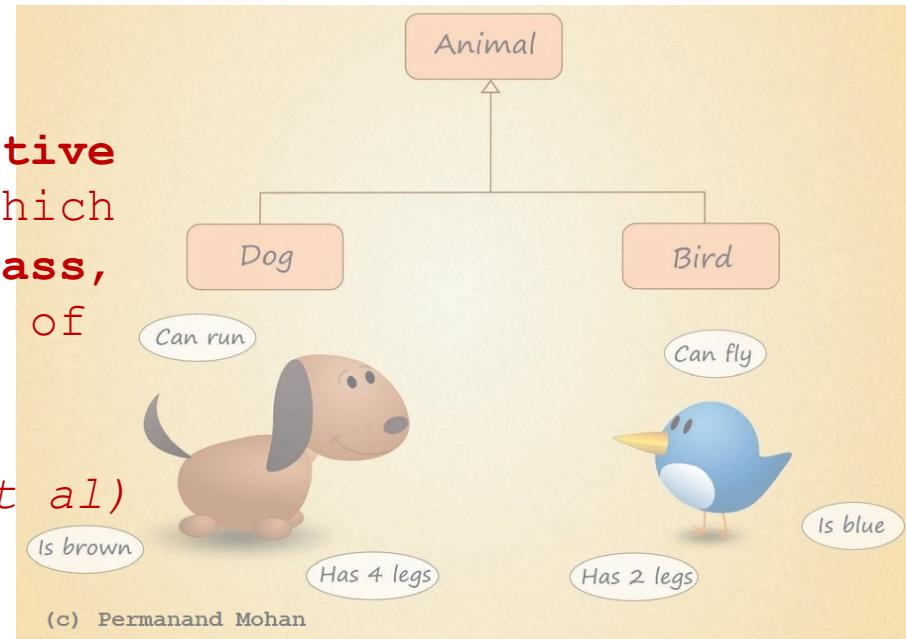


Object-oriented programming is a **method** of implementation in which programs are organized as **cooperative collections of objects**, each of which represents an **instance of some class**, and whose classes are all members of a hierarchy of classes united via inheritance relationships.

(*Grady Booch et al*)



## Chương 2

# LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

---

CT176 – LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

TS. Triệu Thanh Ngoan

tngoan@cit.ctu.edu.vn

Khoa MMT&TT, Trường CNTT-TT

# Nội dung

---

- ◆ 1. Lịch sử của Ngôn ngữ lập trình .....
- ◆ 2. Lập trình hướng đối tượng (OOP) .....
- ◆ 3. Các khái niệm quan trọng .....

# Ngôn ngữ máy & hợp ngữ

- Ngôn ngữ máy:

- Là các lệnh/chỉ thị của các bộ xử lý của máy tính.
- Là dãy các giá trị nhị phân 0, 1
- Không gần với ngôn ngữ của con người  
⇒ khó hiểu, khó nhớ!

- Hợp ngữ:

- Trừu tượng hóa cho ngôn ngữ máy nền tảng.
- Các lệnh máy dưới dạng các dãy số 0, 1 được ký hiệu bằng các chỉ thị gần với ngôn ngữ con người.

Trong giai đoạn này, máy tính được sử dụng chủ yếu để tính toán.

# Ngôn ngữ lập trình cấp cao

---

- Còn được gọi là ngôn ngữ ra lệnh:
  - Trừu tượng hóa cho hợp ngữ.
  - Vẫn đòi hỏi người lập trình suy nghĩ dưới dạng cấu trúc máy tính (do chưa đủ công cụ khái niệm để biểu diễn “thế giới thật” một cách gần gũi).  
⇒ Người lập trình phải thiết lập mối quan hệ giữa mô hình máy (trong không gian giải quyết vấn đề - máy tính) và mô hình của vấn đề (không gian của vấn đề - thế giới thật).

Trong giai đoạn này, máy tính bắt đầu được sử dụng để giải quyết nhiều vấn đề trong thế giới thật

# Ngôn ngữ lập trình HĐT

- Lập trình Hướng đối tượng:



- Cung cấp các công cụ (khái niệm) cho phép người lập trình mô hình hóa thế giới thật trong không gian giải quyết vấn đề một cách dễ dàng.



- Mô hình mà Lập trình hướng đối tượng chọn lựa là biểu diễn vấn đề trong không gian giải pháp như các “sự vật” hay “đối tượng” (object).



- Đây là một sự trừu tượng hóa mạnh mẽ và linh hoạt vì bản chất của thế giới là sự tương tác giữa các “sự vật”.



- ⇒ Nó cho phép mô tả vấn đề dưới dạng vấn đề, thay vì dưới dạng máy tính (nơi giải pháp sẽ chạy)

# Ngôn ngữ lập trình HĐT

- Ý tưởng chủ đạo của OOP là các **sự vật**:
  - Chương trình là một tập các sự vật **tương tác** lẫn nhau.
  - Sự vật trong OOP là **sự tái hiện** của các sự vật trong thế giới **thật**: Mỗi sự vật có những đặc tính (properties/characteristics) và khả năng (capacities) riêng.



# Lịch sử của OOP

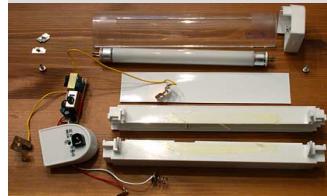
---

- OOP là phương pháp lập trình chính hiện nay:
  - Simula 1967, Smalltalk 1972
  - Giữa thập niên 90's, OOP mới bắt đầu được chú ý và sử dụng rộng rãi.
  - Hầu hết các ngôn ngữ lập trình hiện đại đều hướng đối tượng: **C++, Java, .NET, ...**

*(Dr. Alan Kay, cha đẻ của Smalltalk đã nhận được ACM Turing Award năm 2003)*

# Mở đầu

- Lập trình Hướng đối tượng (Object-Oriented Programming)



Cỗ đèn  
Hướng thủ tục



Hướng đối  
tượng

# Lập trình cổ điển vs. OOP

- Lập trình cổ điển:

## Chương trình

```
typedef struct {
    char *mssv;
    char *hoten;
    float diemTB;
    ...
} Sinhviен;

Sinhvién *dssv1, *dssv2;

void sapxep(Sinhvién *) { ... }
void luu(Sinhvién *, char *) { ... }
void nhap(Sinhvién *) { ... }

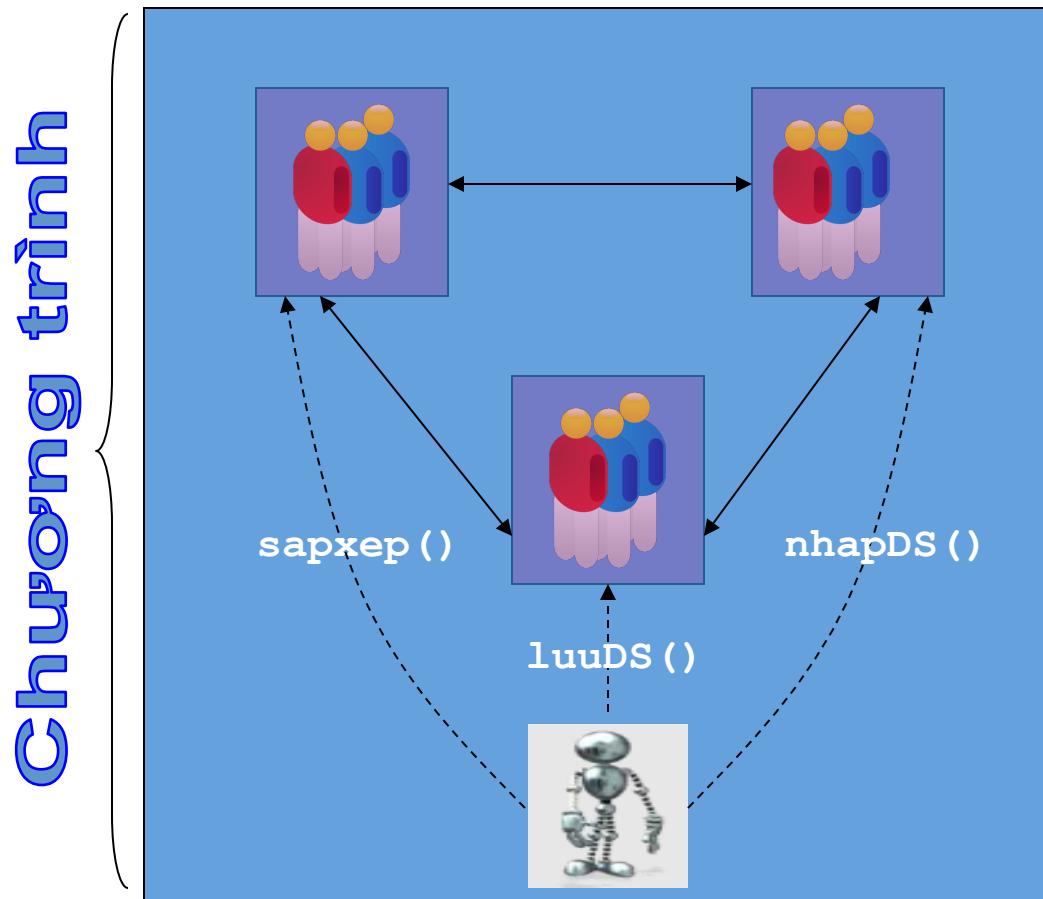
void main() { nhap(dssv1);
              sapxep (dssv2); ... }
```

Cấu trúc dữ liệu

Giải thuật

# Lập trình cổ điển vs. OOP

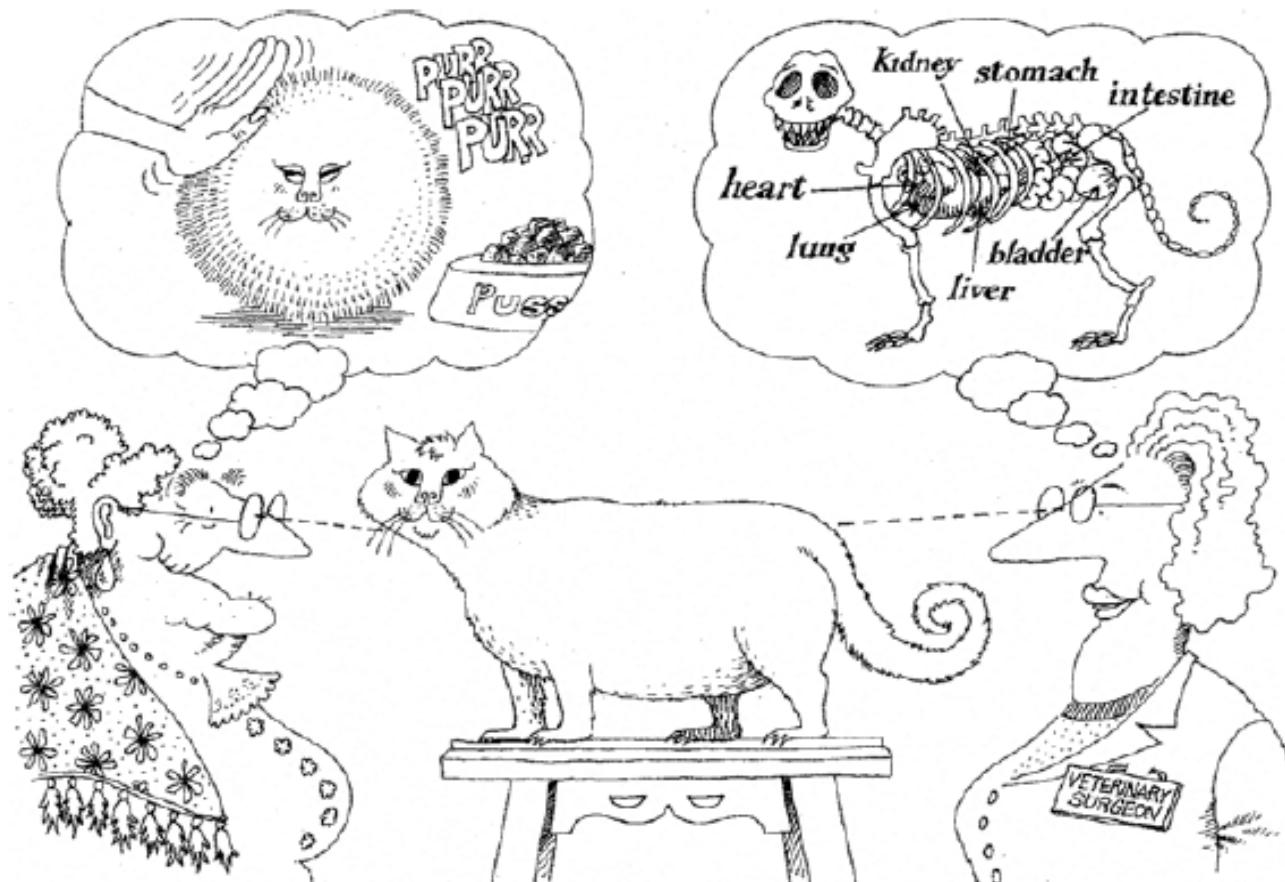
- Lập trình Hướng đối tượng:



**SINHVIEN**

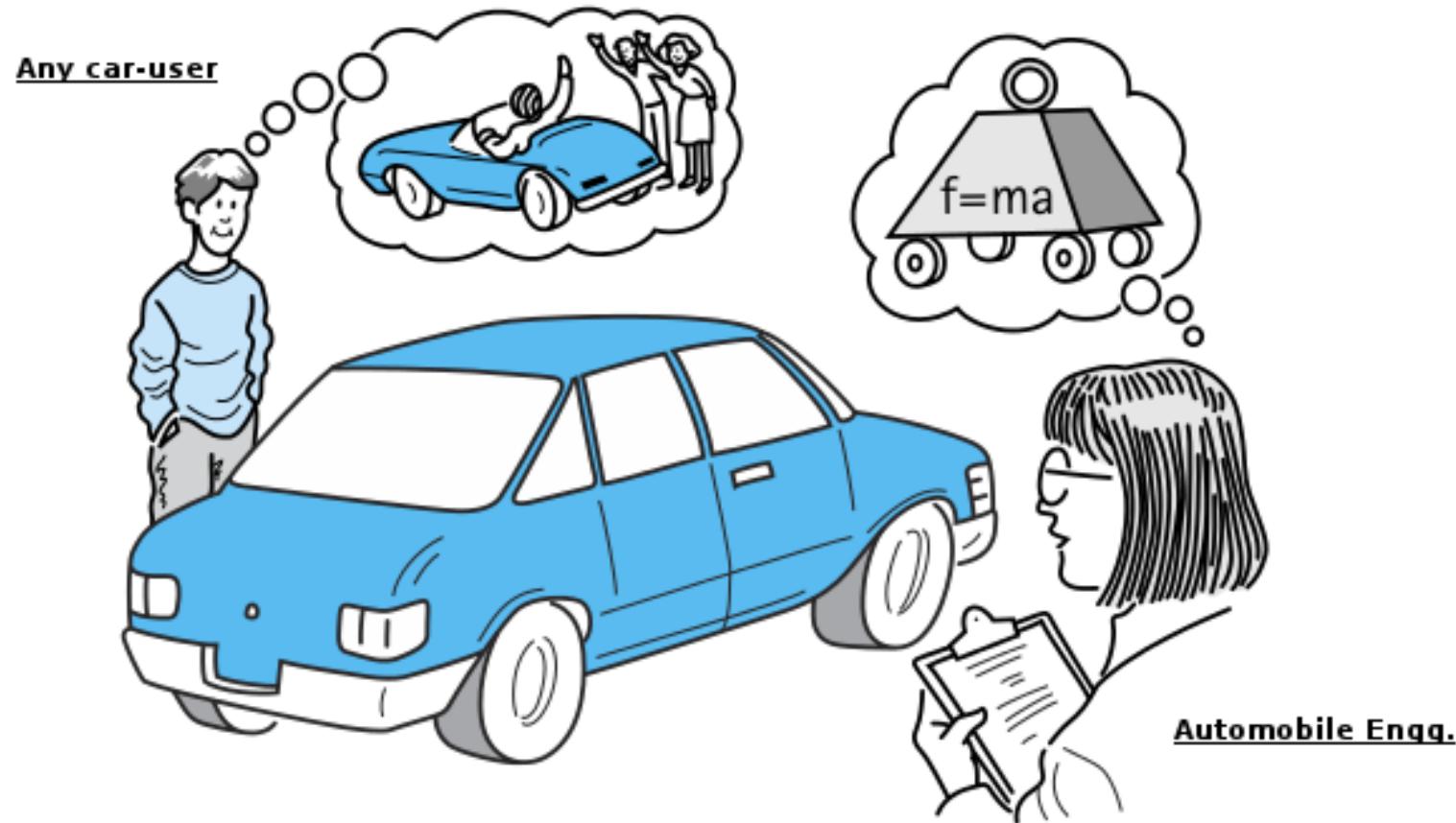


# Trừu tượng hóa (Abstraction)



Abstraction focuses on the essential characteristics of some object, relative to the perspective of the viewer.

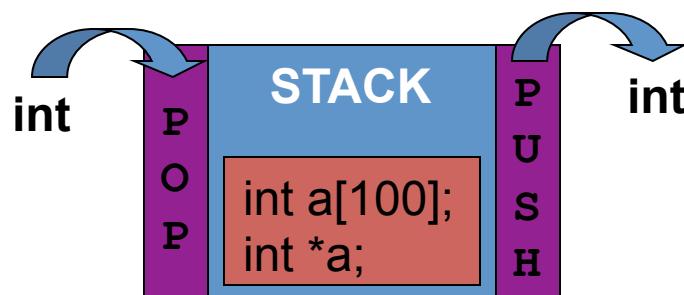
# Trừu tượng hóa (Abstraction)



*An abstraction includes the essential details relative to the perspective of the viewer*

# Tại sao phải OOP?

- Liên kết chặt chẽ giữa dữ liệu và thao tác của một đối tượng.  
⇒ Cho phép ta **tập trung** vào bản chất của vấn đề hơn là các chi tiết bên trong vấn đề.
- Các dữ liệu và thao tác được “bao gói” trong một đối tượng  
⇒ Có thể **che dấu** được những dữ liệu cần thiết, hoặc chỉ cho phép truy xuất thông qua các hàm.



# Đặc trưng của OOP

---

- Trong OOP, vấn đề (chương trình) được mô hình hóa như là **tập hợp của các đối tượng** hoạt động cộng tác với nhau:
  - Chương trình: tương tác của nhóm các đối tượng
  - Tương tác giữa các đối tượng: là việc gửi các thông điệp/yêu cầu cho nhau.

# Đặc trưng của OOP

- Trong OOP, các đối tượng có thể được “nhân hóa”.  
Ví dụ, một “cái cửa” có thể **tự** mở, một “menu” có thể **tự** “hiển thị”,...
- Tuy nhiên, các đối tượng phải được “yêu cầu” khi nào cần thực hiện thao tác gì bởi các đối tượng khác  
⇒ Các đối tượng trong chương trình phải “hợp tác” với nhau để giải quyết một vấn đề.



# Đặc trưng của OOP

---

1. Mọi thứ đều là **đối tượng**.
2. Chương trình là **một nhóm các đối tượng tương tác** với nhau bằng việc gửi các thông điệp cho nhau.
3. Mỗi đối tượng đều có **bộ nhớ riêng**, được tạo nên từ các đối tượng khác.
4. Mọi đối tượng đều **có kiểu** (lớp).
5. Tất cả các đối tượng cùng kiểu (lớp) đều có thể **nhận cùng thông báo**.

*(Thinking in Java)*

# Các khái niệm quan trọng

# Đối tượng (object)

---

- Đây là khái niệm quan trọng nhất trong OOP: Trong OOP, **mọi thứ đều là đối tượng**.
- Là một thực thể được sử dụng bởi máy tính, là “**cái mà ứng dụng muốn đề cập đến**”.
- Mô tả cho **một sự vật hoặc khái niệm trong thực tế**. Một đối tượng có thể là:
  - Một đối tượng thật (real object).
  - Một đối tượng khái niệm (conceptual object).
  - Một đối tượng phần mềm (software object).

# Đối tượng

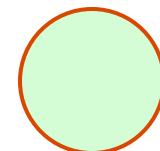
- Đối tượng thế giới thật: Là một đối tượng cụ thể mà ta **có thể sờ, nhìn thấy hay cảm nhận** được.
  - Ví dụ: cái đồng hồ, chiếc xe, con chó,...



- Đối tượng khái niệm: Đây thực sự là những khái niệm, những quá trình (process) trong thực tế được **trừu tượng hóa** thành các đối tượng.
  - Ví dụ: Ngày tháng (Date), chuỗi, hình tròn, ...

02-01-2008

Welcome to OOP



# Đối tượng (object)

---

- Mỗi đối tượng có hai thành phần:
  - Thuộc tính (property, attribute): mô tả các đặc điểm, trạng thái của đối tượng.
  - Hành vi (behavior, method): mô tả các thao tác, các hoạt động mà đối tượng có thể thực hiện (thể hiện “**khả năng**”, “**chức năng**” của một đối tượng)
- Ngoài ra, một đối tượng còn có một **định danh** (object identifier) dùng để phân biệt giữa các đối tượng.

# Đối tượng (Object)

- Ví dụ Đối tượng phần mềm

Định danh: mycar

Thuộc tính:

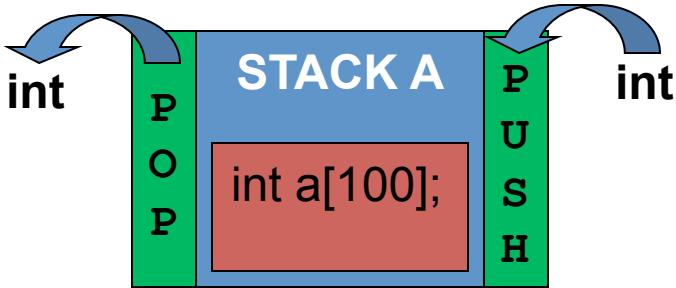
- Model: Mazda 2 Maxx
- Màu: đỏ
- Giá: \$17600



Phương thức:

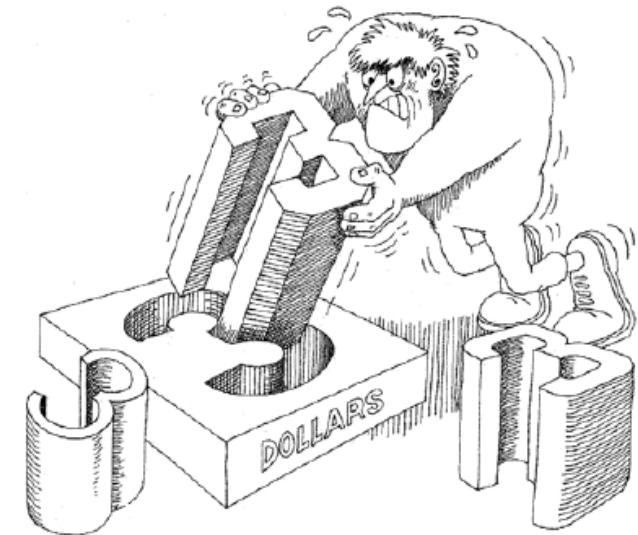
- Dừng
- Chạy
- Tăng tốc

# Đối tượng (object)

Sự vật	Thuộc tính	Hành vi	Ví dụ
Con chó	Tên: Mino Màu sắc: Xám Giống: Nhật Trạng thái: Vui vẻ	Sửa Ăn Chạy Cǎn	
Stack A	List  Empty: NO.	PUSH POP	
Bóng đèn	Nhãn hiệu: ABC Màu: Vàng Trạng thái: Mở Loại: Dây tóc	Bật Tắt Sáng Mờ	

# Lớp (class)

- Còn được gọi là **loại/kiểu** của đối tượng.
- Lớp là một **khuôn mẫu** để tạo ra các đối tượng cùng kiểu.
- Lớp định nghĩa các **thuộc tính** và **phương thức** (hành vi) chung cho tất cả các đối tượng cùng lớp.
- Một đối tượng được gọi là một **thể hiện** (instance) của một lớp và giá trị thuộc tính của chúng có thể khác nhau.



Mỗi lớp xác định một thực thể, trong khi đó mỗi đối tượng là một thể hiện thực sự.

# Lớp (class)

- Thể hiện (instance):

- Các thuộc tính được xác định bằng giá trị cụ thể
- Một đối tượng cụ thể được gọi là một thể hiện



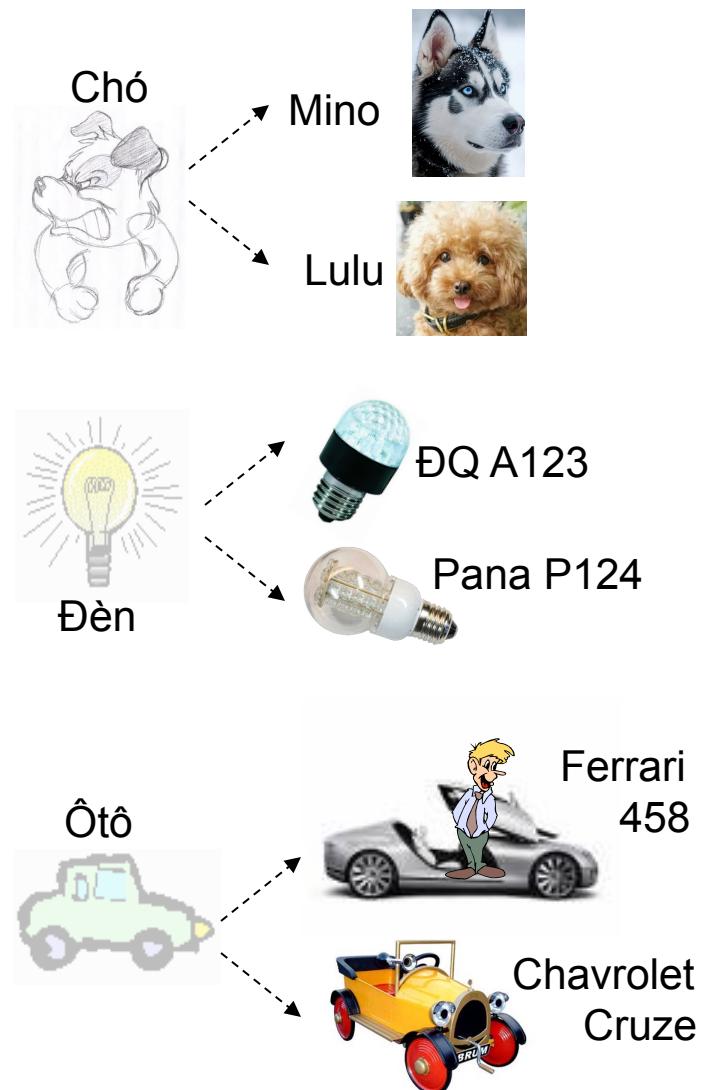
Thuộc tính:  
• Model  
• Màu  
• Giá



Thuộc tính:  
• Model: Maxx sport  
• Màu: đỏ  
• Giá: \$20000

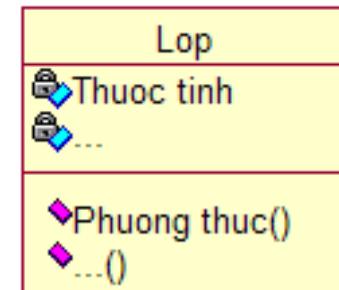
# Lớp (class)

Lớp	Thuộc tính	Hành vi
Chó	Tên Màu sắc Giống Trạng thái	Sửa Ăn Chạy Cắn
Bóng đèn	Nhãn hiệu Màu Trạng thái Loại	Bật Tắt Sáng Mờ
Xe ôtô	Nhãn hiệu Màu sắc Giá Hộp số Tốc độ ...	Chạy Dừng Tăng tốc Giảm tốc ...



# Thuộc tính và phương thức

- Thuộc tính: Mô tả **trạng thái** của một đối tượng  
⇒ Bao gồm: tên và kiểu, ...
- Phương thức: Thể hiện cho **khả năng** của một đối tượng có thể **thực hiện được những hành vi gì?**  
⇒ Bao gồm: chạy, in, nhập, ...



Lớp	Thuộc tính	Hành vi
Xe ôtô	Nhãn hiệu: String Màu sắc: String Giá: double Hộp số: int Tốc độ: int ...	Chạy {...} Dừng {tốc độ = 0; ...} Tăng tốc(km/h) {tốc độ += km/h; ...} Giảm tốc(km/h) {tốc độ -= km/h; ...} ...

