



# KHÁI NIỆM VÀ XÂY DỰNG LỚP

Thời gian: 4 tiết

# NỘI DUNG

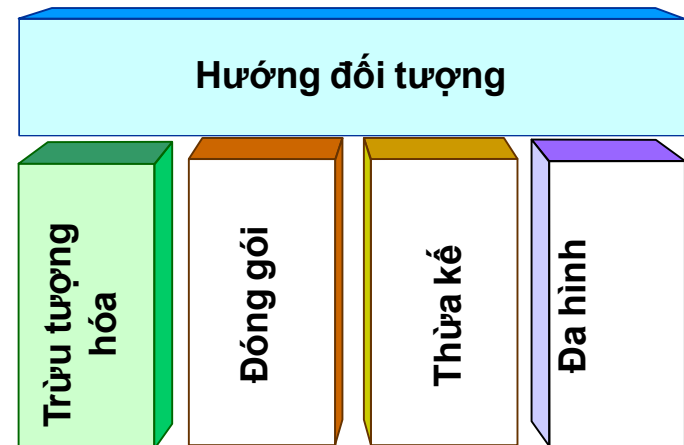
---

- ✓ Khái niệm về Lớp, Đối tượng
- ✓ Trừu tượng hóa
- ✓ Xây dựng lớp trên Java
- ✓ Khai báo và sử dụng đối tượng
- ✓ Thành viên tĩnh và hằng
- ✓ Hàm khởi tạo/hủy bỏ

- Là một trong 4 nguyên lý cơ bản của lập trình HĐT.

Là quá trình loại bỏ đi các thông tin ít quan trọng và giữ lại những thông tin quan trọng, có ý nghĩa với bài toán.

- 2 loại trừu tượng hóa
  - Trừu tượng hóa điều khiển
  - Trừu tượng hóa dữ liệu



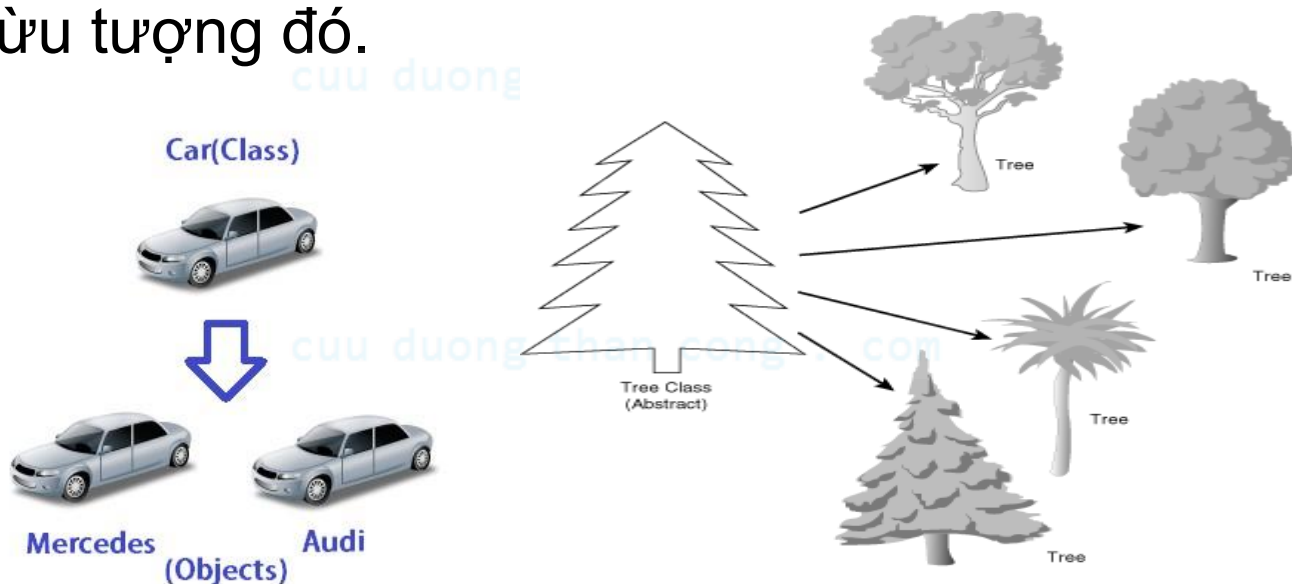
- Trừu tượng hóa điều khiển:
  - Bằng cách sử dụng các chương trình con (subprogram) và các luồng điều khiển (control flow)
  - Ví dụ:  $a := (1 + 2) * 5$ 
    - Nếu không có trừu tượng hóa điều khiển, LTV phải chỉ ra tất cả các thanh ghi, các bước tính toán mức nhị phân...
- Trừu tượng hóa dữ liệu:
  - Xử lý dữ liệu theo các cách khác nhau tùy bài toán



