

Web-based programming with Java



# LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

ThS. Nguyễn Nghiêm  
0913.745.789 - NghiemN@fpt.edu.vn



# NỘI DUNG

---

- Lớp và đối tượng
- Định nghĩa lớp
- Đặc tả truy xuất
- Tham số biến đổi
- Overloading
- Kế thừa
- Overriding
- Static
- Final





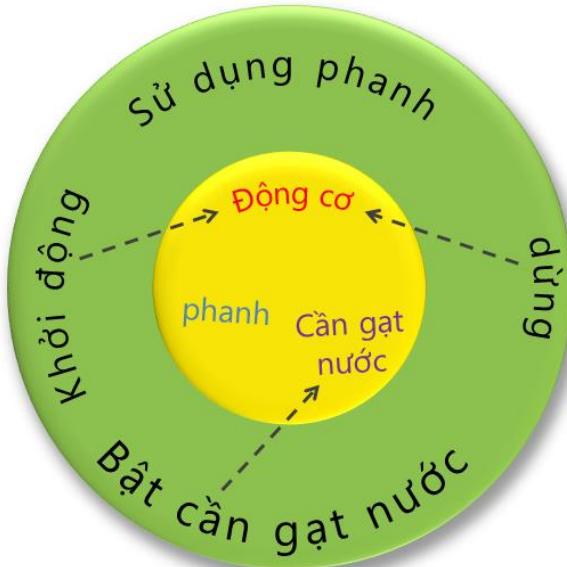
# ĐỐI TƯỢNG

- Biểu diễn 1 đối tượng trong thế giới thực
- Mỗi đối tượng được đặc trưng bởi các thuộc tính và các hành vi riêng của nó





# ĐẶC ĐIỂM VÀ HÀNH VI



## ● Thuộc tính (đặc điểm)

- Hãng sản xuất
- Model
- Năm
- Màu

## ● Ô tô có thể làm gì (hành vi)?

- Khởi động
- Dừng
- Phanh
- Bật cần gạt nước



# THUỘC TÍNH & PHƯƠNG THỨC



## ● Thuộc tính (attributes)

- ★ Hãng sản xuất
- ★ Model
- ★ Năm
- ★ Màu

## ● Phương thức (method)

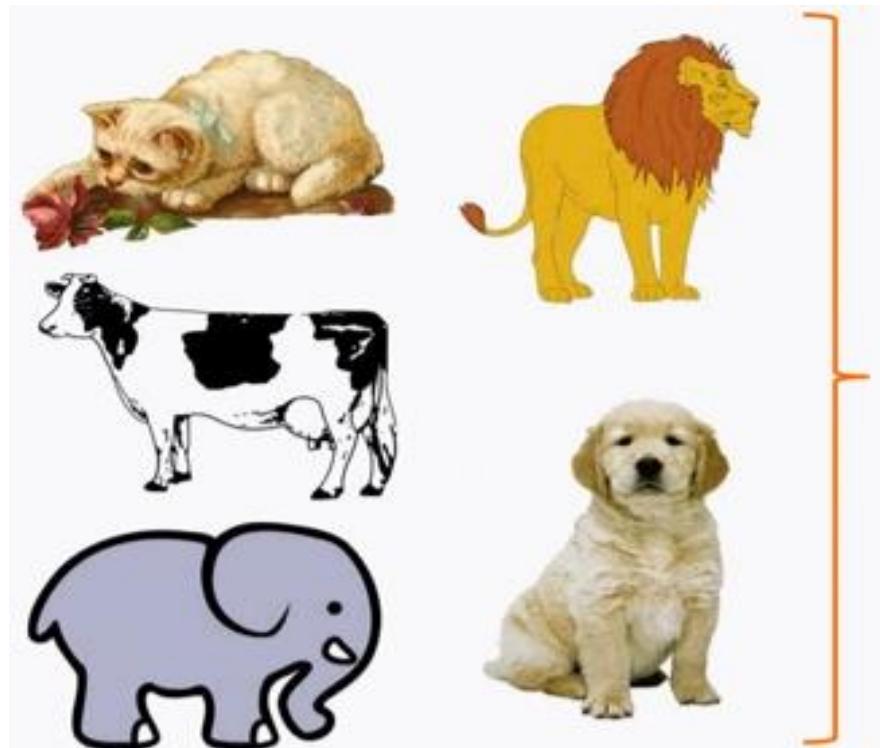
- ★ Khởi động
- ★ Dừng
- ★ Phanh
- ★ Bật cần gạt nước



