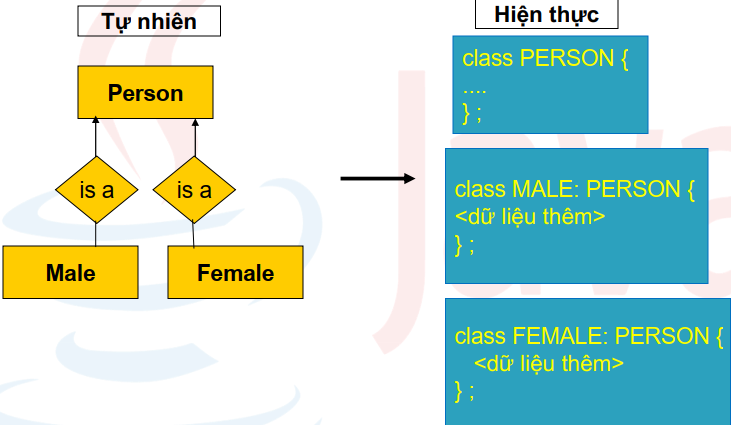
**Object Oriented Programming (OOP)**

* Chương trình: sự hoạt động của các đối tượng 🡪 Giống tự nhiên
* Đối tượng thực thi một hoạt động tức là đối tượng thực hiện một phương thức mà đối tượng này có khả năng
* Một chương trình: là một trật tự các lời yêu cầu đối tượng thực hiện phương thức của mình.

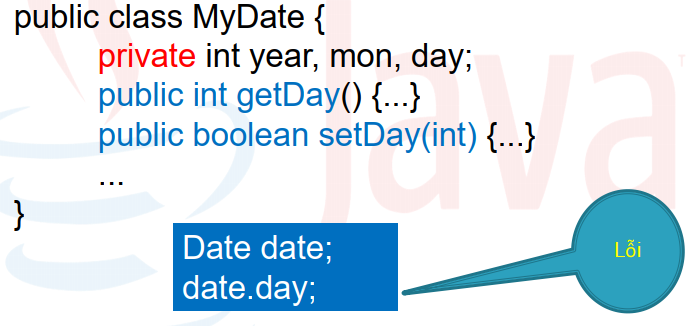
🡺 Chương trình là một kịch bản (script)

**Ưu điểm**

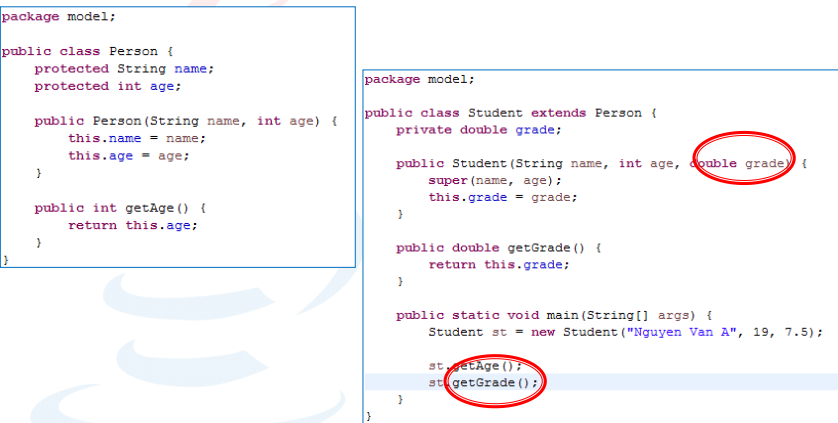
* Dễ mô tả các quan hệ phân cấp trong thế giới tự nhiên



* Có tính bảo mật cao
  + Bên ngoài không thể tuỳ tiện truy cập một dữ liệu thuộc tính



* Tái sử dụng code



**Tính đóng gói (Encapsulation)**

* Là cơ chế ràng buộc dữ liệu và thao tác trên dữ liệu đó thành một thể thống nhất, tránh được các tác động bất ngờ từ bên ngoài. Thể thống nhất này được gọi là đối tượng.
* Đối tượng (object):

