

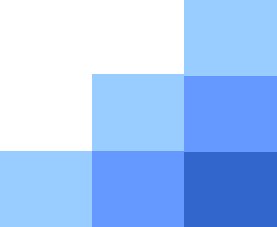



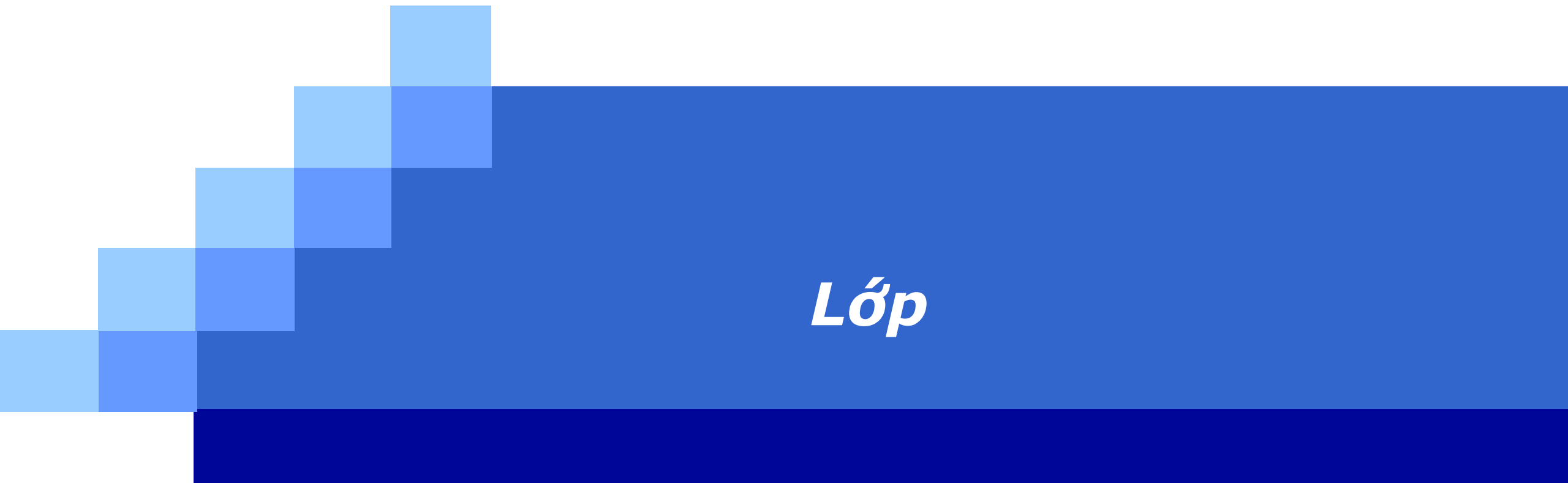
Review

# ***LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG***

Kiến thức cơ bản

**Giảng viên: Huỳnh Tuấn Anh**  
**Khoa CNTT- Đại học Nha Trang**

- 
- 
- ❖ Lớp
  - ❖ Lớp trừu tượng
  - ❖ Giao diện, thực thi giao diện
  - ❖ Thừa kế
  - ❖ Tính đa hình



*Lớp*

# Lớp (class)

- ❖ Class: Sự đóng gói dữ liệu và các phương thức (method) xử lý trên dữ liệu đó
- ❖ Thành phần của lớp
  - Fields
  - Methods
  - Events
  - Properties
  - Các lớp lồng bên trong

# Khai báo lớp trong java

[access modifier] [static] [abstract]

| **class\_name** [extends] [implements]



{

//class-members

...

}

# access-modifier trong java

## Access Modifiers

Modifier	Class	Package	Subclass	Global
Public	✓	✓	✓	✓
Protected	✓	✓	✓	✗
Default	✓	✓	✗	✗
Private	✓	✗	✗	✗

# Khai báo lớp trong C#

```
attributes? unsafe? access-modifier?[ abstract | sealed ]?  
class class-name  
[: base-class | : interface+ | : base-class, interface+ ]?  
{ class-members }
```

# Khai báo lớp trong C#

attributes? unsafe? access-modifier? [ abstract | sealed ]?

class class-name



Bổ từ	Giới hạn truy cập
<b>public</b>	Không hạn chế.
<b>private</b>	Chỉ có thể thấy được ở lớp hiện tại
<b>protected</b>	Chỉ có thể thấy được ở lớp hiện tại và lớp dẫn xuất
<b>internal</b>	Chỉ có thể thấy được trong cùng gói assembly (hợp ngữ) hiện tại
<b>protected internal</b>	Có thể thấy được ở lớp dẫn xuất (trong hay ngoài khối hợp ngữ) và bất cứ lớp nào trong cùng khối hợp ngữ hiện tại



# Khai báo lớp trong C#

```
attributes? unsafe? access-modifier?[ abstract | sealed ]?  
class class-name  
[: base-class | : interface+ | : base-class, interface+ ]?  
{ class-members }
```

# static member

- ❖ Là thành phần của một lớp
- ❖ Các biến, phương thức tĩnh được truy cập thông qua tên lớp
  - static member hoạt động giống như biến, phương thức toàn cục trong lập trình cấu trúc.
  - Phương thức tĩnh không thể truy cập đến một thành viên non-static trong cùng lớp
- ❖ Khai báo (trong c#/java):
  - [bổ từ truy cập] <static> <tên kiểu> <tên thành viên>
  - Ví dụ:
    - public static int count;
    - public static void test(){...}

# properties trong c#

❖ Đặc tính mới trong một số ngôn ngữ LT HĐT dùng để truy cập đến các trường dữ liệu bên trong lớp

❖ Ví dụ: C#

```
public class test
```

```
{
```

```
    private int x;
```

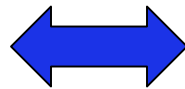
```
    public int X{
```

```
        get {return x;}
```

```
        set {x=value;}
```

```
    }
```

```
}
```