- Những thông tin về các "đối tượng" này?
  - Tất cả là điện thoại Nokia
  - Có loại nắp trượt, có loại nắp gập, có loại dạng bar
  - Một số điện thoại là dòng doanh nhân, một số dòng âm nhạc, 3G...
  - Bàn phím loại tiêu chuẩn, QWERTY hoặc không có bàn phím
  - Màu sắc, chất liệu, kích cỡ... khác nhau
  - v.v...
- Tùy bài toán, chỉ "trích rút" lấy những thông tin quan trọng, phù hợp

- Lớp (Class) là cách phân loại các đối tượng dựa trên đặc điểm chung của các đối tượng đó.
- Lớp chính là kết quả của quá trình *trừu tượng hóa dữ liệu*.
  - Lớp định nghĩa một *kiểu dữ liệu* mới, trừu tượng hóa một tập các đối tượng
  - Một đối tượng gọi là một *thể hiện* của lớp
- Lớp gồm các *phương thức* và *thuộc tính* chung của các đối tượng cùng một loại.

- ❖ Lớp là một mô tả trừu tượng của nhóm các đối tượng cùng bản chất, ngược lại mỗi một đối tượng là một thể hiện cụ thể cho những mô tả trừu tượng đó.



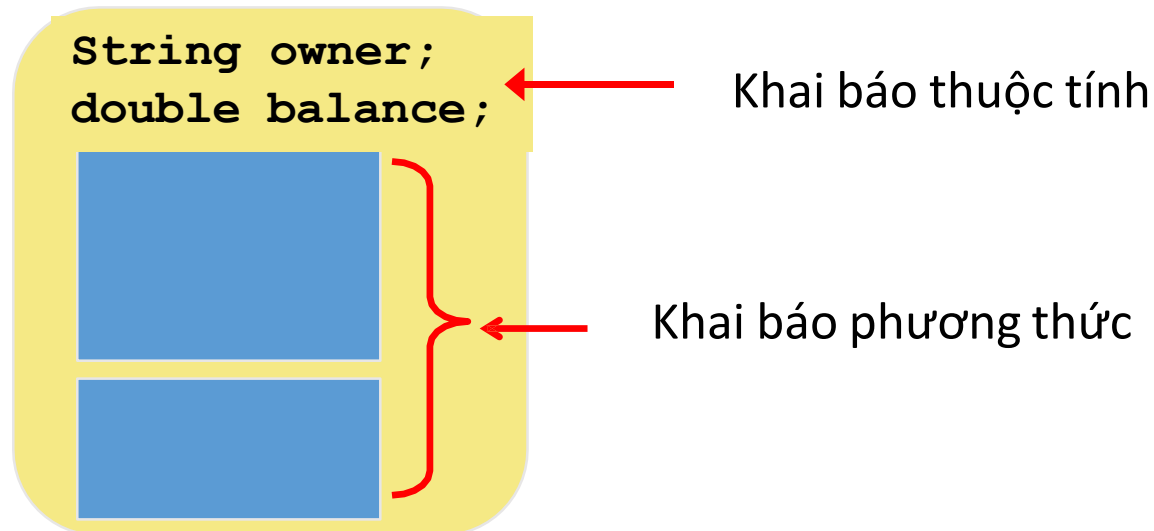
# Thuộc tính

- Thuộc tính
  - Một thuộc tính của một lớp là một trạng thái chung được đặt tên của tất cả các thể hiện của lớp đó có thể có.
  - Ví dụ: Lớp Ô tô có các thuộc tính
    - Màu sắc
    - Vận tốc
- Các thuộc tính của cũng là các giá trị trừu tượng.
- Mỗi đối tượng có bản sao các thuộc tính của riêng nó
  - Ví dụ: một chiếc Ô tô đang đi có thể có màu đen, vận tốc 60 km/h



- Phương thức:
  - Xác định các hoạt động chung mà tất cả các thể hiện của lớp có thể thực hiện được.
  - Xác định cách một đối tượng đáp ứng lại một thông điệp
- Thông thường các phương thức sẽ hoạt động trên các thuộc tính và thường làm thay đổi các trạng thái của đối tượng.
- Bất kỳ phương thức nào cũng phải thuộc về một lớp nào đó.
- Ví dụ: Lớp Ô tô có các phương thức
  - Tăng tốc
  - Giảm tốc

