

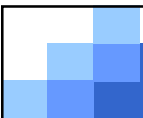


Review

# ***LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG***

Kiến thức cơ bản

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



- ❖ Giới thiệu ngôn ngữ lập trình JAVA
- ❖ Lớp
- ❖ Lớp trừu tượng
- ❖ Giao diện, thực thi giao diện
- ❖ Thừa kế
- ❖ Tính đa hình

Huỳnh Tuấn Anh - ĐHNT

2

## Lập trình hướng đối tượng

- ❖ Lập trình hướng đối tượng (OOP) là một mô hình lập trình dựa trên khái niệm "đối tượng", mỗi đối tượng có thể chứa dữ liệu và mã lệnh.
  - Dữ liệu bao gồm các trường hay còn gọi là thuộc tính.
  - Mã lệnh ở dạng các thủ tục thường được gọi là phương thức. Dữ liệu được xử lý bởi các mã lệnh của đối tượng.

## Giới thiệu về ngôn ngữ lập trình JAVA

- ❖ JAVA là một ngôn ngữ lập trình hướng đối tượng có nhiều điểm tương đồng với ngôn ngữ C#
- ❖ Chương trình JAVA đơn giản

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

- ❖ Chương trình của JAVA được chia thành các gói (package) tương đương với khái niệm namespace trong C#
  - Khi sử dụng thư viện hoặc một lớp thuộc package khác thì phải import thư viện/package tương ứng

## Một số thư viện thường được sử dụng

- ❖ Xuất dữ liệu trên màn hình Console: `System.out.println`
- ❖ List, ArrayList: Lớp quản lý danh sách các đối tượng
- ❖ Dictionary, HashMap, HashTable: Quản lý tập hợp dữ liệu dưới dạng Key-Value
- ❖ Iterator: Lớp dùng để duyệt tập hợp
- ❖ Các khái niệm thường sử dụng trong môn học
  - abstract
  - interface
  - Lập trình trừu tượng
  - Đối tượng trừu tượng
  - Đối tượng đại diện

Huỳnh Tuấn Anh - ĐHNT

5

***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
  - Properties
  - Các lớp lồng bên trong

## class, package trong JAVA

```

1  package CanBan;
2  public class SinhVien {
3      1 usage
4      String maSV, tenSV;
5      4 usages
6      float diemTB;
7      public SinhVien(String maSV, String tenSV, float diemTB) {
8          this.maSV = maSV;
9          this.tenSV = tenSV;
10         this.diemTB = diemTB;
11     }
12     public String xepLoai(){
13         if(diemTB<5) return "Yếu";
14         else
15             if(diemTB<7) return "Trung bình";
16             else
17                 if(diemTB<8) return "Khá";
18                 else return "Giỏi";
19     }
20 }

```

## access-modifier trong java

Modifier	Mô tả	Đối tượng khai báo
public	Thành phần public có thể được truy cập cho tất cả các lớp	lớp, thuộc tính, phương thức
private	Thành phần private chỉ có thể được truy cập trong lớp được khai báo	thuộc tính, phương thức
default	Thành phần <i>default</i> chỉ có thể được truy cập trong cùng một package. Được sử dụng khi không khai báo bỏ từ truy cập cho một thành phần nào đó.	lớp, thuộc tính, phương thức
protected	Thành phần protected có thể được truy cập trong cùng gói và các lớp con	thuộc tính, phương thức

Huỳnh Tuấn Anh - ĐHNT

9

## static member

- ❖ Là thành phần nằm trong 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 (java):
  - [bỏ từ truy cập] <static> <tên kiểu> <tên thành viên>
  - Ví dụ:
    - public static int count;
    - public static void test(){...}