C# 4.0

```
public class test
```

```
{
```

```
    public int x {get;set;}
```

```
}
```

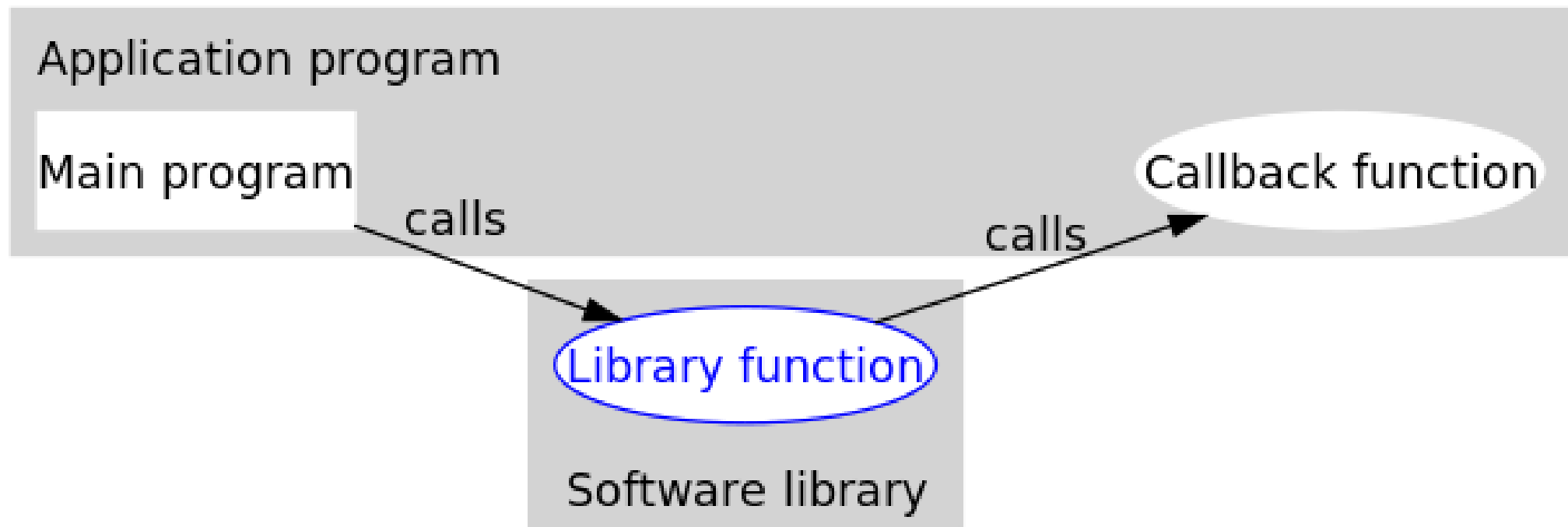
# getter, setter trong java

- ❖ Trong java không hỗ trợ khái niệm properties như C#. Thay vào đó java đưa ra khái niệm getter, setter
- ❖ Để truy cập an toàn vào dữ liệu của lớp, nên khai báo các biến dữ liệu của lớp là private và thực hiện gán giá trị cho các biến thông qua các setter, lấy giá trị của biến thông qua các getter

```
public class SV {  
    String name;  
    int tuoi;  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public int getTuoi() {  
        return tuoi;  
    }  
    public void setTuoi(int tuoi) {  
        this.tuoi = tuoi;  
    }  
}
```

# Callback, Delegate, Event

- ❖ Callback: Là một đoạn mã, có thể là một phương thức, được xem như là một tham số đối với một đoạn mã khác muốn thực hiện (execute) tham số này tại một thời điểm thích hợp nào đó.



# Callback, Delegate, Event trong C#

❖ Delegate: Từ khóa định nghĩa kiểu trong C# dùng để hỗ trợ cơ chế CALLBACK

- Khai báo:

access-modifier delegate return-type **delegate\_name**(argument-list);

❖ Event: Khai báo sự kiện để gọi phương thức ở thành phần Client

- Cú pháp:

access-modifier **event** **delegate\_name** **event\_name**;

❖ Sử dụng Event:

- Library function: **event\_name**(passed\_arguments);

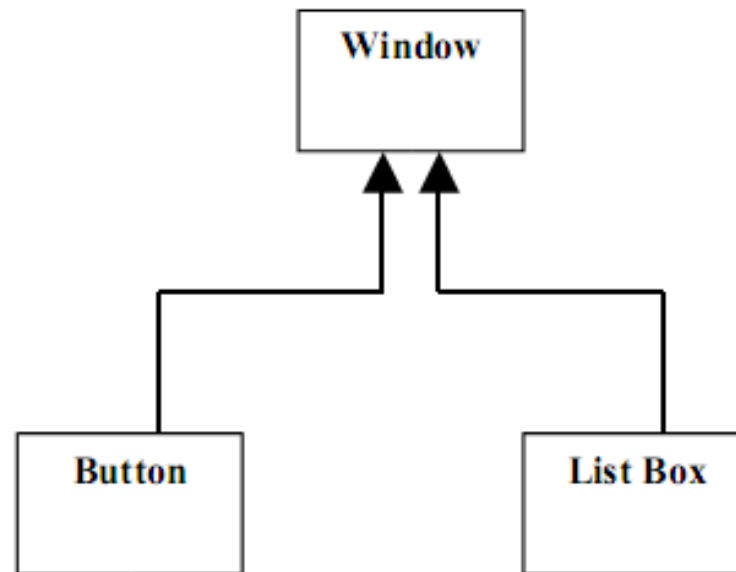
- Client: **event\_name** += **callback\_function\_name**;



