Chương 2.
Lập trình hướng đối tượng trong Java

Mục tiêu

- Giới thiệu về lập trình hướng đối tượng
- Các đặc tính của lập trình hướng đối tượng
- Lập trình hướng đối tượng trong Java

2.1. Giới thiệu về lập trình hướng đối tượng

Nhắc lại về lập trình hướng thủ tục

- Trước đây chúng ta đã được học về lập trình thủ tục:
  - ✓Dữ liệu được tách riêng so với chương trình xử lý chúng.
  - ✓Người lập trình có trách nhiệm tự tổ chức các đoạn mã theo các đơn vị riêng rẽ.
  - ✓Không có sự trợ giúp từ các trình biên dịch hay ngôn ngữ cho việc thực thi các mô đun, …

It's hard to build large systems (not impossible, just hard).

OOP là gì?

- Viết tắt của "**O**bject **O**riented **P**rogramming".
- OOP là một triết lý về thiết kế.
- OOP là một mô hình lập trình.
- Sử dụng "đối tượng" – là cấu trúc dữ liệu bao gồm các trường dữ liệu và phương pháp cùng với các tương tác của nó - để thiết kế các ứng dụng và chương trình máy tính.

So sánh

Procedural	**Object-oriented**
procedure	method
module	object
procedure call	message
variable	attribute

OOP will NOT

- Thiết kế chương trình cho bạn
- Ngăn ngừa bạn thiết kế một chương trình tồi
- Cung cấp các thuật toán
- Thực hiện quản lý dữ liệu

Các từ khóa quan trọng

"object" Đối tượng	Là một thành phần chứa các dữ liệu cụ thể
"class" Lớp	Mô tả một tập các đối tượng
"instance" Thuộc tính	Một thuộc tính (biến instance) của một lớp là đối tượng của lớp đó
"method" Phương thức	Thực hiện một nhiệm vụ (hành vi) nào đó của đối tượng
"message" Thông báo	Yêu cầu để thực hiện một phương thức (giống như lời gọi thủ tục)

Các từ khóa quan trọng (tiếp)

- Một đối tượng sở hữu
  - ✓ Data values or *attributes*
  - ✓ Behavioral abilities or *methods*
- Các đối tượng với thuộc tính và phương thức của nó được biểu diễn trong một tập gọi là *class*.
- Một đối tượng là một thành viên của một lớp còn được gọi là thể hiện của lớp đó.

2.2. Các yếu tố cơ bản của OOP

4 đặc tính chính của OOP

- Tính bao đóng (đóng gói)
- Tính trừu tượng
- Tính kế thừa
- Tính đa hình

Đặc tính thứ nhất

- Tính bao đóng (đóng gói)

Bao đóng là che đi phần hiện thực của code, và chỉ cung cấp cho bên ngoài các chức năng. Mục đích là làm cho việc thay đổi trong code của một module không ảnh hưởng tới các module đã sử dụng nó.

Triển khai (hiện thực) và Giao diện

- Trong OOP, mỗi một lớp có 2 khía cạnh:
  1. Việc triển khai của một lớp (Implementation) - đó là việc triển khai các cấu trúc dữ liệu và mã thực thi.
  2. Giao diện (Interface) là sự phơi bày những gì được sử dụng bởi các lớp khác.

Ví dụ

```
public class Rectangle {

    private String color;
    private int width;
    private int heigh;

    // Constructor
    public Rectangle(String c, int w, int h) {
        this.setColor(c);
        this.setWidth(w);
        this.setHeigh(h);
    }

    private void validateSize() …
    private void checkPosition() …
    private void setCanvas() …
    private void perform() …
```

Implementation side

Implementation side

Ví dụ

```
    // Draw the rectangle
    public String draw() {
        // May be some complex code here
        this.validateSize();
        this.checkPosition();
        this.setCanvas();
        this.perform();
        return "I'm a "
            + this.getColor()
            + " rectangle.";
    }

    // Calculate perimeter
    public int getPerimeter() {
        // May be some complex code here
        return 2 * (this.getWidth()
                 + this.getHeigh());
    }
}
```

Public interface

Implementation side

Public interface

Implementation side

Ví dụ

```
/*
 * Client side code
 * Being a client is so easy
 */

Rectangle red = new Rectangle("Red", 5, 8);
// Simple send a message to draw
System.out.println(red.draw());
// Send another message to get perimeter
System.out.println("Perimeter is " + red.getPerimeter());
```

Client side

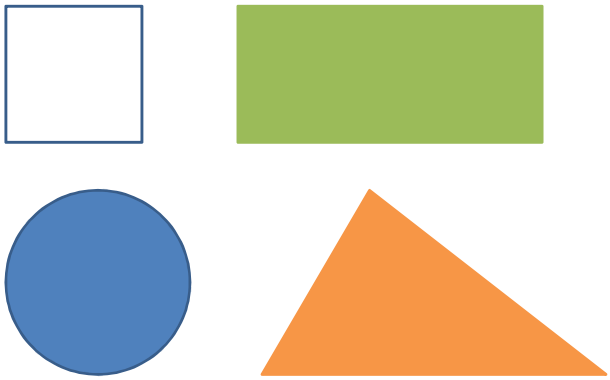
Đặc tính thứ 2

- Trừu tượng

Tính trừu tượng là gì?