Huỳnh Tuấn Anh - ĐHNT

10

## 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

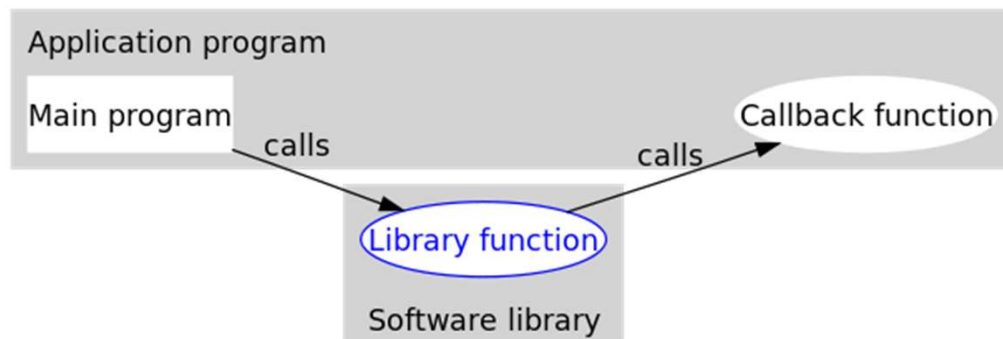
```
public class SV {
    String name;
    int tuoi;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getTuoi() {
        return tuoi;
    }
    public void setTuoi(int tuoi) {
        this.tuoi = tuoi;
   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

Huỳnh Tuấn Anh - ĐHNT

11

## 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.



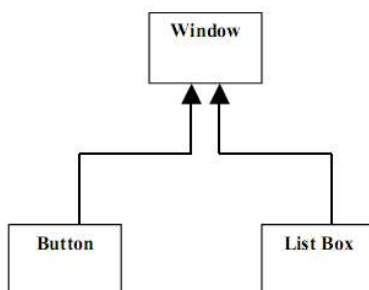
Huỳnh Tuấn Anh - ĐHNT

12

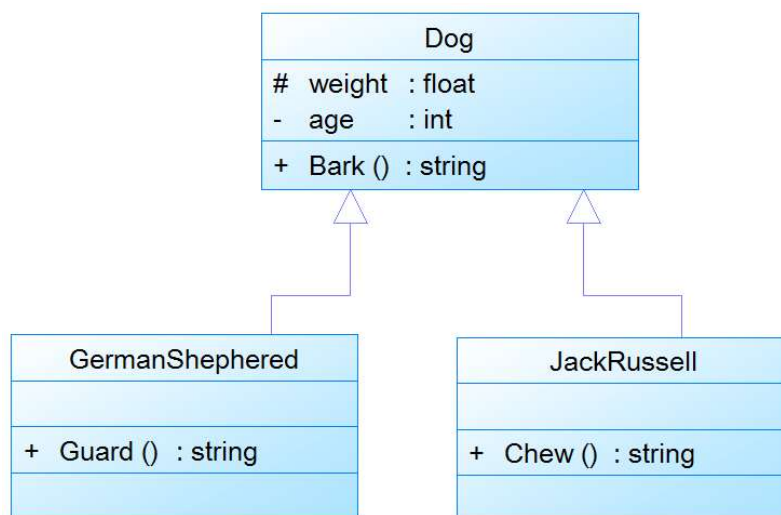
## *Kế thừa, đa hình (inheritance, polymorphism)*

### 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



Huỳnh Tuấn Anh - ĐHNT

15

## 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: <bỏ từ truy cập> tên lớp **extends** <tên lớp cơ sở>

Huỳnh Tuấn Anh - ĐHNT

16



## Polymophysm

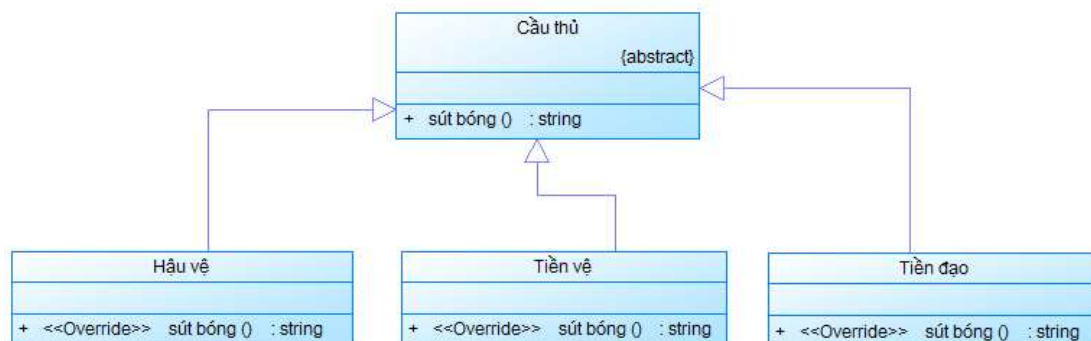
- ❖ Polymophysm=Một chức năng hay đối tượng được thực thi theo nhiều cách khác nhau
- ❖ Polymophysm có thể được áp dụng cho:
  - Objects
  - Operaions (methods)

Huỳnh Tuấn Anh - ĐHNT

17

## polymorphic object

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

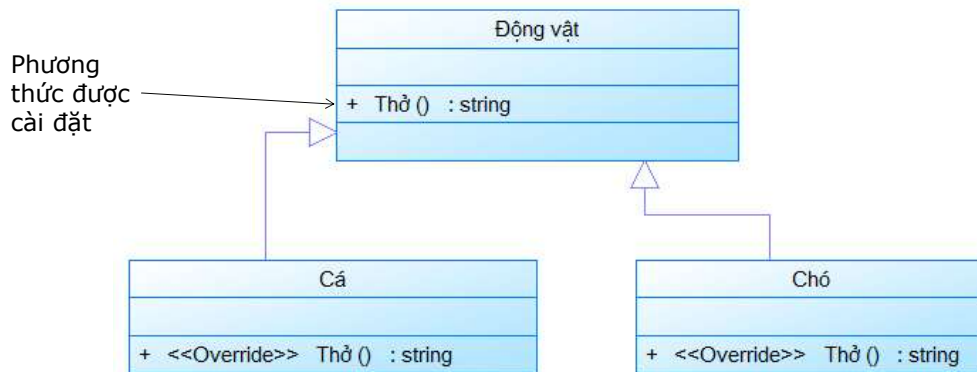


Huỳnh Tuấn Anh - ĐHNT

18

## 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:



Huỳnh Tuấn Anh - ĐHNT