# *Kế thừa, đa hình (inheritance, polymorphism)*

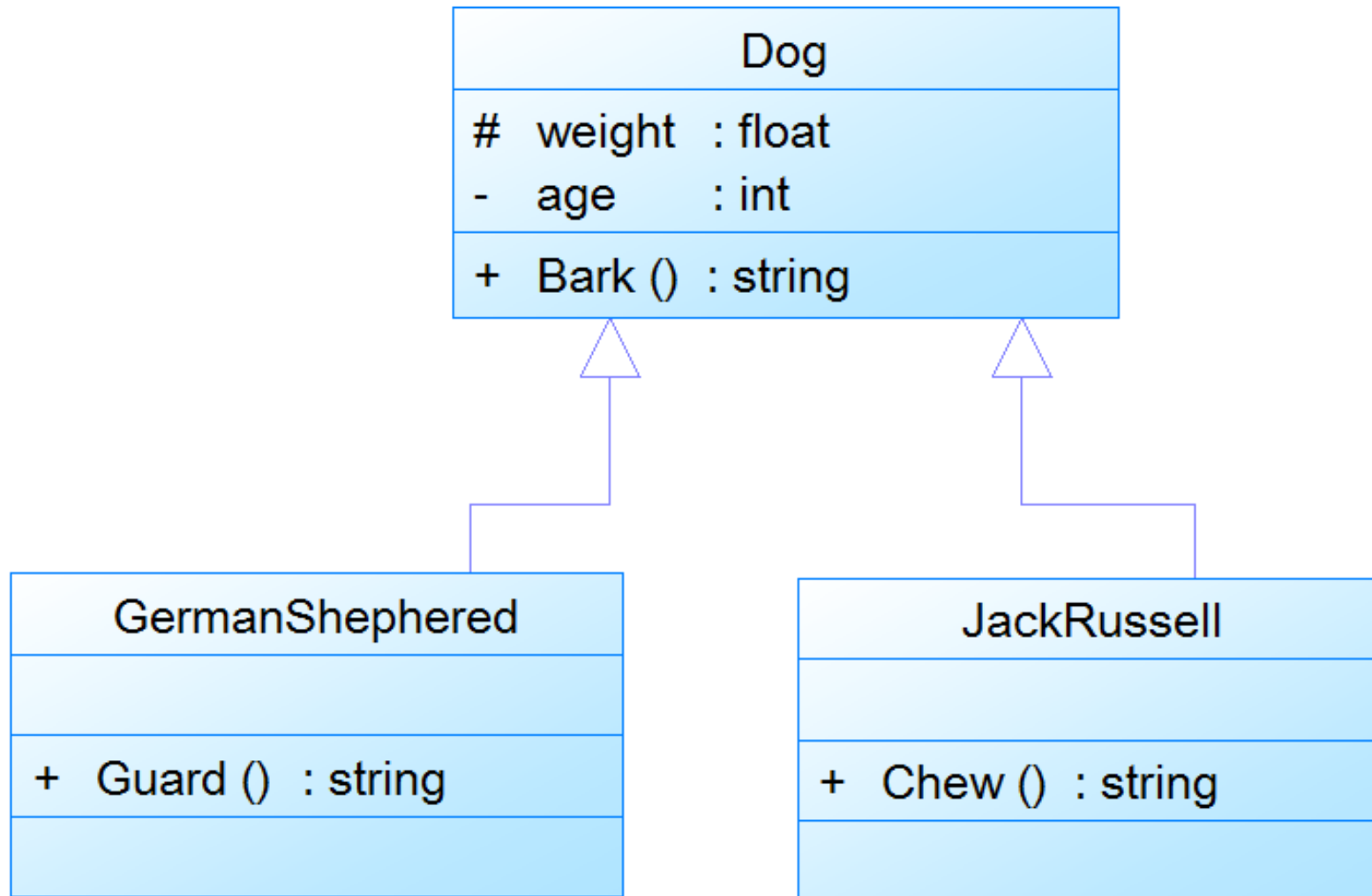
# inheritance

- ❖ cho phép một số phương thức (method), biến thành viên ở lớp cơ sở (base class) được sử dụng ở lớp dẫn xuất (derived class, subclass)





# inheritance



# Đa kế thừa, đơn kế thừa

- ❖ Đa kế thừa (Multi-inheritance): Một lớp được thừa kế từ nhiều lớp cơ sở
  - C++
- ❖ Đơn kế thừa (single inheritance): Một lớp chỉ được phép thừa kế từ một lớp cơ sở
  - C#, Java
- ❖ Cú pháp khai báo:
  - Java: <bổ từ truy cập> tên lớp **extends** <tên lớp cơ sở>
  - C#: <bổ từ truy cập> tên lớp: <tên lớp cơ sở>
  - C++ tên lớp: [kiểu thừa kế] <tên lớp cơ sở,>\*

```
public  
protected  
private
```

## inherited members' access modifier

base class	derived class
protected	private
public	public
internal	internal
protected internal	<b>protected internal</b> (nếu trong cả 2 lớp base và derived cùng khối hợp ngữ) <b>private</b> (nếu derived class khác khối hợp ngữ với base class)

