



**Bộ môn Công nghệ phần mềm**  
**VIỆN CÔNG NGHỆ THÔNG TIN TRUYỀN THÔNG**  
**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**



# **LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG**

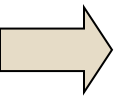
**Bài 03. Đóng gói và xây dựng lớp,  
tạo và sử dụng đối tượng.**

**Cao Tuấn Dũng**  
**Nguyễn Thị Thu Trang**

# Mục tiêu bài học

- Nêu được bản chất, vai trò của trừu tượng hóa
- Giải thích về đóng gói và che giấu thông tin
- Xây dựng lớp
  - Định nghĩa lớp, thực hiện ẩn
  - Tạo các phương thức, các trường/thuộc tính
- Tạo và sử dụng đối tượng
  - Phương thức khởi tạo
  - Khai báo và khởi tạo đối tượng
  - Sử dụng đối tượng

# Nội dung



1. Trừu tượng hóa dữ liệu
2. Đóng gói và xây dựng lớp
3. Tạo và sử dụng đối tượng

# 1.1. Trừu tượng hóa

- Giảm thiểu và tinh lọc các chi tiết nhằm tập trung vào một số khái niệm/vấn đề quan tâm tại một thời điểm.
  - “abstraction – a concept or idea not associated with any specific instance”.
  - Ví dụ: Các định nghĩa toán học
- 2 loại trừu tượng hóa
  - Trừu tượng hóa điều khiển (control abstraction)
  - Trừu tượng hóa dữ liệu (data abstraction)

## 1.1. Trừu tượng hóa (2)

- Trừu tượng hóa điều khiển: Sử dụng các chương trình con (subprogram) và các luồng điều khiển (control flow)
  - Ví dụ:  $a := (1 + 2) * 5$ 
    - Nếu không có trừu tượng hóa điều khiển, LTV phải chỉ ra tất cả các thanh ghi, các bước tính toán mức nhị phân...
- Trừu tượng hóa dữ liệu: Xử lý dữ liệu theo các cách khác nhau
  - Ví dụ: Kiểu dữ liệu
    - Sự tách biệt rõ ràng giữa các thuộc tính trừu tượng của kiểu dữ liệu và các chi tiết thực thi cụ thể của kiểu dữ liệu đó.

## 1.2. Trừu tượng hóa dữ liệu trong OOP

- Đối tượng trong thực tế phức tạp



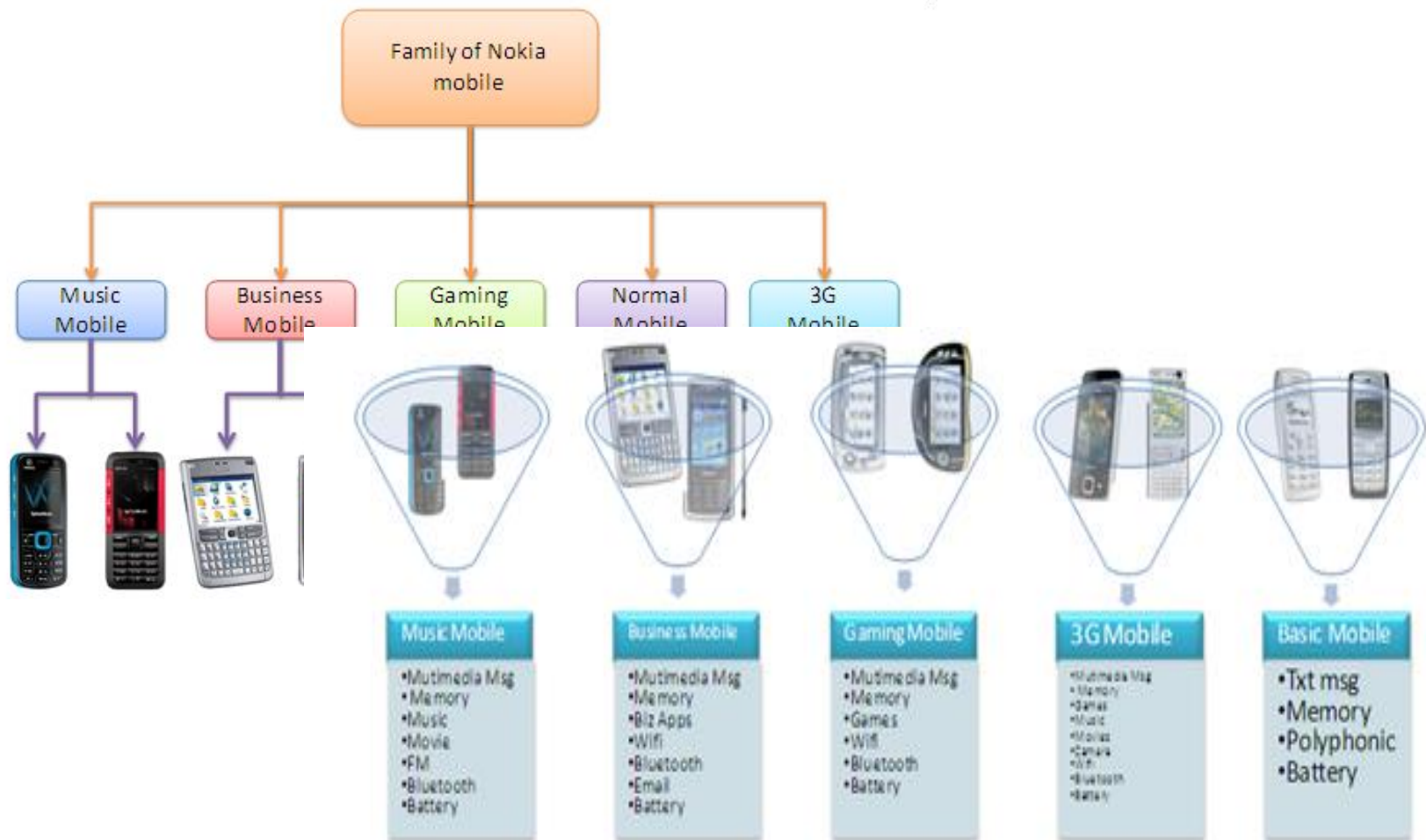
- Cần đơn giản hóa, bỏ qua những chi tiết không cần thiết
- Chỉ “trích rút” lấy những thông tin liên quan, thông tin quan tâm, quan trọng với bài toán

# Ví dụ: Trừu tượng hóa từ các chi tiết sau



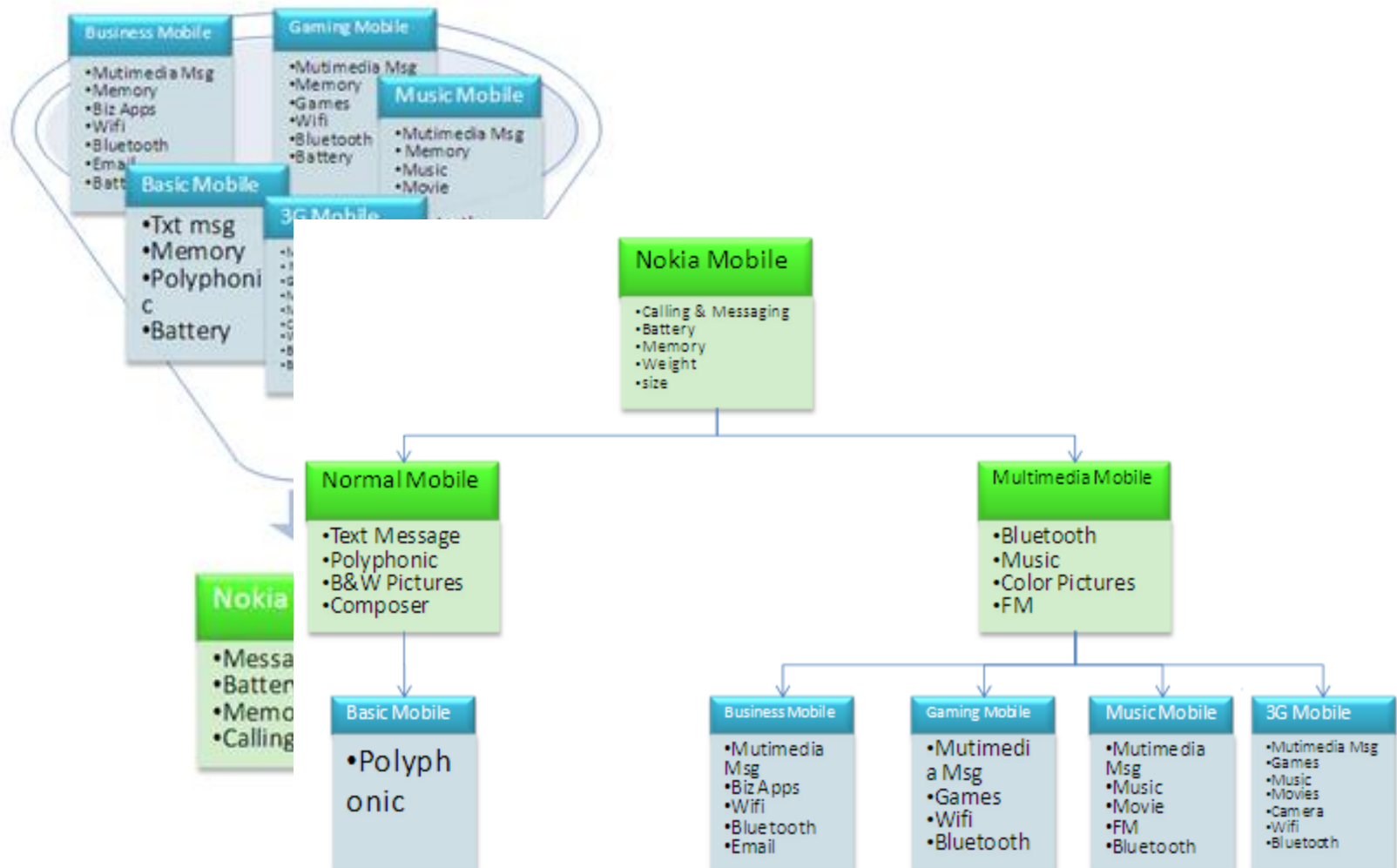
- Các thực thể này có những điểm gì chung – các đặc thù nào?
  - Điều là điện thoại Nokia
  - Dạng trượt, gập, phổ thông
  - Điện thoại cho doanh nhân, Music, 3G
  - Bàn phím QWERTY , Dạng cơ bản và không phím.
  - Màu sắc, kích thước, ...