# CLASS LÀ GÌ?



Nhóm các **Xe hơi**

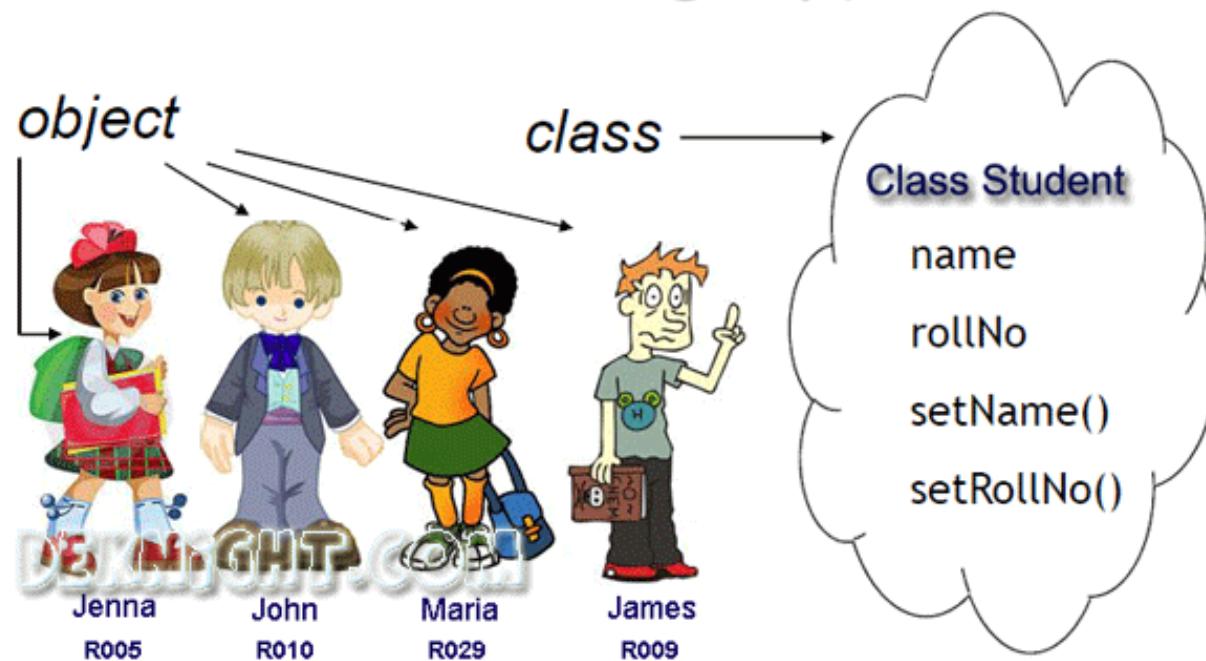


Nhóm các **Động vật**



# ĐỊNH NGHĨA LỚP

- Lớp là một khuôn mẫu mô tả các đối tượng cùng loại.
- Lớp bao gồm các thuộc tính (trường dữ liệu) và các phương thức (hàm tác động lên các thuộc tính bên trong lớp)





# MÔ HÌNH HÓA LỚP VÀ ĐỐI TƯỢNG

## Car

### Thuộc tính

- Năm
- Nhà sản xuất
- Model
- Màu

### Phương thức

- Khởi động
- Dừng
- Phanh
- Rửa xe

## Ô tô của Dũng

### Thuộc tính

- Năm = 2010
- Nhà sản xuất=Honda
- Model = Accord
- Màu = Xanh

### Phương thức

- Khởi động
- Dừng
- Phanh
- Rửa xe

## Ô tô của Mai

### Thuộc tính

- Năm = 2012
- Nhà sản xuất=BMW
- Model = CS30
- Màu = Bạc

### Phương thức

- Khởi động
- Dừng
- Phanh
- Rửa xe



# CẤU TẠO CHUNG CỦA CLASS

```
class className
{
    type field1;
    type field2;
    ...
    type method1(parameters) {
        ...
    }
    type method2(parameters) {
        ...
    }
    ...
}
```



# ĐỊNH NGHĨA CLASS

```
public class Employee{  
    public String fullname;  
    public double salary;  
  
    public void input(){  
        Scanner scanner = new Scanner(System.in);  
        System.out.print(" > Full Name: ");  
        this.fullname = scanner.nextLine();  
  
        System.out.print(" > Salary: ");  
        this.salary = scanner.nextDouble();  
    }  
  
    public void output(){  
        System.out.println(this.fullname);  
        System.out.println(this.salary);  
    }  
}
```

Trường



Phương thức



# TẠO ĐỐI TƯỢNG

```
public static void main(String[] args) {
```

```
    Employee emp = new Employee();  
    emp.input();  
    emp.output();
```

```
}
```

- Toán tử **new** cấp phát bộ nhớ động cho đối tượng
- Biến emp chứa tham chiếu tới đối tượng
- Sử dụng dấu chấm (.) để truy xuất thành viên của lớp





# HÀM TẠO

- **Hàm tạo (constructor) là hàm được sử dụng để tạo đối tượng.**
  - ★ Tên trùng với tên lớp
  - ★ Không trả lại giá trị
- **Có thể định nghĩa nhiều hàm tạo khác tham số. Mỗi hàm tạo cung cấp 1 cách tạo đối tượng.**
- **Nếu không khai báo hàm tạo thì Java tự động sinh ra hàm tạo mặc định (không tham số)**



**Tạo lớp mô tả hình chữ nhật**



# PHƯƠNG THỨC (METHOD)

- Phương thức là một module mã thực hiện một công việc cụ thể nào đó
- Phương thức có thể trả về void hoặc một kiểu dữ liệu nào đó
- Phương thức có thể có một hoặc nhiều tham số

```
return-type name(parameters)  
{  
    // body of method  
}
```



access	return type	name	parameters
public	void	add	(int a, int b)
public void add(int a, int b) { // do stuff here }			