# access-modifier trong java

## Access Modifiers

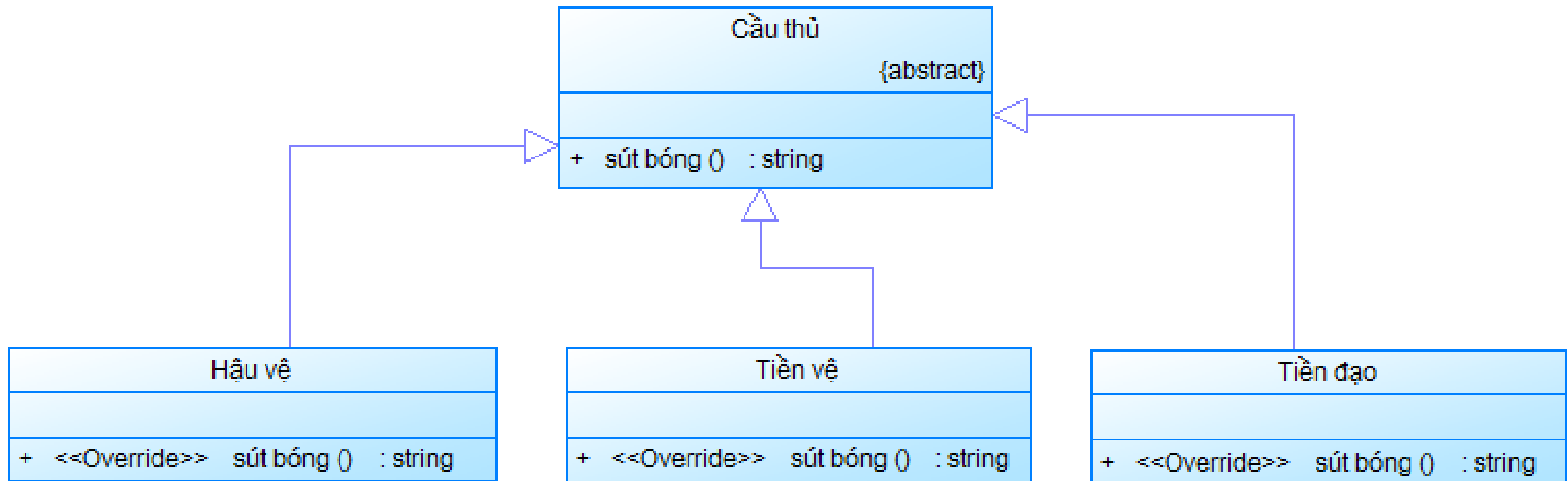
Modifier	Class	Package	Subclass	Global
Public	✓	✓	✓	✓
Protected	✓	✓	✓	✗
Default	✓	✓	✗	✗
Private	✓	✗	✗	✗

# Polymorphysm

- ❖ Polymorphysm=Một chức năng hay đối tượng được thực thi theo nhiều cách khác nhau
- ❖ Polymorphysm có thể được áp dụng cho:
  - Objects
  - Operaions (methods)

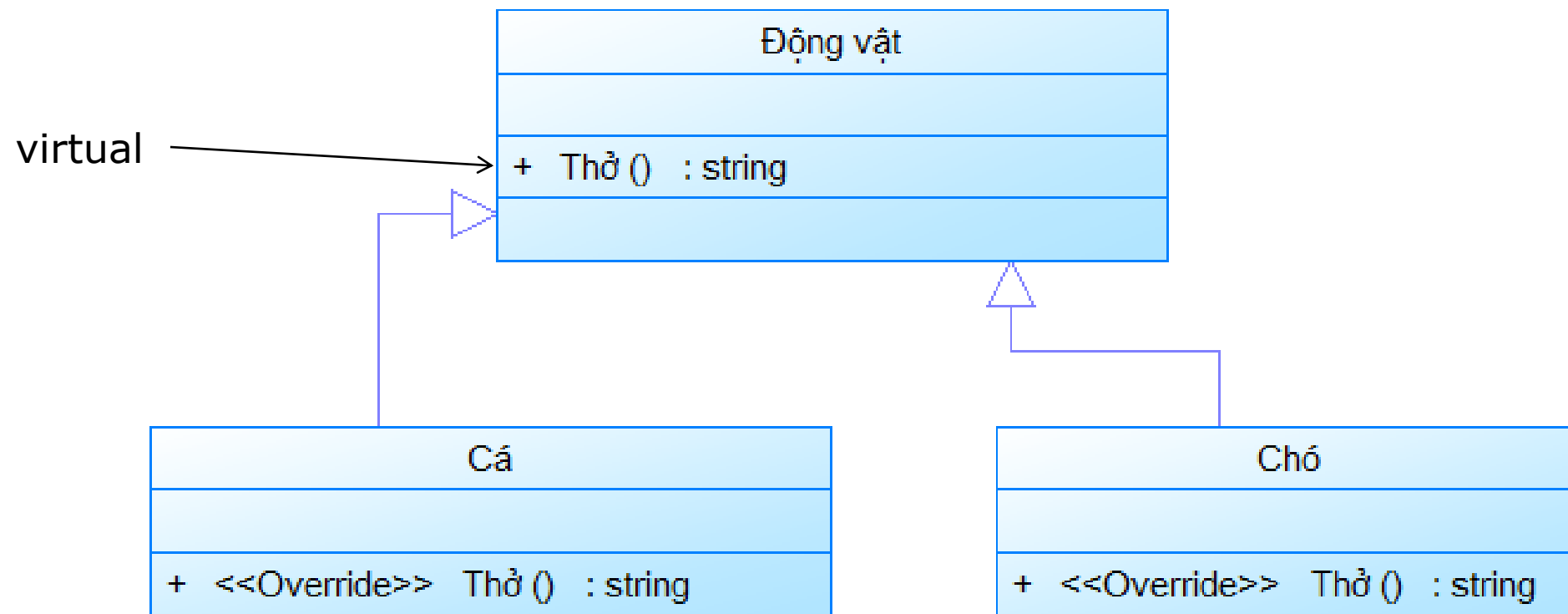
# polymorphic object

- ❖ polymorphic object: đối tượng của một lớp con là dẫn xuất của một super class
- ❖ Ví dụ:



# polymorphic method

- ❖ polymorphic operation: Một phương thức có thể thực hiện, trả về các kết quả khác nhau tùy thuộc vào lớp đối tượng thực hiện nó
- ❖ Ví dụ:



# Lớp trừu tượng (abstract class)

- ❖ Được khai báo với từ khóa abstract
  - ví dụ: `abstract class A {...}`
- ❖ Được thiết lập như là cơ sở cho các lớp dẫn xuất
  - Có một hay nhiều phương thức trừu tượng
  - Các phương thức trừu tượng chỉ có phần khai báo
  - Sự thực thi các phương thức trừu tượng trong lớp trừu tượng được giao cho lớp dẫn xuất
  - Không thể tạo một instance cho lớp trừu tượng
- ❖ Khi một lớp có một phương thức được khai báo là abstract phía trước chỉ dẫn truy cập thì lớp đó là lớp trừu tượng
- ❖ Lớp dẫn xuất từ lớp trừu tượng nhưng không thực thi phương thức trừu tượng cũng được xem là lớp trừu tượng.
  - Khi đó việc thực thi phương thức trừu tượng sẽ được giao cho các lớp con của lớp đó



# polymopic method

## ❖ Từ khóa virtual, override trong c#:

- Phương thức với từ khóa virtual ở lớp base có thể bị ghi đè (overridden) ở lớp dẫn xuất
  - Ví dụ: `public virtual void test(){...}`
- Từ khóa override: dùng để ghi đè phương thức virtual trong lớp base
  - ví dụ: `public override void test(){...}`
- Chú ý:
  - Nếu phương thức virtual bị ghi đè thì nó sẽ bị phương thức override thay thế ở lớp derived
  - Nếu không bị ghi đè thì phương thức virtual sẽ được sử dụng ở lớp derived
  - Phương thức virtual và override phải cùng tên

## Từ khóa new và override

- ❖ override: Phủ quyết một cách tường minh một phương thức virtual ở lớp base
- ❖ new: Chỉ ra rằng phương thức ở lớp derived không phủ quyết bất cứ phương thức nào ở lớp base
- ❖ Trong Java, Override là một chỉ dẫn dùng để chỉ ra rằng phương thức đang ghi đè một phương thức ở lớp cha
  - VD:  
@Override  
public void method1(){...}

## Lớp cô lập (sealed class)

- ❖ Được khai báo với từ khóa sealed
- ❖ Không cho phép các lớp dẫn xuất từ nó

# Lớp lồng nhau

- ❖ Lớp có thể chứa lớp bên trong nó
  - Lớp bên trong: nested/Inner class
  - Lớp ngoài: Outer class
- ❖ Nested class có thể truy cập đến tất cả các thành viên của lớp ngoài
  - Phương thức của lớp nested có thể truy cập đến biến thành viên private của lớp ngoài
  - Nested class có thể ẩn với các lớp khác bên ngoài lớp chứa nó khi được khai báo với từ khóa private

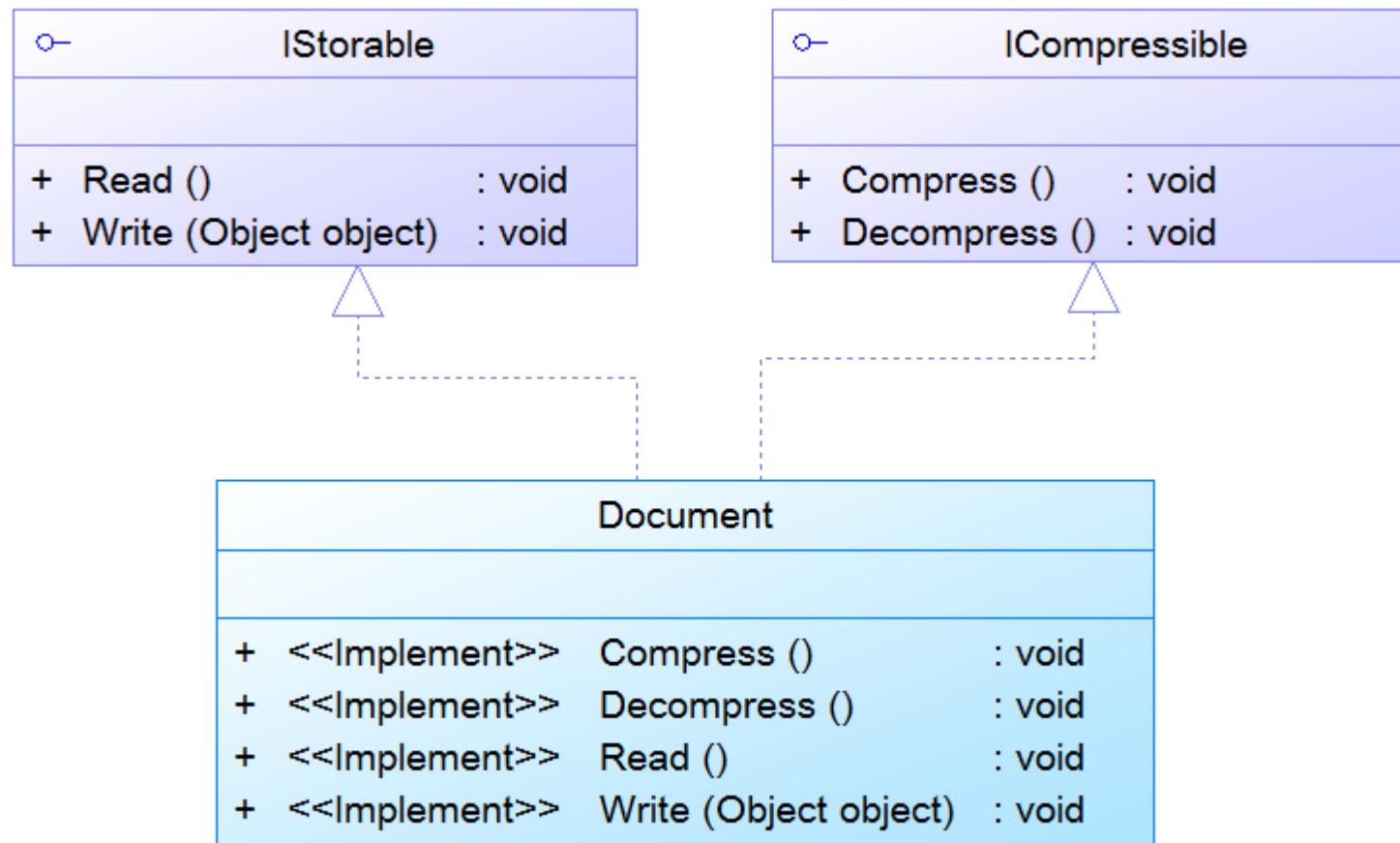
# Giao diện (interface)