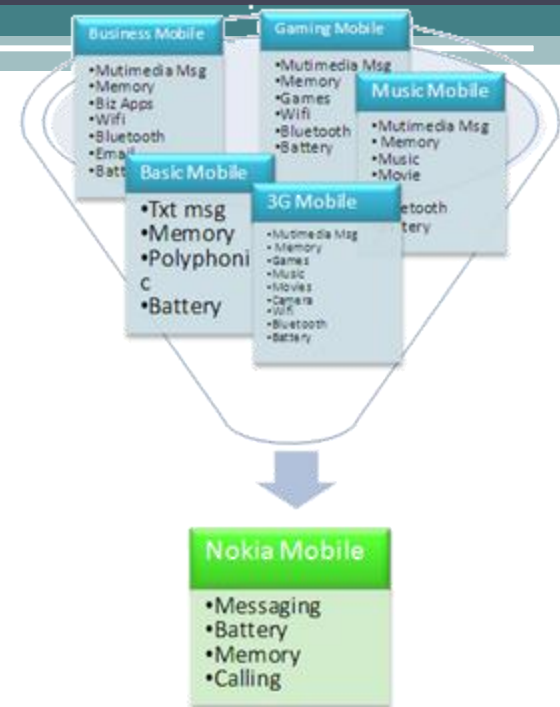
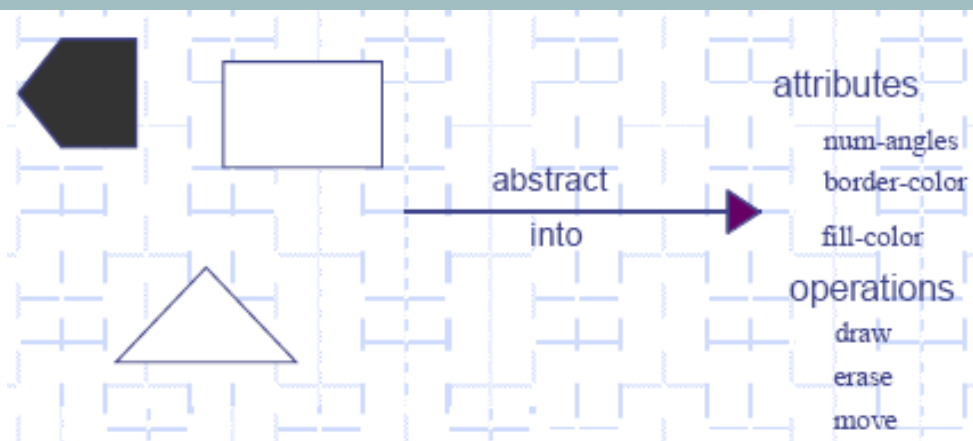
# Phân loại mobile - tìm hiểu các đặc tính riêng





# Từ chi tiết quay lại khái quát





## 1.2. Trừu tượng hóa dữ liệu (3)

- Any model that includes the most important, essential, or distinguishing aspects of something while suppressing or ignoring less important, immaterial, or diversionary details. The result of removing distinctions so as to emphasize commonalties (*Dictionary of Object Technology*, Firesmith, Eykholt, 1995).
  - Cho phép quản lý các bài toán phức tạp bằng cách tập trung vào các đặc trưng quan trọng của một thực thể nhằm phân biệt nó với các loại thực thể khác.

## 1.2. Trừu tượng hóa dữ liệu (2)

- Trừu tượng hóa là một cách nhìn hoặc cách biểu diễn một thực thể chỉ bao gồm các thuộc tính liên quan trong một ngữ cảnh nào đó.
- Tập hợp các thể hiện của các thực thể thành các nhóm có chung các thuộc tính gọi là Lớp (class).



unclassified  
"things"



Pine



Eucalypt



Jumbo



Sher Khan



Daisy



Bugs



Jane



John



Big Al

- organisms,  
mammals,  
humans



Pine



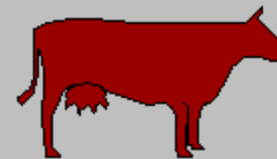
Eucalypt



Jumbo



Sher Khan



Daisy



Bugs



Jane



John



Big Al

- organisms,  
mammals,  
dangerous  
mammals



Pine



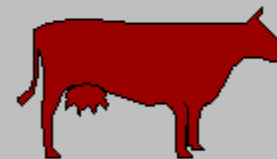
Eucalypt



Jumbo



Shere Khan



Daisy



Bugs



Jane



John



Big Al

## 1.3. Lớp vs. Đối tượng

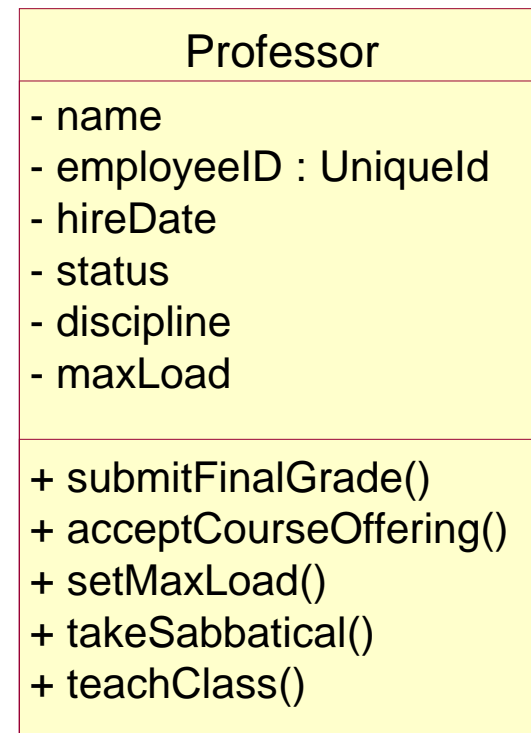
- Lớp là mô hình khái niệm, mô tả các thực thể
- Lớp như một bản mẫu, định nghĩa các *thuộc tính và phương thức chung* của các đối tượng
- Một lớp là sự trừu tượng hóa của một tập các đối tượng
- ◆ Đối tượng là sự vật thật, là thực thể thực sự
- ◆ Đối tượng là một thể hiện (instance) của một lớp, *dữ liệu của các đối tượng khác nhau là khác nhau*
- ◆ Mỗi đối tượng có một lớp xác định dữ liệu và hành vi của nó.



# Biểu diễn lớp trong UML

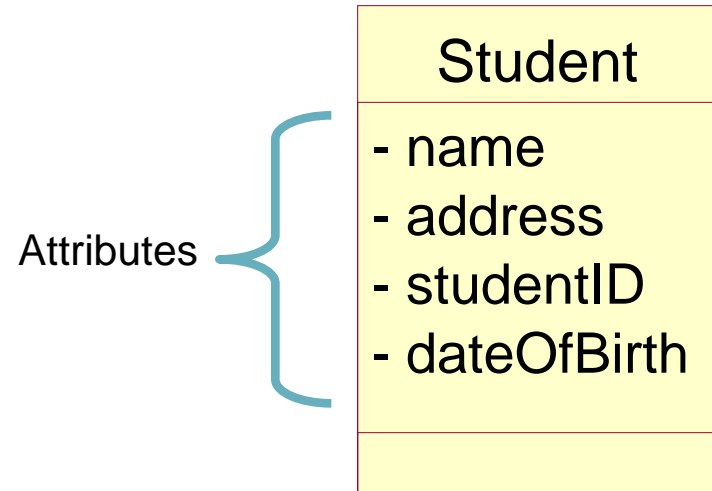
- Lớp (class) được biểu diễn bằng 1 hình chữ nhật với 3 thành phần:

- Tên lớp
- Cấu trúc (thuộc tính)
- Hành vi (thao tác)

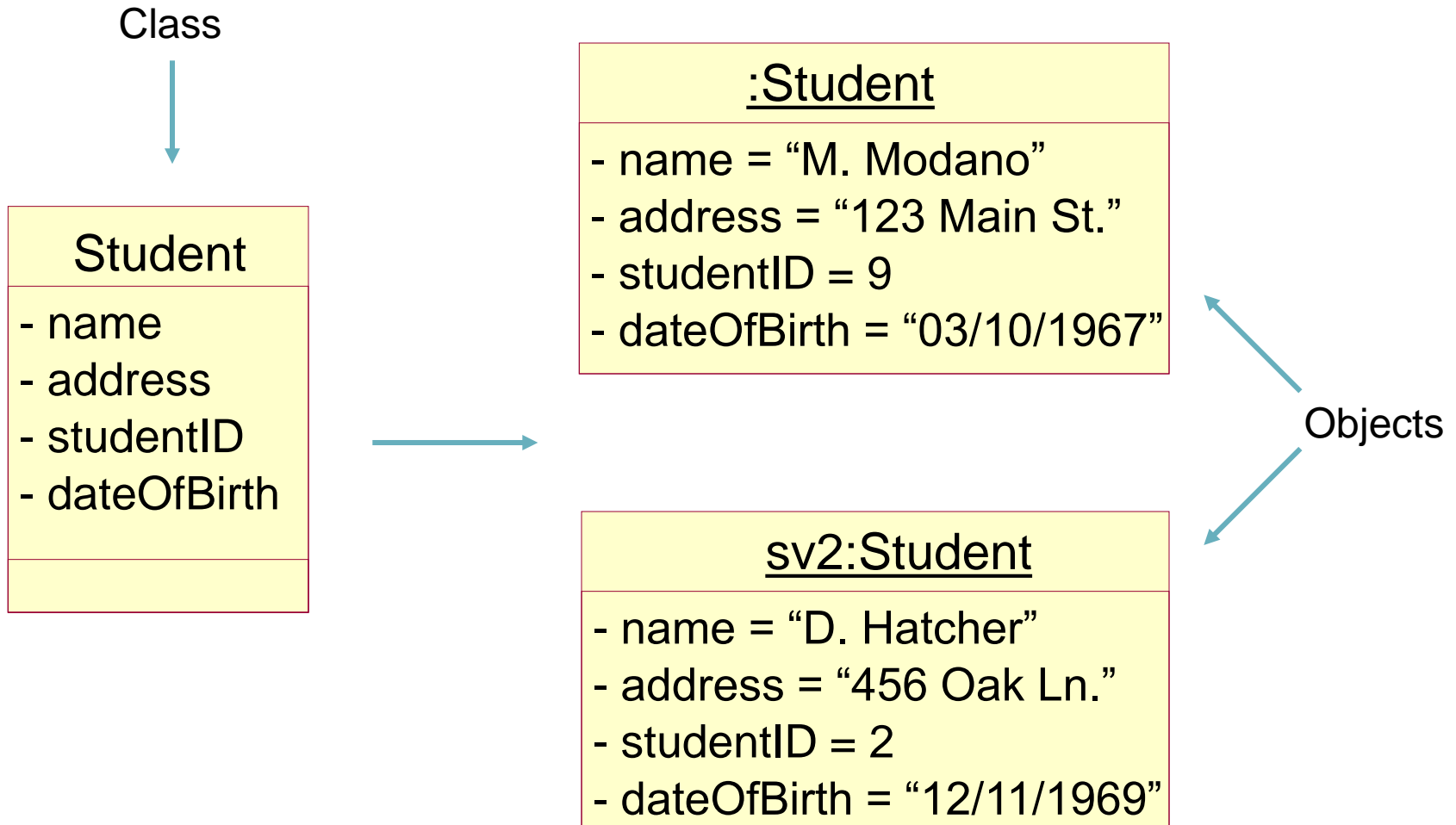


# Thuộc tính (attribute) là gì?

- Một thuộc tính là một đặc tính được đặt tên của một lớp mô tả khoảng giá trị mà các thể hiện của đặc tính đó có thể chứa.
  - Một lớp có thể không có thuộc tính nào hoặc có số lượng thuộc tính bất kỳ.