# ĐỊNH NGHĨA PHƯƠNG THỨC

```
public class Employee{  
    public String fullname;  
    public double salary;  
  
    public void input(){...}  
    public void output(){...}  
  
    public void setInfo(String fullname, double salary) {  
        this.fullname = fullname;  
        this.salary = salary;  
    }  
  
    public double incomeTax(){  
        if(this.salary < 5000000){  
            return 0;  
        }  
        double tax = (this.salary - 5000000) * 10/100;  
        return tax;  
    }  
}
```

Kiểu trả về là **void** nên thân hàm **không** chứa lệnh **return giá trị**

Kiểu trả về là **double** nên thân hàm phải chứa lệnh **return số thực**



# THAM SỐ BIẾN ĐỔI

- Các hàm có tham số biến đổi đã sử dụng
  - ★ System.out.printf(format, a, b, c...)
  - ★ String.format(format, a, b, c...)
- Các hàm này nhận lượng tham số tùy ý (a, b, c...)
- Tham số được truyền vào với số lượng tùy thích như thế được gọi là tham số biến đổi.



# THAM SỐ BIẾN ĐỔI

```
public class MyClass{  
  
    public void myMethod1(int[] args) {  
        for(int a: args){  
            // xử lý phần tử a  
        }  
    }  
  
    public void myMethod2(int...args) {  
        for(int a: args){  
            // xử lý phần tử a  
        }  
    }  
}
```

Tham số biến đổi

```
MyClass myObject = new MyClass();  
  
int[] args = {5, 6, 2};  
myObject.myMethod1(args);  
myObject.myMethod2(args);  
myObject.myMethod2(5, 6, 2);
```

Truyền tham  
số biến đổi



## THAM SỐ BIẾN ĐỔI

---

- Thực chất của tham số biến đổi là tham số mảng.
- Tham số loại này cho phép bạn truyền nguyên mảng hoặc trực tiếp các phần tử mảng.
- Trong một hàm, chỉ có thể khai báo duy nhất một tham số kiểu varargs và phải là tham số cuối cùng
- Cho phép bạn viết phương thức với số lượng tham số tùy ý.



Tính tổng các phần tử  
của một dãy số bất kỳ

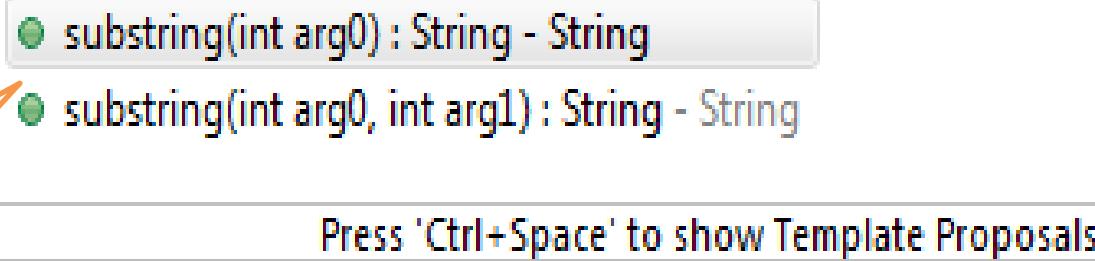


# OVERLOADING

- Overloading là trường hợp có nhiều phương thức cùng tên nhưng khác tham số.
- Ví dụ: lớp String có 2 phương thức substring là
  - \* public String substring(int start, int end)
  - \* public String substring(int start)

```
String hello = "Hello World !";
```

```
hello.subst
```



2 phương thức  
cùng tên nhưng  
khác tham số



# OVERLOADING

```
class HangHoa{  
    String ten;  
    double gia;  
    HangHoa(String name, double price){  
        this.ten = name;  
        this.gia = price;  
    }  
    void hienThi(String tieuDe){  
        System.out.println(tieuDe);  
        System.out.printf("\t%s\r\n\t%.3f\r\n", ten, gia);  
    }  
    void hienThi(){  
        this.hienThi("Thông tin hàng hóa: ");  
    }  
}
```

