



BỘ MÔN CÔNG NGHỆ PHẦN MỀM
VIỆN CNTT & TT
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI



IT3100

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Bài 2: Cơ bản về Java và UML

Nội dung 1: Cú pháp Java cơ bản

Bài giảng e-learning đi kèm

- Vào trang <https://www.udacity.com>
- Đăng ký và Theo dõi bài giảng có tên “Java Programming Basics”

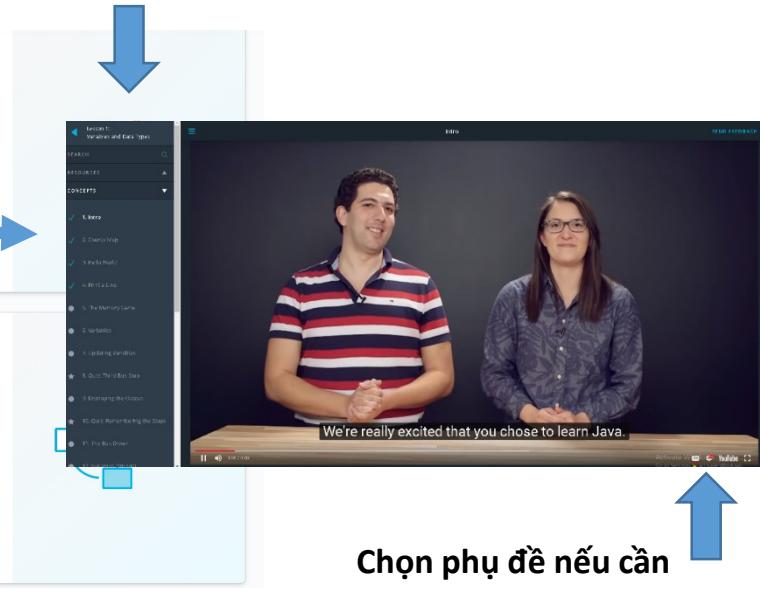
The image shows a screenshot of a course page from Udacity. At the top left is a photo of two people, a man and a woman, sitting at a table. To the right of the photo, the text "School of Programming" is above "Java Programming Basics". A green "FREE" button is located to the right of the course title. Below the title, "Skills Covered" is listed with "Java Programming, Java Syntax, Java Functions, Loops". At the bottom left is a blue "+ COURSE DETAILS" button. At the bottom right is a "Beginner" skill level indicator.

Bài giảng e-learning đi kèm (2)

<https://www.udacity.com/course/java-programming-basics--ud282>

The screenshot shows the Udacity website with the Java Programming Basics course highlighted. The course title 'Java Programming Basics' is displayed prominently with a 'FREE COURSE' badge and a 'LEARN MORE' button. Below the main banner, there's a section titled 'SCHOOL OF Programming and Development' with a brief description. On the right, there's a sidebar for 'THIS COURSE' which includes a summary of the course, a list of 'SAMPLE NANO DEGREE PROGRAMS' (Java Developer, C++, Data Structures and Algorithms), and a large blue 'START FREE COURSE' button. A yellow arrow points to this 'START FREE COURSE' button.

Cấu trúc bài giảng



Bài giảng e-learning đi kèm (3)

- Java Programming Basics:
 - Lesson 1: Variables and Data Types
 - Lesson 2: Control Flow and Conditionals
 - Lesson 3: Functions (sẽ trình bày ở các bài giảng sau)
 - Lesson 4: Loops
 - Lesson 5: IntelliJ and Debugging (tham khảo)

Mục tiêu bài học

- Xây dựng chương trình Java đầu tiên
- Quy ước đặt tên, cách tạo ra các định danh hợp lệ
- Giới thiệu các kiểu dữ liệu cơ bản
- Biến, Kiểu và Giá trị
 - Khai báo sử dụng các biến
 - Phạm vi của biến
 - Chuyển kiểu
- Các toán tử cơ bản
- Các câu lệnh điều khiển
 - Cấu trúc điều kiện
 - Cấu trúc lặp
 - Rẽ nhánh
- Sử dụng Mảng trong Java

Nội dung

1. Chương trình JAVA đầu tiên
2. Định danh
3. Biến
4. Các kiểu dữ liệu cơ bản
5. Toán tử
6. Chuyển đổi kiểu dữ liệu
7. Cấu trúc điều khiển
8. Mảng

1. Chương trình Java đầu tiên

Tham khảo Lesson 1 – Session 3

Lệnh Java đầu tiên

- Lệnh Java đầu tiên:

```
System.out.println();
```

Hello, World!