# Lớp và đối tượng trong UML



# Nhắc lại về đối tượng

- Đối tượng là sự biểu diễn của một thực thể, hoặc trong thế giới thực như bàn, ghế, con người, hoặc là những thực thể trừu tượng
- ***Đối tượng là là sự trừu tượng hoá có ranh giới rõ ràng và có ý nghĩa đối với ứng dụng.***
- Từng đối tượng trong hệ thống bao giờ cũng có ba đặc tả:
  - Trạng thái
  - Hoạt động
  - ***Đặc điểm nhận dạng***

# Trạng thái của đối tượng

- Trạng thái của một đối tượng là một trong số những hoàn cảnh mà đối tượng có thể tồn tại. Thông thường, trạng thái của đối tượng thay đổi theo thời gian
- Trạng thái của đối tượng được định nghĩa là tập tất cả các đặc tính, các giá trị của các đặc tính đó, cộng với mối quan hệ của đối tượng với các đối tượng khác.

# Hành vi của đối tượng

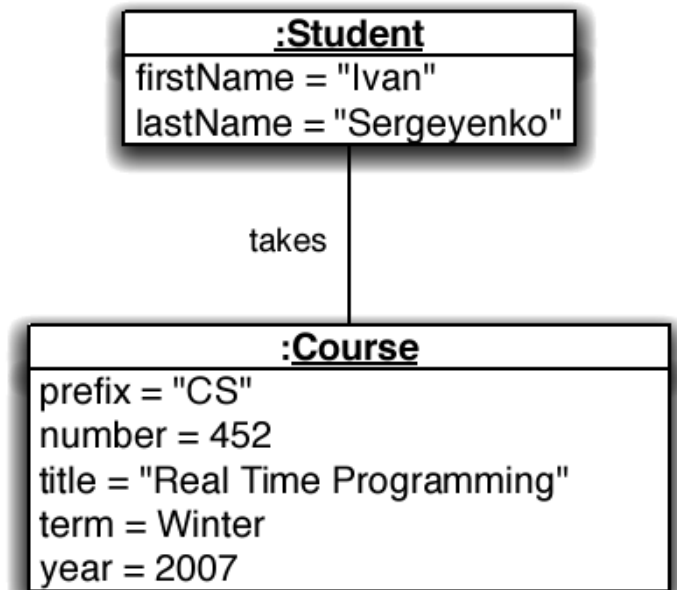
- Hoạt động của đối tượng xác định cách thức đối tượng đáp ứng các yêu cầu từ các đối tượng khác và đó là tất cả những gì đối tượng có thể làm. Hoạt động của đối tượng được thực hiện bởi một tập các thao tác cho đối tượng.
- **Đặc điểm nhận dạng**
  - Đặc điểm nhận dạng là một đặc tính của đối tượng cho phép phân biệt nó với các đối tượng khác.

# Mối quan hệ giữa các đối tượng

- Toàn bộ hệ thống được xây dựng từ rất nhiều lớp và đối tượng.
- Có hai loại quan hệ giữa các đối tượng là :
  - liên kết(*link*)
  - kết tập (*aggregation*)

# Mối quan hệ liên kết (link)

- **Mối** quan hệ liên kết là sự kết nối vật lý hoặc logic giữa các đối tượng. Một đối tượng phối hợp với các đối tượng khác thông qua các liên kết của nó với các đối tượng này. Nói một cách khác, một liên kết biểu diễn một liên hợp (association) xác định, trong đó một đối tượng(client) sử dụng những dịch vụ của đối tượng khác(supplier).



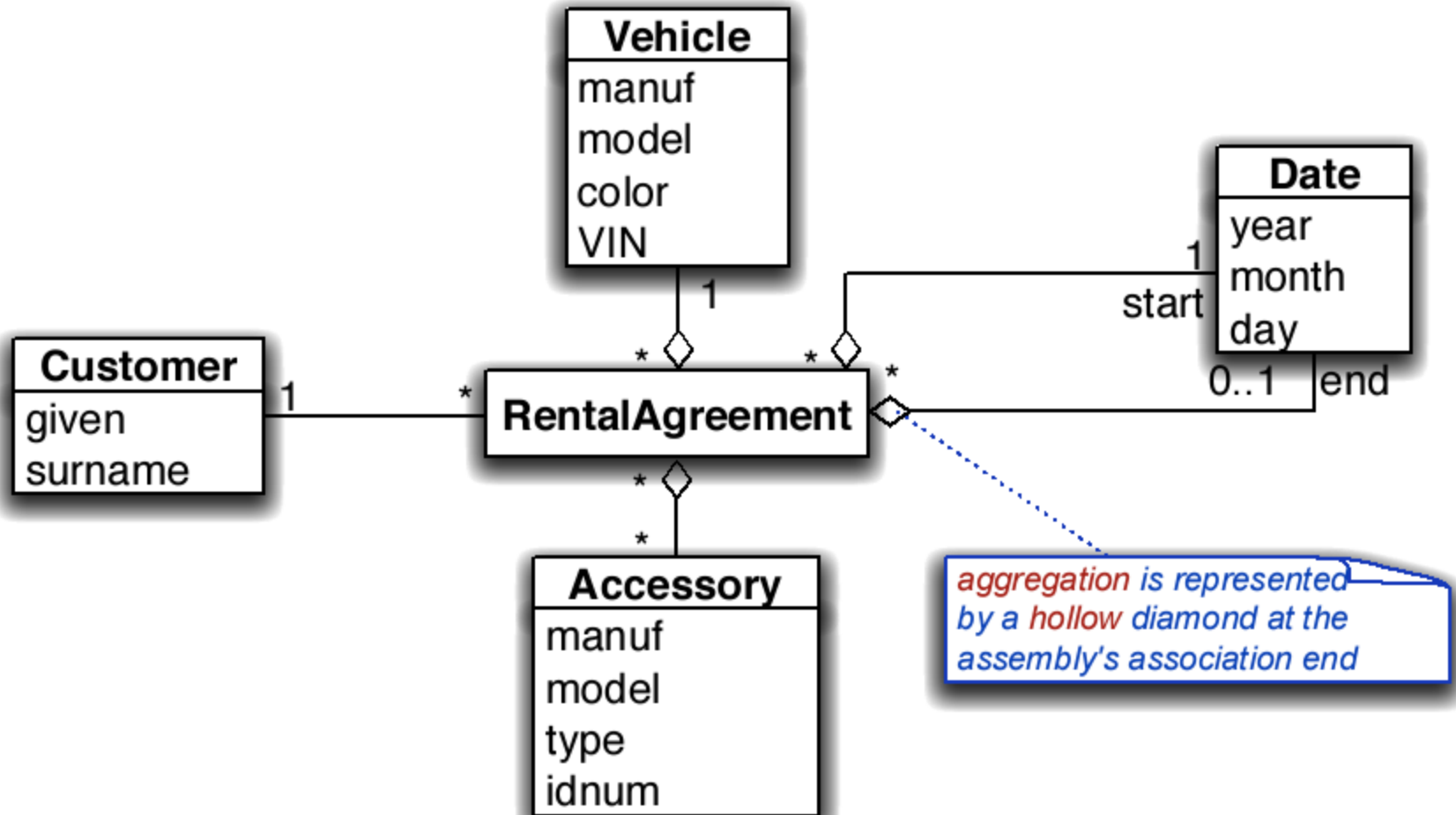


# Mối quan hệ liên kết (link)

- Actor: Một đối tượng có thể hoạt động trên các đối tượng khác chứ không bị thao tác bởi các đối tượng khác.
- Server: Một đối tượng không bao giờ hoạt động trên các đối tượng khác; nó chỉ có thể bị thao tác bởi các đối tượng khác.
- Agent: Là đối tượng vừa có thể hoạt động trên các đối tượng khác, lại vừa có thể bị các đối tượng khác thao tác.

# Mối quan hệ kết tập (aggregation)

- Mối quan hệ kết tập chỉ là một dạng đặc biệt của mối quan hệ liên hợp trong đó một đối tượng là sự tổng hợp của các đối tượng thành phần.
- Ví dụ một chiếc xe ô tô có 4 bánh, một cần lái, một hộp số, một động cơ ...



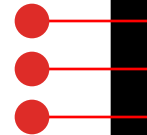
# Nội dung

1. Trừu tượng hóa dữ liệu
2. Đóng gói và xây dựng lớp
3. Tạo và sử dụng đối tượng

## 2.1. Đóng gói (Encapsulation)

- Một đối tượng có hai khung nhìn:
  - Bên trong: Chi tiết về các thuộc tính và các phương thức của lớp tương ứng với đối tượng
  - Bên ngoài: Các dịch vụ mà một đối tượng có thể cung cấp và cách đối tượng đó tương tác với phần còn lại của hệ thống

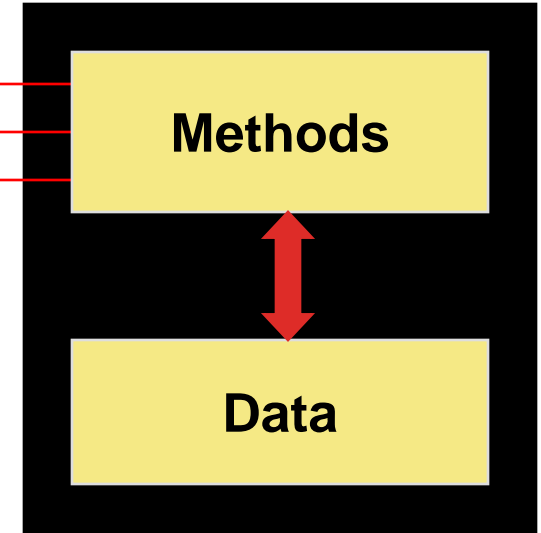
**Client**



**Methods**

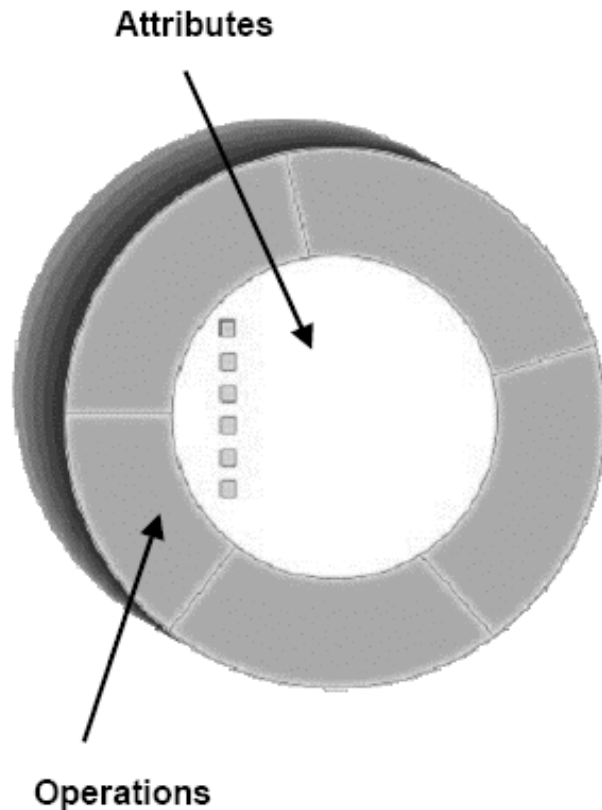


**Data**



## 2.1. Đóng gói (2)

- Dữ liệu/thuộc tính và hành vi/phương thức được đóng gói trong một lớp → Encapsulation