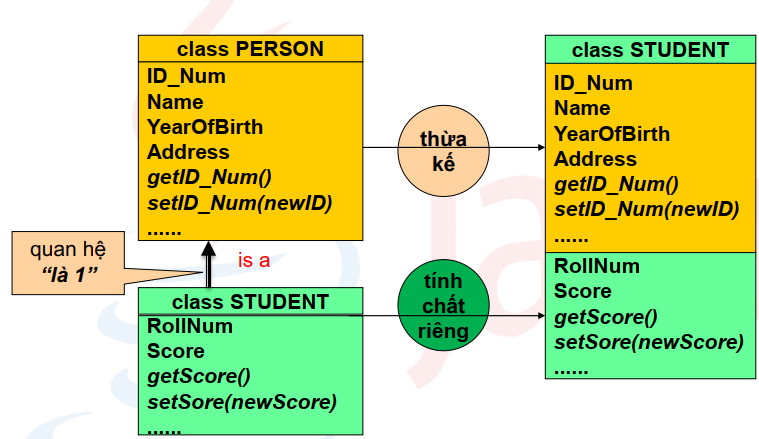
➔ Bao gói dữ liệu + phương thức.

* Đối tượng phải thuộc một lớp (class).
* Muốn làm việc trên đối tượng, ta phải xây dựng lớp

➔ Class = data (biến, thuộc tính)+ methods (code)

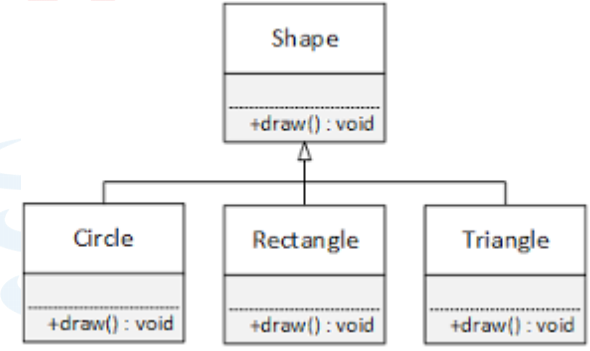
* Che giấu đi dữ liệu.
* Bên ngoài chỉ tương tác được với đối tượng qua một số phương thức.
* Phương thức giao tiếp (public, interface methods): phương thức giúp đối tượng giao tiếp với môi trường.
* Phương thức riêng/nội (private, internal): phương thức xử lý dữ liệu nội tại của đối tượng).

**Tính kế thừa (Inheritance)**



**Tính đa hình (Polymorphism)**

* Kỹ thuật cho phép thay đổi nội dung của cùng một phương thức trong 2 lớp cha con.
* Bản chất: Sửa code của cùng một phương thức ở 2 lớp cha-con để các code này khác nhau.
* Ví dụ: Lớp Shape có phương thức draw() nhưng ở ba lớp con Rectangle, Triangle, Circle phương thức này phải được thể hiện khác nhau



**Tính trừu tượng (Abstraction)**

* Trừu tượng trong thực tế còn có thể hiểu là cái gì đó không có thực.
* Trong OOP: giúp che giấu các hoạt động bên trong và chỉ hiển thị những tính năng thiết yếu của đối tượng tới người dùng
* Java trừu tượng hóa thông qua các lớp trừu tượng (*Abstract class*) và các giao diện (*Interface*)
  + Abstract class:
    - Không thể được dùng để tạo ra các đối tượng như những lớp bình thường khác; được xem như là một cái sườn để tạo ra các lớp con
  + Interface:
    - Một bản thiết kế của một lớp; chỉ có các phương thức trừu tượng; không có thuộc tính (attributes)

**Lớp và đối tượng**

* Lớp (class) là khuôn mẫu để sinh ra đối tượng
* Đối tượng là thể hiện (instance) của một lớp.
* Đối tượng có
  + Định danh
  + Thuộc tính (dữ liệu)
  + Hành vi (phương thức)

**Hệ thống hướng đối tượng**

* Bao gồm một tập các đối tượng
  + Mỗi đối tượng chịu trách nhiệm một công việc
* Các đối tượng tương tác thông qua trao đổi thông điệp (message)
* Các đối tượng có thể tồn tại phân tán/có thể hoạt động song song

**Mô hình hoá đối tượng**

class MyDate {  
 - int day;  
 - int month;  
 - int year;

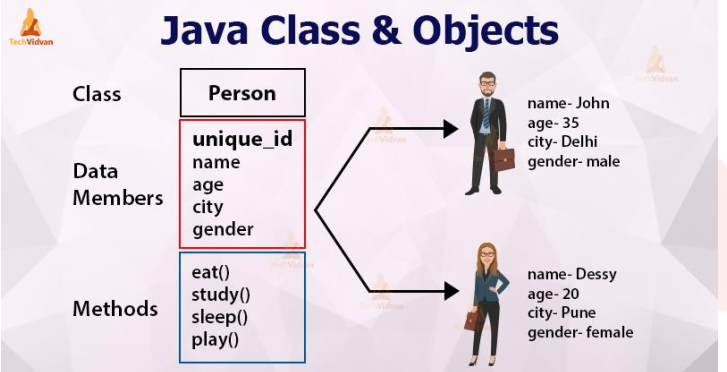
+ int getDay() {…}  
+ void setDay(int day) {…}  
+ int getMonth() {…}  
+ void setMonth(int month) {…}  
+ int getYear() {…}  
+ void setYear(int year) {…}  
- boolean isValidDate() {…}