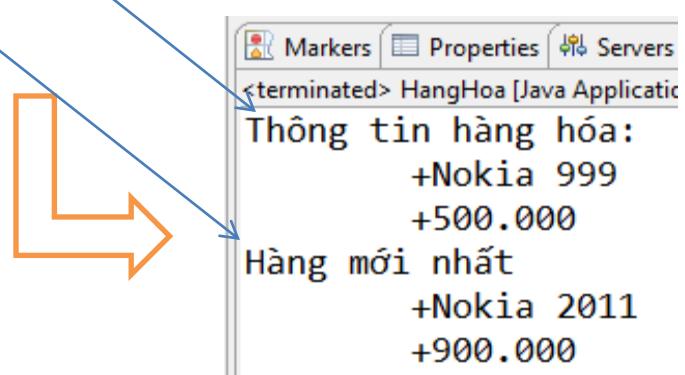
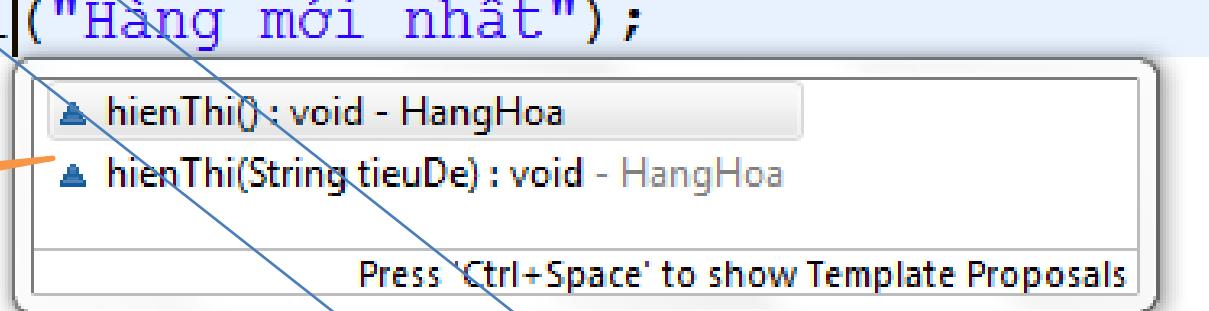
2 phương thức  
cùng tên



# OVERLOADING

```
public static void main(String[] args) {  
    HangHoa hh1 = new HangHoa ("Nokia 999", 500);  
    HangHoa hh2 = new HangHoa ("Nokia 2011", 900);  
    hh1.hienThi();  
    hh2.hienThi("Hàng mới nhất");  
}
```

Gợi ý của công cụ





# PACKAGE

- Package được sử dụng để chia các class và interface thành từng gói khác nhau.
  - \* Việc làm này tương tự quản lý file trên ổ đĩa trong đó class (file) và package (folder)
- Ví dụ sau tạo lớp MyClass thuộc gói com.poly

```
package com.poly;  
public class MyClass{...}
```

- Trong Java có rất nhiều gói được phân theo chức năng
  - \* java.util: chứa các lớp tiện ích
  - \* java.io: chứa các lớp vào/ra dữ liệu
  - \* java.lang: chứa các lớp thường dùng...



# IMPORT PACKAGE

- Lệnh import được sử dụng để chỉ ra lớp đã được định nghĩa trong một package
- Các lớp trong gói java.lang và các lớp cùng định nghĩa trong cùng một gói với lớp sử dụng sẽ được import ngầm định

```
package com.polyhcm;
import com.poly.MyClass;
import java.util.Scanner;
public class HelloWorld{
    public static void main(String[] args){
        MyClass obj = new MyClass();
        Scanner scanner = new Scanner(System.in);
    }
}
```



## ĐẶC TẢ TRUY XUẤT

- Đặc tả truy xuất (**private**, **protected**, **public**, **{default}**) được sử dụng để định nghĩa khả năng cho phép truy xuất đến các thành viên của lớp.
  - \* **private**: chỉ được phép sử dụng nội bộ class
  - \* **public**: công khai hoàn toàn
  - \* **{default}** : public với các lớp cùng gói và private với các lớp khác gói.
  - \* **protected**: tương tự **{default}** nhưng cho phép kế thừa cho dù lớp con và cha khác gói.