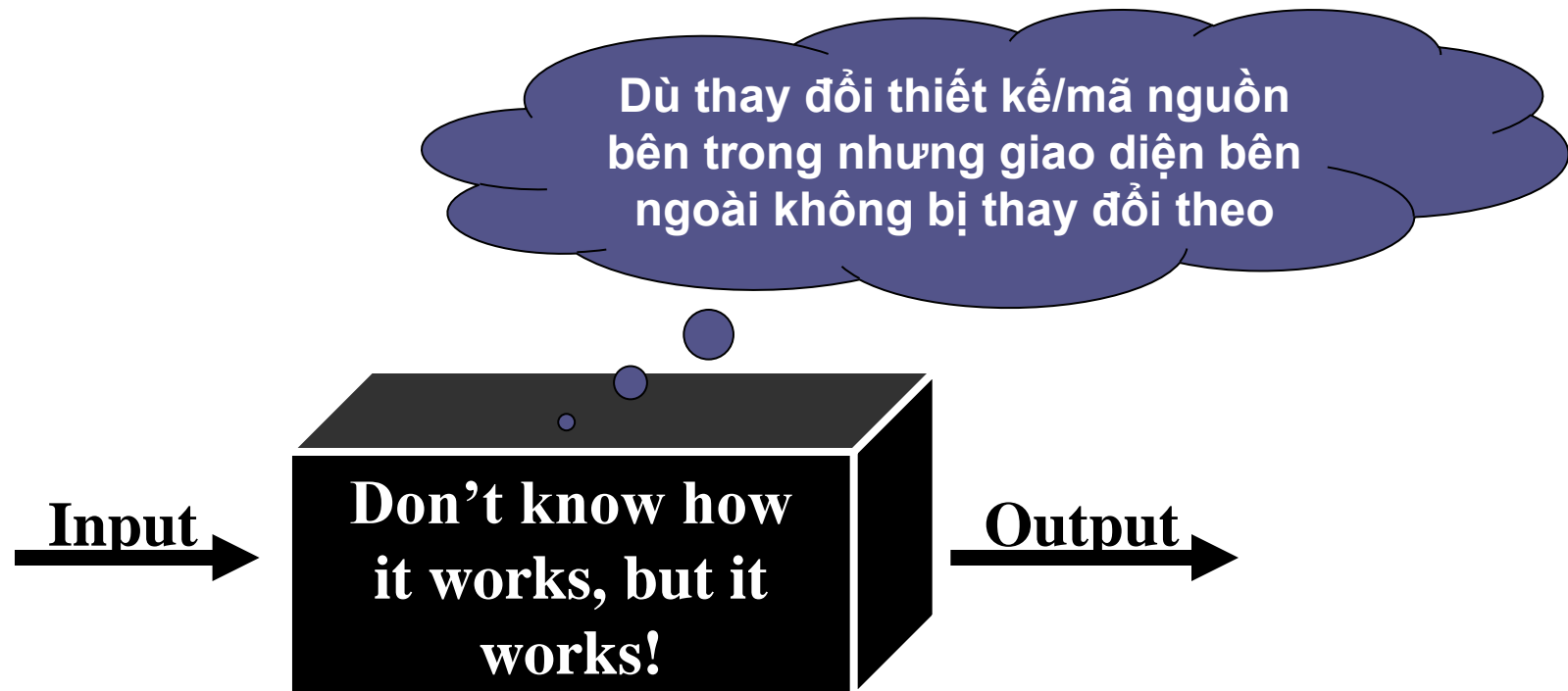
### BankAccount

- owner: String  
- balance: double

+ debit(double): boolean  
+ credit(double)

## 2.1. Đóng gói (3)

- Một đối tượng là một thực thể được đóng gói, cung cấp tập các dịch vụ nhất định
- Một đối tượng được đóng gói có thể được xem như một hộp đen – các công việc bên trong là ẩn so với client



## 2.2. Xây dựng lớp

- Thông tin cần thiết để định nghĩa một lớp
  - *Tên (Name)*
    - Tên lớp nên mô tả đối tượng trong thế giới thật
    - Tên lớp nên là số ít, ngắn gọn, và xác định rõ ràng cho sự trừu tượng hóa.
  - *Danh sách các phần tử dữ liệu*
    - Các phần dữ liệu cần lấy ra khi trừu tượng hóa
  - *Danh sách các thông điệp*
    - Các thông điệp mà đối tượng đó có thể nhận được

### BankAccount

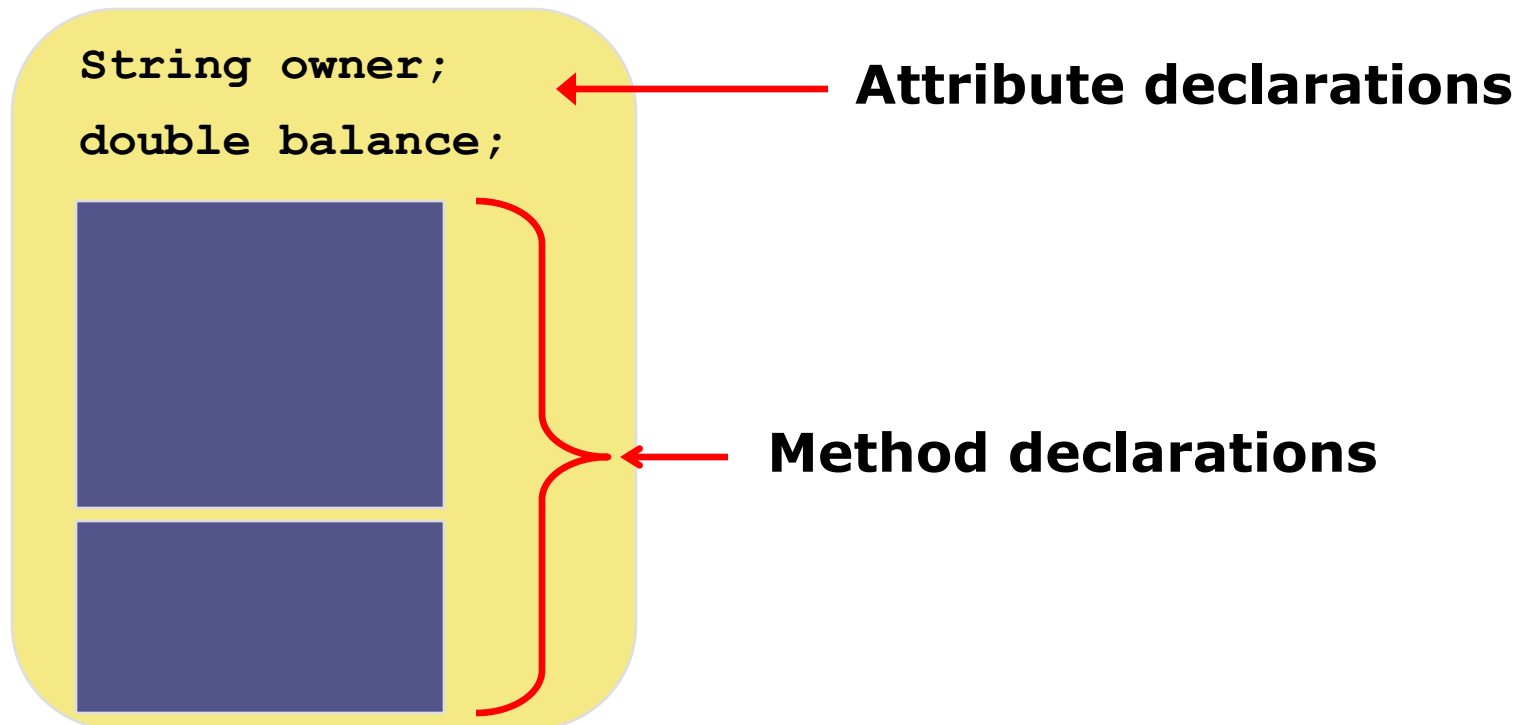
- owner: String
- balance: double

- + debit(double): boolean
- + credit(double)



## 2.2. Xây dựng lớp (2)

- Lớp đóng gói các thành viên (member)
  - Các trường/thuộc tính (field/attribute)
  - Các hàm/phương thức (function/method):



# C++: Lớp Point

```
class point {  
    double x,y;           // private data members  
public:  
    point (int xo, int yo);    // public methods  
    point () { x = 0; y = 0;}; // a constructor  
    void move (int dx, int dy);  
    void rotate (double alpha);  
    int distance (point p);  
};
```

# Lớp hình chữ nhật

```
class T_hcn {
    int x,y;
    single w,h;
public :
    void set_value(int,int,single,single);
    single area(void)
    {
        return (w*h);
    };
};
void T_hcn::set_value(int a,in b,single
    c,single d)
{x=a; y=b; w=c; h=d;};
```

```
int main() {
    T_hcn hcn;
    hcn.set_value(0,0,3
,4);
    cout << " dien tich
= " << hcn. area();
}
```

# Cài đặt phương thức (C++)

- Giao diện của phương thức luôn đặt trong định nghĩa lớp, cũng như các khai báo thành viên dữ liệu.
- Phần cài đặt (định nghĩa phương thức) có thể đặt trong định nghĩa lớp hoặc đặt ở ngoài.
- Trong:
  - Khai báo inline: phương thức được định nghĩa bên trong khai báo lớp
- Ngoài:
  - Khai báo thông thường: phương thức được định nghĩa ngoài thông báo lớp, và dùng toán tử phạm vi ::

# Phương thức C++

```
class Car {  
    ...  
    void drive(int speed,  
               int distance)  
    ;  
    ...  
    void Car::drive (int speed,  
                    int distance)  
    {...}
```

```
class Car {  
    ...  
    void drive(int speed,  
               int distance) {  
        // định nghĩa tại đây  
    }  
};
```

# C++: Lớp Account

```
class Account {  
public:  
    Account();  
    float account_balance() const;    // Return the balance  
    float withdraw( const float );    // Withdraw from account  
    void deposit( const float );      // Deposit into account  
    void set_min_balance( const float ); // Set minimum balance  
private:  
    float the_balance;                // The outstanding balance  
    float the_min_balance;            // The minimum balance  
};
```

# C++: Lớp Account

```
class Account {  
public:  
    Account();  
    float account_balance() const;    // Return the balance  
    float withdraw( const float );    // Withdraw from account  
    void deposit( const float );      // Deposit into account  
    void set_min_balance( const float ); // Set minimum balance  
private:  
    float the_balance;                // The outstanding balance  
    float the_min_balance;            // The minimum balance  
};
```

# C++: Lớp Account

```
Account::Account()
{
    the_balance = the_min_balance = 0.00;
}

float Account::account_balance() const
{
    return the_balance;
}

float Account::withdraw( const float money )
{
    if ( the_balance-money >= the_min_balance
        )
    {
        the_balance = the_balance - money;
        return money;
    } else { return 0.00; }
}
```

```
void Account::deposit( const float
money )
{
    the_balance = the_balance +
money;
}
```

```
void Account::set_min_balance(
const float money )
{
    the_min_balance = money;
}
```




# C++, tách đặc tả lớp và cài đặt phương thức

- Nói chung, ta nên tách phần khai báo phương thức ra khỏi phần cài đặt (định nghĩa),
- Việc phân tách này cho hai ích lợi quan trọng trong đóng gói:
  - Do tách định nghĩa khỏi phần khai báo lớp, người dùng không cần quan tâm đến chi tiết (một khai báo lớp sẽ dài và khó đọc như thế nào nếu nó kèm theo khoảng 10 đến 20 phương thức, mỗi phương thức dài hàng trăm dòng lệnh?)
  - Tách giao diện phương thức ra khỏi cài đặt cho phép ta thay đổi chi tiết cài đặt mà không ảnh hưởng đến người dùng.