**Ưu điểm của lập trình hướng đối tượng**

* Tăng năng suất lập trình (năng suất phát triển)
* Tăng chất lượng phần mềm
* Tăng tính hiểu được của phần mềm
* Tăng vòng đời của phần mềm

**Lớp và đối tượng**

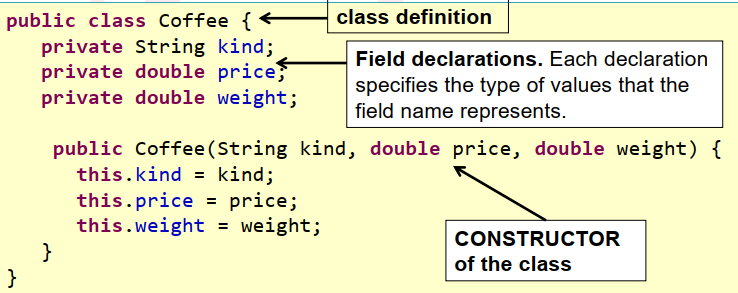
* Lớp (class) là khuôn mẫu để sinh ra đối tượng
* Đối tượng là thể hiện (instance) của một lớp.
* Khi định nghĩa lớp (class) ta cần định nghĩa:  
  ◦ định danh  
  ◦ thuộc tính (biến)  
  ◦ hành vi (phương thức)



|  |  |
| --- | --- |
| [public] | Lớp được truy xuất chung cho các Package khác, mặc định chỉ có các đoạn mã trong cùng một gói mới có quyền truy xuất nó |
| [abstract] | Lớp trừu tượng, không thể khởi tạo |
| [final] | Lớp bằng không có lớp con, không kế thừa |
| Class *ClassName* | Tên lớp |
| [extends *SuperClass]* | Kế thừa lớp cha *SuperClass* |
| [implements *Interfaces*] | Giao diện được cài đặt bởi Class |
| {//Member Variables Declarations | Khai báo các biến |
| // Methods Declarations | Khai báo các phương thức |
| } |  |

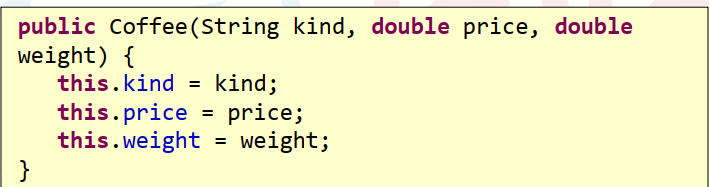
**Xây dựng lớp**

* Hàm dựng(constructor) để tạo đối tượng Coffee với 3 thuộc tính kind, price, và weight



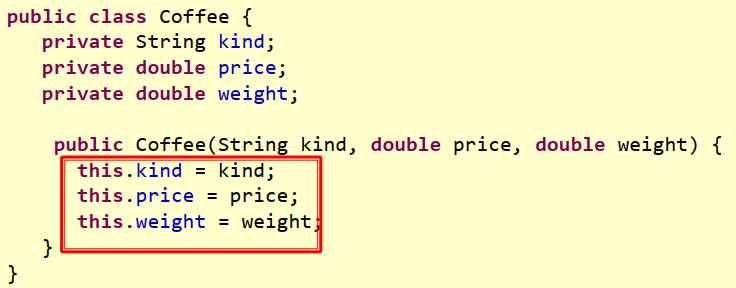
**Constructor**

* Constructor: có tên trùng với tên lớp
* Các tham số được đặt trong cặp dấu (), cách nhau bởi dấu ,
* Chứa nhiều lệnh gán giá trị cho các thuộc tính: **this**.*fieldName* = *parameterName*;

