Bài 3

Xây dựng lớp

Trịnh Thành Trung

trungtt@soict.hust.edu.vn

Nội dung

1. Trừu tượng hóa dữ liệu
2. Lớp và thành phần của lớp
3. Xây dựng lớp
4. Tạo và sử dụng đối tượng
5. Thành viên hằng & tĩnh
6. Biểu đồ lớp

1

Trừu tượng hóa dữ liệu

Data abstraction

Trừu tượng hóa

- Abstraction
  - *"a concept or idea not associated with any specific instance"*
- Giảm thiểu và tinh lọc các chi tiết nhằm tập trung vào một số khái niệm/vấn đề quan tâm tại một thời điểm.
  - Ví dụ: Các định nghĩa toán học: Ký hiệu x được dùng để thể hiện cho các phép nhân

Trừu tượng hóa

- 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.

Trừu tượng hóa dữ liệu trong LTHĐT

- 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: Điện thoại Nokia

- Những thông tin gì có thể cảm nhận được khi nhìn các "đối tượng" này?
  - Tất cả là điện thoại Nokia
  - Các điện thoại này 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…

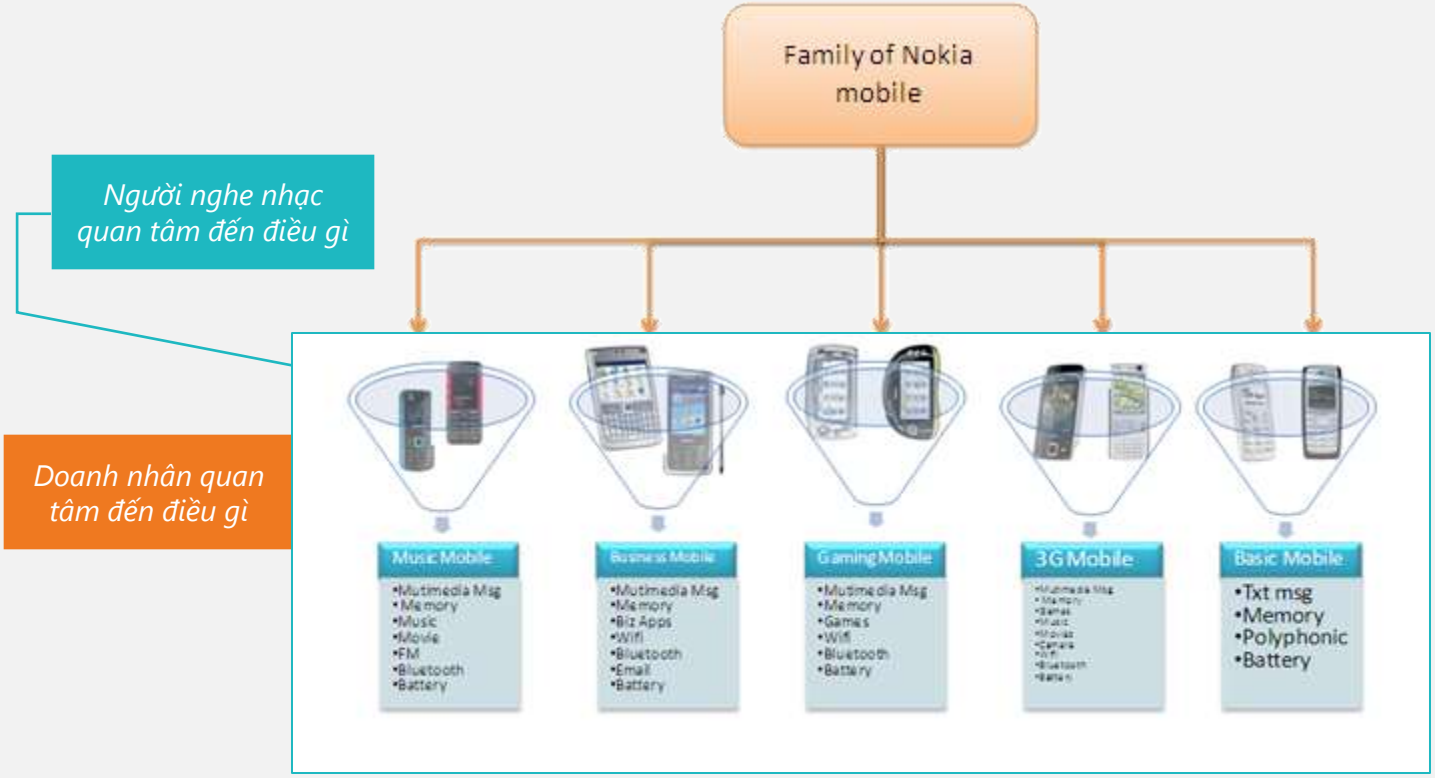
Ví dụ: Điện thoại Nokia

- Tổ chức là đối tượng điện thoại này vào các đặc tính chung

  – Vẫn còn rất khái quát

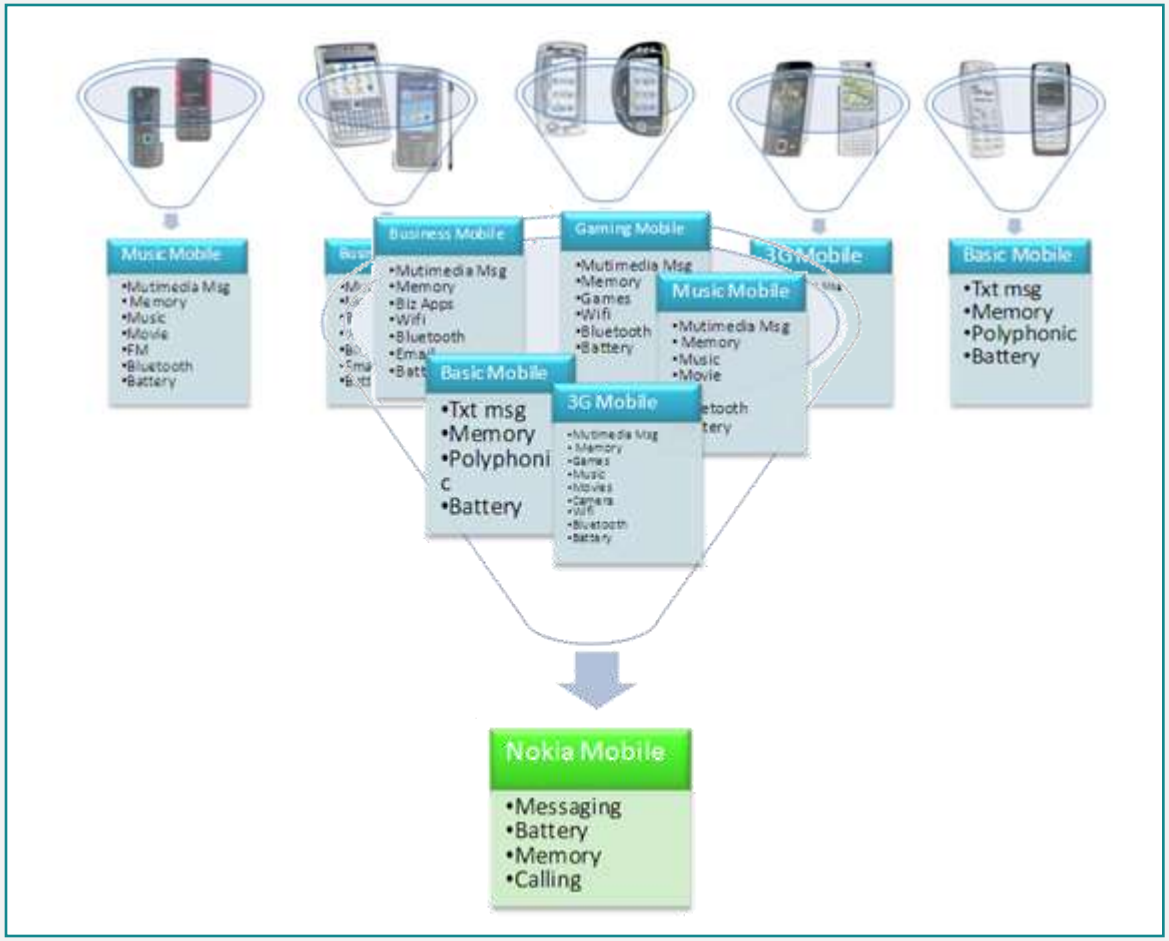
Ví dụ: Điện thoại Nokia

- Chia thành các danh mục nhỏ hơn
  - Ví dụ: Theo chức năng
- Xác định các đặc tính riêng