# Lớp Rectangle (C++)

```
#include <iostream>
using namespace std;
class Rectangle
{
    private:
        float width, length;
    public:
        void setWidth(float);
        void setLength(float);
        float getWidth(),
            getLength(),
            getArea();
};
```



Đặc tả Lớp, với các nguyên mẫu  
hàm thành phần + dữ liệu

```
void Rectangle::setWidth(float w)
{width = w;}
void Rectangle::setLength(float
    len)
{length = len;}
float Rectangle::getWidth()
{return width;}
float Rectangle::getLength()
{return length;}
float Rectangle::getArea()
{return width * length;}
```



Định nghĩa hàm thành phần bên  
ngoài lớp

# Đặt khai báo lớp riêng rẽ (C++)

- Để đảm bảo tính đóng gói, ta thường đặt khai báo của lớp trong file header
  - tên file thường trùng với tên lớp. Ví dụ khai báo lớp Car đặt trong file “car.h”
- Phần cài đặt (định nghĩa) đặt trong một file nguồn tương ứng
  - “car.cpp” hoặc “car.cc”
- Quy ước đặt khai báo/định nghĩa của lớp trong file trùng tên lớp được chấp nhận rộng rãi trong C++
  - là quy tắc bắt buộc đối với các lớp của Java

# File header Car.h

```
// car.h
#ifndef CAR_H
#define CAR_H
class Car {
    public:
        //...
        void drive(int speed, int distance);
        //...
        void stop();
        //...
        void turnLeft();
    private:
        int vin; //...
        string make; //...
        string model; //...
        string color; //...
};
#endif
```

## 2.2. Xây dựng lớp (3) (Java)

- Các lớp được nhóm lại thành package
  - Package bao gồm một tập hợp các lớp có quan hệ logic với nhau,
  - Package được coi như các thư mục, là nơi tổ chức các lớp, giúp xác định vị trí dễ dàng và sử dụng các lớp một cách phù hợp.
- Ví dụ:
  - Một số package có sẵn của Java: java.lang, javax.swing, java.io...
  - Package có thể do ta tự đặt
    - Cách nhau bằng dấu “.”
    - Quy ước sử dụng ký tự thường để đặt tên package
    - Ví dụ: **package oop.k52.cnpm;**

## 2.2.1. Khai báo lớp

- Cú pháp khai báo:

```
package tenpackage;  
chi_dinh_truy_cap class TenLop {  
    // Than lop  
}
```

- **chi\_dinh\_truy\_cap:**

- **public:** Lớp có thể được truy cập từ bất cứ đâu, kể cả bên ngoài package chứa lớp đó.
- **private:** Lớp chỉ có thể được truy cập trong phạm vi lớp đó
- Không có (mặc định): Lớp có thể được truy cập từ bên trong package chứa lớp đó.

## Ví dụ - Khai báo lớp

```
package oop.k52.cnpm;
```

```
public class Student {  
    ...  
}
```

## 2.2.2. Khai báo thành viên của lớp

- Các thành viên của lớp cũng có chỉ định truy cập tương tự như lớp.

	<code>public</code>	Không có	<code>private</code>
Cùng lớp			
Cùng gói			
Khác gói			



## 2.2.2. Khai báo thành viên của lớp (2)

- Các thành viên của lớp cũng có chỉ định truy cập tương tự như lớp.

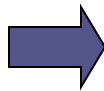
	<b>public</b>	<b>Không có</b>	<b>private</b>
Cùng lớp	Yes	Yes	Yes
Cùng gói	Yes	Yes	No
Khác gói	Yes	No	No

## a. Thuộc tính

- Các thuộc tính phải được khai báo bên trong lớp
- Mỗi đối tượng có bản sao các thuộc tính của riêng nó
  - Giá trị của một thuộc tính thuộc các đối tượng khác nhau là khác nhau.

### Student

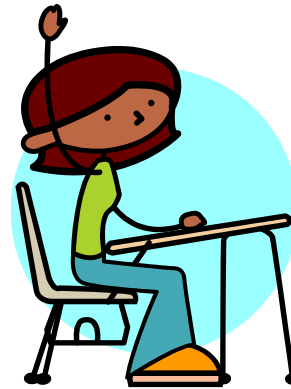
- name
- address
- studentID
- dateOfBirth



Nguyễn Hoàng Nam  
Hà Nội...



Nguyễn Thu Hương  
Hải Phòng...



...



## a. Thuộc tính (2)

- Thuộc tính có thể được khởi tạo khi khai báo
  - Các giá trị mặc định sẽ được sử dụng nếu không được khởi tạo.

access modifier

type

name

```
package com.megabank.models;  
  
public class BankAccount {  
    private String owner;  
    private double balance = 0.0;  
}
```

BankAccount

- owner: String  
- balance: double

+ debit(double): boolean  
+ credit(double)

## b. Phương thức

- Xác định cách một đối tượng đáp ứng lại thông điệp
- Phương thức xác định các hoạt động của lớp
- Bất kỳ phương thức nào cũng phải thuộc về một lớp nào đó

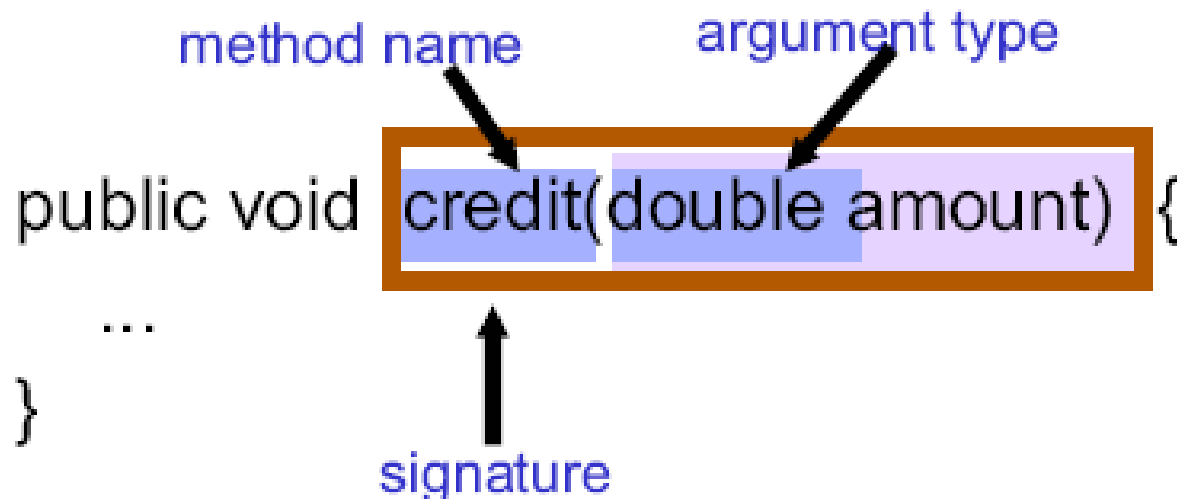
The diagram illustrates the components of a Java method signature. Four labels with arrows point to specific parts of the code snippet below:

- access modifier** points to `public`
- return type** points to `boolean`
- method name** points to `debit`
- parameter list** points to `(double amount)`

```
public boolean debit(double amount) {  
    // Method body  
    // Java code that implements method behavior  
}
```

## \* Chữ ký phương thức (signature)

- Mỗi phương thức phải có một chữ ký riêng gồm:
  - Tên phương thức
  - Số lượng các tham số và kiểu của chúng



The diagram illustrates the components of a method signature in a Java-like code snippet. The code is: `public void credit(double amount) {`. The text is enclosed in a brown rectangular box. Three labels with arrows point to specific parts: 'method name' points to 'credit', 'argument type' points to 'double', and 'signature' points to the entire 'credit(double amount)' expression. The word 'public' is followed by 'void', and there are ellipses '...' and a closing brace '}' below the boxed part.

```
public void credit(double amount) {  
    ...  
}
```

## \* Kiểu dữ liệu trả về

- Khi phương thức trả về ít nhất một giá trị hoặc một đối tượng thì bắt buộc phải có câu lệnh return để trả điều khiển cho đối tượng gọi phương thức.
- Nếu phương thức không trả về 1 giá trị nào (void) và có thể không cần câu lệnh return
- Có thể có nhiều lệnh return trong một phương thức; câu lệnh đầu tiên mà chương trình gặp sẽ được thực thi.