Xe Oto
Nhanhieu : String Mau sac : String Gia : Float Hop so : Integer  Chay() Dung() Tang toc() Giam toc()

# Hàm và việc truyền thông điệp

- Trong một chương trình OOP, các đối tượng hoạt động **cộng tác** với nhau thông qua việc **truyền thông điệp** cho nhau.
- Thông điệp (message): là một **yêu cầu** thực hiện một thao tác, hoạt động. Gồm có:
  - Tên thông điệp (tên của phương thức).
  - Các tham số của thông điệp (tham số của phương thức)
- Truyền thông điệp: gửi thông điệp đến đối tượng được yêu cầu. Bao gồm:
  - Đối tượng cần nhận thông điệp.
  - Thông điệp



# Hàm và việc truyền thông điệp

- Việc thực hiện phương thức của một đối tượng **chỉ ảnh hưởng đến dữ liệu của chính đối tượng đó chứ không ảnh hưởng đến các đối tượng khác trong cùng lớp**



## Class

Definition of objects that share structure, properties and behaviours



Building  
*class*



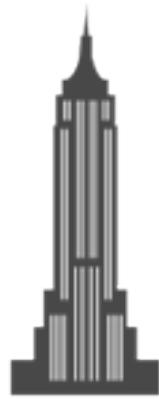
Dog  
*class*



Computer  
*class*

## Instance

Concrete object, created from a certain class.



Empire State  
*instance of Building*



Lassie  
*instance of Dog*



Your computer  
*instance of Computer*

- ❖ Đối tượng và lớp

# Đối tượng và lớp

# Đối tượng và lớp (object & class)

---

- Đối tượng:
  - Đối tượng là một thực thể phần mềm, hay là sự mô hình hóa của một sự vật hay khái niệm trong thực tế
  - Một đối tượng **tồn tại trong bộ nhớ** của chương trình, thực hiện 1 **tác vụ** hay **chức năng** nào đó
  - Đối tượng cũng là 1 **biến** (variable) trong chương trình

# Đối tượng và lớp (object & class)

- Biến kiểu dữ liệu cơ bản:
  - Chứa dữ liệu
  - Tác động lên dữ liệu: gọi các hàm
- Biến đối tượng:
  - Chứa dữ liệu + các thao tác (phương thức) trên dữ liệu
  - Tác động lên dữ liệu của đối tượng: gọi các phương thức của chính đối tượng

```
int i = 5;  
  
Math.sqrt(i);
```

```
String s = new String("Hello  
World");  
  
s = s.replace("u", "o");  
  
int len = s.length();
```

# Đối tượng và lớp (object & class)

---

- Lớp:

- Là một **đặc tả** (specification) của một kiểu dữ liệu mới
- Mô tả các thành phần của kiểu dữ liệu đó:
  - Các **thuộc tính** mà một đối tượng thuộc lớp đó có thể lưu trữ
  - Các **phương thức** của các đối tượng thuộc lớp đó
- Ví dụ: String, Scanner là các lớp được định nghĩa sẵn của Java
  - Các đối tượng thuộc lớp String có thể chứa các chuỗi ký tự  
`(String s = "Hello World";)` và các thao tác trên chuỗi ký tự `(s.toLowerCase();`  
`s.toUpperCase();)`

# Tạo lớp căn bản

---

- Một lớp có thể có một trong các khả năng sau:
  - Hoặc chỉ có thuộc tính, không có phương thức.
  - Hoặc chỉ có phương thức, không có thuộc tính.
  - Hoặc có cả thuộc tính và phương thức, trường hợp này là phổ biến nhất.
  - Đặc biệt, lớp không có thuộc tính và phương thức nào là các lớp trừu tượng. Các lớp này không có đối tượng tương ứng.

# Tạo lớp căn bản

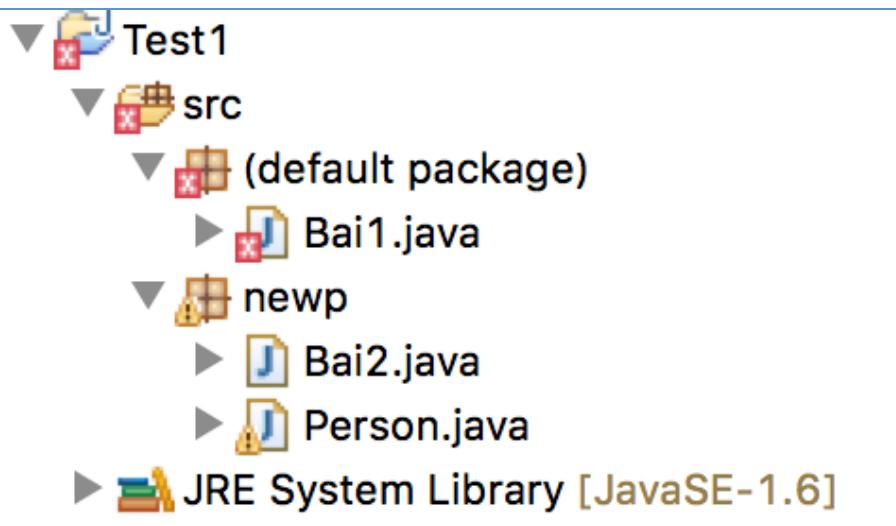
- Cú pháp tạo lớp:

```
[phạm vi truy cập] class <tên lớp> {  
    //các thành phần của Lớp: thuộc tính + phương thức  
}
```

- Phạm vi truy cập: public, default(no access modifier).
- Trong một tập tin có thể định nghĩa nhiều lớp, tuy nhiên chỉ được có nhiều nhất một lớp có phạm vi truy cập là public
- Tên lớp: đặt theo qui tắc đặt tên định danh. Thông thường đặt theo qui tắc title case (chữ cái đầu tiên của mỗi từ được viết hoa)

Hello.java => public class Hello{}

# Tạo lớp căn bản



17 class SinhVien{  
18 private int MSSV;  
19 private String lop;  
20  
21 public SinhVien(){  
22 MSSV = 12345;  
23 lop = "DI1234";  
24 }  
25  
26 public void inThongTin(){  
27 System.out.println(MSSV);  
28 }  
29 }  
30

1 package newp;  
2  
3 public class Bai2 {  
4 public static void main(String [] args ){  
5 Person a = new Person();  
6 a.inThongTin();  
7 System.out.println("Hello "+a.tuoi);  
8 SinhVien b = new SinhVien();  
9 b.inThongTin();  
10 System.out.println("Hello "+b.tuoi);  
11 }  
12 }  
13 }  
14  
15

```
3 import newp.Person;
4 import newp.SinhVien;
5
6
7 public class Bai1 {
8     public static void main(String [] args ){
9         Person p = new Person();
10        p.inThongTin();
11        System.out.println("Hello "+p.tuoi);
12        System.out.println("Hello "+p.tuoi);
13        SinhVien b = new SinhVien();
14        b.inThongTin();
15 }
```

- Cấu trúc khai báo lớp chung như sau:

```
<access modifier> class classname {  
    // declare instance variables  
    <access modifier> type var1;  
    <access modifier> type var2;  
    // declare methods  
    <access modifier> type method1 (parameters) {  
        // body of method  
    }  
    <access modifier> type method2 (parameters) {  
        // body of method  
    }  
}
```

# Tạo lớp căn bản

- Ví dụ tạo ra lớp **Vehicle** để mô tả thông tin chung của các đối tượng như xe hơi, xe khách...
- Lớp này bao gồm 3 thông tin: số người mà xe có thể chở, dung tích bình chứa nhiên liệu, mức tiêu hao nhiên liệu bình quân.

```
public class Vehicle {  
    private int passengers; // number of passengers  
    private int fuelcap; // fuel capacity in gallons  
    private int mpg; // fuel consumption in miles per gallon  
}
```

- Định nghĩa một lớp là việc tạo ra một kiểu dữ liệu mới. Trong trường hợp này kiểu dữ liệu là **Vehicle**.

# Thuộc tính của lớp

- **Thuộc tính:** tương tự cú pháp khai báo biến

[phạm vi truy cập] <kiểu dữ liệu> <tên thuộc tính>;

- Phạm vi truy cập: xác định thuộc tính có thể được truy cập từ đâu, chỉ bên trong lớp hay có thể từ bên ngoài lớp,...
  - **public:** thuộc tính có thể được truy cập từ bất kỳ phạm vi nào
  - **private:** thuộc tính chỉ có thể được truy cập từ bên trong lớp
  - **protected:** thuộc tính có thể được truy cập từ trong package hoặc lớp con của nó ở package khác
  - **no modifier:** thuộc tính có thể được truy cập từ trong package
- Kiểu dữ liệu: có thể là một kiểu dữ liệu nguyên thủy hay lớp.
- Tên thuộc tính: theo qui tắc và qui ước đặt tên biến
- **Lưu ý:** **các thuộc tính thường được khai báo là private** và việc truy xuất các thuộc tính sẽ thông qua các phương thức

# Phương thức

# Phương thức

---

- Phương thức trong java là một nhóm các câu lệnh được đặt tên và có thể gọi để thực thi đơn giản bằng cách gọi tên đó.
  - Ví dụ: `System.out.println ("Hello World!");`
- Chúng ta có thể viết và gọi phương thức của riêng mình.  
Ví dụ: `public static void main ( String args[] ) { }`
- Phương thức được khai báo **public** có thể được gọi từ bất cứ nơi đâu trong chương trình.
- **static**: phương thức chỉ có một thể hiện trên một lớp. Mặc nhiên phương thức là không phải static
- **void**: kiểu dữ liệu trả về của phương thức
- **main**: tên phương thức, `args` là đối số truyền vào

# Phương thức của lớp

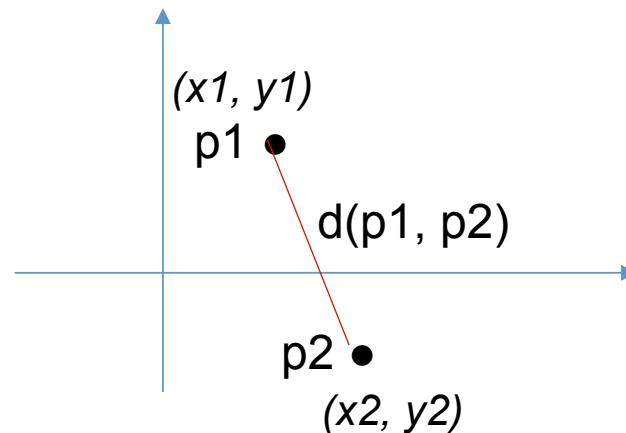
- **Phương thức:** tương tự cú pháp tạo hàm

```
[access modifier] <return type> <methodname> ([parameters]) {  
    //thân phương thức (hàm)  
}  
                                prototype (nguyên mẫu/khai báo)
```

- Thuộc tính truy cập (access modifier): qui định phương thức có thể được truy cập từ đâu (tương tự như thuộc tính)
- Kiểu dữ liệu trả về (return type): kiểu dữ liệu của giá trị trả về của hàm, `void` nếu hàm không trả giá trị về
- Danh sách tham số (parameters): các dữ liệu được truyền vào cho phương thức
- Thân hàm: các lệnh thực hiện tác vụ của hàm

# Lớp Diem2D

- Ví dụ 1: Tạo một lớp điểm trong không gian hai chiều (2D)
  - Dữ liệu: mỗi điểm gồm tọa độ x, y
  - Phương thức: thiết đặt giá trị x, y; hiển thị tọa độ ra màn hình; lấy giá trị của x, lấy giá trị của y, nhập giá trị cho x, y; tính khoảng cách đến 1 điểm khác;...



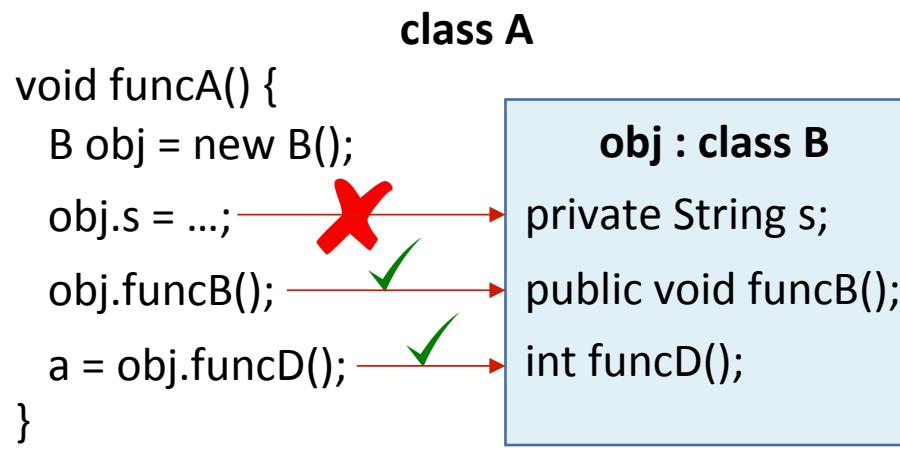
Diem2D
<pre>int x int y</pre>
<pre>void nhap() void hienthi() void ganXY(int, int) void ganX(int x1) void ganY(int y1) int layX() int layY() float khoangcach(Diem2D) ...</pre>

# Lớp Diem2D

```
public class Diem2D {  
    private int x, y; //thuộc tính x, y  
  
    public void ganXY(int h, int t) {  
        /* thân hàm */  
    }  
  
    public void hienthi() {  
        /* thân hàm */  
    }  
  
    public float khoangcach(Diem2D d) {  
        /* thân hàm */  
    }  
  
    public static void main( String [] args){  
    }  
}
```

# Phạm vi truy cập

- Chỉ định phạm vi truy cập tới các phương thức:
  - public: có thể truy cập từ bên trong lớp lẫn bên ngoài lớp
  - private: chỉ có thể được truy cập từ bên trong lớp
  - protected: có thể được truy cập từ bên trong lớp, trong cùng package và các lớp con thừa kế lớp
  - no modifier(không khai báo): trong cùng lớp hoặc trong cùng package



	class	package	sub-class	world
public	x	x	x	x
protected	x	x	x	
no acc	x	x		
private	x			