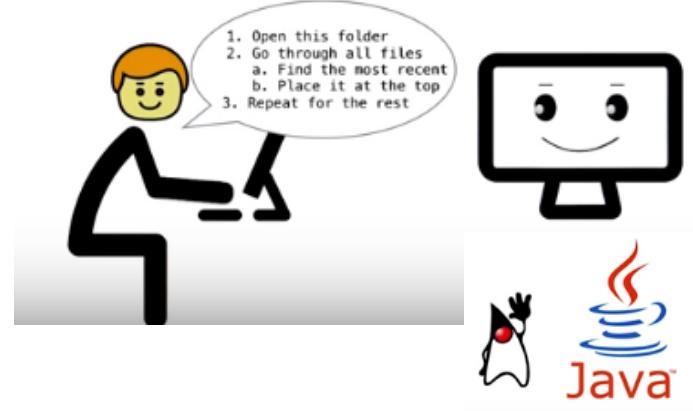
Đây là một câu lệnh của “System”

```
System.out.println();
```

Làm việc với đầu ra (mặc định là màn hình)

Thực hiện in ra màn hình một dòng mới

Programming a computer



This is the message we want to print

```
System.out.println("Hello, World!");
```

Hoặc một thông điệp nào đó trong hai dấu “...”

Lệnh Java đầu tiên (2)

- Lưu ý:
 - Java phân biệt chữ hoa, chữ thường
 - Cặp dấu “ ” để xây dựng một chuỗi sẽ được dùng nguyên dạng, không đổi
 - Câu lệnh trong Java kết thúc bằng dấu chấm phẩy ;
 - Nhiều lệnh cùng xuất hiện trên 1 dòng

```
a=0; b=1; c=2;
```
 - Một câu lệnh
- ```
System.out.println(
 "This is part of the same line");
```

# Chú thích trong Java

- Dùng để mô tả, chú thích cho một dòng/đoạn code, Trình thông dịch sẽ bỏ qua các chú thích này.

```
// Chú thích trên một dòng
/* Chú thích một đoạn,
 nhiều dòng
*/
/** Javadoc * chú thích dạng Javadoc */
```

```
int x = 5;
int y = 2;
double div = x/y;
System.out.println(div);
double accurate = (double)x/y;
System.out.println(accurate);
```



```
int x = 5;
int y = 2;
//dividing x by y
double div = x/y;
//this won't work
System.out.println(div);
//casting x to double
double accurate = (double)x/y;
//this looks better
System.out.println(accurate);
```

# Chương trình Java đầu tiên

- Tạo một file mã nguồn với tên HelloWorld.java, nội dung như sau.

```
/* This is a simple Java program.
 FileName : "HelloWorld.java". */
class HelloWorld
{
 // Your program begins with a call to main().
 // Prints "Hello, World" to the terminal window.
 public static void main(String args[])
 {
 System.out.println("Hello, World");
 }
}
```

Đóng gói trong 1 lớp cùng tên

Chương trình bắt đầu với lời gọi tới hàm main()

Các lệnh cần thực hiện

- Dịch file mã nguồn: gõ lệnh “javac HelloWorld.java”
- Chạy file nhị phân: gõ lệnh “java HelloWorld”
- Kết quả in ra màn hình

Hello, World

# 3 cách đọc dữ liệu từ bàn phím

- Cách 1: sử dụng lớp BufferedReader

```
// Java program to demonstrate BufferedReader
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class Test
{
 public static void main(String[] args) throws IOException
 {
 //Enter data using BufferedReader
 BufferedReader reader =
 new BufferedReader(new InputStreamReader(System.in));

 // Reading data using readLine
 String name = reader.readLine();

 // Printing the read line
 System.out.println(name);
 }
}
```