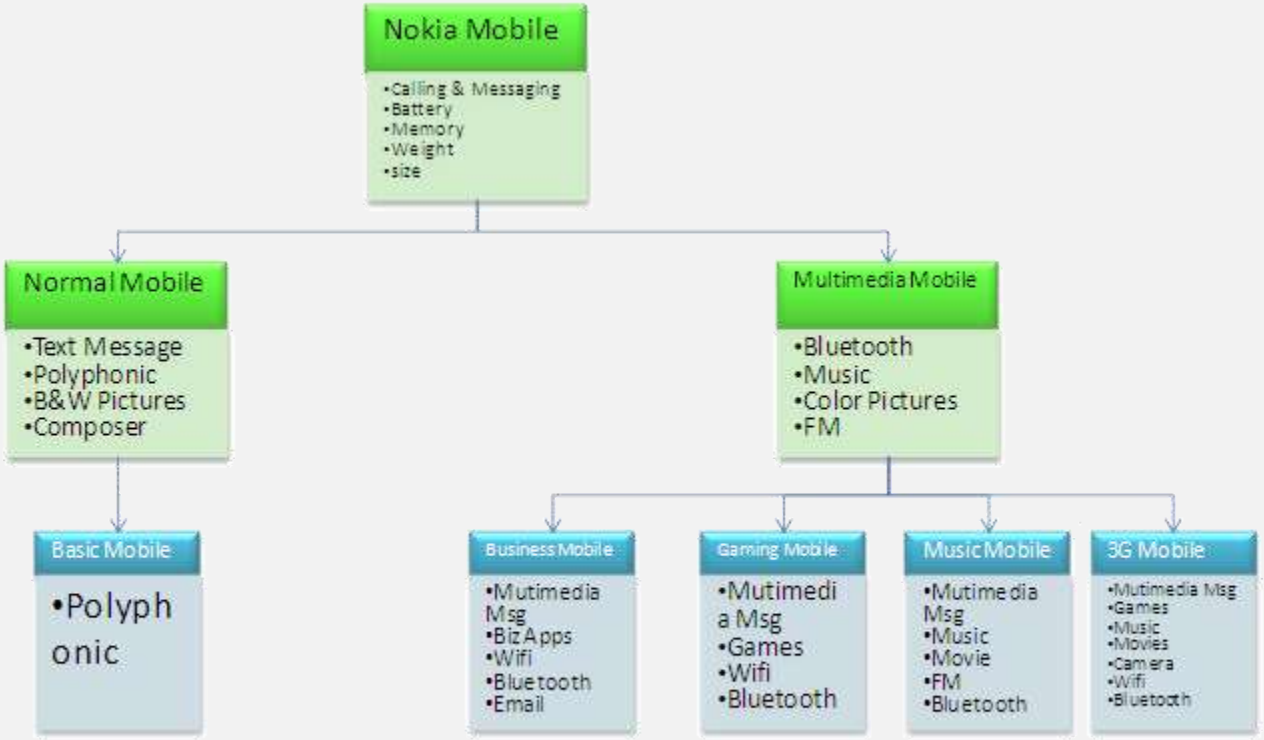
Ví dụ: Điện thoại Nokia

- Quay lại khái quát



Ví dụ: Điện thoại Nokia

- Trừu tượng hóa các đối tượng điện thoại Nokia



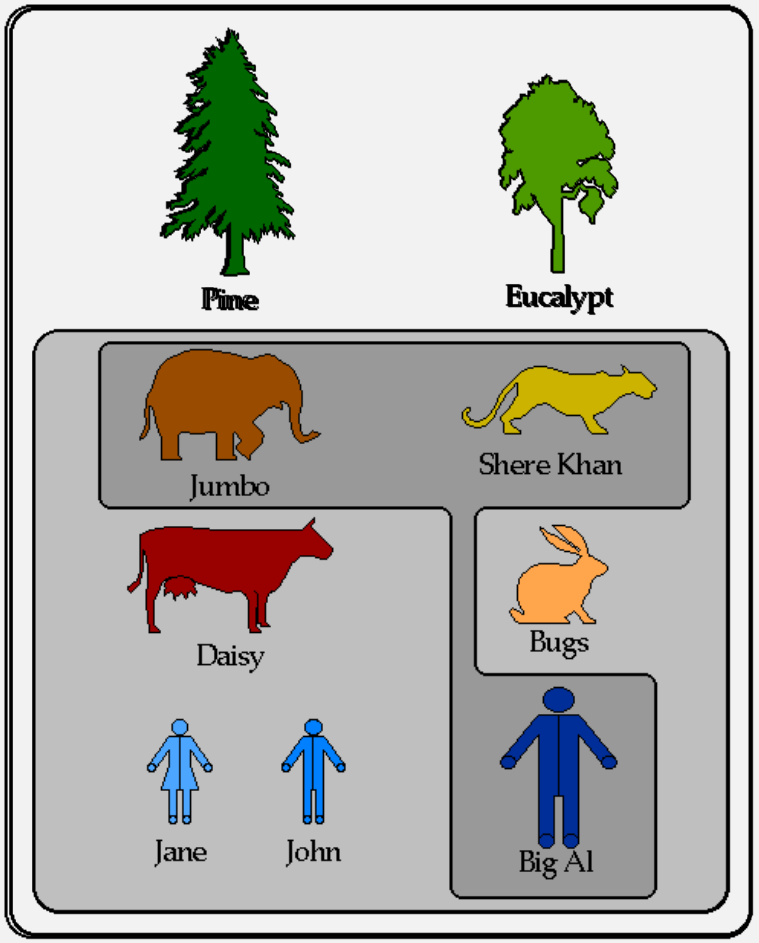
Ví dụ

- Trừu tượng hóa các đối tượng

Chưa phân loại

Sinh vật – Động vật – Loài người

Sinh vật – Động vật – Động vật nguy hiểm



2

Lớp và thành phần của lớp

Các khái niệm chung về lớp và các thành phần của lớp

Lớp

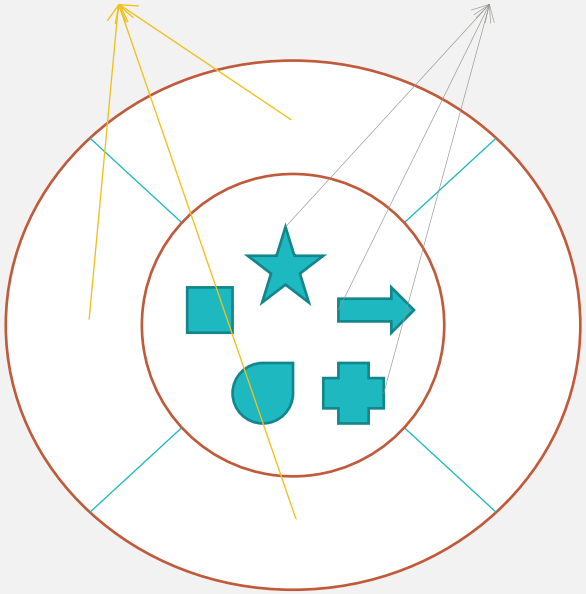
- Lớp (**Class**) là cách phân loại (*classify*) các đối tượng dựa trên đặc điểm chung của các đối tượng đó.
- Lớp có thể coi là khuôn mẫu để tạo các đối tượng
  - Ví dụ: Người, Sinh Vật, Màu sắc…
- Lớp chính là kết quả của quá trình *trừu tượng hóa dữ liệu*
  - Lớp định nghĩa một *kiểu dữ liệu* mới, trừu tượng hóa một tập các đối tượng
  - Một đối tượng gọi là một *thể hiện* của lớp

Các thành phần của lớp

- Lớp đóng gói các *phương thức* và *thuộc tính* chung của các đối tượng cùng một loại

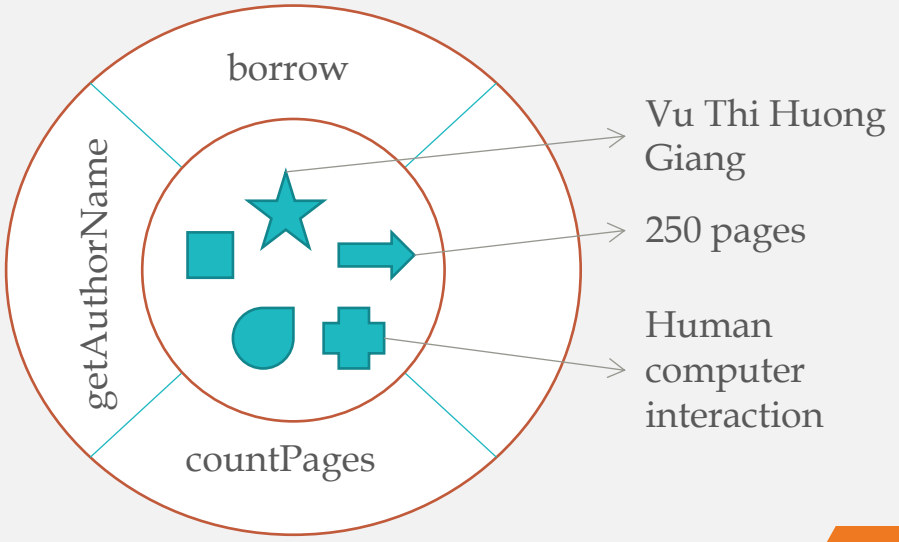
Phương thức: các hành vi đối tượng có thể thực hiện

Thuộc tính: các thông tin liên quan đến thuộc tính



Class

Thể hiện: Một đối tượng cụ thể
Thuộc tính thể hiện: Những giá trị gán cho các thuộc tính của một đối tượng cụ thể



Object MyBook

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

- 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 lớp.
  - 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

Phạm vi

- Phạm vi nhìn thấy được xác định khả năng nhìn thấy được của một thành phần của chương trình với các thành phần khác của chương trình
- Đối với lớp
  - Phạm vi nhìn thấy được có thể được áp dụng cho các thành phần của lớp
    + private: chỉ truy cập được bên trong lớp đó
    + public: có thể truy cập được tại mọi nơi

3

Xây dựng lớp

Xây dựng các lớp cùng các thuộc tính và phương thức của nó trong Java

Gói

- Gói (package) giống như thư mục giúp:
  - Tổ chức và xác định vị trí lớp dễ dàng và sử dụng các lớp một cách phù hợp.
  - Tránh cho việc đặt tên lớp bị xung đột (trùng tên)
    + Các package khác nhau có thể chứa các lớp có cùng tên
  - Bảo vệ các lớp, dữ liệu và phương thức ở mức rộng hơn so với mối quan hệ giữa các lớp.
- Một package cũng có thể chứa các package khác
- Còn được gọi là *không gian tên (namespace)* trong một số ngôn ngữ lập trình (C/C++...)