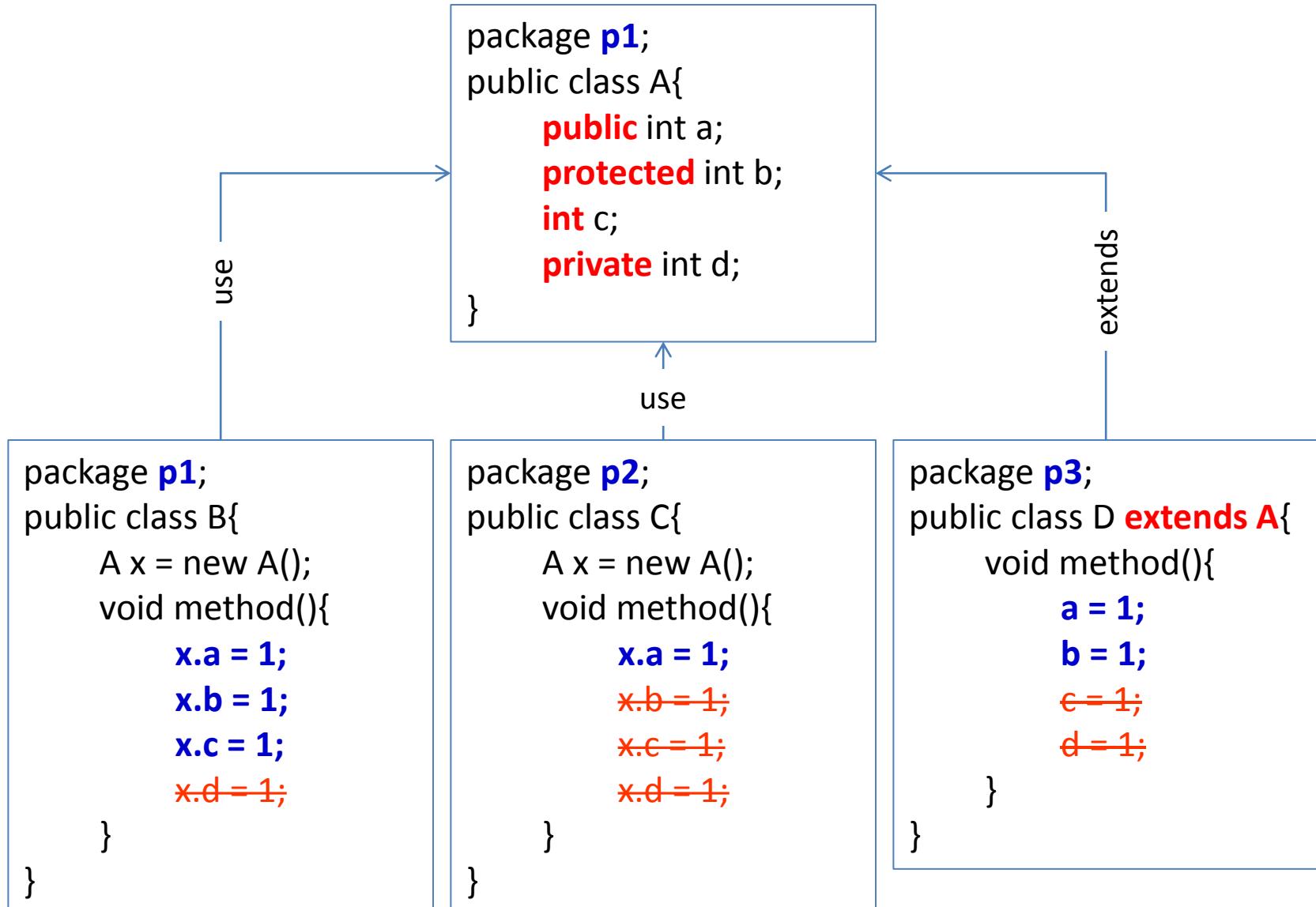


# ĐẶC TẢ TRUY XUẤT

- Đặc tả truy xuất được sử dụng để định nghĩa khả năng cho phép truy xuất đến các thành viên của lớp. Trong java có 4 đặc tả khác nhau:
  - \* **private**: chỉ được phép sử dụng nội bộ trong class
  - \* **public**: công khai hoàn toàn
  - \* **{default}**:
    - Là public đối với các lớp truy xuất cùng gói
    - Là private với các lớp truy xuất khác gói.
  - \* **protected**: tương tự {default} nhưng cho phép kế thừa dù lớp con và cha khác gói.
- Mức độ che dấu tăng dần theo chiều mũi tên  
**public** → **protected** → **{default}** → **private**



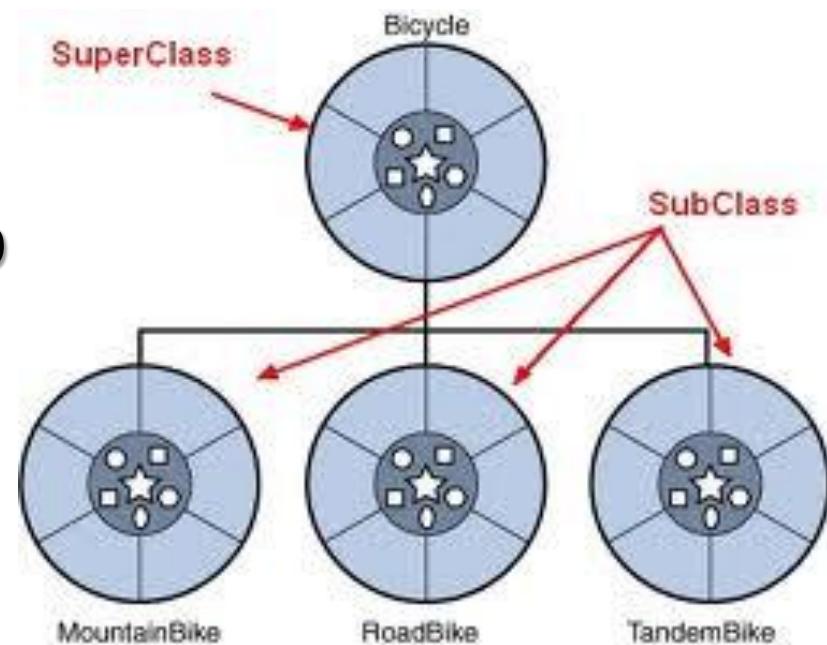
# ĐẶC TẢ TRUY XUẤT





# THÙA KẾ

- Các lớp trong Java tồn tại trong một hệ thống thứ bậc, gọi là cây thừa kế
- Các lớp ở bậc trên một lớp đã cho trong một hệ thống thứ bậc là lớp cha (superclass) của lớp đó
- Lớp cụ thể là một lớp con (subclass) của tất cả các lớp bậc cao hơn