# 3 cách đọc dữ liệu từ bàn phím (2)

- Cách 2: sử dụng lớp Scanner

```
// Java program to demonstrate working of Scanner in Java
import java.util.Scanner;

class GetInputFromUser
{
 public static void main(String args[])
 {
 // Using Scanner for Getting Input from User
 Scanner in = new Scanner(System.in);

 String s = in.nextLine();
 System.out.println("You entered string "+s);

 int a = in.nextInt();
 System.out.println("You entered integer "+a);

 float b = in.nextFloat();
 System.out.println("You entered float "+b);
 }
}
```

# 3 cách đọc dữ liệu từ bàn phím (3)

- Cách 3: sử dụng lớp Console (không làm việc trên IDE)

```
// Java program to demonstrate working of System.console()
// Note that this program does not work on IDEs as
// System.console() may require console
public class Sample
{
 public static void main(String[] args)
 {
 // Using Console to input data from user
 String name = System.console().readLine();

 System.out.println(name);
 }
}
```

## 2. Định danh

*Tham khảo Lesson 1 - Session 15*

## 2.1. Khái niệm Định danh

- **Định danh:**
  - Xâu ký tự thể hiện tên các biến, các phương thức, các lớp và nhãn
  - là duy nhất trong chương trình
- **Quy định với định danh hợp lệ:**
  - Gồm các ký tự có thể là chữ cái, chữ số, ký tự '\$' hoặc '\_'
  - **Không** được phép:
    - Bắt đầu bởi một chữ số
    - Trùng với từ khóa
    - Chứa dấu cách
  - Phân biệt chữ hoa chữ thường
    - Yourname, yourname, YourName và yourName là 4 định danh khác nhau

```
An_Identifier
a_2nd_Identifier
Go2
$10
```

```
An-Identifier
2nd_Identifier
goto
10$
```

## 2.1. Khái niệm Định danh (2)

- Quy ước với định danh (naming convention):
  - Phải mang tính gợi nhớ
    - Ví dụ: nên dùng định danh “bookPrice” hơn là “bp” để lưu thông tin về giá 1 quyển sách
  - Bắt đầu bằng chữ cái
  - Gói (package): tất cả sử dụng chữ thường
    - `theexample`
  - Lớp (Class): viết hoa chữ cái đầu tiên trong các từ ghép lại
    - `TheExample`
  - Phương thức/thuộc tính (method/field): Bắt đầu bằng chữ thường, viết hoa chữ cái đầu tiên trong các từ còn lại
    - `theExample`
  - Hằng (constants): Tất cả viết hoa
    - `THE_EXAMPLE`

## 2.2. Các từ khóa

- Người lập trình không được phép sử dụng các từ khóa như một định danh
- Literals:

null true false

- Từ khóa (keyword):

abstract assert boolean break byte case catch char class  
continue default do double else extends final finally float  
for if implements import instanceof int interface long  
native new package private protected public return short  
static strictfp super switch synchronized this throw throws  
transient try void volatile while

- Từ dành riêng (reserved word):