Gói trong Java

- Java đã xây dựng sẵn một số package
  - java.lang
  - javax.swing
  - java.io
  - ...
- Có thể tự tạo ra các gói để tổ chức các lớp
  - Cú pháp:

```
package <tên gói>;
```

Gói trong Java

- Tên gói phải được viết trên cùng của file mã nguồn
- Chỉ được phép có 1 câu khai báo gói trong mỗi file mã nguồn, và khai báo này sẽ được áp dụng cho tất cả các dữ liệu trong file đó.
- Một gói có thể được đặt trong một gói khác
  - Phân cách bằng dấu .
  - Ví dụ `package trungtt.oop.k59;`

Quy ước đặt tên gói

- Tên gói được viết toàn bộ bằng chữ thường để tránh xung đột với tên lớp hay giao diện
- Đối với các công ty có tên miền Internet: Sử dụng tên miền đảo để đặt tên gói
  - Ví dụ: Một lập trình viên tại công ty example.com sẽ đặt tên gói là com.example.mypackage

Các package trong Java

- java.applet
- java.awt
- java.beans
- java.io
- java.lang
- java.math
- java.net
- java.nio
- java.rmi
- java.security
- java.sql
- java.text
- java.util
- javax.accessibility
- javax.crypto
- javax.imageio
- javax.naming
- javax.net
- javax.print
- javax.rmi
- javax.security
- javax.sound
- javax.sql
- javax.swing
- javax.transaction
- javax.xml
- org.ietf.jgss
- org.omg.CORBA
- org.omg.CosNaming
- org.omg.Dynamic
- org.omg.IOP
- org.omg.Messaging
- org.omg.PortableInterceptor
- org.omg.PortableServer
- org.omg.SendingContext
- org.omg.stub.java.rmi
- org.w3c.dom
- org.xml

Các package trong Java

- Các package cơ bản trong Java
  - java.lang
    - Cung cấp các lớp cơ bản cho thiết kế ngôn ngữ lập trình Java
    - Bao gồm wrapper classes, String và StringBuffer, Object, ...
    - Import ngầm định vào tất cả các lớp
  - java.util
    - Bao gồm tập hợp framework, mô hình sự kiện, date time, và nhiều tiện ích khác.
  - java.io
    - Cung cấp khả năng vào/ra hệ thống với các luồng dữ liệu và hệ thống file.

Các package trong Java

- Các package cơ bản trong Java
  - java.math
    - Cung cấp các lớp thực thi các phép toán với số nguyên và các phép toán thập phân
  - java.sql
    - Cung cấp các API cho phép truy nhập và xử lý dữ liệu được lưu trữ trong một nguồn dữ liệu (thường sử dụng cơ sở dữ liệu quan hệ)
  - javax.swing
    - Cung cấp các lớp và giao diện cho phép tạo ra các ứng dụng đồ họa.
  - ...

Không gian tên trong C++/C#

– Cú pháp:

```
namespace <tên namespace>
{
        // Khai báo các lớp ở đây
}
```

– Đối không gian tên nằm trong không gian tên khác

```
namespace <tên namespace ngoài>
{
        namespace <tên namespace trong>
        {
                // Khai báo các lớp ở đây
        }
}
```

Khai báo lớp

- Cú pháp: sử dụng từ khóa `class`

```
class <Tên Lớp> {
    // Nội dung lớp
}
```

Ví dụ

```
class Dog {
    // Nội dung lớp
}
```

- Cú pháp khai báo lớp sử dụng chỉ định truy cập:

```
accessmodifier class <Tên Lớp> {
    // Nội dung lớp
}
```

Khai báo lớp sử dụng chỉ định truy cập

- Chỉ định truy cập:
  - + **`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
  - + mặc định: Lớp có thể được truy cập từ bên trong package chứa lớp đó.

inner class

	public	*mặc định*	**private**
Cùng lớp	✔	✔	✔
Cùng gói	✔	✔	✘
Khác gói	✔	✘	✘

Thuộc tính

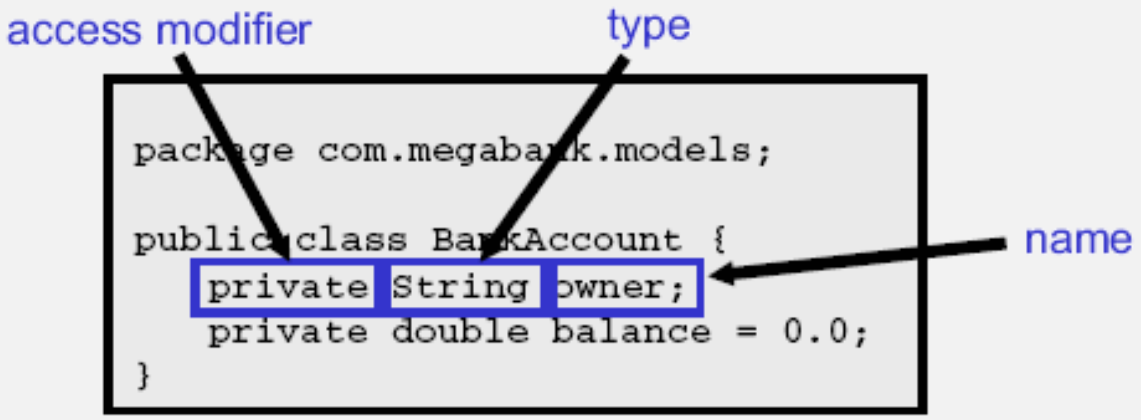
- Là các thông tin, trạng thái mà đối tượng của lớp đó có thể mang
- Các thuộc tính phải được khai báo bên trong lớp
- Mỗi đối tượng có bản sao các thuộc tính của riêng nó
  - Giá trị của một thuộc tính thuộc các đối tượng khác nhau là khác nhau.
- Bản chất của các thuộc tính là các thành phần dữ liệu của đối tượng
  - Khai báo: tương tự như biến

Thuộc tính

- Cú pháp khai báo thuộc tính

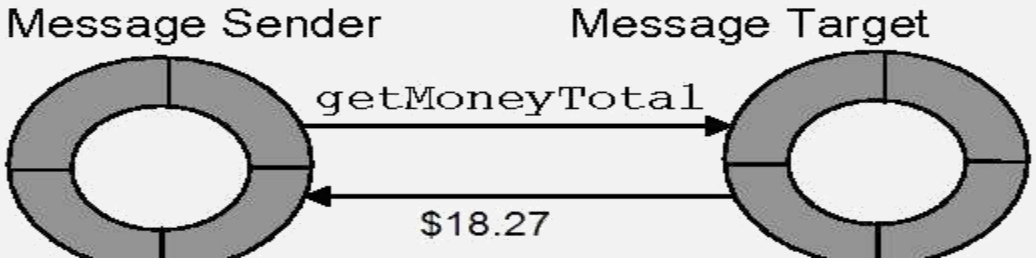
```
accessmodifier kiểu tênThuộcTính;
```