## c. Thành viên hằng (Java)

- Một thuộc tính/phương thức không thể thay đổi giá trị/nội dung trong quá trình sử dụng.
- Cú pháp khai báo:

```
chi_dinh_truy_cap final kieu_du_lieu  
TEN_HANG = gia_tri;
```

- Ví dụ:

```
final double PI = 3.141592653589793;  
public final int VAL_THREE = 39;  
private final int[] A = { 1, 2, 3, 4, 5, 6 };
```

```
package com.megabank.models;  
public class BankAccount {  
    private String owner;  
    private double balance;  
  
    public boolean debit(double amount) {  
        if (amount > balance)  
            return false;  
        else {  
            balance -= amount; return true;  
        }  
    }  
  
    public void credit(double amount) {  
        balance += amount;  
    }  
}
```

### BankAccount

- owner: String
- balance: double
- + debit(double): boolean
- + credit(double)



# Đối tượng trong C++ và Java

- C++: đối tượng của một lớp được tạo ra tại dòng lệnh khai báo:
  - `Point p1;`
- Java: Câu lệnh khai báo một đối tượng thực chất chỉ tạo ra một tham chiếu, sẽ trỏ đến đối tượng thực sự khi gặp toán tử new
  - `Box x;`
  - `x = new Box();`
  - Các đối tượng được cấp phát động trong bộ nhớ heap

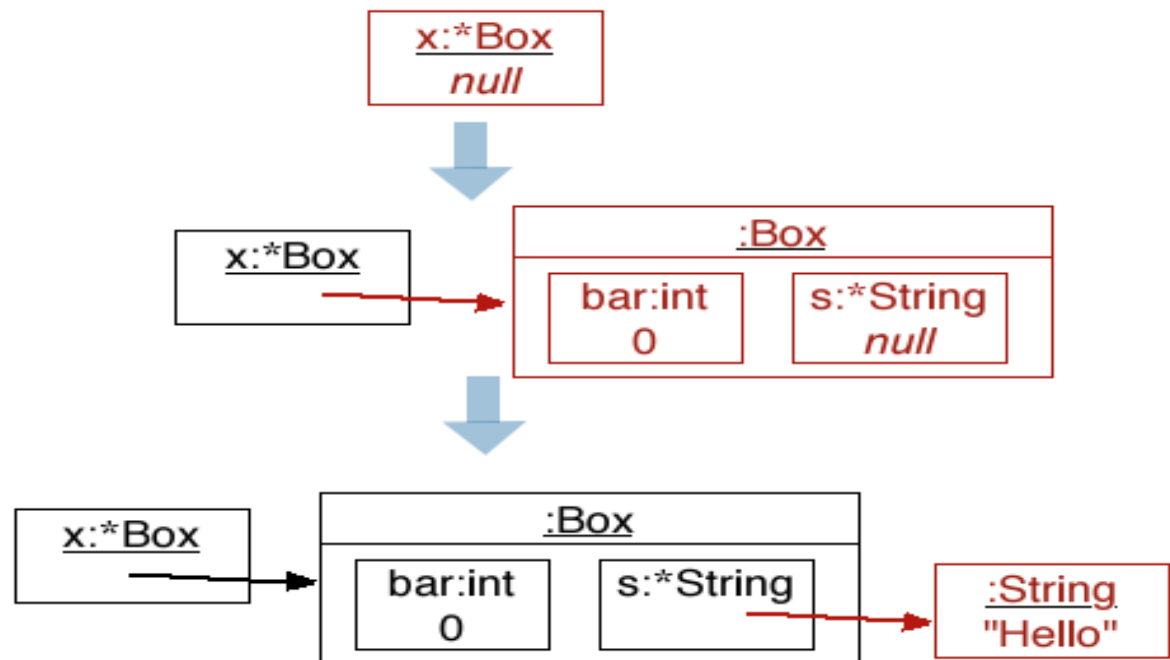
# Đối tượng trong Java

```
class Box
{
    int bar;
    String s;
}
```

Box x;

x = new Box ();

x.s = "Hello";



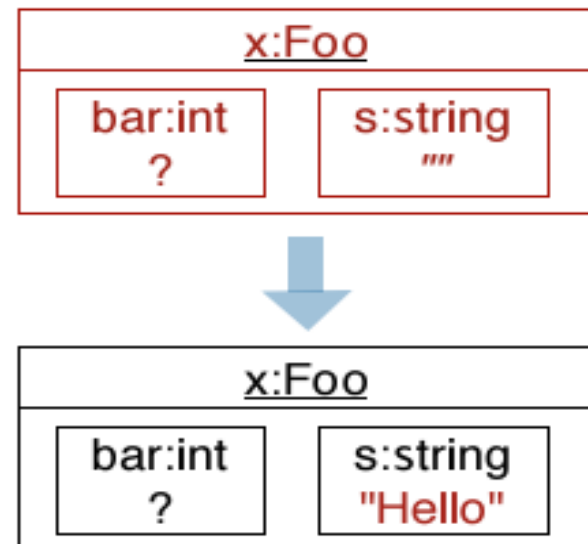
# Đối tượng trong C++

```
class Foo
{
    int bar;
    string s;
};
```

*gotta have that semi*

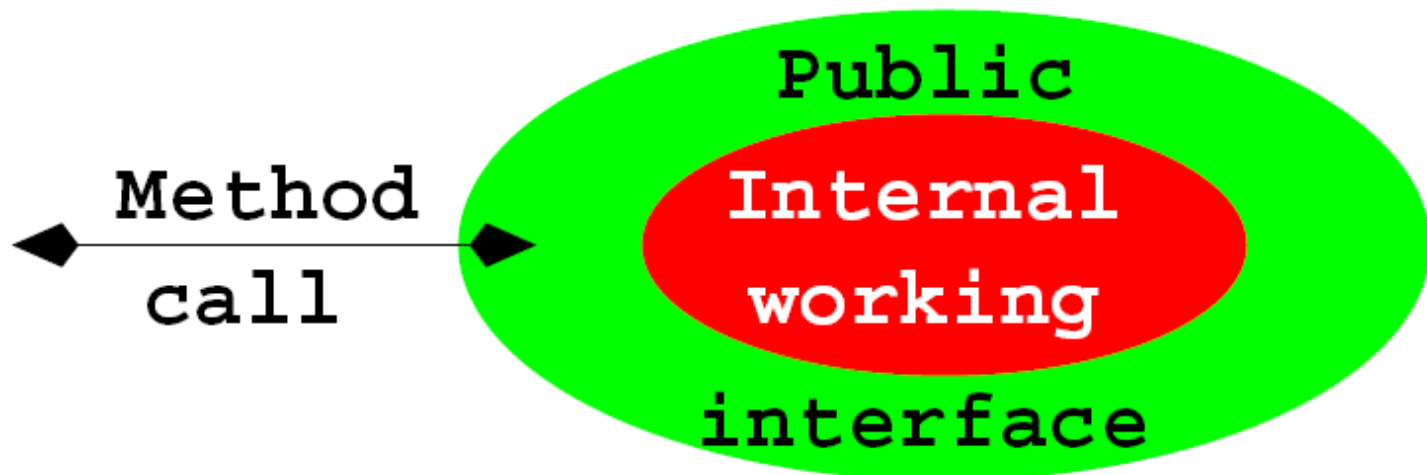
```
Foo x;
```

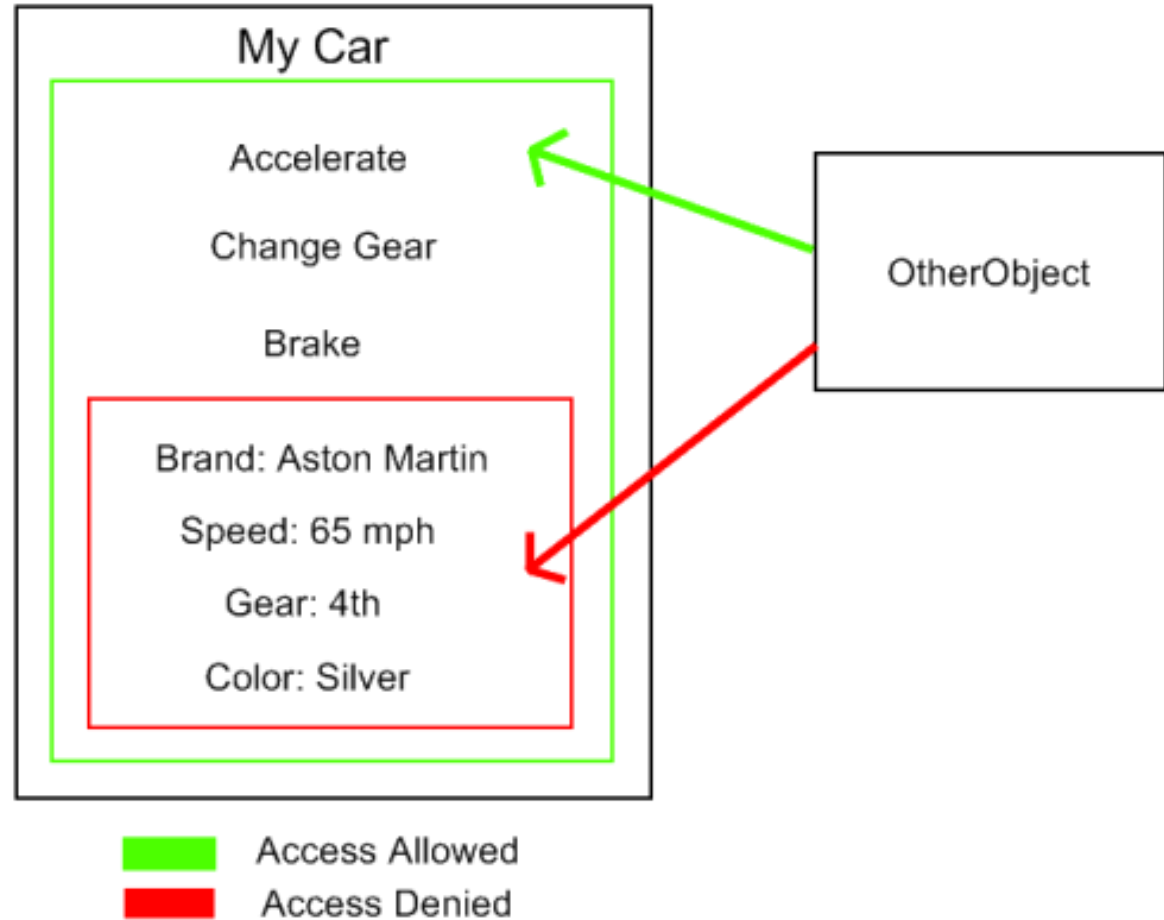
```
x.s = "Hello";
```



## 2.3. Che giấu dữ liệu (Data hiding)

- Dữ liệu được che giấu ở bên trong lớp và chỉ được truy cập và thay đổi ở các phương thức bên ngoài
  - Tránh thay đổi trái phép hoặc làm sai lệch dữ liệu



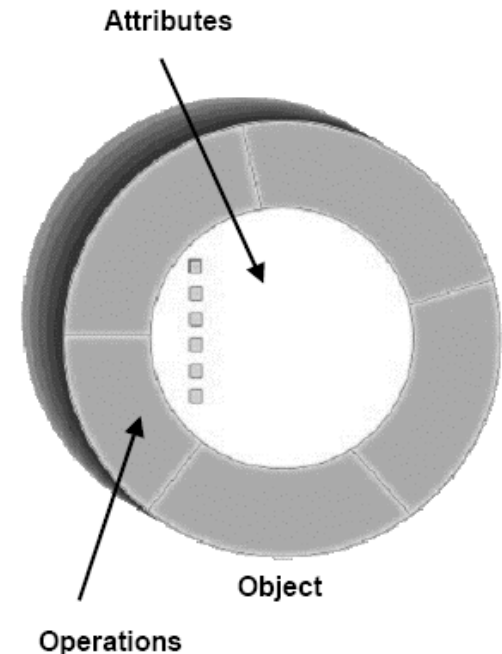


# Thảo luận

- Đóng gói dữ liệu có phải là che giấu dữ liệu không?