# TÍNH THỪA KẾ

Phương tiện  
di chuyển

Xe đạp



Xe con



Xe tải



Xe thể thao



Xe trọng tải nhỏ





## TỔ CHỨC KẾ THỪA

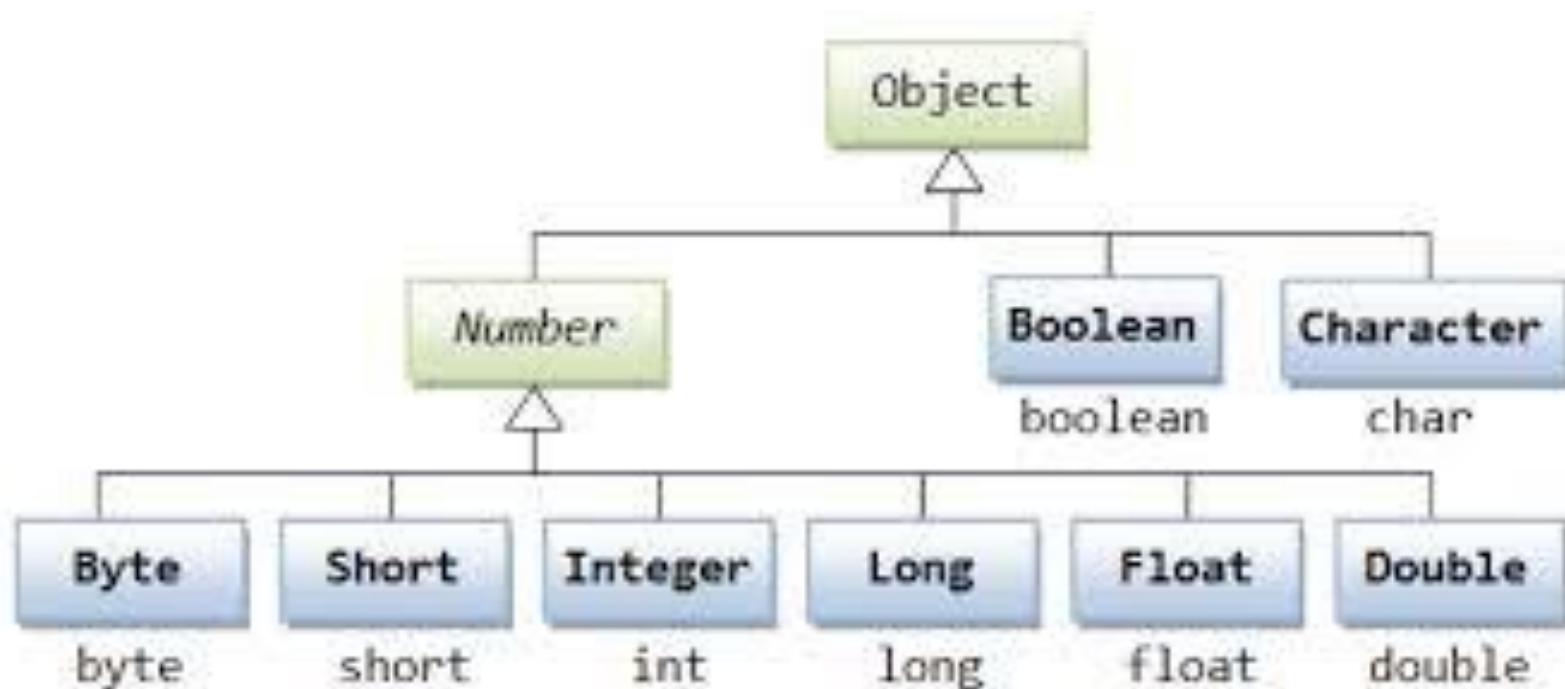
---

- class PhuongTien{...}
- class XeDap **extends PhuongTien{...}**
- class XeCon extends PhuongTien{...}
- class XeTai extends PhuongTien{...}
- class XeTheThao extends XeCon{...}
- class XeTrongTaiNho extends XeCon{...}



# LỚP OBJECT

- Các lớp đều mặc định kế thừa lớp Object  
(Không phải khai báo kế thừa lớp Object khi khai báo một lớp)





# KẾ THỪA

- Mục đích của kế thừa là để tái sử dụng
- Lớp con được phép sử dụng lại những gì trong lớp cha đã định nghĩa

```
class Parent{  
    Public void m1(){...}  
}  
  
class Child extends Parent{  
    public void m2(){  
        this.m1();  
    }  
}
```

```
Child object = new Child();  
object.m1();
```



# KẾ THỪA

```
class DSNhanVien extends ArrayList<NhanVien> {  
    public void print() {  
        for(int i=0;i<this.size();i++) {  
            NhanVien nv = this.get(i);  
            nv.hienThi();  
        }  
    }  
}  
  
public static void main() {  
    DSNhanVien dsnv = new DSNhanVien();  
    dsnv.add(new NhanVien("My Hoa", 500));  
    dsnv.add(new NhanVien("My Hạnh", 350));  
    dsnv.print();  
}
```

- Lớp DSNhanVien thực chất là `ArrayList<NhanVien>` có bổ sung phương thức `print()`.