- ❖ Chứa các khai báo của các phương thức qui định cho các lớp thực thi giao diện
  - Các phương thức đều được mặc định là public
  - Không khai báo bổ từ truy cập cho các phương thức trong giao diện
  - Khi thực thi giao diện, các phương thức phải được khai báo public
  - Khi một lớp thực thi một giao diện, nó phải thực thi tất cả các phương thức trong giao diện đó
  - Nội dung thực hiện các phương thức do các lớp thực thi giao diện qui định → **polymorphism**
- ❖ Khai báo giao diện:
  - C#
  - java

# Sử dụng interface, abstract class

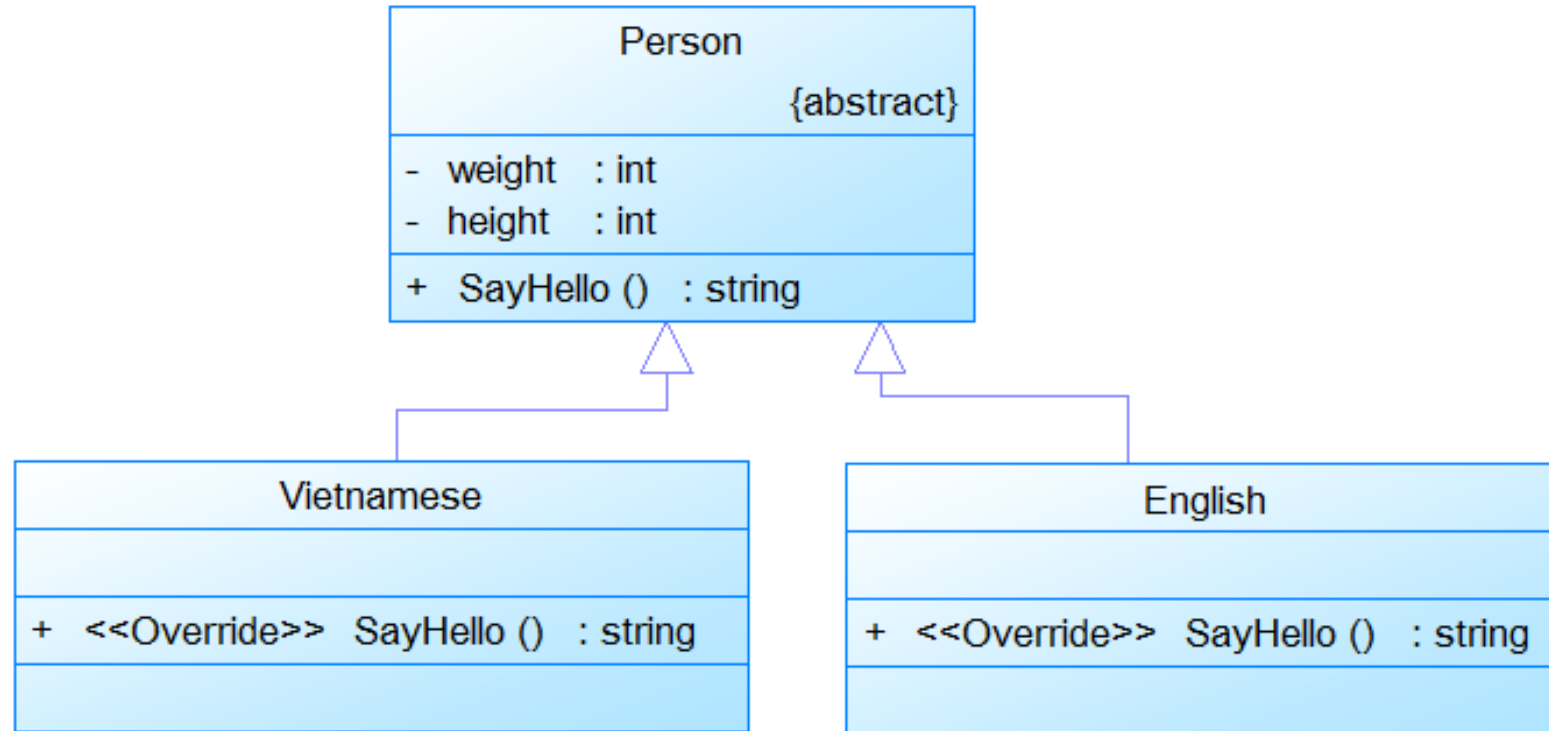
## ❖ Thực thi giao diện:

- Một lớp có thể thực thi một hoặc nhiều giao diện



# Sử dụng interface, abstract class

❖ C#/JAVA: Một lớp chỉ có thể thừa kế một lớp trừu tượng



Sử dụng:

Person person=new English();