# Cơ chế che giấu dữ liệu

- Các thành viên dữ liệu
  - Chỉ có thể truy cập từ các phương thức bên trong lớp
  - Chỉ định truy cập là **private** để bảo vệ dữ liệu
- Các đối tượng khác muốn truy nhập vào dữ liệu riêng tư này phải thông qua các phương thức **public**



## BankAccount


- owner: String
- balance: double
+ debit(double): boolean
+ credit(double)

## Cơ chế che giấu dữ liệu (2)

- Vì dữ liệu là riêng tư → Thông thường một lớp cung cấp các dịch vụ để truy cập và chỉnh sửa các giá trị của dữ liệu
  - Accessor (getter): Trả về giá trị hiện tại của một thuộc tính (dữ liệu)
  - Mutator (setter): Thay đổi giá trị của một thuộc tính
  - Thường là getX và setX, trong đó x là tên thuộc tính

```
package com.megabank.models;  
  
public class BankAccount {  
    private String owner;  
    private double balance = 0.0;  
}
```

```
public String getOwner() {  
    return owner;  
}
```





# Phương thức Get (Truy vấn)

- Các phương thức truy vấn (query method, accessor) là các phương thức dùng để hỏi về giá trị của các thành viên dữ liệu của một đối tượng
- Có nhiều loại câu hỏi truy vấn có thể:
  - truy vấn đơn giản (*“giá trị của x là bao nhiêu?”*)
  - truy vấn điều kiện (*“thành viên x có lớn hơn 10 không?”*)
  - truy vấn dẫn xuất (*“tổng giá trị của các thành viên x và y là bao nhiêu?”*)
- Đặc điểm quan trọng của phương thức truy vấn là nó không nên thay đổi trạng thái hiện tại của đối tượng
  - không thay đổi giá trị của thành viên dữ liệu nào.

*restricted access: private*  
members are *not*  
*externally accessible*; but  
we need to know and  
modify their values

*set methods: public*  
methods that allow  
clients to *modify*  
*private* data; also  
known as *mutators*

```
public class Time {
    private int hour;
    private int minute;
    private int second;

    public Time () {
        setTime(0, 0, 0);
    }
```

```
    public void setHour (int h) { hour = ( ( h >= 0 && h < 24 ) ? h : 0 ); }
```

```
    public void setMinute (int m) { minute = ( ( m >= 0 && m < 60 ) ? m : 0 ); }
```

```
    public void setSecond (int s) { second = ( ( s >= 0 && s < 60 ) ? s : 0 ); }
```

```
    public void setTime (int h, int m, int s) {
        setHour(h);
        setMinute(m);
        setSecond(s);
    }
```

```
    public int getHour () { return hour; }
```

```
    public int getMinute () { return minute; }
```

```
    public int getSecond () { return second; }
```

```
}
```

*get methods: public*  
methods that allow  
clients to *read private*  
data; also known as  
*accessors*

# Bài tập 1

- Viết mã nguồn cho lớp NhanVien như trong hình bên biết:

- $Lương = Lương\ cơ\ bản * Hệ\ số\ lương$
- Phương thức inTTin() hiển thị thông tin của đối tượng NhanVien tương ứng.

- Phương thức tangLuong(double) tăng hệ số lương hiện tại lên một lượng bằng giá trị tham số double truyền vào. Nếu điều này làm cho lương của nhân viên > lương tối đa cho phép thì không cho phép thay đổi, in ra thông báo và trả về false, ngược lại trả về true.

- Viết các phương thức get và set cho các thuộc tính của lớp NhanVien.

## NhanVien

-tenNhanVien: String

-luongCoBan: double

-heSoLuong: double

+LUONG\_MAX: double

+tangLuong(double): boolean

+tinhLuong(): double

+inTTin()

# Nội dung

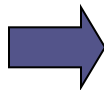
1. Trừu tượng hóa dữ liệu
2. Đóng gói và xây dựng lớp
3. Tạo và sử dụng đối tượng

## 3.1. Khởi tạo dữ liệu

- Dữ liệu cần được khởi tạo trước khi sử dụng
  - Lỗi khởi tạo là một trong các lỗi phổ biến
- Với kiểu dữ liệu đơn giản, sử dụng toán tử =
- Với đối tượng → Cần dùng phương thức khởi tạo

### Student

- name
- address
- studentID
- dateOfBirth



Nguyễn Hoàng Nam

Hà Nội...



Nguyễn Thu Hương

Hải Phòng...



...



# Khởi tạo và hủy bỏ đối tượng

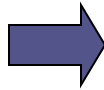
- Mỗi đối tượng khi tồn tại và hoạt động được hệ điều hành cấp phát một vùng nhớ để lưu lại các giá trị của dữ liệu thành phần
- Khi tạo ra đối tượng HĐH sẽ gán giá trị khởi tạo cho các dữ liệu thành phần
  - Phải được thực hiện tự động trước khi người lập trình có thể tác động lên đối tượng
  - Sử dụng hàm/phương thức khởi tạo
- Ngược lại khi kết thúc cần phải giải phóng hợp lý tất cả các bộ nhớ đã cấp phát cho đối tượng.
  - Java: JVM
  - C++: Hàm hủy (destructor)

## 3.2. Phương thức khởi tạo

- Là phương thức đặc biệt được gọi tự động khi tạo ra đối tượng
- Mục đích chính: Khởi tạo cho các thuộc tính của đối tượng

Student

- name
- address
- studentID
- dateOfBirth



Nguyễn Hoàng Nam  
Hà Nội...



Nguyễn Thu Hương  
Hải Phòng...



...



## 3.2. Phương thức khởi tạo (2)

- Mỗi lớp phải chứa ít nhất một constructor
  - Có nhiệm vụ tạo ra một thể hiện mới của lớp
  - Tên của constructor trùng với tên của lớp
  - Constructor không có kiểu dữ liệu trả về
- Ví dụ:

```
public BankAccount(String o, double b) {  
    owner = o;  
    balance = b;  
}
```



## 3.2. Phương thức khởi tạo (3)

- Phương thức khởi tạo **có thể dùng** các chỉ định truy cập
  - **public**
  - **private**
  - Không có (mặc định – phạm vi package)
- Một phương thức khởi tạo **không thể dùng** các từ khóa **abstract, static, final, native, synchronized**.
- Các phương thức khởi tạo không được xem như là *thành viên của lớp*.

## 3.2. Phương thức khởi tạo (4)

- Phương khởi tạo mặc định (default constructor)
  - Là phương thức khởi tạo **KHÔNG THAM SỐ**

```
public BankAccount() {  
    owner = "noname"; balance = 100000;  
}
```
  - Nếu ta không viết một phương khởi tạo nào trong lớp
    - JVM mới cung cấp phương thức khởi tạo mặc định
    - Phương thức khởi tạo mặc định do JVM cung cấp có chỉ định truy cập giống như lớp của nó
  - Một lớp nên có phương thức khởi tạo mặc định

### 3.3. Khai báo và khởi tạo đối tượng

- Đối tượng được tạo ra, thể hiện hóa (instantiate) từ một mẫu chung (lớp).
- Các đối tượng phải được khai báo *kiểu* của đối tượng trước khi sử dụng:
  - Kiểu của đối tượng là lớp các đối tượng
  - Ví dụ:
    - `String strName;`
    - `BankAccount acc;`

### 3.3. Khai báo và khởi tạo đối tượng (2)

- Đối tượng cần được khởi tạo trước khi sử dụng
  - Sử dụng toán tử = để gán
  - Sử dụng từ khóa **new** với constructor để khởi tạo đối tượng:
    - Từ khóa **new** dùng để tạo ra một đối tượng mới
    - Tự động gọi phương thức khởi tạo tương ứng
  - Một đối tượng được khởi tạo mặc định là **null**
- Đối tượng được thao tác thông qua *tham chiếu* (~ con trỏ).
- Ví dụ:

```
BankAccount acc1;  
acc1 = new BankAccount();
```

### 3.3. Khai báo và khởi tạo đối tượng (3)

- Có thể kết hợp vừa khai báo vào khởi tạo đối tượng

- **Cú pháp:**

```
Ten_lop ten_doi_tuong = new  
                        Pthuc_khoi_tao(ds_tham_so) ;
```

- **Ví dụ:**

```
BankAccount account = new BankAccount() ;
```

### 3.3. Khai báo và khởi tạo đối tượng (4)

- Phương thức khởi tạo **không có giá trị trả về**, nhưng khi sử dụng với từ khóa **new** trả về một tham chiếu đến đối tượng mới

```
public BankAccount(String name) {  
    setOwner(name);  
}
```

Constructor  
definition



```
BankAccount account = new BankAccount("Joe Smith");
```

Constructor use



### 3.3. Khai báo và khởi tạo đối tượng (5)

- Mảng các đối tượng được khai báo giống như mảng dữ liệu cơ bản
- Mảng các đối tượng được khởi tạo mặc định với giá trị **null**.
- Ví dụ:

```
Employee emp1 = new Employee(123456);
```

```
Employee emp2;
```

```
emp2 = emp1;
```

```
Department dept[] = new Department[100];
```

```
Test[] t = {new Test(1), new Test(2)};
```

## Ví dụ 1

```
class BankAccount{  
    private String owner;  
    private double balance;  
}  
  
public class Test{  
    public static void main(String args[]){  
        BankAccount acc1 = new BankAccount();  
    }  
}
```

→ Phương thức khởi tạo mặc định do Java cung cấp.



## Ví dụ 2

```
public class BackAccount{  
    private String owner;  
    private double balance;  
    public BankAccount(){  
        owner = "noname";  
    }  
}  
public class Test{  
    public static void main(String args[]){  
        BankAccount acc1 = new BankAccount();  
    }  
}
```

→ Phương thức khởi tạo mặc định tự viết.

## Ví dụ 3

```
public class BankAccount {
    private String owner;
    private double balance;
    public BankAccount(String name) {
        setOwner(name);
    }
    public void setOwner(String o) {
        owner = o;
    }
}

public class Test{
    public static void main(String args[]){
        BankAccount account1 = new BankAccount(); //Error
        BankAccount account2 = new BankAccount("Hoang");
    }
}
```

# Constructor với đối số ngầm định (C++)

```
class Automobile {  
public:  
    Automobile() ;  
  
    Automobile( string make, int doors,  
                int cylinders = 4, int engineSize =  
                2000 ) ;  
  
    Automobile( const Automobile & A ) ;  
    // copy constructor
```

# Hàm thiết lập sao chép

- Nhiệm vụ của hàm thiết lập sao chép là tạo đối tượng và sao chép nội dung từ một đối tượng đã có sang đối tượng vừa được tạo ra.
- Dạng khai báo của hàm thiết lập là :  
`<Name> (<type> &) ;`  
 hoặc `<Name> (const <type> &) ;`
- Từ khoá `const` trong khai báo tham số hình thức nhằm ngăn cấm mọi thay đổi nội dung của tham số truyền cho hàm.
- Ta cũng có thể tạo ra đối tượng mới giống đối tượng cũ một số đặc điểm, không hoàn toàn như phép gán. Đây là phương thức thiết lập có tham số là tham chiếu đến đối tượng thuộc chính lớp này.