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



Phương thức

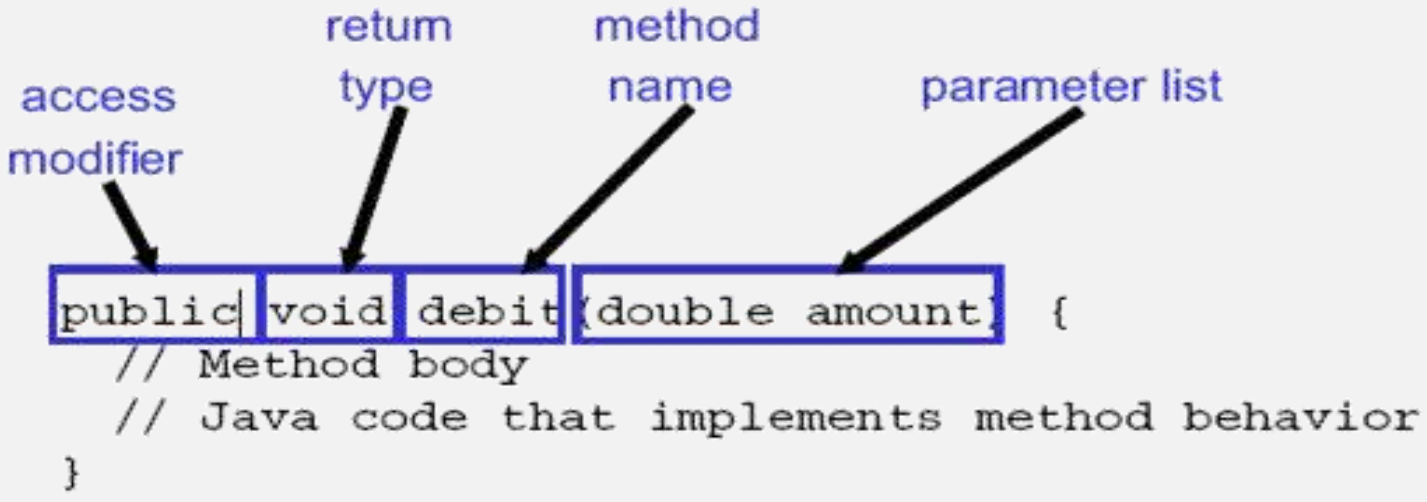
- Xác định cách một đối tượng đáp ứng lại thông điệp
  - Khai báo: tương tự khai báo hàm
- Phương thức xác định các hoạt động của lớp
- Bất kỳ phương thức nào cũng phải thuộc về một lớp nào đó



Phương thức

- Cú pháp

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



Chỉ định truy cập

- Chỉ định truy cập cho thành viên của lớp:
  - + **`public:`** Thuộc tính hoặc phương thức có thể được truy cập từ bất cứ đâu, kể cả bên ngoài package chứa lớp đó.
  - + **`private:`** Thuộc tính hoặc phương thức chỉ có thể được truy cập trong phạm vi lớp đó
  - + mặc định: Thuộc tính hoặc phương thức có thể được truy cập từ bên trong package chứa lớp đó.

	public	*mặc định*	**private**
Cùng lớp	✔	✔	✔
Cùng gói	✔	✔	✘
Khác gói	✔	✘	✘

Ví dụ: Lớp BankAccount

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

4

Tạo và sử dụng đối tượng

Khai báo và khởi tạo đối tượng. Truy cập đến các phương thức và thuộc tính của đối tượng

Khai báo và khởi tạo dữ liệu

- Trong Java, mọi dữ liệu cần phải được *khai báo* và *khởi tạo* trước khi sử dụng
- Ví dụ
  - Khai báo

```
int i;
```

  - Khởi tạo

```
i = 3;
```

  - Hoặc kết hợp khai báo và khởi tạo

```
int i = 3;
```

Khai báo và khởi tạo dữ liệu

- Trong C++, dữ liệu có thể tự động được khởi tạo khi khai báo
  - Ví dụ: `int i;`
  - Dữ liệu sẽ được khởi tạo với giá trị mặc định của kiểu dữ liệu tương ứng
- Trong Java, dữ liệu KHÔNG được tự động khởi tạo khi khai báo
  - Ví dụ:

```
int i;
System.out.println(i); //LỖI
```

Khai báo

- Để khai báo dữ liệu, ta cần phải xác định *kiểu* của dữ liệu đó
  - Cú pháp: `<kiểu> tên biến;`
  - Ví dụ: `int i; // Biến i là kiểu int`
- Trong OOP, **lớp** có thể coi là kiểu dữ liệu trừu tượng do người dùng định nghĩa và **đối tượng** chính là biến của kiểu dữ liệu đó
- Khai báo: tương tự khai báo biến
  - Cú pháp: `<Tên Lớp> tên đối tượng;`
  - Ví dụ:

```
BankAccount acc; // Đối tượng acc là một BankAccount
```

Khởi tạo

- Đối với kiểu dữ liệu nguyên thủy: Dùng toán tử **=**
  - Ví dụ: `i = 3;`
- Đối với kiểu dữ liệu tham chiếu [1]: Khởi tạo bằng toán tử **new**
  - Ví dụ: `acc1 = new BankAccount();`
- Nếu không được khởi tạo: đối tượng mang giá trị *null*

[1] *tương tự con trỏ trong C++*

Khởi tạo

- Khi đối tượng được khởi tạo, các thành phần dữ liệu (thuộc tính) của đối tượng được khởi tạo với giá trị mặc định của kiểu dữ liệu tương ứng
  - number data type ← 0;
  - reference type ← null
  - boolean ← false

Kết hợp khai báo và khởi tạo dữ liệu

- Có thể kết hợp khai báo và khởi tạo dữ liệu
  - Cú pháp:

```
<Tên Lớp> tên đối tượng = new <Tên Lớp>();
```

  - Ví dụ

```
BankAccount acc = new BankAccount();
```

Truy cập đến phương thức và thuộc tính của đối tượng

- Sử dụng toán tử .
- Không cần thiết nếu truy cập từ trong cùng một lớp

```
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);
}
```