*hoặc:*

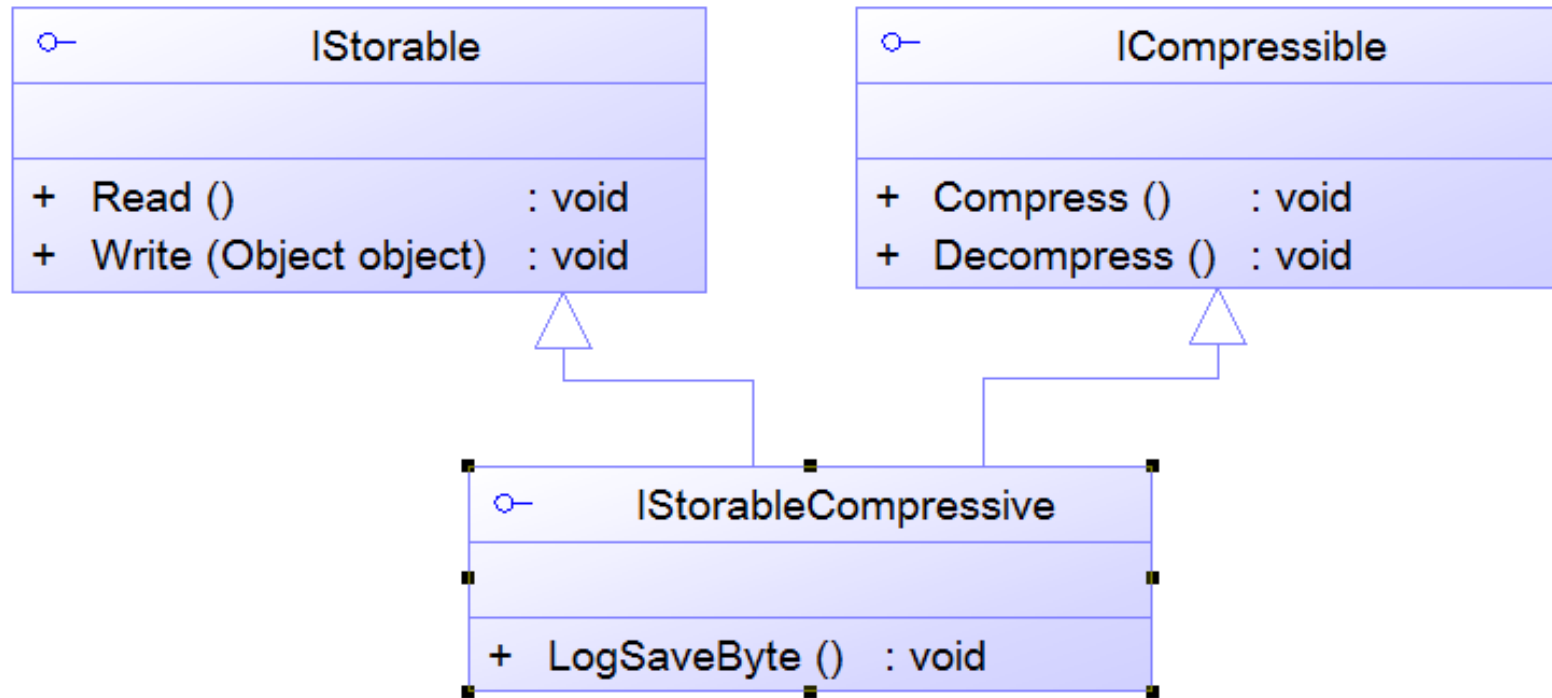
English en=new English();

Person person= en as Person;

Huỳnh Tuấn Anh - ĐHNT

# Mở rộng interface, abstract class

- ❖ Có thể thêm các thành viên mới cho một interface bằng cách thừa kế



- ❖ Có thể mở rộng một abstract class bằng cách tạo lớp abstract dẫn xuất từ lớp đó



## Hạn chế của abstract class

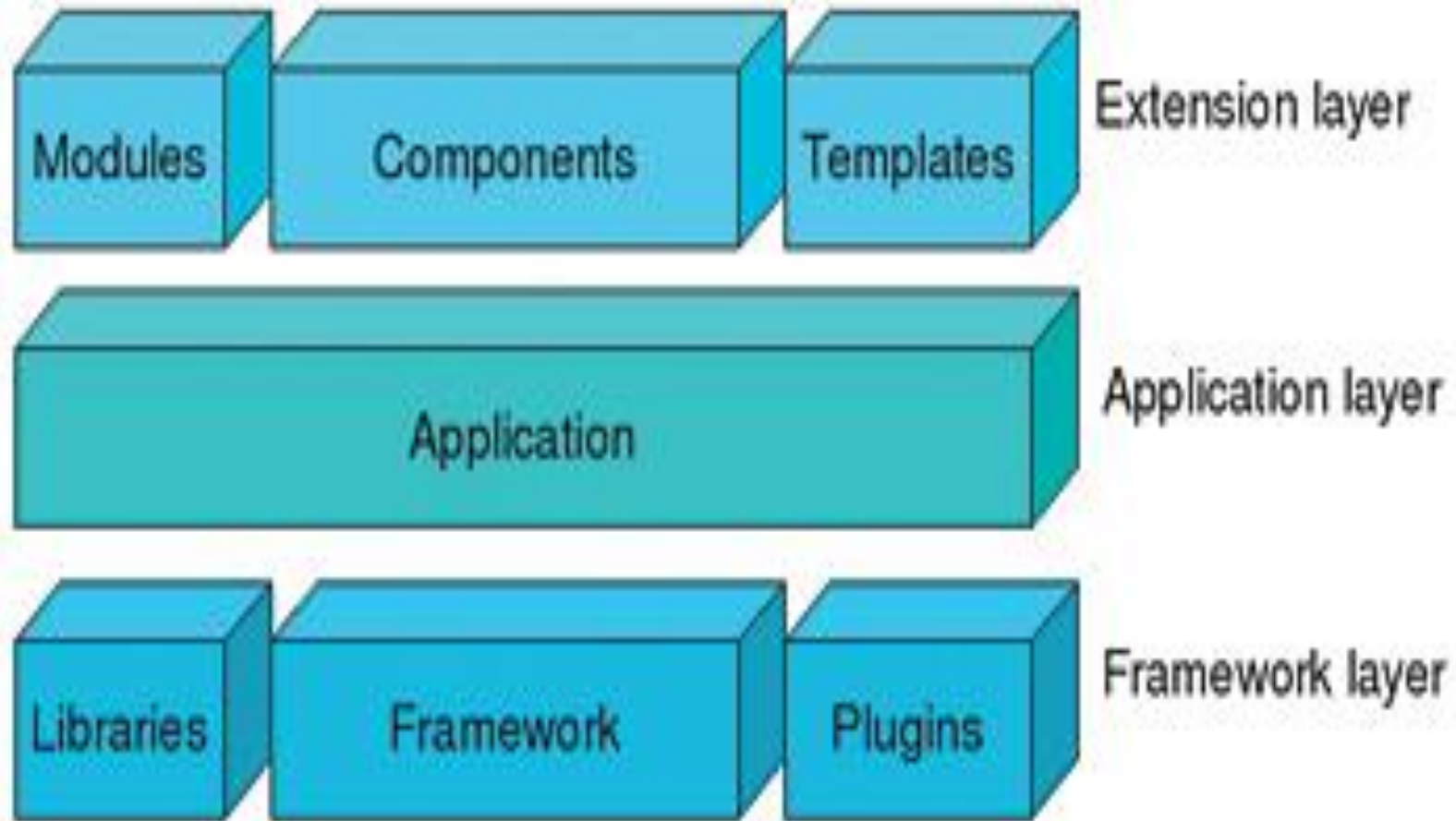
- ❖ Abstract class không bắt buộc các lớp dẫn xuất ghi đè các phương thức abstract của nó.
- ❖ C#/JAVA: Một lớp dẫn xuất chỉ có thể thừa kế từ một base class
- ❖ Khắc phục những hạn chế của các abstract class bằng cách thay thế chúng bằng các interface