- Trừu tượng là quá trình mà các dữ liệu và các chương trình được định nghĩa với một đại diện tương tự như ý nghĩa của nó, trong khi ẩn dấu đi những chi tiết thực hiện.
- Tính trừu tượng (Abstraction) trong Java hướng đến khả năng tạo một đối tượng trừu tượng trong lập trình hướng đối tượng. Một lớp trừu tượng là một lớp mà không được khởi tạo. Tất cả các chức năng khác của lớp vẫn tồn tại, và tất cả các trường, phương thức, và hàm khởi tạo đều được truy cập với một cách giống nhau. Không thể tạo một đối tượng với một lớp trừu tượng hóa.
- Sử dụng từ khóa **abstract** để khai báo một lớp trừu tượng, trước từ khóa class trong khai báo lớp.

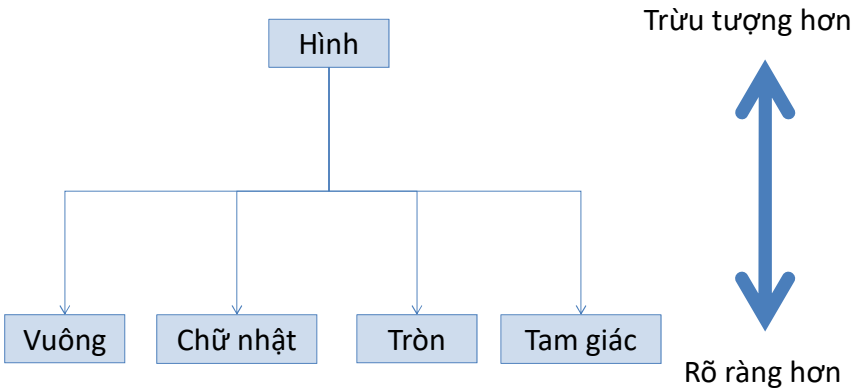
Ví dụ



Tính trừu tượng

- Đối với các hình Vuông, Chữ nhật, Tròn và Tam giác:
  - ✓ Có màu sắc
  - ✓ Có thể hiển thị
- Các hình đều có tất cả các đặc điểm trên, ta gọi "Hình" là một kiểu trừu tượng của Vuông, Chữ nhật, Tròn và Tam giác.

Tính trừu tượng



Đặc tính thứ 3

- Tính kế thừa

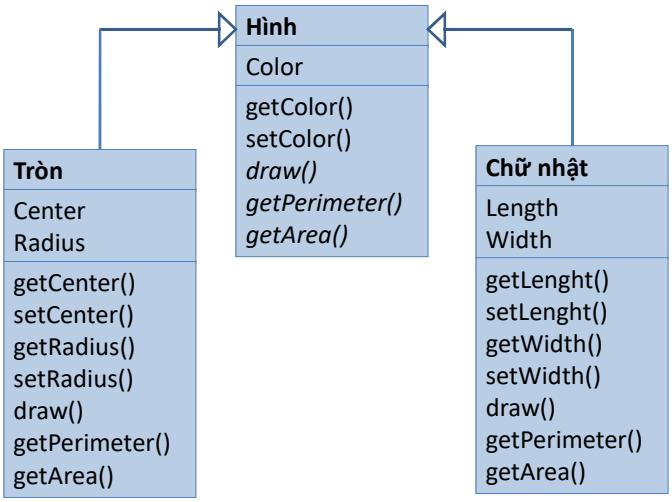
Tính kế thừa

- Nếu một lớp mới được tạo ra từ một lớp đã tồn tại bằng cách mở rộng nó, được gọi là tính kế thừa (thừa kế).
- Tính kế thừa trong Java là môt kỹ thuật mà trong đó một đối tượng thu được tất cả thuộc tính và hành vi của đối tượng cha.
- Khi kế thừa từ một lớp đang tồn tại, có thể tái sử dụng các phương thức và các thuộc tính của lớp cha, và cũng có thể bổ sung thêm các phương thức và các thuộc tính khác.

Một số khái niệm liên quan tính kế thừa

- Superclass / Subclass (lớp cha/ lớp con)
- Inheritance (kế thừa)
- Overriding (Ghi đè)
- Một thể hiện (đối tượng) của lớp con cũng là một thể hiện của lớp cha.
- Từ khóa thường được sử dụng là **extends** (đối với class) và **implements** (đối với interface).