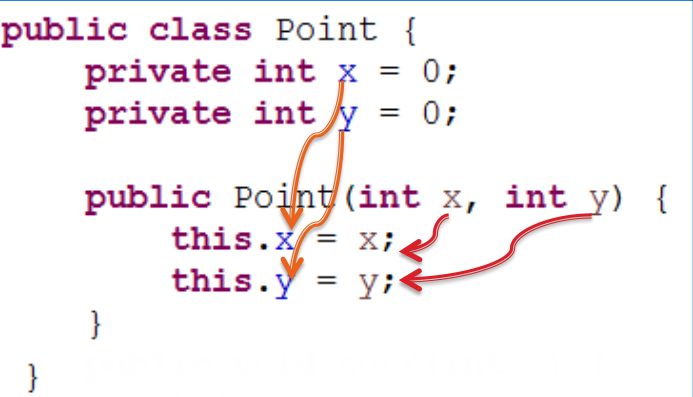


**Từ khoá this**

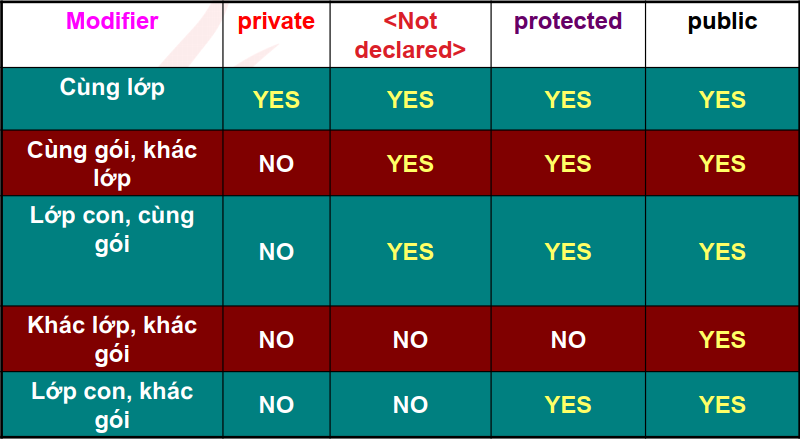
* Từ khóa **this**: dùng để tham khảo đến đối tượng hiện tại trong phương thức hoặc hàm .dựng
* Trong hàm dựng: ‘**this**’ dùng để gán giá trị cho các thuộc tính của lớp.

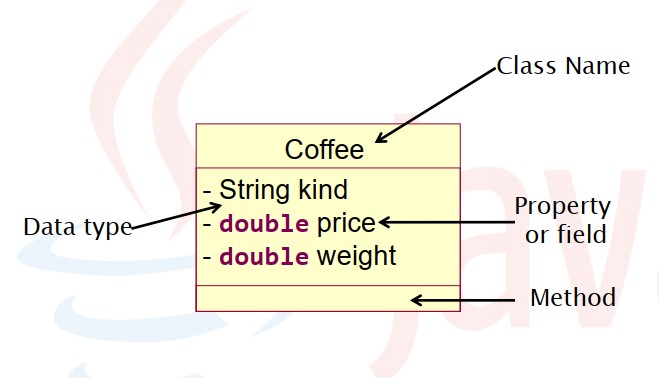


* Dùng để phân biệt thuộc tính với tham số!



**Các phạm vi**

****

**Class Diagram** ****

**Sử dụng đối tượng**

* Có thể sử dụng một đối tượng theo 2 cách

◦ Tham chiếu tới các thuộc tính

objectReference.variableName;

◦ Ví dụ: System.out.println("Chieu cao cua rectOne: "+

rectOne.height);

◦ Gọi các phương thức

objectReference.methodName(argumentList);

◦ Ví dụ: System.out.println(“Dien tich cua rectOne: " + rectOne.area());

**Truyền tham số cho phương thức**

**Truyền tham số cho phương thức**

* Java chỉ hỗ trợ truyền tham số kiểu tham trị (pass-by-value):

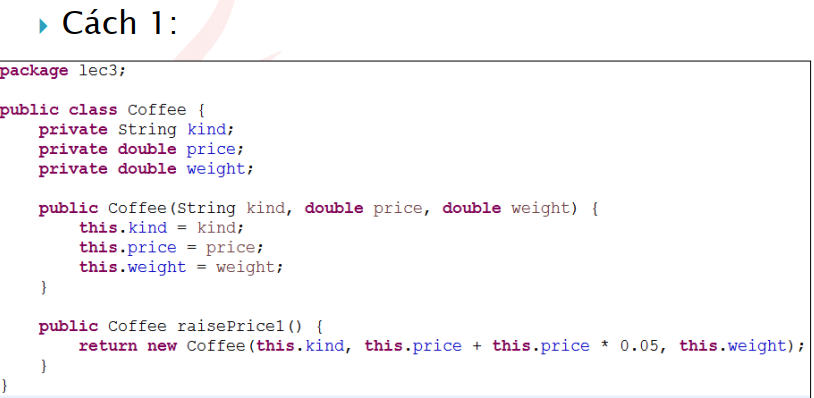
◦ Đối với kiểu dữ liệu nguyên thủy (primitive data);