## ***Design Principle***

*Program to an interface, not an implementation.*

# Lập trình module



# Lập trình module

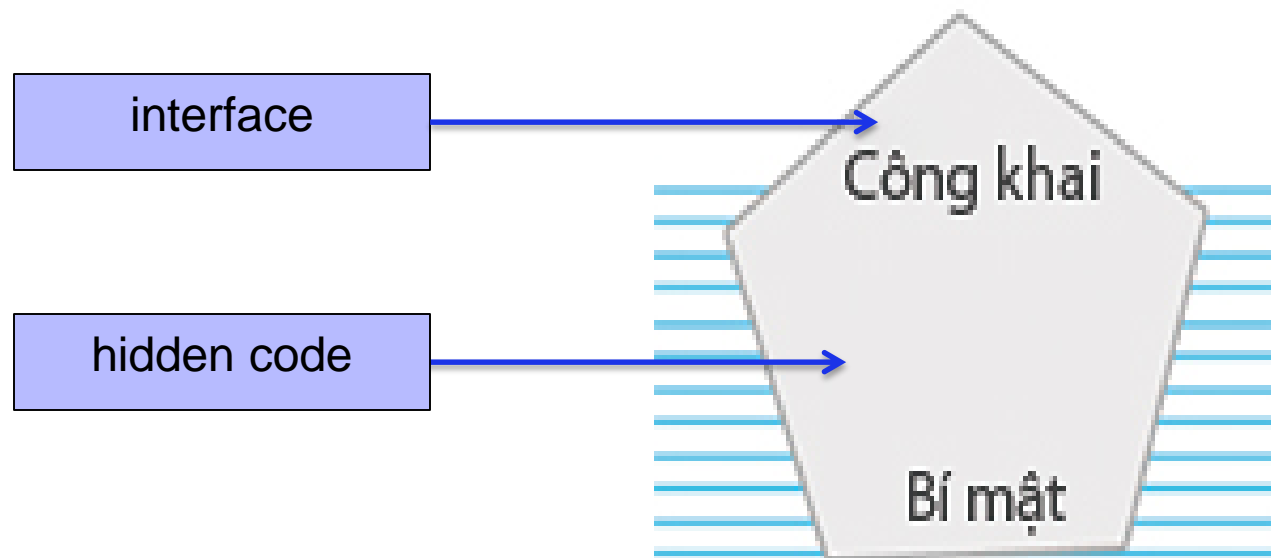
- ❖ Kiến trúc mô-đun (module) cho phép chia nhỏ bài toán (hay yêu cầu) của phần mềm thành các phần hầu như không trùng lặp
  - Hỗ trợ làm việc song song trên các module
  - Dễ bảo trì
  - Khả năng tái sử dụng các thành phần của hệ thống
  - Khả năng mở rộng tốt hơn.
- ❖ Flex, Ruby: cho phép biên dịch các module một cách độc lập và có thể gắn kết vào hệ thống lúc thực thi.
- ❖ C#, C++: hỗ trợ cơ chế như thư viện liên kết động (DLL) để biên dịch các module thành các thư viện độc lập và có thể gắn kết động vào hệ thống.

## interface: Công cụ giao tiếp của các modul

- ❖ Các module được gắn kết với nhau trong chương trình thông qua các "interface".
  - interface của module mô tả những thành phần được cung cấp và cần được cung cấp.
  - Các thành phần này được các module khác "thấy" và sử dụng.
- ❖ Interface của modul khác với interface của C# hay Java
  - Interface là những thành phần được các modul khác nhìn thấy
  - Để dễ dàng bảo trì nên dùng khái niệm Interface của NNLT

...

- ❖ interface của module nên được thiết kế chỉ bao gồm những phần hần như không thay đổi,
  - Những thành phần này được gọi là thành phần "công khai".
  - Những chi tiết ẩn bên dưới các interface thường được gọi là các thành phần "bí mật" hoặc "riêng tư"



## Tài liệu tham khảo

- ❖ Jesse Liberty and Donald Xie. Programming C# 3.0. O'Reilly 2008
- ❖ Eric Freeman, Elisabeth Freeman, Kathy Sierra, Bert Bates. Head First Design pattern. O'Reilly 2006.