# Định nghĩa phương thức

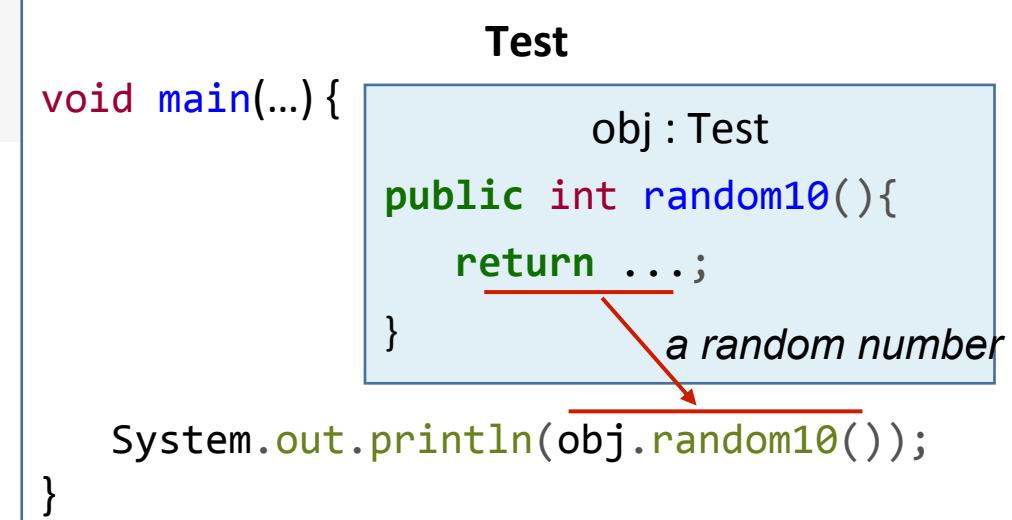
---

- **Kiểu dữ liệu trả về (return type):**
  - Phương thức: thực hiện một tác vụ (task) nào đó  
⇒ có thể trả về **1 giá trị**
  - Lệnh `return` được dùng để trả về 1 giá trị cho lời gọi hàm
  - Hàm có thể trả về 1 giá trị có kiểu dữ liệu nguyên thủy (**int**, **float**, **boolean**...) hoặc là trả về một đối tượng
  - Ví dụ:
    - Hàm `Math.sqrt()` tính căn bậc 2 của của 1 số, trả về 1 số thực  
⇒ kiểu dữ liệu trả về là `double`: **double sqrt(...);**
    - Hàm `System.out.println()` hiển thị 1 chuỗi ra màn hình  
⇒ không có giá trị trả về: **void println(...);**

# Định nghĩa phương thức

```
class Test {  
    public static void main(String []args) {  
        Test obj = new Test();  
        System.out.println(obj.random10());  
    }  
  
    //generate a random value in range of [0, 10)  
    public int random10() {  
        int r = (int)(Math.random() * 10);  
        return r;  
    }  
}
```

- Cú pháp lệnh return:  
**return <biểu thức>;**



# Định nghĩa phương thức

---

- **Tham số của hàm (parameters):**

- Một số hàm cần phải được cung cấp dữ liệu để thực hiện
- **Tham số:** các dữ liệu truyền vào cho hàm
  - Cú pháp khai báo tham số: <kiểu dữ liệu> <tên đối số>
  - Các đối số cách nhau bằng dấu phẩy
- Về bản chất, **tham số chính là biến cục bộ** của phương thức
- Ví dụ:
  - **double sqrt(double n):** hàm tính căn bậc hai, nhận vào một số thực cần tính căn bậc hai
  - **void PTB1(int a, int b):** hàm giải phương trình bậc 1, nhận 2 tham số kiểu int, là hai hệ số a và b

# Định nghĩa phương thức

- **Tham số của hàm (tt):**

- Tham số hình thức: là các tham số trong định nghĩa hay khai báo hàm
- Tham số thực tế: các biến, biểu thức được truyền vào trong lời gọi hàm

*tham số hình thức*

```
class Math {  
    public static double sqrt(double n) {  
        //định nghĩa hàm  
    }  
}
```

```
public class Test {  
    public static void main(String args[]) {  
        double a = 2.0, b;  
        b = Math.sqrt(a);  
    }  
}
```

*tham số thực tế*

# Phương thức của lớp

---

- Truyền tham số cho phương thức:

```
class ChkNum {  
    public boolean isEven(int x) {  
        if ( (x%2) == 0 )  
            return true;  
        else  
            return false;  
    }  
}  
  
public class ParmDemo {  
    public static void main ( String args[] ) {  
        ChkNum e = new ChkNum();  
        if ( e.isEven(10) ) System.out.println ("10 is even.");  
        if ( e.isEven(9) ) System.out.println ("9 is even.");  
        if ( e.isEven(8) ) System.out.println ("8 is even.");  
    }  
}
```

# Phương thức của lớp

---

- Java có hỗ trợ phương thức đệ quy.
- Ví dụ:  $n!$  được định nghĩa là  $n \times (n-1) \times (n-2) \times \dots \times 2 \times 1$

Như vậy  $n! = n \times (n-1)!$

```
public static long factorial (int n) {  
    if (n < 0) {  
        return -1;  
    }  
    else if (n == 0) {  
        return 1;  
    }  
    else {  
        return n*factorial(n-1);  
    }  
}
```

# Khai báo & tạo đối tượng

---

- Khai báo đối tượng: <tên lớp> <tên đối tượng>;
- Đối tượng cũng là biến  $\Rightarrow$  cú pháp giống khai báo biến, trong đó tên lớp đóng vai trò kiểu dữ liệu.
- Tạo đối tượng: dùng toán tử new: new <tên lớp>;
- Để tạo ra đối tượng thật sự cần khai báo như sau:  
`classname objectname = new classname();`
- Ví dụ tạo đối tượng tên xe thuộc lớp Vehicle.  
`Vehicle xe = new Vehicle();`

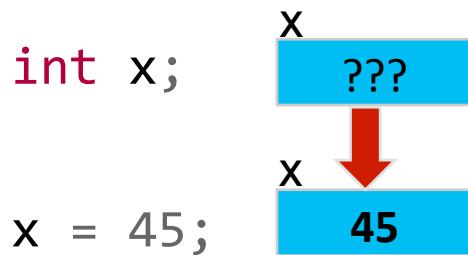
# Khai báo và tạo đối tượng

---

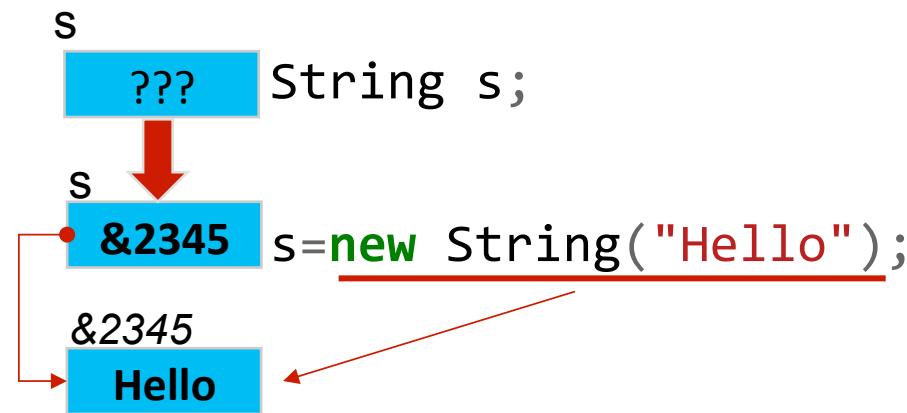
- Khi tạo ra đối tượng thì đối tượng cũng sẽ có một bản sao các thuộc tính của lớp (passengers, fuelcap, mpg).
- Một biến đối tượng được gọi là **biến tham chiếu** (reference variable):
  - Bản thân biến **không chứa dữ liệu**, chỉ là một **tham chiếu** đến đối tượng
  - Hay nói cách khác, biến tham chiếu chứa “**địa chỉ**” của đối tượng mà nó tham chiếu đến
  - Muốn cho biến tham chiếu đến đối tượng, dùng phép gán =

# Khai báo và tạo đối tượng

- **Biến kiểu nguyên thủy** (*primitive datatype variable*)
  - Lưu trực tiếp dữ liệu vào vùng nó được cấp phát



- **Biến tham chiếu** (*reference variable*)
  - Lưu địa chỉ của đối tượng mà nó tham chiếu đến



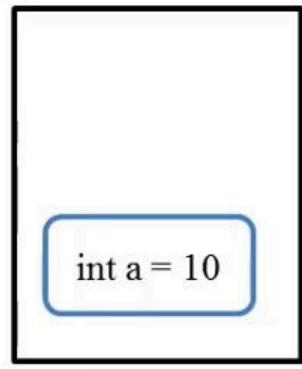
Có thể khai báo ngắn gọn:

`String s = new String("Hello");`

# Khai báo và tạo đối tượng

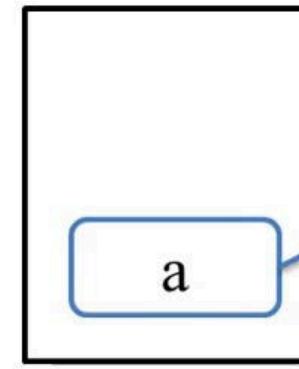
## Stack and Heap

```
int a = 10; // local variable
```

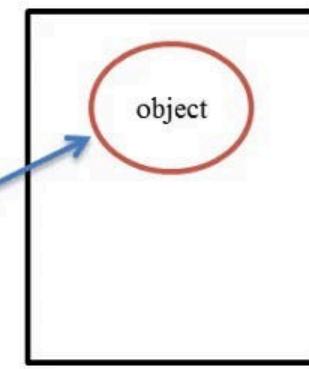


Stack

```
Test a = new Test();
```



Stack



Heap

# Truy xuất các thành phần của đối tượng

---

- Các thành phần của một đối tượng được truy xuất thông qua biến tham chiếu đến nó
- Cú pháp: <biến tham chiếu>.<tên thành phần>
- Ví dụ:  
`s.replace("u", "o");`  
`s.toUpperCase();`
- Lưu ý:
  - Từ bên ngoài lớp, chỉ được truy xuất đến các thành phần **public**, **protected** hoặc **default** (không khai báo thuộc tính truy cập)
  - Trong cùng một lớp, 1 phương thức có thể truy xuất tất cả các thành phần khác

# Khai báo và tạo đối tượng

```
class Vehicle {  
    int passengers; // number of passengers  
    int fuelcap; // fuel capacity in gallons  
    int mpg; // fuel consumption in miles per gallon  
}  
  
public class VehicleDemo {  
    public static void main (String []arg) {  
        Vehicle minivan = new Vehicle();  
        minivan.passengers = 7; // 7 people  
        minivan.fuelcap = 16; // 16 gallon full tank  
        minivan.mpg = 21; // 21 miles per gallon  
        // compute the range assuming a full tank of gas  
        int range = minivan.fuelcap * minivan.mpg;  
        System.out.println ("Minivan carry " +  
            minivan.passengers + " with " + range);  
    }  
}
```

- ❖ Khởi tạo và hủy đối tượng

# Khởi tạo và hủy đối tượng

# Hàm xây dựng (constructor)

---

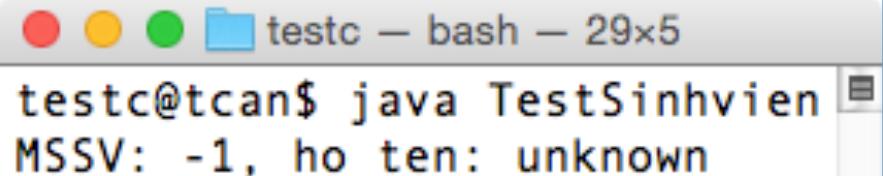
- Là một phương thức đặc biệt của lớp, dùng để khởi tạo đối tượng (khởi tạo các dữ liệu thành viên)
- Có 2 loại:
  - Hàm khởi tạo không có tham số, còn được gọi là hàm xây dựng mặc nhiên (default constructor)
  - Các hàm khởi tạo có tham số
- Cú pháp:
  - **Phạm vi truy xuất:** public
  - **Tên hàm:** trùng với tên lớp
  - **Kiểu dữ liệu trả về:** không có, kể cả void

# Hàm xây dựng (constructor)

- Hàm xây dựng mặc định được cung cấp khi chưa khai báo hàm xây dựng
- Khi đã khai báo hàm xây dựng, thì hàm xây dựng mặc định sẽ không thể dùng được nữa

```
class Sinhvien {  
    private int mssv;  
    private String hoten;  
  
    public Sinhvien() {  
        mssv = -1;  
        hoten = "unknown";  
    }  
  
    public void hienthi() {  
        System.out.println("MSSV: " + mssv +  
                           ", ho ten: " + hoten);  
    }  
}
```

```
class Test {  
    public static void main(String args[]) {  
        Sinhvien sv1 = new Sinhvien();  
        sv1.hienthi();  
    }  
}
```



A screenshot of a terminal window titled "testc – bash – 29x5". The command "java TestSinhvien" is run, and the output shows the constructor's default values: "MSSV: -1, ho ten: unknown".

```
testc@tcan$ java TestSinhvien  
MSSV: -1, ho ten: unknown
```

# Hàm xây dựng (constructor)

---

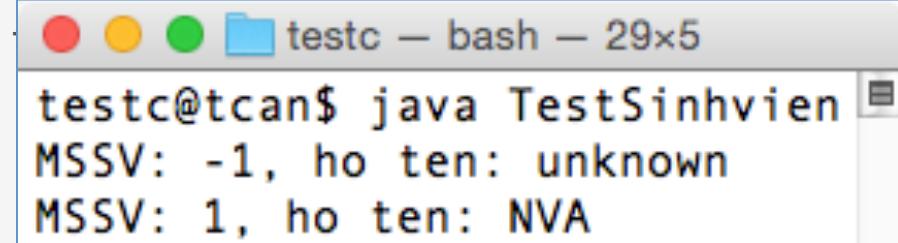
- Một số tính chất:
  - Hàm xây dựng sẽ **tự động được gọi** khi đối tượng được tạo ra
  - Hàm xây dựng cũng là 1 phương thức của lớp nên nó có thể **được tái định nghĩa**
  - Trong trường hợp có nhiều hàm xây dựng, hàm nào được gọi là tùy vào danh sách tham số truyền vào trong câu lệnh tạo đối tượng
  - Lớp **có bao nhiêu hàm xây dựng, có bấy nhiêu cách để tạo đối tượng**
  - Hàm xây dựng không thể được gọi trực tiếp

- ❖ Khởi tạo và hủy đối tượng

# Hàm xây dựng (constructor)

```
class Sinhvien {  
    private int mssv;  
    private String hoten;  
  
    public Sinhvien() {  
        mssv = -1;  
        hoten = "unknown";  
    }  
  
    public Sinhvien(int ms, String ht) {  
        mssv = ms;  
        hoten = ht;  
    }  
  
    public void hienthi() {  
        System.out.println("MSSV: " + mssv +  
                           ", ho ten: " + hoten);  
    }  
}
```

```
public class TestSinhvien {  
    public static void main(String args[]) {  
        Sinhvien sv1 = new Sinhvien();  
        Sinhvien sv2 = new Sinhvien(1, "NVA");  
        sv1.hienthi();  
        sv2.hienthi();  
    }  
}
```