19

## 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

Huỳnh Tuấn Anh - ĐHNT

20

## polymopic method

- ❖ Một phương thức ở lớp cha/base có thể bị ghi đè ở lớp con
- ❖ Từ khóa `@override`: dùng để ghi đè một phương thức trong lớp base
  - ví dụ: `public override void test(){...}`
- ❖ Chú ý:
  - Nếu một phương thức ở lớp base bị ghi đè thì nó sẽ bị phương thức override thay thế ở lớp derived.
  - Phương thức bị ghi đè và phương thức override phải có cùng tên.

## 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)

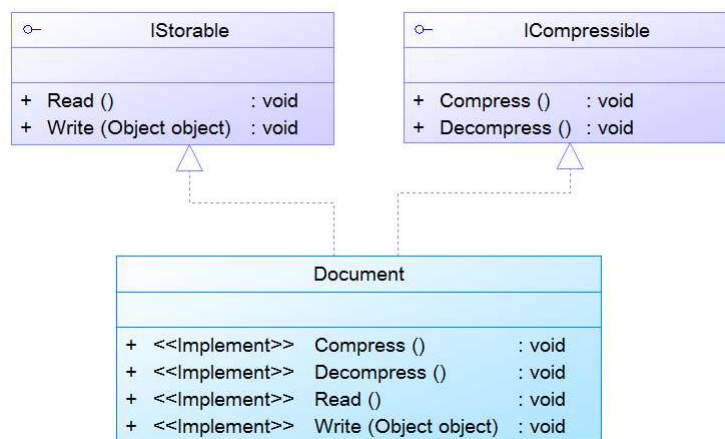
- ❖ Chứa các khai báo của các phương thức qui định cho các lớp thực thi giao diện
  - Các phương thức đều được mặc định là public
  - Không khai báo bỏ từ truy cập cho các phương thức trong giao diện
  - Khi thực thi giao diện, các phương thức phải được khai báo public
  - Khi một lớp thực thi một giao diện, nó phải thực thi tất cả các phương thức trong giao diện đó
  - Việc thực hiện các phương thức do các lớp thực thi giao diện cài đặt → **polymorphism**
- ❖ Trong java, giao diện được khai báo với từ khóa **interface**

Huỳnh Tuấn Anh - ĐHNT

23

## Sử dụng interface, abstract class

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

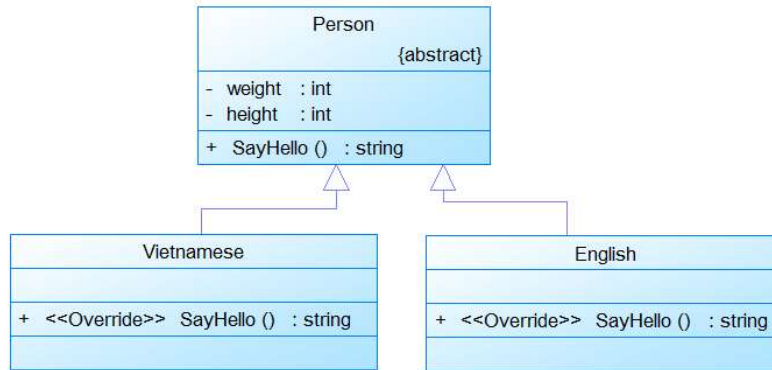


Huỳnh Tuấn Anh - ĐHNT