- 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 thuộc tính
  - Danh sách các phương thức



- Cú pháp: sử dụng từ khóa **class**

```
class Ten_Lop {  
    // Nội dung lớp  
    // Khai báo các thuộc tính  
    // Khai báo và cài đặt các phương thức  
}
```

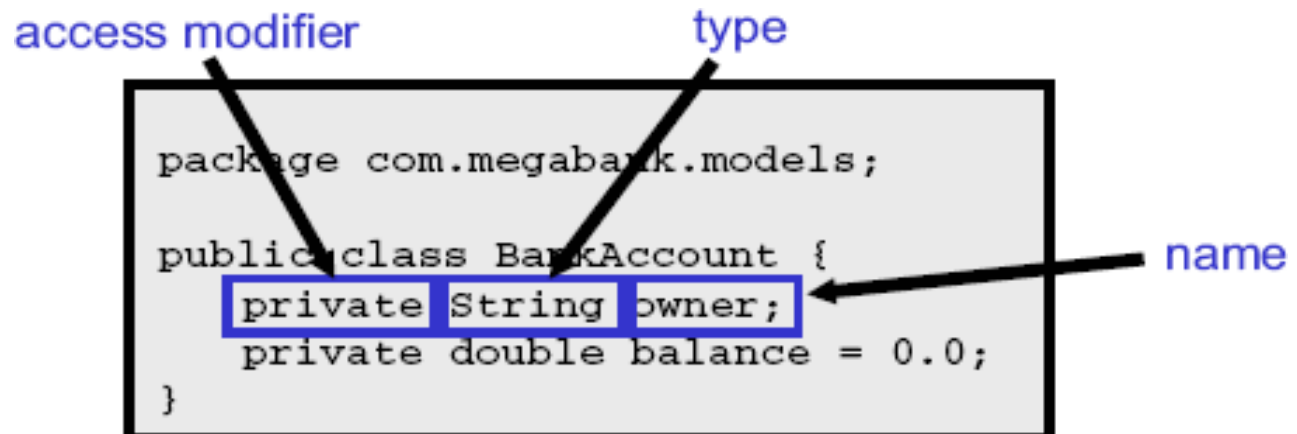
Ví dụ:

```
class Student {  
    // Nội dung lớp  
    ...  
}
```

- Cú pháp khai báo thuộc tính: Tương tự khai báo biến

`phamViTruyCap kieu tenThuocTinh;`

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



The diagram shows a Java code snippet with annotations. The code is enclosed in a black box. The annotations are as follows:

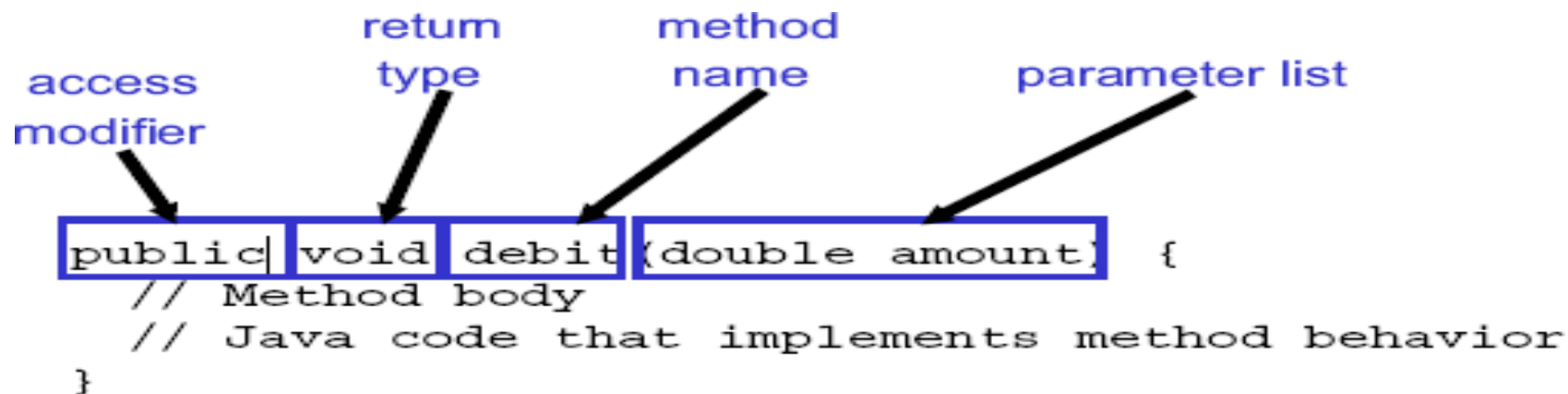
- access modifier**: Points to the word `public` in the class declaration.
- type**: Points to the word `String` in the field declaration.
- name**: Points to the word `owner` in the field declaration.