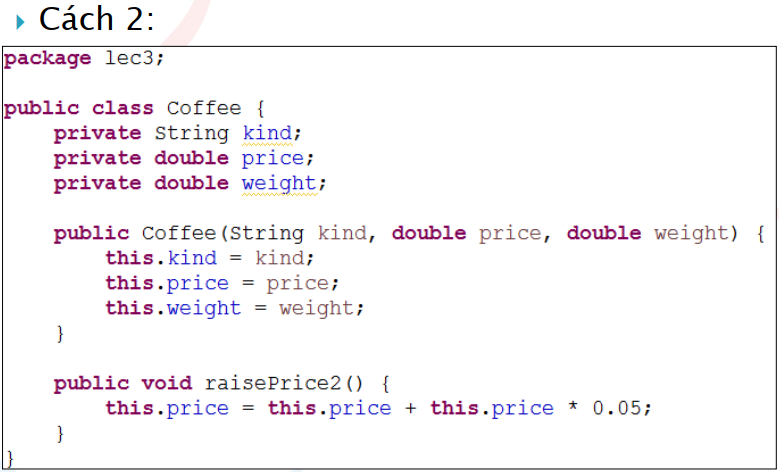
* Việc thay đổi giá trị của biến chỉ có ý nghĩa trong phương thức.
* Khi phương thức kết thúc → giá trị trở lại giá trị ban đầu

◦ Kiểu dữ liệu đối tượng (object):

* + - **Thay đổi giá trị thuộc tính của đối tượng** bên trong phương thức sẽ ảnh hưởng tới đối tượng dùng làm tham số.
    - Thay đổi bằng phép gán (=) chỉ có ý nghĩa trong phương thức.

**Phương thức Mutable vs Immutable**

****

****

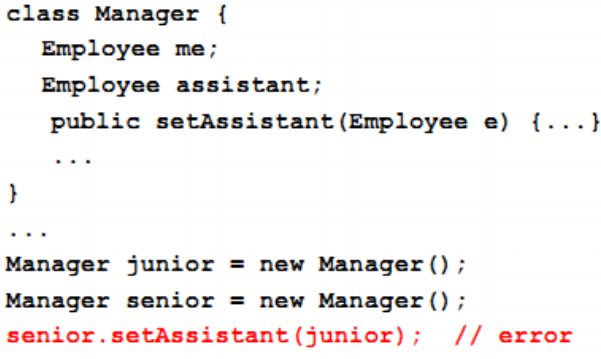
**Kế thừa (Inheritance)**

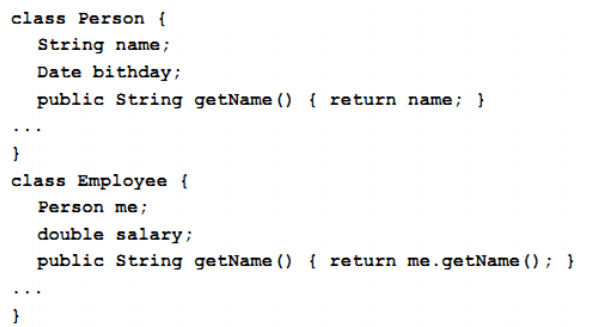
**Thuật ngữ**

* Kế thừa (inheritance)
* Lớp cơ sở (Base class), lớp cha
* Lớp dẫn xuất (Derived class), lớp con

**Kế thừa trong Java**

* Khả năng một lớp thừa hưởng data và code từ một hay nhiều lớp khác.
* Kỹ thuật giúp tái sử dụng code ➔ Tiết kiệm công sức lập trình, công sức kiểm tra code.
* Một lớp được xây dựng thừa kế từ một lớp khác gọi là lớp dẫn xuất (derived class) hay lớp con.
* Lớp dùng để xây dựng lớp dẫn xuất gọi là lớp cơ sở (base class) hay lớp cha

Ví dụ has\_a



* Dựa trên quan hệ is\_a
* Thừa hưởng lại các thuộc tính và phương thức đã có
* Chi tiết hóa cho phù hợp với mục đích sử dụng mới
  + Thêm các thuộc tính mới
  + Thêm hoặc hiệu chỉnh các phương thức

**Từ khoá super**

**Phạm vi truy xuất**