24

## 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();

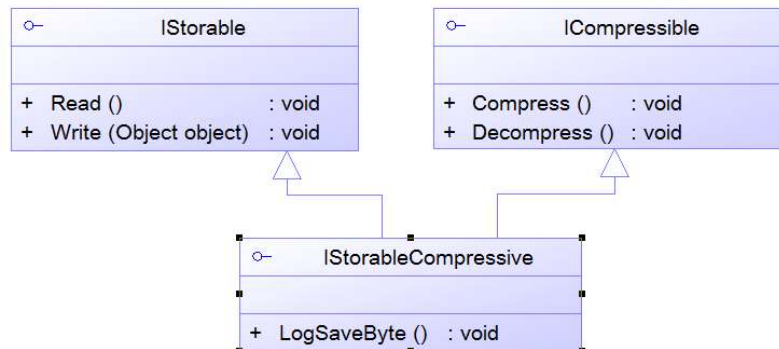
Person person= en as Person;

Huỳnh Tuấn Anh - ĐHNT

25

## 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

Huỳnh Tuấn Anh - ĐHNT

26

## Hạn chế của abstract class

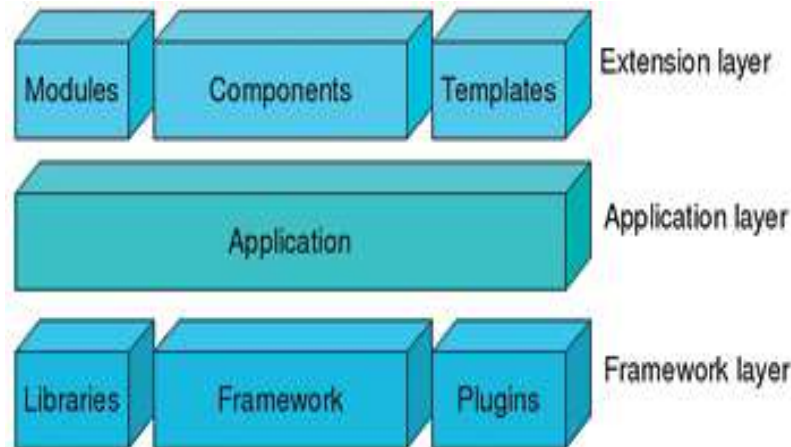
- ❖ Abstract class không bắt buộc các lớp dẫn xuất ghi đè các phương thức abstract của nó.
- ❖ C#/JAVA: Một lớp dẫn xuất chỉ có thể thừa kế từ một base class
- ❖ Khắc phục những hạn chế của các abstract class bằng cách thay thế chúng bằng các interface



### ***Design Principle***

*Program to an interface, not an implementation.*

## Lập trình module



Huỳnh Tuấn Anh - ĐHNT

29

## Lập trình module

- ❖ Kiến trúc mô-đun (module) cho phép chia nhỏ bài toán (hay yêu cầu) của phần mềm thành các phần nhỏ 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.

Huỳnh Tuấn Anh - ĐHNT

30

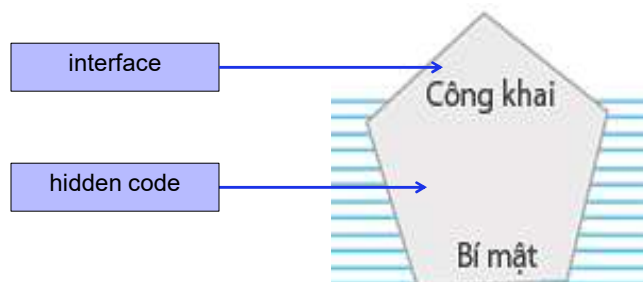
## interface: Công cụ giao tiếp của các modul

- ❖ 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

Huỳnh Tuấn Anh - ĐHNT

31

- ❖ interface của module nên được thiết kế chỉ bao gồm những phần hầu 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"



Huỳnh Tuấn Anh - ĐHNT

32





## 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.