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

- Khai báo: tương tự khai báo hàm
- Cú pháp:

```
phạmViTruyCap kiểuTrảVề tênPhươngThức (ds tham số) {  
    // Nội dung phương thức  
}
```

- Ví dụ:



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

The diagram illustrates the components of the Java method declaration `public void debit(double amount) {` with arrows pointing to each part:

- access modifier**: `public`
- return type**: `void`
- method name**: `debit`
- parameter list**: `(double amount)`

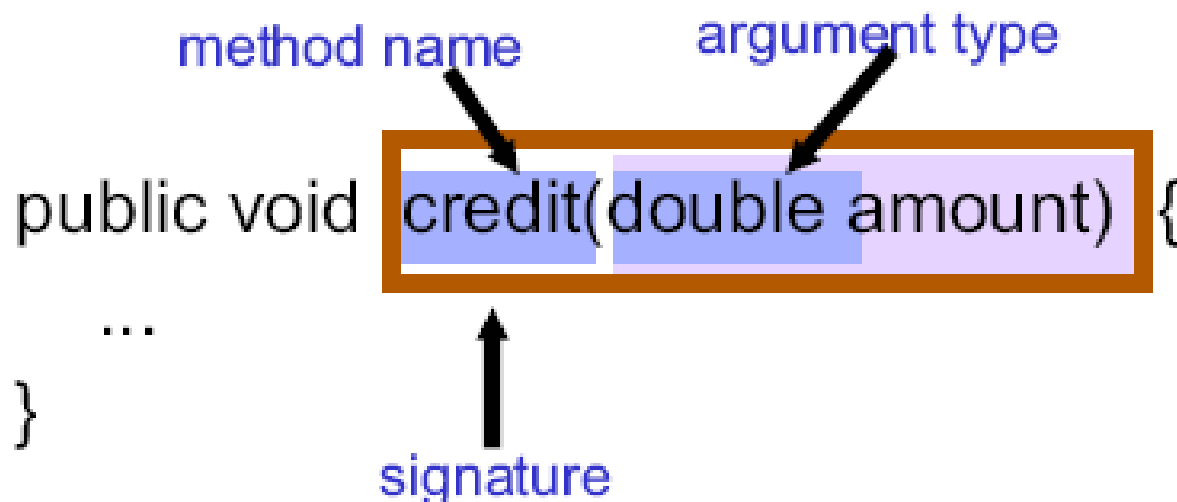
## Khai báo Phương thức

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

method name      argument type

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

signature



- 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 giá trị nào (**void**) 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.

## Ví dụ - Khai báo phương thức

```
public boolean checkOdd(int i) {  
    if (i %2 ==0)  
        return true;  
    else  
        return false;  
}
```

```
public boolean checkOdd(int i) {  
    return true;  
    return false; //error  
}
```



```
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;  
    }  
}
```

- Các phương thức truy vấn Get là các phương thức dùng để hỏi 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.

- Các phương thức thiết lập Set là các phương thức dùng để thay đổi giá trị các thành viên dữ liệu
- Ưu điểm của việc sử dụng các phương thức setter là kiểm soát tính hợp lệ của các thành phần dữ liệu
- Kiểm tra giá trị đầu vào trước khi gán vào các thuộc tính

## Ví dụ: phương thức get, set

```
class Student{  
    private String name;  
    public String getName() {  
        return this.name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
}
```

```
class Student{
    private String name;
    public String getName() {
        return this.name;
    }
    public void setName(String name)
    {
        this.name = name;
    }
}

class Manager{
    private Student[] students;
    public initialize()
    {
        students = new Student[10];
        students[0] = new Student();
        //students[0].name = "Hung"; error
        students[0].setName("Hung");
    }
}
```

- Trong Java
  - Các thành viên bình thường là thành viên thuộc về đối tượng
  - Thành viên thuộc về lớp được khai báo là static
- Cú pháp khai báo thành viên static:  
`chi_dinh_truy_cap static kieu_du_lieu tenBien;`
- Ví dụ:

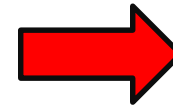
```
public class MyDate {  
    public static long getMillisSinceEpoch() {  
        ...  
    }  
}  
...  
long millis = MyDate.getMillisSinceEpoch();
```

# Thành viên static

- Thay đổi giá trị của một thành viên **static** trong một đối tượng của lớp sẽ thay đổi giá trị của thành viên này của tất cả các đối tượng khác của lớp đó.
- Các phương thức **static** chỉ có thể truy cập vào các thuộc tính **static** và chỉ có thể gọi các phương thức **static** trong cùng lớp.