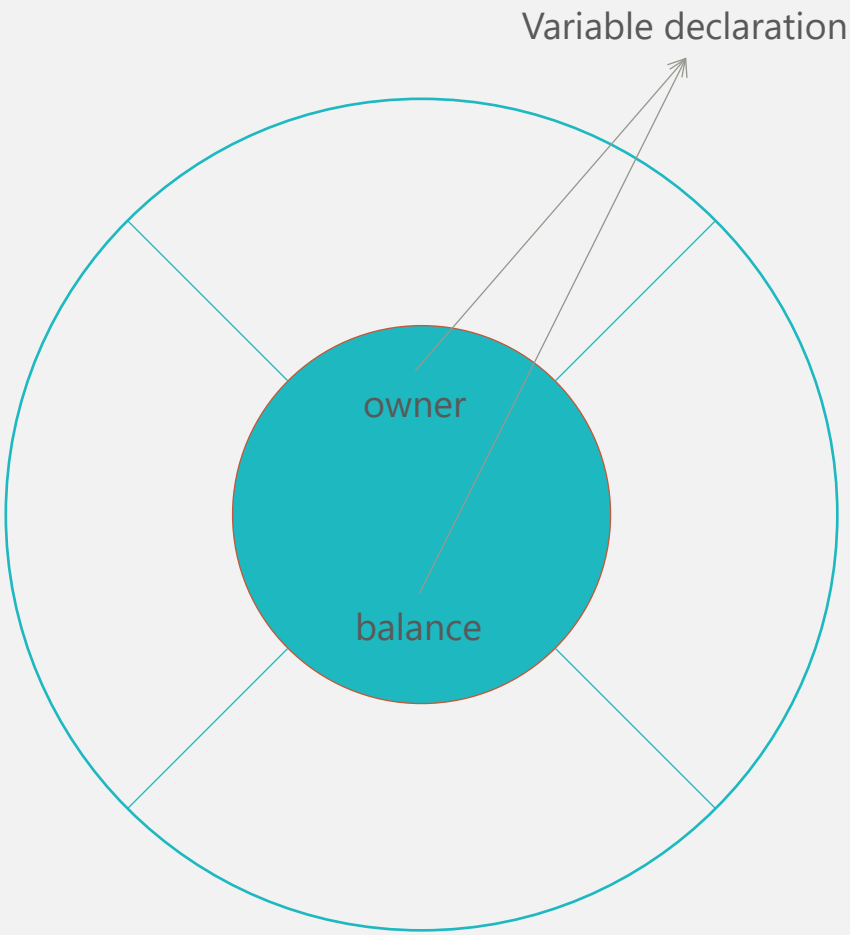
Tự tham chiếu

- Sử dụng từ khóa **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

Ví dụ: từ khóa this

```
public class Account {
   // instance variable
   String owner; // Account name
   long balance; // Balance
   //...
   // value setting method
   void setAccountInfo(String owner, long balance) {
     this.owner = owner;
     this.balance = balance;
   }
   //...
}
```

Ví dụ:
Khởi tạo đối tượng



Class Account

Account object of Mrs. Giang

Account object of Mr. Tuan

Ví dụ:
Truy cập đến các thuộc tính

```
public class Account {
    String name; //Account name
    long balance; //Balance

    void display(){
        System.out.println(…);
    }
    void deposit (long money){
        balance += money;
    }
}

Account acc1 = new Account();
acc1.name = “Vu T Huong Giang”;
acc1.balance = “2000000”;
Account acc2 = new Account();
acc2.name = “Nguyen Manh Tuan”;
acc2.balance = “1000000”;
```

Account object of Ms. Giang

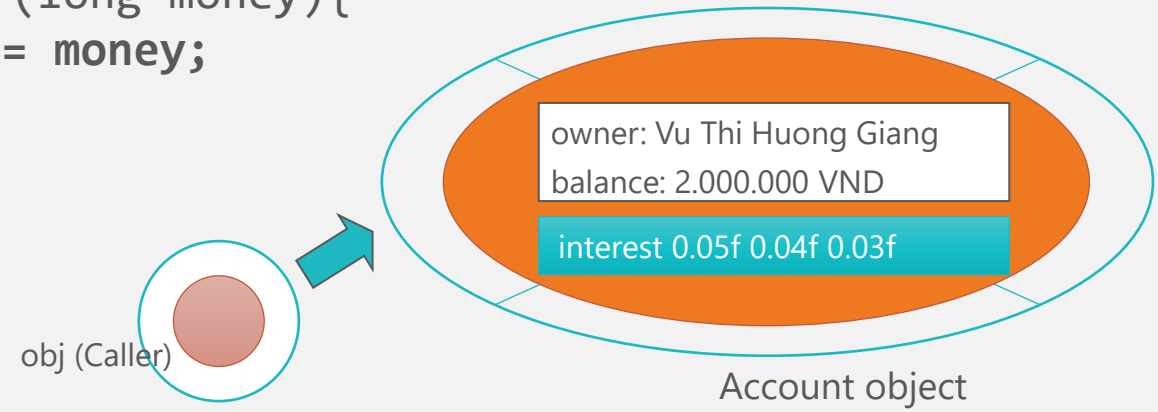
Account object of Mr. Tuan

Ví dụ:
Truy cập đến các phương thức

```
public class Account {
    String name; //Account name
    long balance; //Balance

    void display(){
        System.out.println(…);
    }
    void deposit (long money){
        balance += money;
    }
}
```

```
// Class that uses
// methods of Account object
Account obj = new Account()
obj.display();
obj.deposit(1000000);
```



Tham chiếu đến lớp khác gói

- Đối với lớp trong cùng một gói: chỉ cần tên lớp
  - Ví dụ: `BankAccount`
- Đối với lớp khác gói: phải cung cấp đầy đủ tên lớp và tên gói
  - Ví dụ trong Java: `oop.trungtt.k59.BankAccount`
- Sử dụng lệnh `import` để khai báo các package hoặc các lớp để khi sử dụng không cần nêu tên đầy đủ.

Ví dụ

```
import javax.swing.JOptionPane;
String result;
result = JOptionPane.showInputDialog("Hay nhap ten ban:");
JOptionPane.showMessageDialog(null, "Xin chao " + result + "!");
```

hoặc `import javax.swing.*;`
để import tất cả các lớp trong gói

Đối tượng trong C++

- Trong C++, có 2 cách sử dụng đối tượng
  - Khởi tạo đối tượng truyền thống

```
BankAccount acc;
acc.deposit(1000000);
```

  - Con trỏ đối tượng ← tương tự đối tượng trong Java

```
BankAccount *acc = new BankAccount();
acc->deposit(1000000);
```

5

Thành viên hằng & thành viên tĩnh

final, static

Thành viên tĩnh

- Trong lập trình cấu trúc: Các biến địa phương khai báo cục bộ trong hàm:
  - Trong trường hợp các biến địa phương không khai báo là biến static thì mỗi lần gọi hàm chương trình dịch lại đăng ký tạo ra biến mới
  - Khi chúng ta khai báo các biến địa phương là các biến static thì chương trình dịch sẽ chỉ khởi tạo duy nhất một lần (ở lần gọi đầu tiên) biến địa phương này và thông qua con trỏ stack ở những lần gọi sau chi tham chiếu tới biến đã tạo ra này để sử dụng lại chúng mà không tạo ra biến mới
  - Tạo một lần/tham chiếu nhiều lần/lưu giá trị của lần tham chiếu trước

Biến địa phương static

Biến địa phương static:

```
void f()
{ static int x=0;
   x++;
}
```

Lần gọi 1: f() 0

Lần gọi 2: f() 1

Biến địa phương không static

```
void f()
{ int x=0;
   x++;
}
```

Lần gọi 1: f() 0

Lần gọi 2: f() 0

Thành viên tĩnh trong OOP

- Trong lập trình hướng đối tượng
  - Các thành viên bình thường là thành viên thuộc về đối tượng
  - Các thành viên tĩnh (static) là các thành viên thuộc về lớp
- Cú pháp khai báo thành viên static:

```
<chỉ định truy cập> static <kiểu> tên biến;
```

Thuộc tính static

- Là thuộc tính mang thông tin chung của một lớp
- 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.