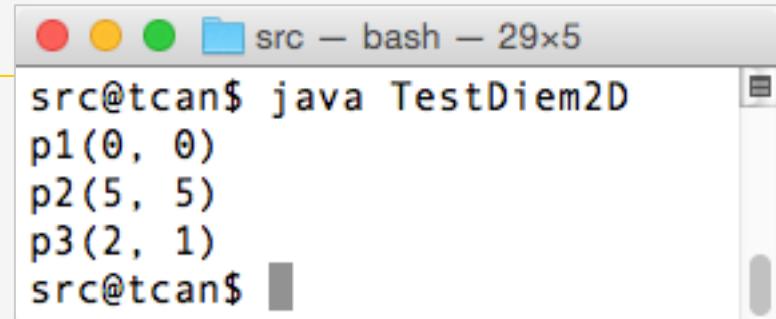
A terminal window titled 'testc – bash – 29x5' showing the output of a Java program. The command 'java TestSinhvien' is run, followed by two lines of output: 'MSSV: -1, ho ten: unknown' and 'MSSV: 1, ho ten: NVA'.

```
testc@tcan$ java TestSinhvien  
MSSV: -1, ho ten: unknown  
MSSV: 1, ho ten: NVA
```

# Hàm xây dựng (constructor)

```
class Diem2D {  
    private int x, y;  
  
    public Diem2D() {  
        x = y = 0;  
    }  
  
    public Diem2D(int th) {  
        x = y = th;  
    }  
  
    public Diem2D(int h, int t) {  
        x = h; y = t;  
    }  
  
    public void hienthi(String s) {  
        System.out.println(s + "(" + x + ", " + y + ")");  
    }  
}
```

```
class TestDiem2D {  
    public static void main(String args[]) {  
        Diem2D p1 = new Diem2D();  
        Diem2D p2 = new Diem2D(5);  
        Diem2D p3 = new Diem2D(2, 1);  
  
        p1.hienthi("p1");  
        p2.hienthi("p2");  
        p3.hienthi("p3");  
    }  
}
```



```
src@tcan$ java TestDiem2D  
p1(0, 0)  
p2(5, 5)  
p3(2, 1)  
src@tcan$
```

# Hàm xây dựng (constructor)

```
class Clock {  
    private int h, m, s;  
  
    public Clock() {  
        h = m = s = 0;  
    }  
}
```

```
public Clock(int g, int p, int gi) {  
    h = g; m = p; s = gi;  
}  
  
public void setTime(int g, int p, int gi) {  
    h = g; m = p; s = gi;  
}  
  
public void display() {  
    System.out.println(h + ":" + m + ":" + s);  
}  
}
```

```
public class TestClock {  
    public static void main(String args[]) {  
        Clock c1 = new Clock();  
        Clock c2 = new Clock(5, 10, 20);  
        c1.display();  
        c2.display();  
    }  
}
```

```
src@tcan$ java TestClock  
0:0:0  
5:10:20  
src@tcan$
```

# Hàm xây dựng sao chép (copy constructor)

- Là hàm xây dựng với tham số là một đối tượng cùng lớp
- Được sử dụng để tạo 1 đối tượng mới “giống” với 1 đối tượng đã có sẵn
- Cú pháp: `public Classname(Classname obj);`
  - Đây cũng là hàm xây dựng nên cú pháp cũng theo qui tắc hàm xây dựng: tên hàm trùng tên lớp, không có giá trị trả về
  - Hàm luôn có 1 đối số, chính là 1 đối tượng cùng lớp

```
//hàm xây dựng sao chép của Lớp Clock
public Clock(Clock c) {
    h = c.h;
    m = c.m;
    s = c.s;
}
```

```
Clock c1 = new Clock(5, 10, 20);
Clock c2 = new Clock(c1);
```

# Phép gán đối tượng (object assignment)

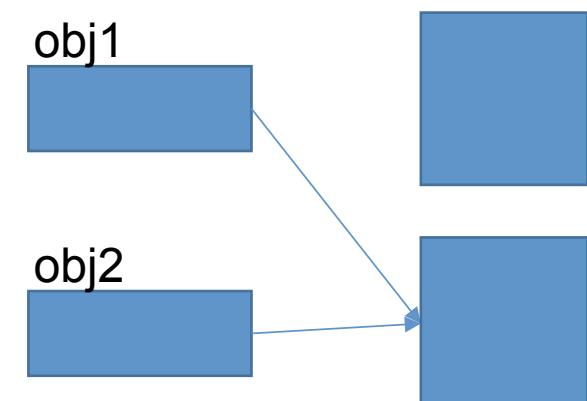
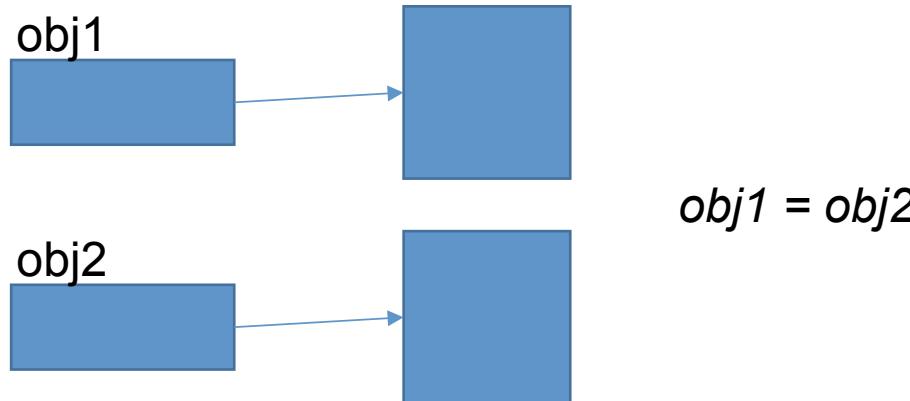
- Phép gán đối tượng còn được gọi là phép sao chép cạn (shadow copy):

- Cái được sao chép chính là tham chiếu: phép gán

$$\text{obj1} = \text{obj2}$$

làm cho obj1 và obj2 tham chiếu đến cùng 1 đối tượng

- Điều này có thể dẫn đến các lỗi ngoài mong đợi trong chương trình: obj1 và obj2 luôn có cùng 1 giá trị



# Phép gán đối tượng (object assignment)

- Ví dụ:

```
Date d;
```

```
Date birthday;
```

```
d = new Date (26, 9, 2005);
```

```
birthday = d;
```

Phép gán không phải là copy thông thường

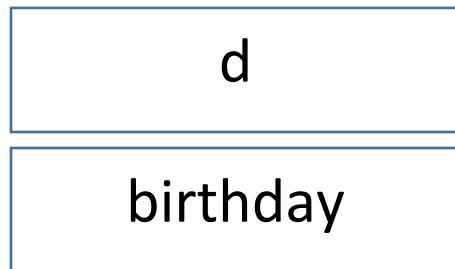
Copy nội dung của tham chiếu

Hai tham chiếu sẽ cùng tham chiếu đến một đối tượng

new operation

assign operation

static/ stack memory



heap memory

26 – 9 – 2005

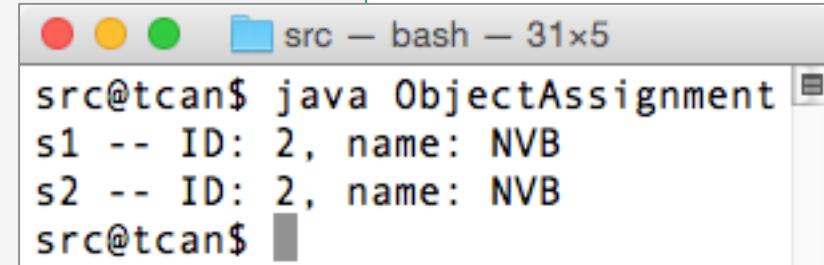
# Phép gán đối tượng (object assignment)

```
public class Student {  
    int ID;  
    String name;  
  
    public Student(int i, String n)  
        ID = i;  
        name = n;  
}
```

```
public void display(String s) {  
    System.out.println(s + " -- ID: " + ID +  
                       ", name: " + name);  
}
```

```
public void setInfo(int i, String n) {  
    ID = i;  
    name = n;  
}
```

```
public class ObjectAssignment {  
    public static void main(String []args) {  
        Student s1 = new Student(1, "NVA"), s2;  
        s2 = s1;  
        s2.setInfo(2, "NVB");  
        s1.display("s1");  
        s2.display("s2");  
    }  
}
```



A terminal window titled 'src - bash - 31x5' showing the execution of the 'ObjectAssignment' program. The output shows two student objects, s1 and s2, both initialized with ID 1 and name NVA. After calling s2.setInfo(2, "NVB"), the display method is called on both objects. Both objects print their updated information: ID 2 and name NVB.

```
src@tcan$ java ObjectAssignment  
s1 -- ID: 2, name: NVB  
s2 -- ID: 2, name: NVB  
src@tcan$
```

# Sao chép sâu (deep copy)

---

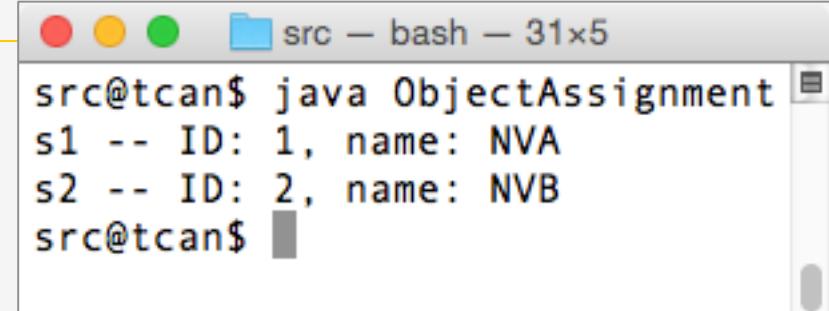
- Muốn thực hiện sao chép dữ liệu của đối tượng thay vì tham chiếu, ta phải tự viết các phương thức để sao chép dữ liệu
- Thao tác **sao chép dữ liệu** của đối tượng được gọi là **sao chép sâu** (deep copy)
- Ví dụ: để thực hiện sao chép hai đối tượng s1 và s2 trong ví dụ trước, ta phải viết thêm phương thức makeCopy () như sau trong lớp Student:
  - void makeCopy (Student) ;
  - Để sao chép s1 cho s2, ta gọi: s2 .makeCopy (s1)

- ❖ Thêm về lớp và đối tượng

# Sao chép sâu (deep copy)

```
public class Student {  
  
    //đữ liệu  
    //các phương thức  
  
    public void makeCopy(Student s) {  
        ID = s.ID;  
        name = s.name;  
    }  
}
```

```
public class ObjectAssignment {  
    public static void main(String []args) {  
        Student s1 = new Student(1, "NVA");  
        Student s2 = new Student();  
  
        s2.makeCopy(s1);  
        s2.setInfo(2, "NVB");  
  
        s1.display("s1");  
        s2.display("s2");  
    }  
}
```



A terminal window titled 'src - bash - 31x5' showing the execution of the Java program. The output shows two student objects: s1 with ID 1 and name NVA, and s2 with ID 2 and name NVB.

```
src@tcan$ java ObjectAssignment  
s1 -- ID: 1, name: NVA  
s2 -- ID: 2, name: NVB  
src@tcan$
```

# Sao chép sâu (deep copy)

- Một phương pháp khác để sao chép sâu dùng chính phép gán = là tạo một đối tượng mới giống với đối tượng cần được sao chép, sau đó sử dụng phép gán

```
public class ObjectAssignment {  
    public static void main(String []args) {  
        Student s1 = new Student(1, "NVA");  
        Student s2 = new Student();  
  
        s2 = s1.getCopy();  
        s2.setInfo(2, "NVB");  
  
        s1.display("s1");  
        s2.display("s2");  
    }  
}
```

```
src@tcan$ java ObjectAssignment  
s1 -- ID: 1, name: NVA  
s2 -- ID: 2, name: NVB
```

```
public class Student {  
  
    //dữ liệu  
    //các phương thức  
  
    public Student getCopy() {  
        Student t = new Student(ID, name);  
        return t;  
    }  
}
```

# Toán tử quan hệ (==)

---

- So sánh nội dung của các dữ liệu kiểu nguyên thủy (int, long, float....)
- So sánh nội dung của tham chiếu chứ không so sánh nội dung của đối tượng do tham chiếu trả tới

```
Integer n1 = new Integer(47);
```

```
Integer n2 = new Integer(47);
```

```
System.out.println(n1 == n2);
```

```
System.out.println(n1 != n2);
```

-----

false

true

# Toán tử quan hệ (==)

```
public class Date {  
    private int year, month, day;  
    public Date ( int d, int m, int y) {  
        year = y; month = m; day = d;    }  
    public boolean equalTo ( Date d) {  
        if (day==d.day & month==d.month & year==d.year)  
            return true;  
        else  
            return false;    }  
}  
//....  
Date d1 = new Date (10,10,1954);  
Date d2 = new Date (10,10,1954);  
System.out.println (d1 == d2);          =>false  
System.out.println (d1.equalTo(d2));    =>true
```

# Hàm hủy (descstructor)

---

- Trong Java, hàm hủy được gọi là **finalizer** (hàm kết thúc)
- Cú pháp hàm hủy:
  - Không có đối số và kiểu dữ liệu trả về, kể cả void
  - Một lớp chỉ được phép có tối đa 1 hàm kết thúc
  - Tên của hàm hủy: finalizer
- Trong Java, thông thường hàm này ít được sử dụng vì JVM đã cài đặt cơ chế Garbage collection để giải phóng các đối tượng không cần thiết

