phương thức (operations), thuộc tính hoặc event mà các lớp khác bắt buộc phải thực thi.
- Chỉ định nghĩa prototype chứ không có định nghĩa thực thi chi tiết (contract)
- Mức truy suất mặc định của contract là `public`

VD Interfaces



```
interface IShape
{
    void SetPosition(int x, int y);
    int CalculateSurface();
}

interface IMovable
{
    void Move(int deltaX, int deltaY);
}

interface IResizable
{
    void Resize(int weight);
    void Resize(int weightX, int weightY);
    void ResizeByX(int weightX);
    void ResizeByY(int weightY);
}
```

Thực thi nhiều Interface (đa kế thừa)

- Các lớp có thể thực thi 1 hoặc nhiều interface (thể hiện đa kế thừa)

```
class Rectangle : IShape, IMovable
{
   private int x, y, width, height;
   public void SetPosition(int x, int y) // IShape
   {
      this.x = x;
      this.y = y;
   }
   public int CalculateSurface() // IShape
   {
      return this.width * this.height;
   }
   public void Move(int deltaX, int deltaY) // IMovable
   {
      this.x += deltaX;
      this.y += deltaY;
   }
}
```

VD đa hình với Interface

```
public interface IPerson
{
    string Name  // property Name
    { get; set; }
    DateTime DateOfBirth  // property DateOfBirth
    { get; set; }
    int Age  // property Age (read-only)
    { get; }
}

public class Person : IPerson {…}
public class Trainer : IPerson {…}
public class Student : IPerson {…}
```

VD đa hình với Interface (2)

```
static void Main()
{
    Interface.IPerson[] iPersons =
    {
        new Interface.Person(),
        new Interface.Trainer(),
        new Interface.Student()
    };

    foreach (Interface.IPerson p in iPersons)
    {
        Console.WriteLine(p);
    }
}
```

Abstract Classes

- Abstract classes là một dạng class đặc biệt
  - Lai giữa class và interface
  - Có thể implemented một phần hoặc không implement như interface
  - Các phương thức không implement được khai báo với từ khóa abstract
  - Không thể khởi tạo trực tiếp mà phải thông qua các lớp con

VD Abstract Class

```
abstract class MovableShape : IShape, IMovable
{
   private int x, y;
   public void Move(int deltaX, int deltaY)
   {
      this.x += deltaX;
      this.y += deltaY;
   }
   public void SetPosition(int x, int y)
   {
      this.x = x;
      this.y = y;
   }
   public abstract int CalculateSurface();
}
```

VD Abstract Class

```
public abstract class Animal : IComparable<Animal>
{
  // Abstract methods
  public abstract string GetName();
  public abstract int Speed { get; }

  // Non-abstract method
  public override string ToString()
  {
    return "I am " + this.GetName();
  }

  // Interface method
  public int CompareTo(Animal other)
  {
    return this.Speed.CompareTo(other.Speed);
  }
}
```

VD Abstract Class

```
public class Turtle : Animal
{
  public override int Speed { get { return 1; } }

  public override string GetName()
  { return "turtle"; }
}

public class Cheetah : Animal
{
  public override int Speed { get { return 100; } }

  public override string GetName()
  { return "cheetah"; }
}
```

So sánh Interfaces vs. Abstract Classes

- C# interfaces khác abstract classes ở các điểm:
  - Không chứa phương thức với chi tiết implement
  - Members không có scope modifier, tất cả là publish
  - Không thể định nghĩa fields, constants, inner types và constructors

Tóm lược

- OOP bao gồm 4 tính chất cơ bản:
  - Tính trừu tượng áp dụng cho việc mô hình hóa các đối tượng, loại bỏ những đặc tính không cần thiết
  - Kế thừa tăng khả năng reuse code, loại bỏ duplicate code, giúp thiết kế dễ mở rộng hơn. .NET không hỗ trợ đa kế thừa.
  - Tính đóng gói cần nhớ về che dấu dữ liệu sử dụng các access modifier. Khi thiết kế class thì phải luôn ghi nhớ điều này
  - Tính đa hình trợ giúp trong việc thiết kế linh hoạt và mềm dẻo. Có thể sử dụng 3 cách để implement dung virtual method, abstract class, interface


THANK YOU