# Hàm thiết lập sao chép

MyClass x(5);

MyClass y = x; **hoặc** MyClass y(x);

- C++ cung cấp sẵn một copy constructor, nó chỉ đơn giản copy từng thành viên dữ liệu từ đối tượng cũ sang đối tượng mới.
- Tuy nhiên, trong nhiều trường hợp, ta cần thực hiện các công việc khởi tạo khác trong copy constructor
  - Ví dụ: lấy giá trị cho một ID duy nhất từ đâu đó, hoặc thực hiện sao chép “sâu” (chẳng hạn khi một trong các thành viên là con trỏ giữ bộ nhớ cấp phát động)
- Trong trường hợp đó, ta có thể định nghĩa lại copy constructor

# Khai báo điển hình

```
Foo(const Foo& existingFoo);
```

tham số là đối tượng  
được sao chép

Kiểu tham số là tham chiếu  
đến đối tượng kiểu Foo

từ khoá const được dùng để đảm bảo đối  
tượng được sao chép sẽ không bị sửa đổi

```

class Person {
    public:
        Person(const char *nameO="", int
            ageO=0);
            Person(const Person &p);
            void print();
    private:
        char name[30];
        int age;
};

```

Sử dụng tường minh hàm thiết lập sao chép:

```

Person person("Matti", 20);
Person twinBrother(person);

```

```

Person::Person(const
    Person &p) {
    strcpy(name, p.name);
    age = p.age;
}

```

Sử dụng không tường minh:

```

void f(Person p);
void main(void) {
    Person person("Matti", 20);
    f(person);
}

```

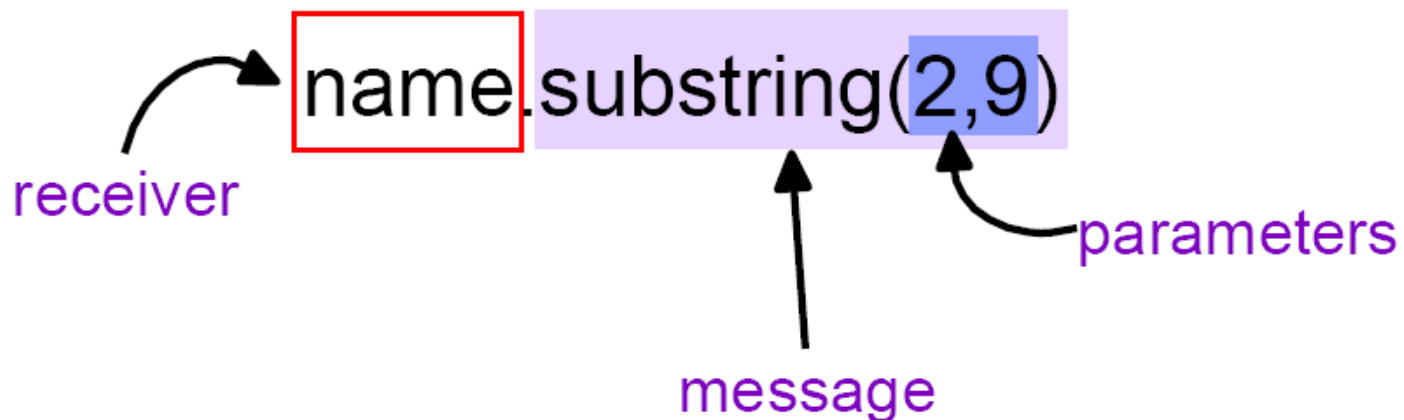
# Hàm thiết lập sao chép

- Chú ý vấn đề rò rỉ bộ nhớ khi viết code cho hàm tạo sao chép
- Trong Java, không có khái niệm copy constructor.



## 3.4. Sử dụng đối tượng

- Đối tượng cung cấp các hoạt động phức tạp hơn các kiểu dữ liệu nguyên thủy
- Đối tượng đáp ứng lại các thông điệp
  - Toán tử "." được sử dụng để gửi một thông điệp đến một đối tượng




## 3.4. Sử dụng đối tượng (2)

- Để gọi thành viên (dữ liệu hoặc thuộc tính) của lớp hoặc đối tượng, sử dụng toán tử “.”
- Nếu gọi phương thức ngay trong lớp thì toán tử “.” không cần thiết.

```
BankAccount account = new BankAccount();  
account.setOwner("Smith");  
account.credit(1000.0);  
System.out.println(account.getBalance());  
...
```

BankAccount method



```
public void credit(double amount) {  
    setBalance(getBalance() + amount);  
}
```

```
public class BankAccount{
    private String owner;
    private double balance;
    public BankAccount(String name) {
        setOwner(name) ;
    }
    public void setOwner(String o){ owner = o; }
    public String getOwner(){ return owner; }
}

public class Test{
    public static void main(String args[]){
        BankAccount acc1 = new BankAccount("");
        BankAccount acc2 = new BankAccount("Hong");
        acc1.setOwner("Hoa") ;
        System.out.println(acc1.getOwner()
                           + " " + acc2.getOwner() ) ;
    }
}
```

# Hàm hủy: Destructor (C++)

- Ngược lại với quá trình khởi tạo đối tượng, khi giải phóng đối tượng chúng ta phải giải phóng toàn bộ bộ nhớ đã được cấp phát cho đối tượng. Chức năng của hàm hủy sẽ thực hiện vai trò này:
- Ví dụ: 

```
class A {  
    int n;  
    public:  
        A(); //constructor  
        ~A(); // destructor  
};
```
- Java: không dùng hàm hủy.

# Hàm hủy

- Trước khi HDH giải phóng bộ nhớ đã cấp phát để lưu trữ các dữ liệu thành phần của đối tượng, sẽ thực hiện hàm hủy. Vì vậy chúng ta lưu ý khi xây dựng các lớp, trong hàm hủy chỉ cần giải phóng những gì mà HDH không tự động giải phóng cho chúng ta.
- Nguyên tắc cần lưu ý ở đây là:
  - Cần đặc biệt lưu ý tới các lớp trong đó có dữ liệu thành phần là các con trỏ
  - Không bỏ sót (không hiệu quả sử dụng bộ nhớ) và không giải phóng các bộ nhớ cấp phát hai lần (sẽ báo lỗi)

# Quá trình hủy DT: hàm hủy

```
#include <iostream>
#include <conio.h>
#include <stdio.h>

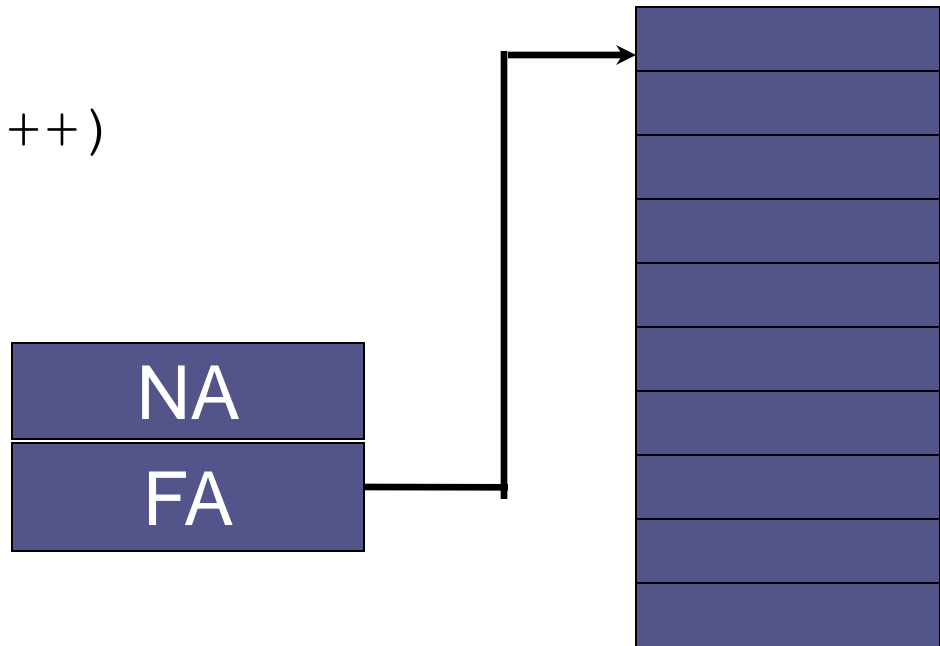
class A
{ int NA;
  float *FA;
public:
  A(int m);
  void display();
};
```

```
A::A(int m)
{
  NA=m;
  FA = new float [m];
  for (int i=0; i<m; i++)
    { FA[i]=i*10.0; }
}
```

# Quá trình hủy DT: hàm hủy

```
void A::display() {  
    for (int i=0; i<NA; i++)  
        { cout << FA[i];  
          cout << "  ";  
        }  
}
```

```
void main()  
{ A A1(5);  
  A1.display();  
}
```



*Giải phóng A1,  
mảng vẫn còn?*

*Giải quyết triệt để giải phóng bộ nhớ?*

# Hàm hủy

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>

class A
{ int NA;
  float *FA;
public:
  A(int m);
  void display();
  ~A();
};
```

```
A::A(int m)
{
  NA=m;
  FA = new float [m];
  for (int i=0; i<m;
i++)
    { FA[i]=i*10.0; }
}

A::~~A()
{ delete FA[];}
```



# Tự tham chiếu - this

- Cho phép truy cập vào đối tượng hiện tại của lớp.
- Quan trọng khi hàm/phương thức thành phần thao tác trên hai hay nhiều đối tượng.
- Xóa đi sự nhập nhằng giữa một biến cục bộ, tham số với thành phần dữ liệu của lớp
- Không dùng bên trong các khối lệnh static

# C++

```
int point::coincide(point pt)    {  
    return(this->x==pt.x && this->y==pt.y);  
}
```

```
void point::display()    {  
    cout<<"Dia chi : "<<this<<"Toa do :  
    "<<x<<" "<<y<<"\n";  
}
```

# Java

```
public class Person
{
    ...
    public void setName(String name)
    {
        this.name = name;
    }
    ...
    private String name;
    private int age;
} // end class Person
```

```
public class BankAccount{
    private String owner;
    private double balance;
    public BankAccount() { }
    public void setOwner(String owner){
        this.owner = owner;
    }
    public String getOwner(){ return owner; }
}

public class Test{
    public static void main(String args[]){
        BankAccount acc1 = new BankAccount();
        BankAccount acc2 = new BankAccount();
        acc1.setOwner("Hoa");
        acc2.setOwner("Hong");
        System.out.println(acc1.getOwner() + " " +
                           acc2.getOwner());
    }
}
```

## Bài tập 2

- Viết mã nguồn cho lớp NhanVien (đã làm)
- Viết phương thức khởi tạo với các tham số cần thiết để khởi tạo cho các thuộc tính của lớp NhanVien.
- Viết lớp TestNV trong đó tạo ra 2 đối tượng của lớp NhanVien, thực hiện truyền thông điệp đến các đối tượng vừa tạo để hiển thị thông tin, hiển thị lương, tăng lương...

### NhanVien

```
-tenNhanVien: String
-luongCoBan: double
-heSoLuong: double
+LUONG_MAX: double
+tangLuong(double): boolean
+ tinhLuong(): double
+ inTTin()
```