# Giải phóng bộ nhớ động

---

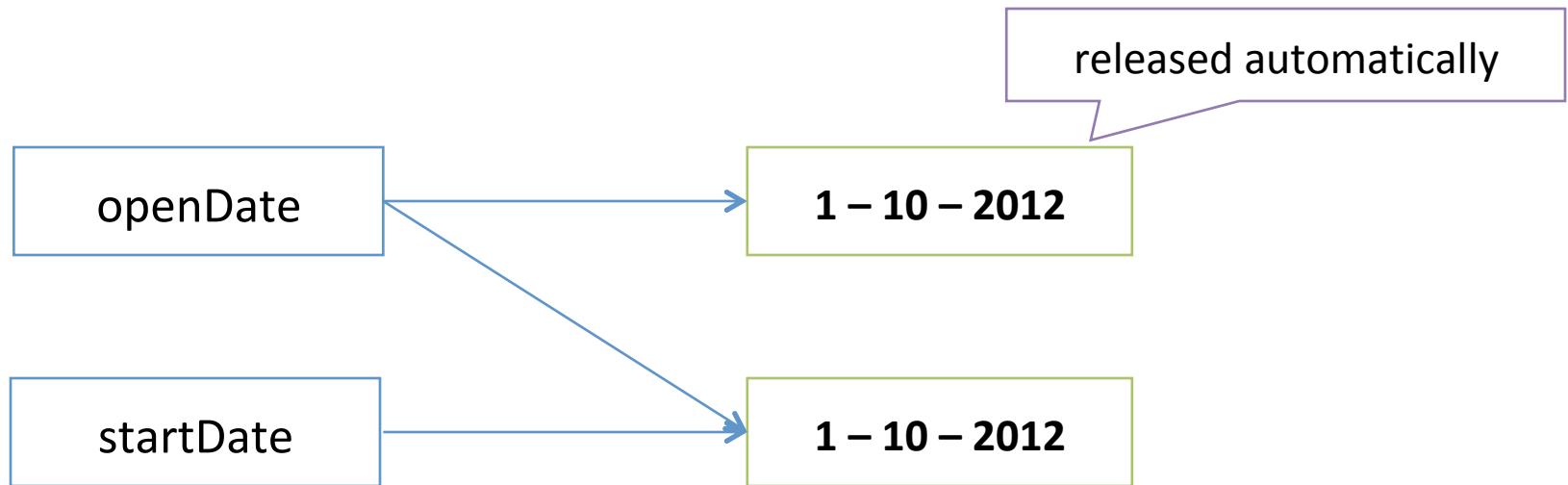
- GC giúp người lập trình không cần phải viết code để giải phóng đối tượng do đó không bao giờ quên giải phóng đối tượng
- GC sử dụng cơ chế đếm
  - Mỗi đối tượng có một số đếm các tham chiếu trỏ tới
  - Giải phóng đối tượng khi số đếm bằng 0
- Giải phóng các đối tượng không còn sử dụng
  - Kiểm tra tất cả các tham chiếu
  - Đánh dấu các đối tượng còn được tham chiếu
  - Giải phóng các đối tượng không còn được tham chiếu

# Giải phóng bộ nhớ động

- Ví dụ:

```
Date openDate = new Date (1,10,2012);
```

```
Date startDate = new Date (10,10,2012);
```



- ❖ Thêm về lớp và đối tượng

# Thêm về lớp và đối tượng *(more on class & object)*

# Định nghĩa phương thức

- **Tham số của hàm (tt):**

- Tham số kiểu dữ liệu nguyên thủy:
  - **Giá trị** của tham số thực tế được truyền vào cho tham số hình thức  $\Rightarrow$  **truyền bằng giá trị**
  - Thay đổi giá trị tham số hình thức **không làm thay đổi** giá trị của tham số thực tế
- Tham số kiểu dữ liệu tham chiếu (đối tượng):
  - **Địa chỉ** của đối tượng được tham chiếu bởi tham số thực tế được truyền vào cho tham số hình thức  $\Rightarrow$  **truyền bằng tham chiếu**
  - Cả t/số hình thức và t/số thực tế tham chiếu đến cùng 1 đối tượng  
 $\Rightarrow$  **có thể thay đổi** giá trị (thuộc tính) của đối tượng được tham chiếu bởi tham số thực tế thông qua tham số hình thức  
 $\Rightarrow$  là một cách để **truyền giá trị trả về** cho lời gọi hàm

# Định nghĩa phương thức

- Truyền tham trị

```
class Test {  
    void noChange ( int i, int j) {  
        i = i + j;  
        j = -j;  
        System.out.println ("i and j inside call: " + i + " " + j);  
    }  
}  
  
public class CallByValue {  
    public static void main ( String args[]) {  
        Test ob = new Test ();  
        int a = 15, b = 20;  
        System.out.println ("a and b before call: " + a + " " + b);  
        ob.noChange (a, b);  
        System.out.println ("a and b after call: " + a + " " + b);  
    }  
}
```

a and b before call: 15 20  
i and j inside call: 35 -20  
a and b after call: 15 20

# Định nghĩa phương thức

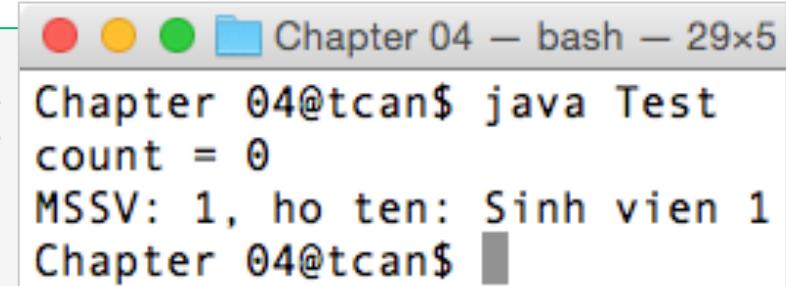
- Truyền tham chiếu:

```
class Test {  
    private int a, b;  
    public Test ( int i, int j) {  
        a = i;  
        b = j;  }  
    public static void change ( Test ob) {  
        ob.a = ob.a + ob.b;  
        ob.b = -ob.b;  }  
}  
  
public class CallByRef {  
    public static void main ( String args[]) {  
        Test ob = new Test (15, 20);  
        System.out.println ("ob.a and ob.b before call: " + ob.a + " " + ob.b);  
        Test.change(ob);  
        System.out.println ("ob.a and ob.b after call: " + ob.a + " " + ob.b);  
    }  
}
```

ob.a and ob.b before call: 15 20  
ob.a and ob.b after call: 35 -20

# Định nghĩa phương thức

```
class Test {  
    public static void main(String args[]) {  
        int count = 0;  
        Sinhvien sv1 = new Sinhvien();  
        test(count, sv1);  
        System.out.println("count = " + count);  
        sv1.hienthi();  
    }  
  
    public static void test(int c, Sinhvien sv) {  
        c = c + 1;  
        sv.mssv = c;  
        sv.hoten = "Sinh vien " + c;  
    }  
}
```



```
Chapter 04 – bash – 29x5  
Chapter 04@tcan$ java Test  
count = 0  
MSSV: 1, ho ten: Sinh vien 1  
Chapter 04@tcan$
```

```
class Sinhvien {  
    public int mssv;  
    public String hoten;  
  
    public void hienthi() {  
        System.out.println("MSSV: " + mssv +  
                           ", ho ten: " + hoten);  
    }  
}
```

# Định nghĩa phương thức

```

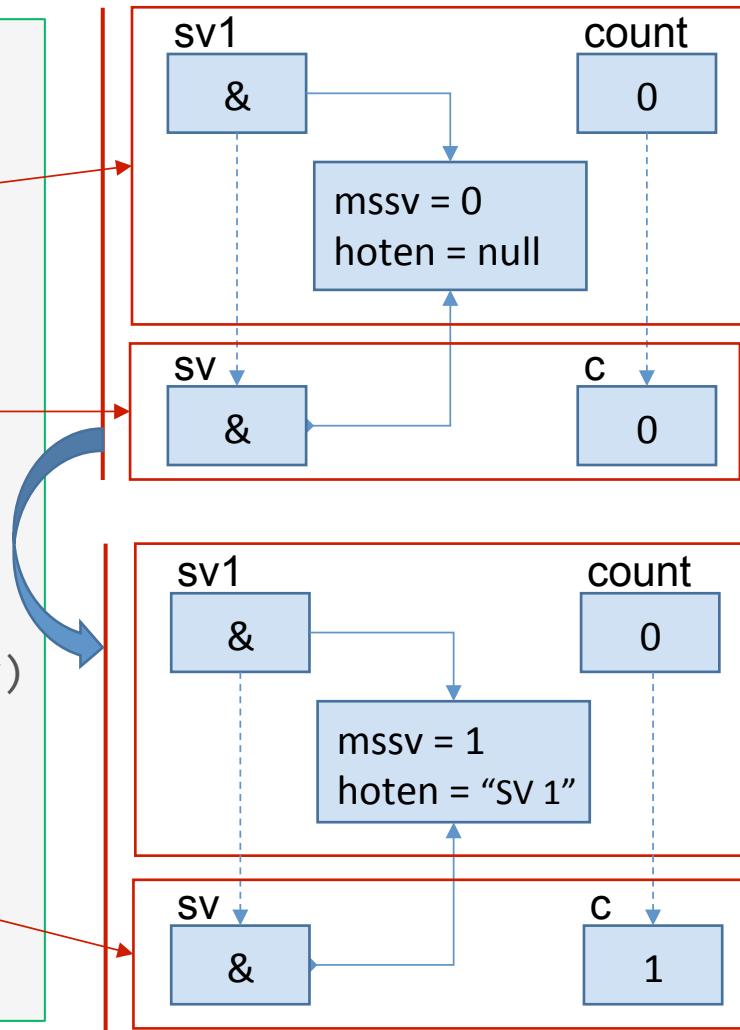
class Test {
    public static void main(String args[]) {
        int count = 0;
        Sinhvien sv1 = new Sinhvien();

        test(count, sv1);

        System.out.println("count = " + count);
        sv1.hienthi();
    }

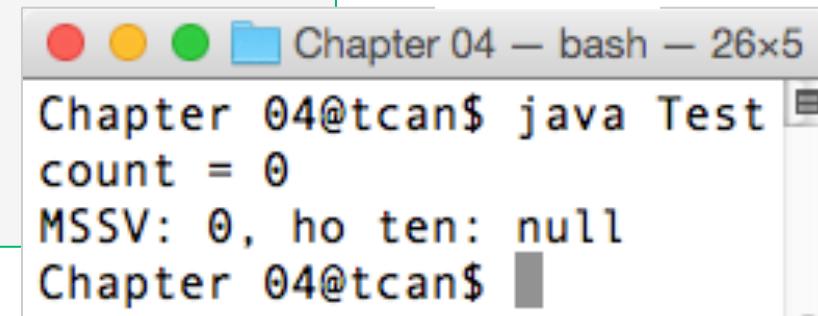
    public static void test(int c, Sinhvien sv)
    {
        c = c + 1;
        sv.mssv = c;
        sv.hoten = "SV " + c;
    }
}

```



# Định nghĩa phương thức

```
class Test {  
    public static void main(String args[]) {  
        int count = 0;  
        Sinhvien sv1 = new Sinhvien();  
        test(count, sv1);  
        System.out.println("count = " + count);  
        sv1.hienthi();  
    }  
  
    public static void test(int c, Sinhvien sv) {  
        c = c + 1;  
        sv = new Sinhvien();  
        sv.mssv = c;  
        sv.hoten = "Sinh vien " + c;  
    }  
}
```



The terminal window shows the following output:

```
Chapter 04 – bash – 26x5  
Chapter 04@tcan$ java Test  
count = 0  
MSSV: 0, ho ten: null  
Chapter 04@tcan$
```

# Định nghĩa phương thức

---

- Ưu điểm của truyền bằng tham chiếu:
  - Tiết kiệm bộ nhớ: tham số kiểu tham chiếu chỉ chứa địa chỉ được truyền vào, không phụ thuộc kích thước của đối tượng
  - Có thể sử dụng để trả về nhiều hơn một giá trị cho lời gọi phương thức
  - Tăng tốc độ gọi hàm: chỉ sao chép tham chiếu chứ không sao chép toàn bộ đối tượng (dữ liệu)

# Trả về một đối tượng

```
class Date {  
    private int year, month, day;  
    public Date ( int y, int m, int d) {  
        year = y; month = m; day = d; }  
    public Date nextMonth() {  
        Date d = new Date (day, month+1, year);  
        return d;  
    }  
    public void printDay() {  
        System.out.print (day + "/" + month+ "/" + year);  
        System.out.println();  
    }  
}  
-----  
Date d1 = new Date (2005, 9, 26);  
Date d2;  
d2 = d1.nextMonth();
```

# Tái định nghĩa phương thức

- Tái định nghĩa phương thức (**method overloading**): định nghĩa các phương thức trùng tên
- Các phương thức trùng tên phải:
  - Khác nhau về **số lượng tham số**
  - Khác nhau về **kiểu tham số**

```
class Point2D {  
    public void hienthi() {  
        System.out.println("(" + x + ", " + y + ")");  
    }  
  
    public void hienthi(String s) {  
        System.out.println(s + "(" + x + ", " + y + ")");  
    }  
}
```

# Tái định nghĩa phương thức

---

- Java cho phép ta xây dựng nhiều phương thức trùng tên nhau trong cùng một lớp, hiện tượng này được gọi là overloading phương thức.
- Ta phải cho java biết cần phải gọi phương thức nào để thực hiện dựa vào **sự khác nhau về số lượng đối số cũng như kiểu dữ liệu của các đối số** này để phân biệt các phương thức trùng tên.
- Ta **không thể sử dụng giá trị trả về** của hàm để phân biệt sự khác nhau giữa 2 phương thức overload

# Tái định nghĩa phương thức

---

Hợp lệ:

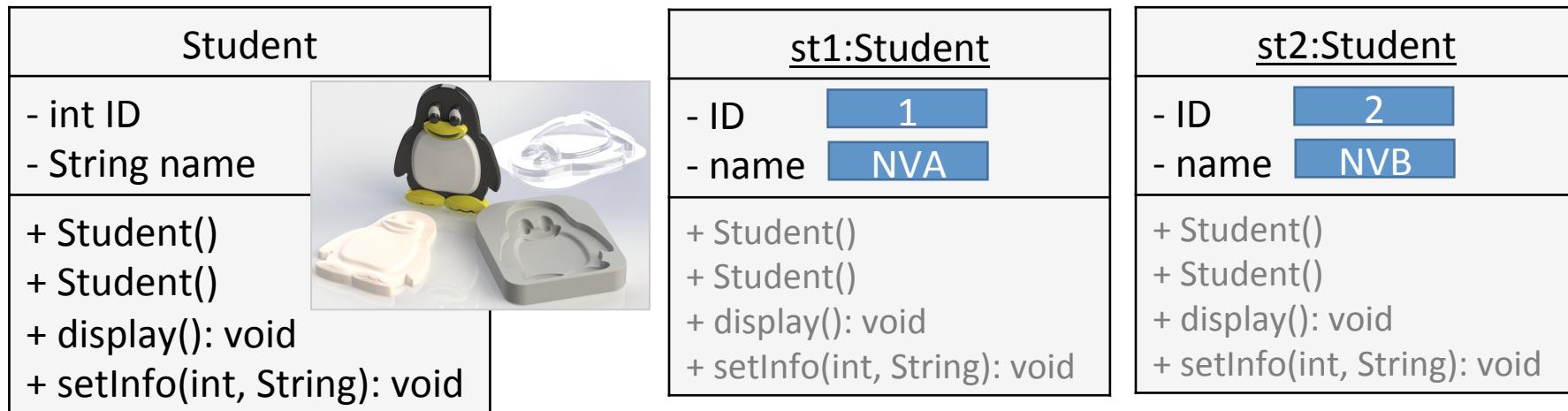
```
public void func1();  
public void func1(int);  
public void func1(long);  
public void func1(String, int);  
public void func1(int, String);
```