Ví dụ về tính kế thừa



Ví dụ về tính kế thừa

- **Tròn** và **Chữ nhật** kế thừa từ **Hình**, vì thế nó có thuộc tính Color.
- **Hình** là lớp Cha (superclass).
- **Tròn** và **Chữ nhật** là các lớp con (subclasses).
- **Tròn** là một **Hình**, nhưng ngược lại thì không.
- Phương thức draw() của **Tròn** ghi đè phương thức draw() của **Hình**.
- Nếu ta bổ sung/ loại bỏ thuộc tính của **Hình** thì nó sẽ ảnh hưởng đến **Tròn** và **Chữ nhật**.

Đặc tính thứ 4

- Đa hình

Tính đa hình (Polymorphism)

- Trong lập trình hướng đối tượng, đa hình (nghĩa từ tiếng Hy Lạp "có nhiều hình thức") là đặc tính của việc có thể gán các ý nghĩa khác nhau cho một đối tượng.
- Tính đa hình muốn nói là *một tên, nhiều hình thức* (*one name, many forms*).

Summary

- Encapsulation (bao đóng)
- Abstraction (trừu tượng)
- Inheritance (kế thừa)
- Polymorphism (đa hình)

2.3. Lập trình hướng đối tượng trong Java

- Khởi tạo một kiểu đối tượng mới bằng từ khóa **`class`**.
- Một định nghĩa lớp có thể chứa:
  - ✓variables (fields) – thuộc tính, trường
  - ✓initialization code – mã khởi tạo
  - ✓Methods – các phương thức


```
class classname {
    field declarations (khai báo trường)
     { initialization code }
      Constructors (phương thức khởi tạo)
      Methods (các phương thức khác)
}
```

Khởi tạo một đối tượng

- Sử dụng *new* như sau:

```
classname varname = new classname();
```

- Một đối tượng phải được khởi tạo trước khi bất kỳ phương thức nào của lớp được gọi.
  - Ngoại trừ các phương thức ***static***.

Phương thức khởi tạo (Constructor)

- Mỗi lớp có thể có nhiều phương thức khởi tạo, chúng khác nhau về tham số.
- Nếu không viết phương thức khởi tạo, trình biên dịch sẽ hiểu là:

```
classname() {}
```

- Nếu có bất kỳ một phương thức khởi tạo nào được viết, trình biên dịch sẽ không tự khởi tạo phương thức khởi tạo ngầm định.

Nhiều phương thức khởi tạo

- Một phương thức khởi tạo có thể gọi các phương thức khởi tạo khác.
- Sử dụng biến "***this***", không sử dụng tên lớp:

```
class Foo {
    int i;
    Foo() {
        this(0);
    }
    Foo( int x ) {
        i = x;
    }
```


Các từ bổ nghĩa truy cập của lớp (Class Access Modifiers)

- **`public`**: bất kỳ lớp nào, ở đâu cũng nhìn thấy lớp này.
  - ✓Chỉ có 1 lớp **public** trong một tệp, và phải có tên lớp như tên tệp.
- Nếu 1 lớp không có modifier thì nó chỉ có tác dụng trong gói chứa nó (package).

Từ bổ nghĩa truy cập của các thuộc tính Fields Access Modifier

Bảng từ bổ nghĩa truy cập thuộc tính và phạm vi của nó

Modifier	Class	Package	Subclass	World
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
`no modifier` (Default)	Y	Y	N	N
`private`	Y	N	N	N

Các thuộc tính (trường) Static

- Khi khai báo một biến là **static**, thì biến đó được gọi là biến tĩnh, hay biến static.
- Biến **static** có thể được sử dụng để tham chiếu thuộc tính chung của tất cả đối tượng.
- Biến **static** được cấp bộ nhớ chỉ một lần.

Ví dụ

BienStatic.java

```
3  class SinhVien {
4      static int id = 0;
5      String hoTen;
6      String lop = "K54";
7
8      SinhVien() {
9          id++;
10         System.out.println(id);
11         System.out.println(lop);
12     }
13 }
14
15 public class BienStatic {
16
17     public static void main(String args[]) {
18
19         SinhVien c1 = new SinhVien();
20         SinhVien c2 = new SinhVien();
21         SinhVien c3 = new SinhVien();
22
23     }
24 }
```


Các trường Final (Final Fields)

- Từ khóa **final** muốn nói: chỉ 1 lần giá trị được thiết lập, không thể thay đổi.
- Được sử dụng để khai báo các hằng.

  static final int BUFSIZE=100;

  static final double PI=3.14159;

Các từ bổ nghĩa phương thức
Method Modifiers

- **`private/protected/public/`*`no modifier`***:
  - ✓Ý nghĩa tương tự như với các trường.
- **`abstract`**: các phương thức không được triển khai mà nó được cung cấp bởi các lớp con.
  - ✓Lớp chứa nó cũng phải khai báo **`abstract`**

Các từ bổ nghĩa phương thức
Method Modifiers

- **`static`**: phương thức không phụ thuộc vào bất kỳ một biến instance hay các phương thức khác, và có thể được gọi mà không cần khởi tạo đối tượng.
- **`final`**: là phương thức không thể thay đổi bởi lớp con.