# Ví dụ 1

```
class TestStatic{  
    public static int iStatic;  
    public int iNonStatic;  
}
```

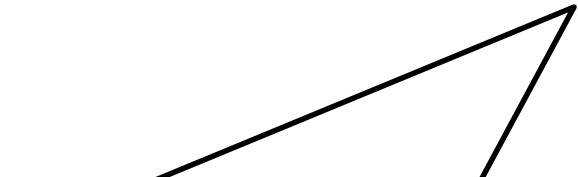


```
10,11  
10,0  
12,11
```

```
public class TestS {  
    public static void main(String[] args){  
        TestStatic obj1 = new TestStatic();  
        obj1.iStatic = 10; obj1.iNonStatic = 11;  
        System.out.println(obj1.iStatic + "," + obj1.iNonStatic);  
        TestStatic obj2 = new TestStatic();  
        System.out.println(obj2.iStatic + "," + obj2.iNonStatic);  
        obj2.iStatic = 12;  
        System.out.println(obj1.iStatic + "," + obj1.iNonStatic);  
    }  
}
```



```
public class Demo {  
    int i = 0;  
    void tang(){ i++; }  
    public static void main(String[] args) {  
        tang();  
        System.out.println("Gia tri cua i la" + i);  
    }  
}
```



non-static method tang() cannot be referenced from a static context

non-static variable i cannot be referenced from a static context

## ■ static có thể áp dụng cho:

- blocks
- variables
- methods
- nested classes

```
class Test
{
    // static variable
    static int a = 10;
    static int b;

    // static block
    static {
        System.out.println("Static block initialized.");
        b = a * 4;
    }

    public static void main(String[] args)
    {
        System.out.println("from main");
        System.out.println("Value of a : "+a);
        System.out.println("Value of b : "+b);
    }
}
```

# Thành viên hằng

- 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 };
```

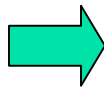
- Thông thường các hằng số liên quan đến lớp được khai báo là **static final** nhằm giúp truy cập dễ dàng

```
public class MyDate {  
    public static final long SECONDS_PER_YEAR =  
        31536000;  
    ...  
}  
...  
long years = MyDate.getMillisSinceEpoch() /  
    (1000*MyDate.SECONDS_PER_YEAR);
```

- 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 Văn A  
Hà Nội



Nguyễn Văn B  
Hải Phòng...



...



- 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)

## Các loại phương thức khởi tạo

- 2 loại phương thức khởi tạo
  - Phương thức khởi tạo mặc định (phương thức khởi tạo không tham số)
  - Phương thức khởi tạo có tham số

# Phương khởi tạo mặc định (default constructor)

- Là phương thức khởi tạo **KHÔNG THAM SỐ**

```
public BankAccount(){  
    owner = "noname";  
    balance = 100000;  
}
```

- Một lớp nên có phương thức khởi tạo mặc định



- Khi LTV không viết một phương khởi tạo nào trong lớp
  - JVM 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.

```
public class MyClass{  
    public static void main(String args) {  
        //...  
    }  
}
```

**MyClass.java**



```
public class MyClass{  
    public MyClass(){  
    }  
    public static void main(String args) {  
        //...  
    }  
}
```

**MyClass.class**

## Phương thức khởi tạo có tham số

- Một phương thức khởi tạo có thể có các tham số truyền vào
- Dùng khi muốn khởi tạo giá trị cho các thuộc tính
- Ví dụ:

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

# 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;`

- 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();
```

# Khai báo và khởi tạo đối tượng

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

- Cú pháp:

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

- Ví dụ:

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

## Khai báo và khởi tạo đối tượng

- 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

- 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(120);
```

```
Employee emp2;
```

```
emp2 = emp1;
```

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

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

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

→ 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");  
    }  
}
```



# Xin cảm ơn!

**TRẦN QUÝ NAM**

*Giảng viên Khoa CNTT  
namtq@dainam.edu.vn*