# SỬ DỤNG TỪ KHÓA SUPER

- Truy cập đến các thành viên của lớp cha bằng cách sử dụng từ khóa **super**
- Có thể sử dụng **super** để truy cập đến hàm tạo của lớp cha

```
class Parent
{
    String name;
}
class Child extends Parent {

    String name;

    void detail()
    {
        super.name = "Parent";
        name = "Child";
    }
}
```



# OVERRIDING

- Trong kế thừa, khi lớp con và lớp cha có các phương thức cùng cú pháp thì xảy ra overriding. Khi đó phương thức của lớp con sẽ nạp đè lên phương thức của lớp cha.
- Trong lớp con hãy sử dụng từ khóa **super** để truy cập đến phương thức của lớp cha.

Hình  
method()

Child  
method()



# OVERRIDING

```
class Hinh{  
    String ten;  
    protected double dienTich;  
    protected double chuVi;  
    public Hinh(String ten){  
        this.ten = ten;  
    }  
    public void xemThongTin() {  
        System.out.println(" >> Tên: " + this.ten);  
        System.out.println(" >> Diện tích: " + this.chuVi);  
        System.out.println(" >> Chu vi: " + this.dienTich);  
    }  
}
```

- Protected có ý nghĩa là cho phép thừa hưởng ngay cả khi lớp con khác package với lớp cha



# OVERRIDING

```
class HinhTron extends Hinh{  
    double banKinh;  
    public HinhTron(double banKinh) {  
        super("Hình Tròn");  
        this.banKinh = banKinh;  
  
        super.dienTich = Math.PI * Math.pow(banKinh, 2);  
        super.chuVi = 2 * Math.PI * banKinh;  
    }  
    public void xemThongTin() {  
        super.xemThongTin();  
        System.out.println(" >> Bán kính: " + this.banKinh);  
    }  
}
```

Gọi hàm tạo của lớp cha

Gọi xemThongTin() của lớp cha



# LỚP TRƯU TƯỢNG

```
abstract class Hinh{  
    public String ten;  
    Hinh(String ten) {  
        this.ten = ten;  
    }  
    abstract public double tinh DienTich();  
    abstract public double tinh ChuVi();  
    public void hienThi() {  
        System.out.printf("Tên hình: %s\r\n",  
            this.ten);  
        System.out.printf("Diện tích: %.3f\r\n",  
            this.tinh DienTich());  
        System.out.printf("Chu vi: %.3f\r\n",  
            this.tinh ChuVi());  
    }  
}
```

Các phương thức trừu tượng  
(phương thức chưa hoàn thiện)



## LỚP TRÙU TƯỢNG

---

- Lớp trừu tượng (Abstract class) là lớp chứa các phương thức trừu tượng (chưa định nghĩa đầy đủ các phương thức).
- Không thể tạo đối tượng từ lớp trừu tượng
- Các lớp con (lớp cụ thể) phải viết mã cho các phương thức trừu tượng



# LỚP TRƯU TƯỢNG

```
class HinhTron extends Hinh{  
    double banKinh;  
    HinhTron(double banKinh) {  
        super("Hình tròn");  
        this.banKinh = banKinh;  
    }  
    @Override  
    public double tinhDienTich() {  
        return Math.PI*Math.pow(this.banKinh, 2);  
    }  
    @Override  
    public double tinhChuVi() {  
        return 2*Math.PI*this.banKinh;  
    }  
}
```

```
class HinhCN extends Hinh{  
    double rong, dai;  
    HinhCN(double rong, double dai) {  
        super("Chữ nhật");  
        this.rong = rong;  
        this.dai = dai;  
    }  
    @Override  
    public double tinhDienTich() {  
        return this.dai*this.rong;  
    }  
    @Override  
    public double tinhChuVi() {  
        return 2*(this.dai+this.rong);  
    }  
}
```



# ĐỊNH NGHĨA INTERFACE

- Interface chứa một nhóm phương thức về một chủ đề nào đó.

```
public interface DbAction {  
    void insert(String sql);  
    void update(String sql);  
    void delete(String sql);  
    void select(String sql);  
}
```



# THỰC THI INTERFACE

- Các phương thức này sẽ được viết mã bởi các lớp implements interface này.

```
public class Product
    implements DbAction{
    public void insert(String sql) {...}
    public void update(String sql) {...}
    public void delete(String sql) {...}
    public void select(String sql) {...}
}
```



# THỰC THI NHIỀU INTERFACE

- Mỗi lớp có thể implements theo nhiều interface khác nhau.

```
public class MyClass
    implements interface1, interface2,...{
    /*
     * viết mã cho các phương thức
     * của tất cả các interface
     */
}
```



# KẾ THỪA INTERFACE

- Một interface có thể kế thừa từ nhiều interface khác.
- Thực chất của kế thừa interface là gộp các phương thức từ nhiều interface

```
public interface I1 {  
    void m1();  
}
```

```
public interface I2 {  
    void m2();  
}
```

```
public interface I3 extends I1, I2 {  
    void m3();  
}
```



# CHÚ Ý VỀ INTERFACE

---

- Interface chỉ chứa các ***phương thức abstract*** và các ***biến final***
- Bổ từ truy xuất của các phương thức trong Interface ***luôn là public*** (mặc định là public)
- Một class thực thi theo một interface thì phải ***viết mã cho các phương thức*** trong interface.
- Interface cho phép ***đa kế thừa*** (có thể được kế thừa từ một hoặc nhiều interface khác).