Ví dụ

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

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);
  }
}
```

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

Phương thức static

- Các phương thức không tương tác với các "thể hiện" của lớp
- Các phương thức "tiện ích", không cần thiết phải khởi tạo đối tượng để sử dụng
- 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

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

Ví dụ

```
class MyUtils {
    public static double mean(int[] p) {
                int sum = 0;
                for (int i=0; i<p.length; i++) {
              sum += p[i];
                }
                return ((double)sum) / p.length;
    }
}

// Gọi phương thức tĩnh bên trong lớp
double avgAtt = mean(attendance);
// Gọi phương thức tĩnh bên ngoài lớp
double avgAtt = MyUtils.mean(attendance);
```

Phương thức mean có thể không phải khai báo static tuy nhiên muốn gọi nó phải thông qua một đối tượng

Thành viên lớp và thành viên đối tượng

Thành viên đối tượng

- Thuộc tính/phương thức chỉ được truy cập thông qua đối tượng
- Mỗi đối tượng có 1 bản sao riêng của 1 thuộc tính đối tượng
- Giá trị của 1 thuộc tính đối tượng của các đối tượng khác nhau là khác nhau.

Thành viên lớp (static)

- Thuộc tính/phương thức có thể được truy cập thông qua lớp
- Các đối tượng có chung 1 bản sao của 1 thuộc tính lớp
- Giá trị của 1 thuộc tính lớp của các đối tượngkhác nhau là giống nhau.

Ví dụ

- Lớp JOptionPane trong javax.swing
  - *Thuộc tính*
  - *Phương thức*

Field Summary

static int	CANCEL_OPTION Return value from class method if CANCEL is cho
static int	CLOSED_OPTION Return value from class method if user closes windo CANCEL_OPTION or NO_OPTION.
static int	DEFAULT_OPTION Type used for showConfirmDialog.
static int	ERROR_MESSAGE Used for error messages.
static int	WARNING_MESSAGE Used for warning messages.
static int	YES_NO_CANCEL_OPTION Type used for showConfirmDialog.
static int	YES_NO_OPTION Type used for showConfirmDialog.
static int	YES_OPTION Return value from class method if YES is chosen.

static void	showMessageDialog(Component parentComponent, Object message) Brings up an information-message dialog titled "Message".
static void	showMessageDialog(Component parentComponent, Object message, String title, int messageType) Brings up a dialog that displays a message using a default icon determined by the messageType parameter.
static void	showMessageDialog(Component parentComponent, Object message, String title, int messageType, Brings up a dialog displaying a message, specifying all parameters

Ví dụ

```
JOptionPane.showMessageDialog(null,
  "Ban da thao tac loi", "Thong bao loi",
  JOptionPane.ERROR_MESSAGE);
```


```
JOptionPane.showConfirmDialog(null,
  "Ban co chac chan muon thoat?",
  "Hay lua chon", OptionPane.YES_NO_OPTION);
```

Ví dụ

```
Object[] options = { "OK", "CANCEL" };
JOptionPane.showOptionDialog(null,
  “Nhan OK de tiep tuc", "Canh bao",
  JOptionPane.DEFAULT_OPTION,
  JOptionPane.WARNING_MESSAGE,
  null,options,options[0]);
```

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: Sử dụng từ khóa final