byvalue cast const future generic goto inner operator outer  
rest var volatile

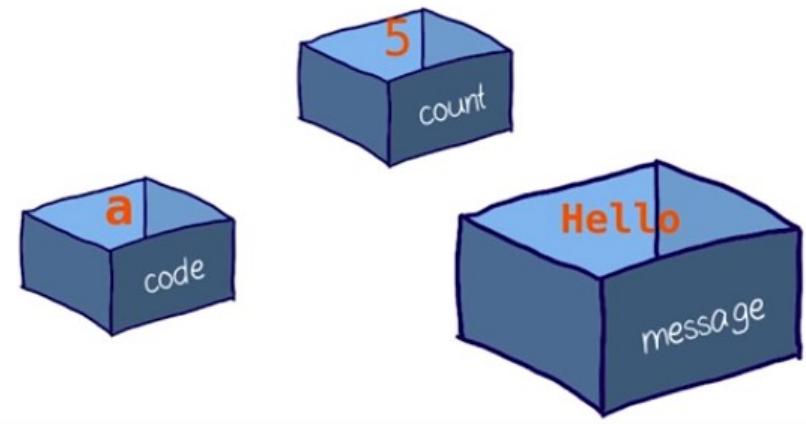
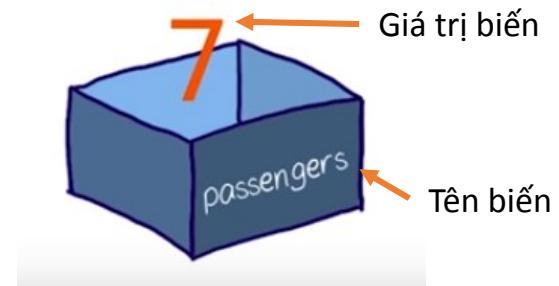
### 3. Biển

*Tham khảo Lesson 1 – Session 6*

### 3.1. Khái niệm biến

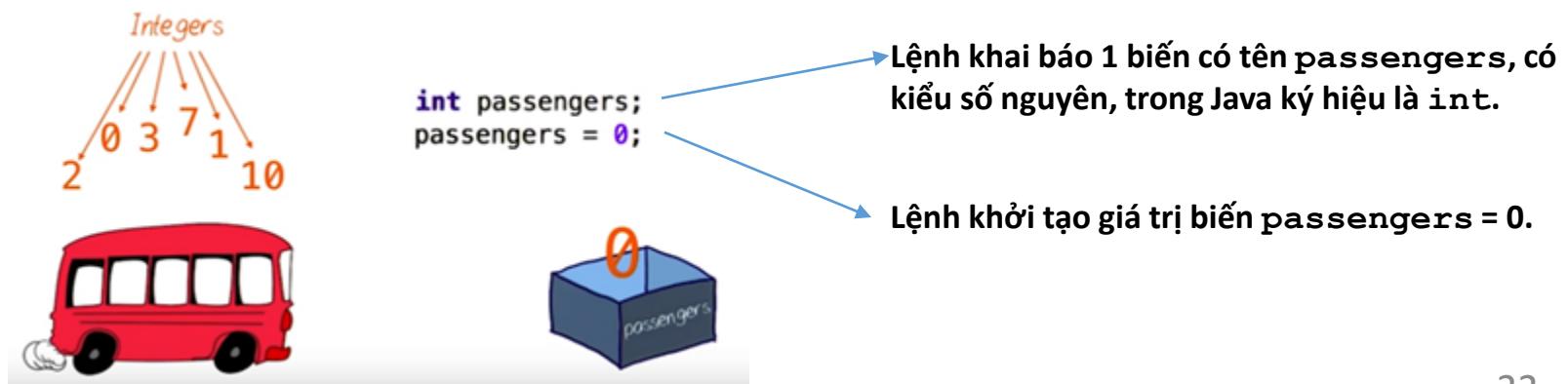
- Biến giống như 1 chiếc hộp trong bộ nhớ, chứa giá trị cho 1 đại lượng nào đó
  - Biến có tên không thay đổi
  - Biến được gán 1 giá trị, có thể thay đổi trong khi chạy
- Biến có thể chứa các giá trị kiểu số, ký tự, văn bản, hay đối tượng
  - và kiểu giá trị này của biến cũng không thay đổi, gọi là kiểu dữ liệu của biến

#### VARIABLES



## 3.2. Khai báo biến

- Biến khi dùng phải khai báo tên bằng một **tên** (định danh) và gán cho một **kiểu dữ liệu** (số, ký tự, văn bản, hay đối tượng, v.v.)
- Các biến đơn cần phải được khởi tạo trước khi sử dụng



## 3.2. Khai báo biến (2)

- Có thể kết hợp khai báo và khởi tạo cùng một lúc.
- Ví dụ:

The diagram illustrates the declaration and initialization of three integer variables. It features two orange horizontal arrows above the code. The first arrow, labeled "Declaring", spans from the start of the first variable declaration to the start of the second. The second arrow, labeled "Initializing", spans from the start of the first assignment operator (=) to the start of the third. The code below the arrows shows three separate declarations, each consisting of a type (int), a variable name, an assignment operator (=), and a value.

```
int price = 0;
int speed = 100;
int stockPrice = 75;
```

### 3.3. Sử dụng biến

```
int passengers;
passengers = 0;
```



```
passengers = passengers + 5;
```



```
passengers = passengers - 3;
```



```
System.out.println(passengers);
```



Lệnh in ra giá trị hiện tại của biến  
passengers (không có "" quanh tên biến)  
Nếu passengers chưa khởi tạo, sẽ báo lỗi

### 3.4. Phạm vi sử dụng của các biến

- Phạm vi của biến là vùng chương trình mà trong đó biến có thể được tham chiếu đến, có thể sử dụng được.
- Phạm vi hoạt động (scope) của các biến cho phép xác định các nguyên lý của tạo biến, sử dụng biến và giải phóng biến
- Phân loại:
  - Biến toàn cục: phạm vi trong cả chương trình
  - Biến cục bộ: được khai báo trong một phương thức/ khối lệnh thì chỉ có thể truy cập trong phương thức/ khối lệnh đó.

### 3.4. Phạm vi sử dụng của các biến (2)

```
boolean isLightGreen = ? ; //true or false

if(isLightGreen) {
 //traffic light is green
 double carSpeed = 100; //in km/hr
 System.out.println("Drive!");
 System.out.println("Speed is: " + carSpeed);
}
```

Block of code where  
a variable can be used

A set of curly braces  
defines a variable scope

{ scope }

isLightGreen  
scope      carSpeed  
              scope

```
boolean isLightGreen = ? ; //true or false

if(isLightGreen) {
 //traffic light is green
 double carSpeed = 100; //in km/hr
 System.out.println("Drive!");
 System.out.println("Speed is: " + carSpeed);
}
carSpeed = carSpeed - 10;
```

Cannot resolve symbol 'carSpeed'

Error!

carSpeed  
scope

# 4. Các kiểu dữ liệu cơ bản

*Tham khảo Lesson 1 - Session 16, 12, 13*

# 4. Các kiểu dữ liệu cơ bản

- Dữ liệu được phân lớp theo các tiêu chí khác nhau => Kiểu dữ liệu
- Mỗi kiểu dữ liệu có một tính chất xác định và có kiểu thể hiện riêng
- Trong Java kiểu dữ liệu được chia thành hai loại:
  - Kiểu dữ liệu nguyên thủy (primitive)
    - Số nguyên (integer)
    - Số thực (float)
    - Ký tự (char)
    - Giá trị logic (boolean)
  - Kiểu dữ liệu tham chiếu (reference)
    - Mảng (array) (xem phần 8)
    - Đối tượng (object) (xem ở các bài học tiếp theo)

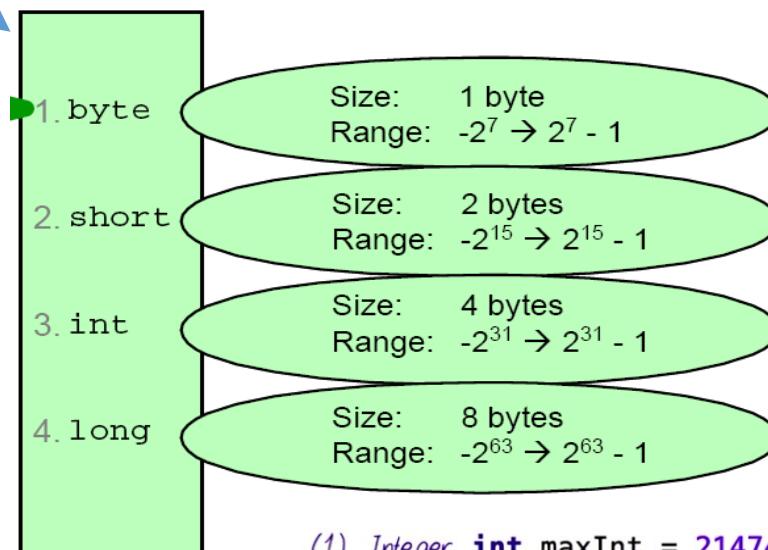
# 4.1. Kiểu dữ liệu nguyên thủy

Categories:

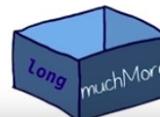
- a. integer
- b. floating point
- c. character
- d. boolean

## a. Số nguyên

- có dấu, không có phần thập phân, khởi tạo với giá trị 0



(1) Integer `int maxInt = 2147483647;`  
(2) Long `long muchMore = 2147483647*10000000;`



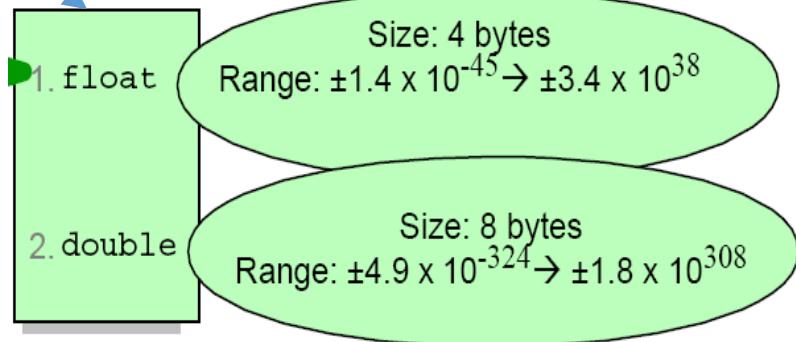
# 4.1. Kiểu dữ liệu nguyên thủy (2)

Categories:

- a. integer
- b. floating point
- c. character
- d. boolean

## b. Số thực

- có dấu, có phần thập phân, khởi tạo với giá trị 0.0



```
double fraction = 99.275;
```

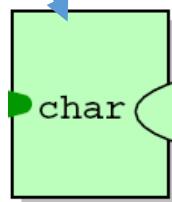
# 4.1. Kiểu dữ liệu nguyên thủy (3)

Categories:

- a. integer
- b. floating point
- c. character
- d. boolean

## c. Ký tự

- Ký tự Unicode không dấu, được đặt giữa hai dấu nháy đơn
- 2 cách gán giá trị:
  - Sử dụng các chữ số trong hệ 16: char uni = '\u05D0';
  - Sử dụng ký tự: char a = 'A';
- Giá trị mặc định là giá trị zero (\u0000)



Size: 2 bytes  
Range: \u0000 → \uFFFF

```
char answer = 'b';
char three = '3';
char ten = '10';
```

# 4.1. Kiểu dữ liệu nguyên thủy (4)

Categories:

- a. integer
- b. floating point
- c. character
- d. boolean

## d. Giá trị logic

- Có thể lưu trữ giá trị hoặc true hoặc false
- Một giá trị int không thể sử dụng thay cho giá trị boolean
- Khởi tạo là false



Size: 1 byte  
Range: true | false

```
boolean fact = true;
```

```
boolean condition = false;
```

## 4.2. Kiểu dữ liệu xâu ký tự

- Kiểu String dùng để lưu trữ một xâu ký tự: tập hợp nhiều ký tự đọc được.

Java is fun  
Hello!  
String  
01.01.2000 email@gmail.com  
@ 4 . a x \$  
# Character !  
7 z - 8 %

- Khai báo và sử dụng biến kiểu String

```
String driver;
driver = "Hamish";
```

↑  
String literal



```
int letters = driver.length();
System.out.println(letters);
```

Print output:  
6

```
driver = driver.toUpperCase();
System.out.println(driver);
```

Print output:  
HAMISH

## 4.2. Kiểu dữ liệu xâu ký tự (2)

- Cộng hai xâu

```
String driverFirstName;
driverFirstName = "Hamish";

String driverLastName;
driverLastName = "Blake";

String driverFullName = driverFirstName + driverLastName;
System.out.println(driverFullName);
```