Không hợp lệ:

```
public void func2(String);  
public void func2(String);  
public int func2(String);  
public func3(int, long);  
public func3(int, long);
```

# Thành phần tĩnh (static members)

- Lớp là khuôn mẫu để tạo ra đối tượng
  - Các thành phần khai báo trong lớp chỉ tồn tại khi đối tượng được tạo ra
  - Mỗi đối tượng sẽ “sở hữu” các thành phần riêng của mình  
⇒ có bao nhiêu đối tượng được tạo ra thì có bấy nhiêu “tập” thành phần trên bộ nhớ



# Thành phần tĩnh (static members)

---

- Thành phần tĩnh:

- Còn được gọi là **thành phần “của lớp”**
- Tồn tại độc lập với các đối tượng: có bao nhiêu đối tượng được tạo ra thì vẫn chỉ có 1 bản (copy) của thành phần tĩnh trong bộ nhớ
- Thành phần tĩnh được truy xuất trực tiếp thông qua lớp

**<đối tượng>.<tên thành phần>**

- Các phương thức tĩnh chỉ được truy xuất các thuộc tính tĩnh
- Các phương thức không tĩnh có thể truy xuất tất cả các thành phần tĩnh lẫn không tĩnh

# Thành phần tĩnh (static members)

- *Thuộc tính tĩnh*: Thuộc tính được khai báo với từ khóa static là thuộc tính tĩnh.
- Ví dụ:

```
class xStatic {  
    static int i = 10; //Thuộc tính tĩnh  
    int j = 5; // Thuộc tính thường  
}
```

```
class xStatic {  
    static int i;  
    static {  
        i = 10;  
    }  
}
```

Khởi tạo ngay khi khai báo

Sử dụng khối khởi đầu tĩnh

Chú ý: ta không thể sử dụng phương thức xây dựng để khởi đầu các thuộc tính tĩnh, bởi vì nó không phải là phương thức tĩnh.

# Thành phần tĩnh (static members)

- Phương thức tĩnh là chung cho cả lớp, nó không lệ thuộc vào một đối tượng cụ thể nào.
- Lời gọi phương thức tĩnh xuất phát từ:
  - tên\_lớp.tên\_phương\_thức\_tĩnh(tham số);
  - tên\_đối\_tương. tên\_phương\_thức\_tĩnh(tham số);
- Vì phương thức tĩnh là độc lập với đối tượng do vậy ở bên trong phương thức tĩnh ta không thể truy cập các thành viên không tĩnh của lớp đó.

```
public class xStatic {  
    private static int i = 10; //Thuộc tính tĩnh  
    // Phương thức tĩnh  
    public static void printFunction(){  
        System.out.println(i);  
    }  
}
```

# Thành phần tĩnh (static members)

```
class ClockStatic {  
    private static int count;  
    private int h, m, s;  
  
    public ClockStatic() {  
        h = m = s = 0;  
        count++;  
    }  
  
    public ClockStatic(int g, int p, int gi) {  
        h = g; m = p; s = gi;  
        count++;  
    }  
  
    static public void noOfInstances() {  
        System.out.println("# of instances: " +  
            count);  
    }  
}
```

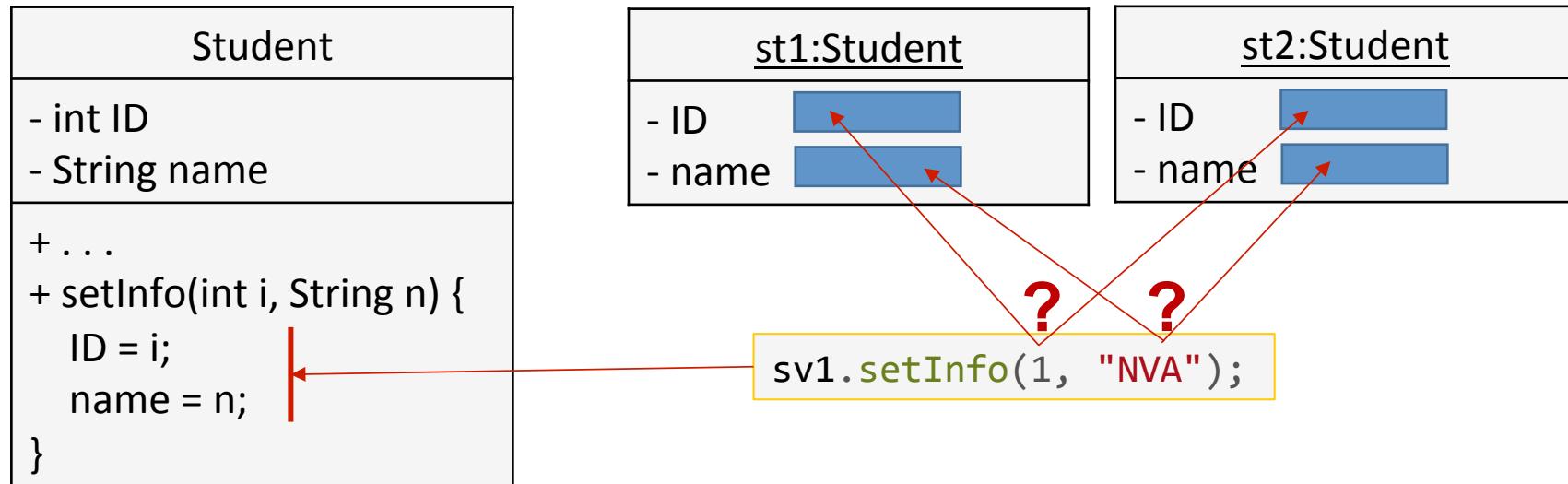
```
class TestClockStatic {  
    public static void main(String args[]) {  
        ClockStatic.noOfInstances();  
  
        ClockStatic c1 = new ClockStatic();  
        ClockStatic.noOfInstances();  
    }  
}
```

src@tcan\$ java TestClockStatic  
# of instances: 0  
# of instances: 1

```
public void display() {  
    System.out.println(h + ":" + m + ":" + s);  
}  
}
```

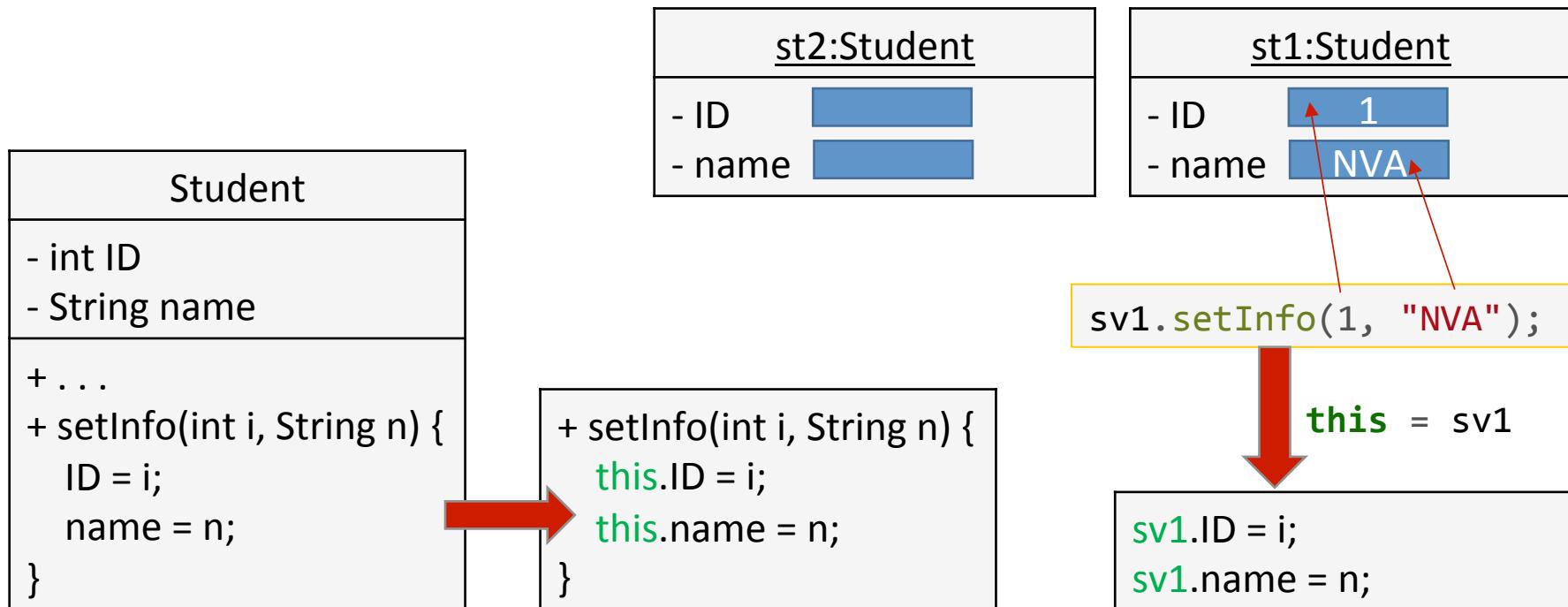
# Biến tham chiếu this

- Đây là một biến tham chiếu đặc biệt, tồn tại trong tất cả các phương thức không tĩnh của lớp
- Biến này tham chiếu đến đối tượng đang gọi phương thức



# Biến tham chiếu this

- Trong tất cả các phương thức **không tĩnh**, tất cả các lệnh truy xuất vào các dữ liệu thành viên không tĩnh sẽ **được tự động thêm vào** tham chiếu this:



# Biến tham chiếu this

---

- Java cung cấp tham chiếu *this* để trỏ tới chính đối tượng đang hoạt động
- Từ khóa *this* được sử dụng vào các mục đích như:
  - Tham chiếu tường minh đến thuộc tính và phương thức của đối tượng (tránh trùng tên)
  - Truyền tham số và giá trị trả về của phương thức
  - Dùng để gọi constructor

# Biến tham chiếu this

- Biến tham chiếu this có thể được sử dụng một cách tường minh để **tránh trùng tên**

```
class Student {  
    private int ID;  
    private String name;
```

//...

```
public void setInfo(int ID, String name) {  
    ID = ID;  
    name = name;  
}  
}
```

```
public void setInfo(int ID, String name) {  
    this.ID = ID;  
    this.name = name;  
}
```

Hai tham số của phương thức sẽ che đi các dữ liệu thành viên

# Biến tham chiếu this

---

- Ví dụ: dùng làm giá trị trả về

```
public class Counter {  
    private int c = 0;  
    public Counter increase() {  
        c++;  
        return this;  
    }  
    public int getValue() {  
        return c;  
    }  
}
```

```
Counter c = new Counter();  
System.out.println ( c.increase().increase().getValue() );  
System.out.println  
( c.increase().increase().getValue().increase().increase().getValue() );
```

# Biến tham chiếu this

---

- Ví dụ: gọi constructor

```
class MyDate {  
    private int year, month, day;  
    public MyDate ( int y, int m, int d ) {  
        //  
    }  
    // copy constructor  
    public MyDate ( MyDate d ) {  
        this ( d.year, d.month, d.day );  
        System.out.println("copy constructor called");  
    }  
}
```

- Constructor chỉ được gọi bên trong một constructor khác và chỉ được gọi một lần ở thời điểm đầu tiên.



# Question?

# Phụ lục – UML

---

- UML: Unified Modeling Language.
- Là một ngôn ngữ dùng để **mô hình hóa** các hệ thống thông tin theo Hướng đối tượng.
- Bao gồm một hệ thống các **ký hiệu** (symbol), **sơ đồ** (diagram).
- Được sử dụng trong nhiều giai đoạn của qui trình phát triển phần mềm.
- Một số công cụ: Rational Rose, Visio, StartUML, ArgoUML, ...

# Phụ lục – UML

---

- UML bao gồm 9 loại sơ đồ:
  1. Use case diagram: Sơ đồ use case.
  2. **Class diagram: Sơ đồ lớp.**
  3. Object diagram: Sơ đồ đối tượng.
  4. Sequence diagram: Sơ đồ trình tự.
  5. State diagram: Sơ đồ trạng thái.
  6. Collaboration diagram: Sơ đồ cộng tác.
  7. Component diagram: Sơ đồ thành phần.
  8. Deployment diagram: Sơ đồ triển khai.
  9. Activity diagram: Sơ đồ hoạt động.

# Phụ lục – UML

---

- Class diagram (sơ đồ lớp):
  - Mô tả cấu trúc của hệ thống theo hướng đối tượng.
  - Cấu trúc của một hệ thống được xây dựng từ các lớp và mối quan hệ giữa chúng.
- Ký hiệu (notation):
  - Lớp.
  - Giao diện.
  - Quan hệ:
    - Kết hợp (Association), tập hợp (Aggregation) và tổng hợp (Composition).
    - Hiện thực hóa (Realization).
    - Thừa kế/Tổng quát hóa (Generalization).