```
<chỉ định truy cập> final <kiểu> tên hằng = giá trị;
```

- Ví dụ:

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

Thành viên hằng & tĩnh

Thông thường các hằng số liên quan đến lớp được khai báo là `static final`

- Tương đương với `const` trong C++

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

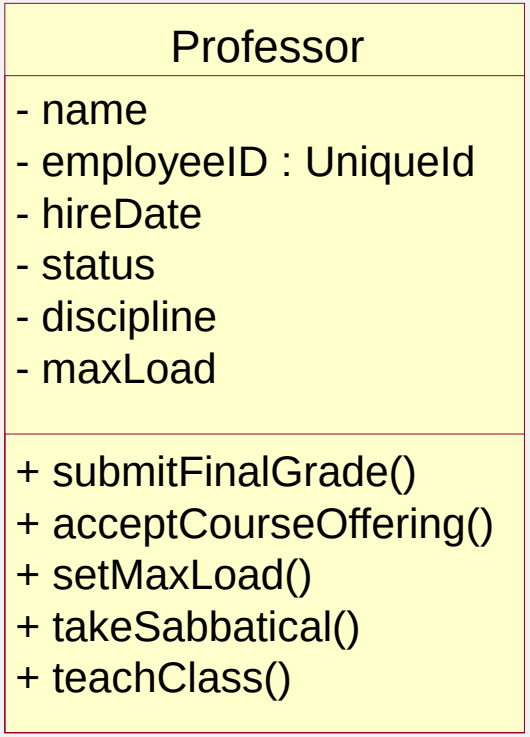
6

Biểu đồ lớp

Biểu diễn lớp trong UML

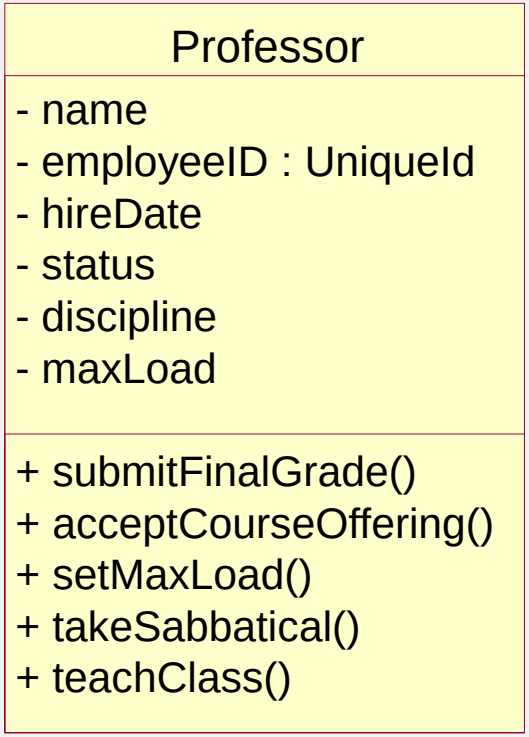
Biểu đồ lớp

- Lớp (class) được biểu diễn bằng 1 hình chữ nhật với 3 thành phần:
  - Tên lớp
  - Cấu trúc (thuộc tính)
  - Hành vi (phương thức)



Chỉ định truy cập

- Trong biểu đồ lớp, dấu – được sử dụng để thể hiện chỉ định truy cập private
- Dấu + được sử dụng để thể hiện chỉ định truy cập public

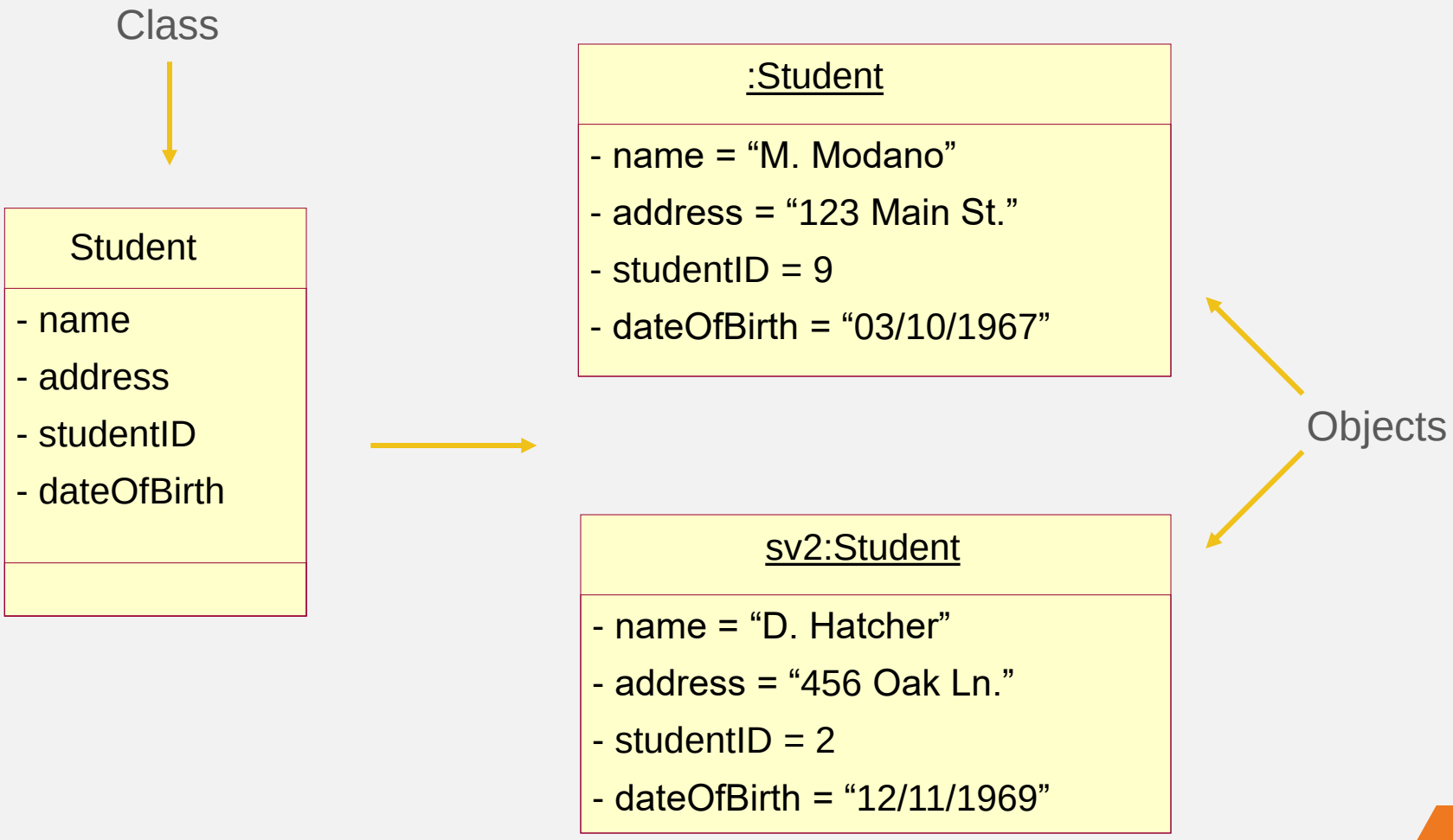


Giá trị tĩnh

- Giá trị tĩnh: Thể hiện bằng dấu gạch chân

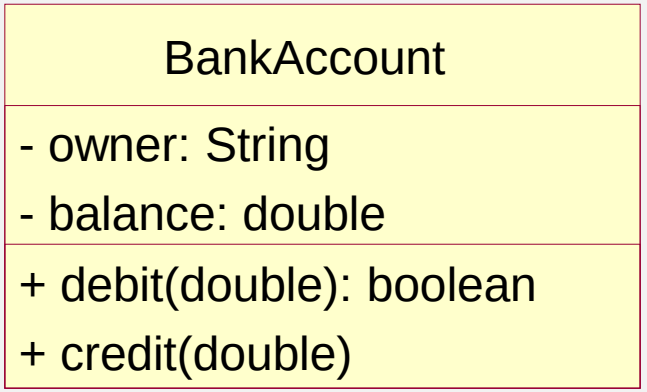
Student
- name - address - studentID - nextAvailID : int
+ addSchedule ([in] theSchedule : Schedule, [in] forSemester : Semester) + getSchedule ([in] forSemester : Semester) : Schedule + hasPrerequisites ([in] forCourseOffering : CourseOffering) : boolean # passed ([in] theCourseOffering : CourseOffering) : boolean + getNextAvailID () : int

Lớp và đối tượng trong UML

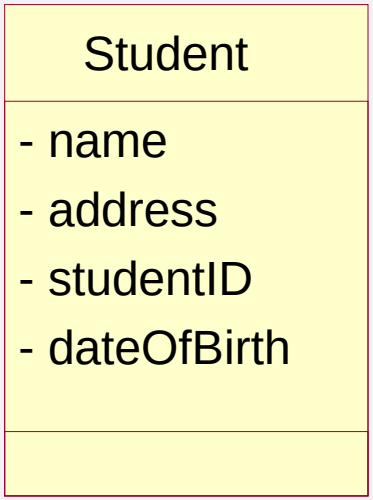


Ví dụ

- Lớp BankAccount



- Lớp Student



Thank you!

Any questions?