# INTERFACE VS ABSTRACT CLASS

## ● Thành phần:

- ★ Abstract class có thể chứa phương thức abstract, phương thức hoàn thiện, biến...
- ★ Interface chỉ chứa phương thức abstract và biến final.

## ● Đặc tả truy xuất:

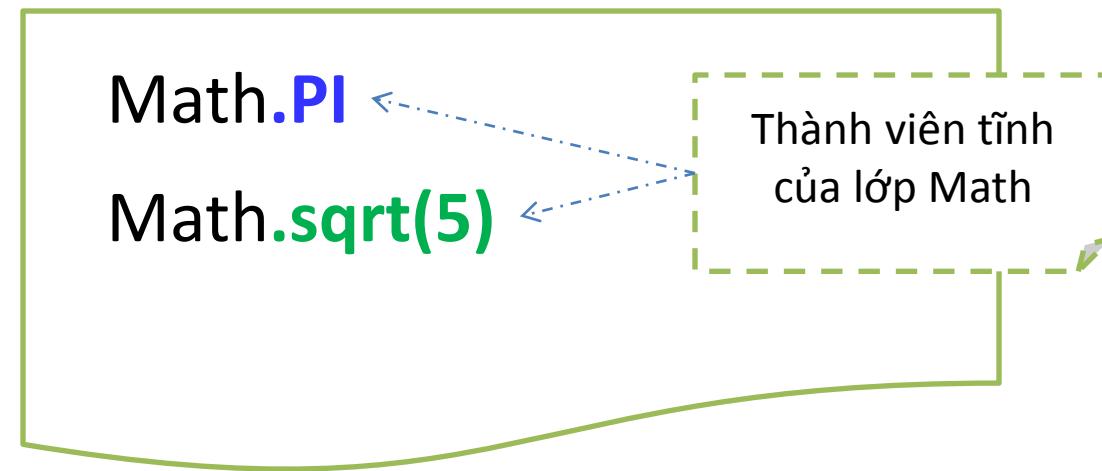
- ★ Abstract class có đặc tả truy xuất tùy ý
- ★ Interface luôn luôn là public.

## ● Kế thừa:

- ★ Một lớp chỉ được kế thừa một lớp abstract
- ★ Một lớp có thể thực thi theo nhiều interface



- Static được sử dụng để định nghĩa cho các thành viên (phương thức, trường) tĩnh.
- Thành viên tĩnh của một lớp được sử dụng độc lập với các đối tượng được tạo từ lớp đó.
- Nên truy cập thành viên tĩnh thông qua tên lớp để tránh nhầm lẫn với các thành viên.

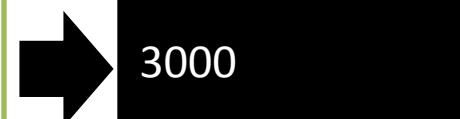




# ĐỊNH NGHĨA VÀ SỬ DỤNG STATIC

```
class MyClass{  
    public static int myField;  
  
    public static void myMethod() {  
        System.out.println(MyClass.myField);  
    }  
  
    static{  
        myField += 1000;  
    }  
}
```

```
MyClass.myField += 2000;  
MyClass.myMethod();
```





# PHƯƠNG THỨC STATIC

```
public class Keyboard {  
  
    static Scanner scanner = new Scanner(System.in);  
  
    public static String readString(String prompt) {}  
  
    public static int readInt(String prompt) {}  
  
    public static double readDouble(String prompt) {}  
  
    public static boolean readBoolean(String prompt) {}  
}
```

```
public static void main(String[] args) {  
    String name = Keyboard.readString("Họ và tên: ");  
    int age = Keyboard.readInt("Tuổi: ");  
}
```



# HẰNG - FINAL

---

- Từ khóa final được sử dụng để định nghĩa hằng.
- Có 3 loại hằng trong Java
  - \* **Lớp hằng** là lớp không cho phép kế thừa
  - \* **Phương thức hằng** là phương thức không được phép override
  - \* **Biến hằng** là biến không thể thay đổi giá trị



- Bạn không thể thay đổi giá trị của một biến hằng sau khi đã gán giá trị

```
public class LuongGiac{  
    public static final int GocVuong = 90  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Math.PI = 3.50;  
        LuongGiac.GocVuong = 120;  
    }  
}
```



# PHƯƠNG THỨC HẰNG

- Bạn có thể định nghĩa phương thức hằng nếu không muốn lớp con override nó

```
public class MyClass {  
    public final void method() { ... }  
}
```

```
public class MySubClass  
    extends MyClass {  
    public void method() { ... }  
}
```



# LỚP HẰNG

- Trong Java có nhiều lớp hằng: Math, String...
- Bạn có thể định nghĩa lớp hằng riêng của mình nếu không muốn người khác kế thừa

```
public final class MyClass {  
    ...  
}
```

```
public class MySubClass extends MyClass { ... }  
public class SubStringClass extends String { ... }
```