# Phụ lục – UML

---

- Lớp:

ClassName
🔒PriAttribute : Datatype
🔑ProAttribute : Datatype
❖PubAttribute : Datatype
❖PriMethod() ❖ProMethod() ❖PubMethod()

ClassName
🔒PriAttribute : Datatype
🔑ProAttribute : Datatype
❖PubAttribute : Datatype
❖PriMethod(arg : dataType, ...) : returnType ❖ProMethod(arg : dataType, ...) : returnType ❖PubMethod(arg : dataType, ...) : returnType

❖Private
❖Protected
❖Public

-Private
#Protected
+Public

Xe Oto
🔒Nhanhieu : String 🔒Mau sac : String 🔒Gia : Float 🔒Hop so : Integer
❖Chay() ❖Dung() ❖Tang toc() ❖Giam toc()

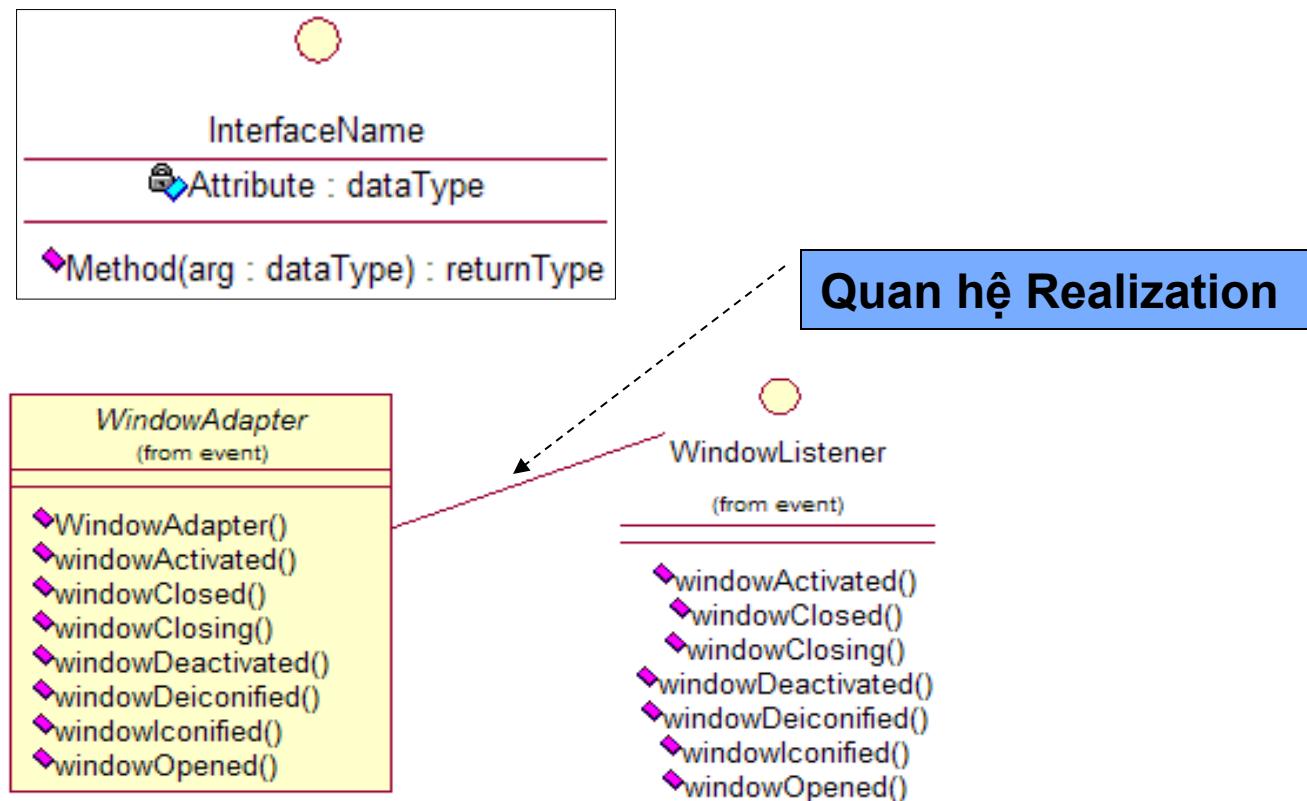
Xe Oto
🔒Nhanhieu : String 🔒Mau sac : String 🔒Gia : Float 🔒Hop so : Integer
❖Chay() : void ❖Dung() : void ❖Tang toc(speed : Integer) : boolean ❖Giam toc(speed : int) : boolean

_Stack_
🔒int *list
❖push(item : int) : boolean
❖pop() : int
❖isEmpty() : boolean

# Phụ lục – UML

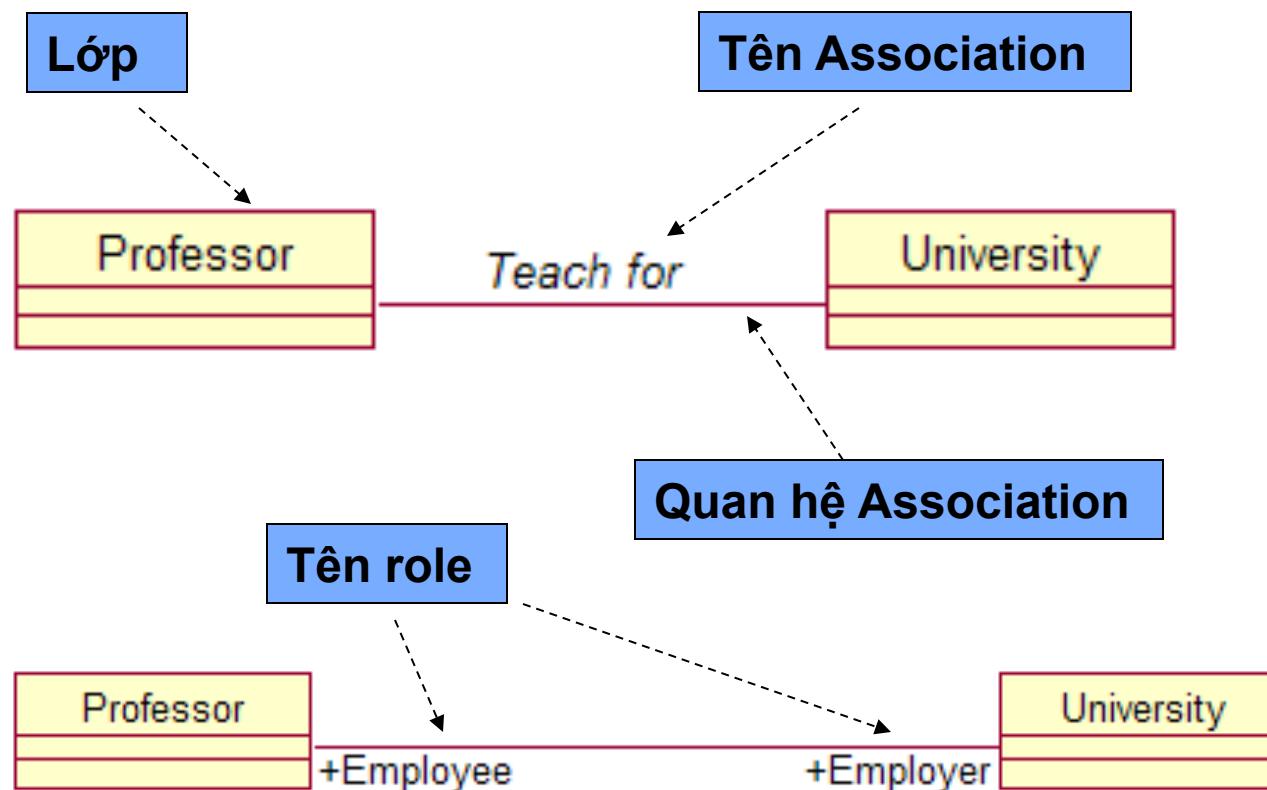
---

- Giao diện và quan hệ Realization: Quan hệ realization là quan hệ giữa giao diện và lớp cài đặt nó.



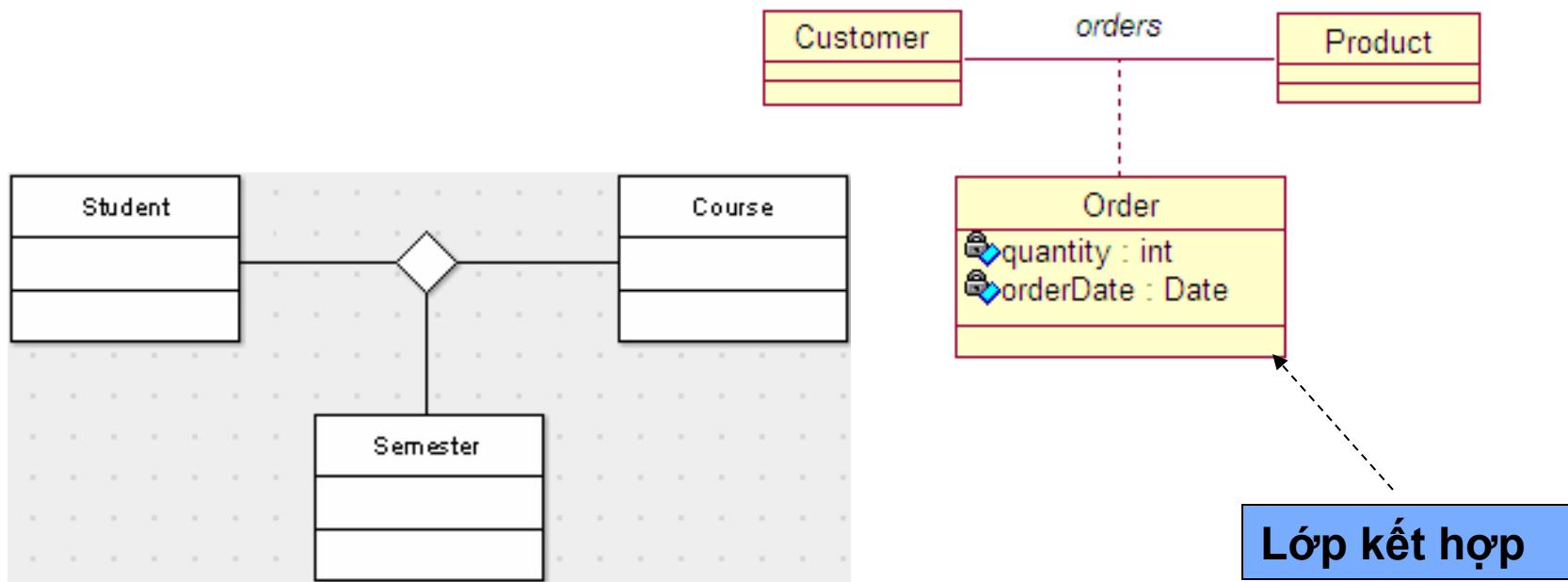
# Phụ lục – UML

- Quan hệ kết hợp (Association): Mô tả mối liên hệ ngữ nghĩa giữa các lớp.



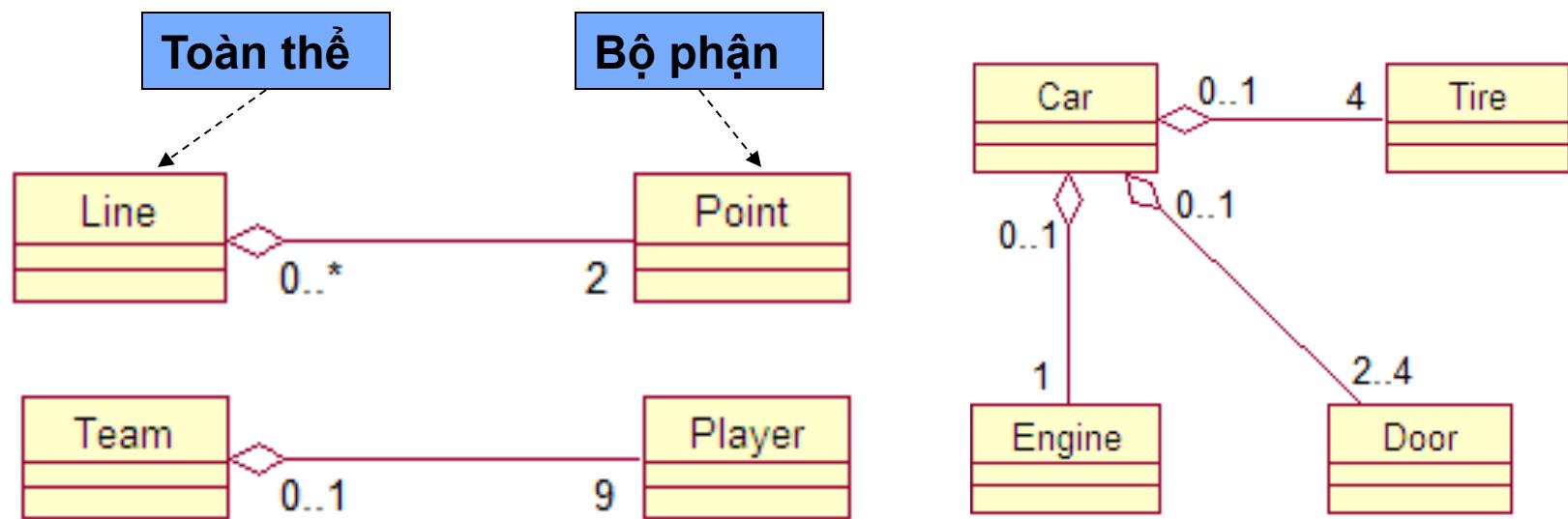
# Phụ lục – UML

- Quan hệ kết hợp nhiều chiều (n-ary)
- Lớp kết hợp (association class): Được xem là thuộc tính của một quan hệ kết hợp.



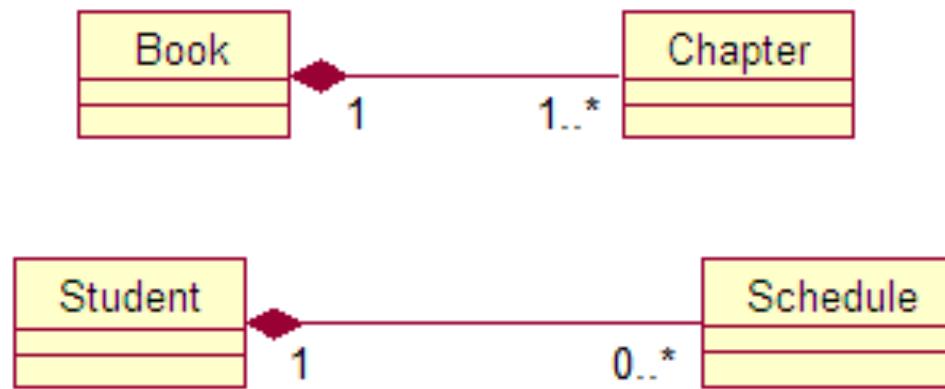
# Phụ lục – UML

- Quan hệ tập hợp (Aggregation): Là một dạng đặc biệt của quan hệ kết hợp, mô tả mối quan hệ toàn thể-bộ phận giữa một thực thể và các bộ phận của một thực thể.



# Phụ lục – UML

- Quan hệ tổng hợp (Composition): Là một dạng đặc biệt của quan hệ tập hợp, có tính sở hữu cao. Trong đó, các bộ phận của một thực thể **không thể sống lâu hơn** thực thể.



**Chú ý:** Nếu không có sinh viên → không có schedule. Hoặc, nếu không tồn tại quyển sách (book) thì không có các chương sách (chapter)

# Phụ lục – UML

- Quan hệ thừa kế (Inheritance):