Print output:  
HamishBlake

```
int stops;
int fare;
stops = 0;
fare = 0;
stops = stops + 1;
fare = fare + 5;
stops = stops + 1;
fare = fare + 3;
stops = stops + 1;
fare = fare + 7;
System.out.println("The bus made $" + fare + " after " + stops + " stops");
```



Print output:  
The bus made \$15 after 3 stops

variable  
variable  
variable  
string literal      string literal      string literal

## 4.3. Biểu diễn nguyên dạng

### a. Biểu diễn 1 số nguyên.

- Hệ Octals (0):  $032 = 011\ 010(2) = 16 + 8 + 2 = 26(10)$
- Hệ Hexadecimals (0x):  $0x1A = 0001\ 1010(2) = 16 + 8 + 2 = 26(10)$
- Kiểu long (kết thúc bằng 'L'): 26L

#### Literals

|                   |      |
|-------------------|------|
| integer.....      | 7    |
| floating point... | 7.0f |
| boolean.....      | true |
| character.....    | 'A'  |
| string.....       | "A"  |

## 4.3. Biểu diễn nguyên dạng (2)

### b. Biểu diễn 1 số thực.

- float kết thúc bằng ký tự f (hoặc F):  
7.1f
- double kết thúc bằng ký tự d (hoặc D):  
7.1D
- e (hoặc E) được sử dụng trong dạng biểu diễn khoa học : 7.1e2
- Một giá trị thực mà không có ký tự kết thúc đi kèm sẽ có kiểu là double: 7.1 giống như 7.1d

#### Literals

|                       |   |
|-----------------------|---|
| integer.....          | 7 |
| floating point...7.0f |   |
| boolean.....true      |   |
| character.....'A'     |   |
| string....."A"        |   |

## 4.3. Biểu diễn nguyên dạng (3)

### c. Biểu diễn logic.

- true, false

### d. Biểu diễn 1 ký tự.

- Được đặt giữa 2 dấu nháy đơn
- Ví dụ: 'a', 'A', '9' hoặc '\uffff'

### e. Biểu diễn 1 xâu ký tự.

- Được đặt giữa hai dấu nháy kép
- Ví dụ: "Hello world", "Xin chao ban",...

### Literals

|                   |      |
|-------------------|------|
| integer.....      | 7    |
| floating point... | 7.0f |
| boolean.....      | true |
| character.....    | 'A'  |
| string.....       | "A"  |

## 4.3. Biểu diễn nguyên dạng (4)

### f. Các ký tự điều khiển nhấn phím

- '\b' backspace
- '\f' form feed
- '\n' newline
- '\r' return (về đầu dòng)
- '\t' tab

### g. Ký tự đặc biệt trong xâu

- '\" quotation mark
- '\'' apostrophe
- '\\ backslash

#### Literals

|                   |      |
|-------------------|------|
| integer.....      | 7    |
| floating point... | 7.0f |
| boolean.....      | true |
| character.....    | 'A'  |
| string.....       | "A"  |

# 5. Toán tử

*Tham khảo Lesson 1 - Session 18*

## 5.1. Toán tử (Operators)

- Kết hợp các giá trị đơn hoặc các biểu thức con thành những biểu thức mới, phức tạp hơn và có thể trả về giá trị.
- Java cung cấp nhiều dạng toán tử sau:
  - Toán tử số học: +, -, \*, /, %
  - Toán tử làm việc trên bit: AND: &, OR: |, XOR: ^, NOT: ~, Dịch bit: <<, >>, >>>
  - Toán tử quan hệ/so sánh: ==, !=, >, <, >=, <=
  - Toán tử logic: AND: &&, OR: ||, NOT: !
  - Toán tử gán: =, +=, -=, \*=, /=, %=, &=, |=, ^=, >>=, <<=

## 5.1. Toán tử (Operators) (2)

- Toán tử một ngôi:
  - Đảo dấu: + -
  - Tăng giảm 1 đơn vị ++, --
  - Phủ định 1 biểu thức: !
- Toán tử điều kiện ( ? : )
- ...

## 5.2. Thứ tự ưu tiên của toán tử

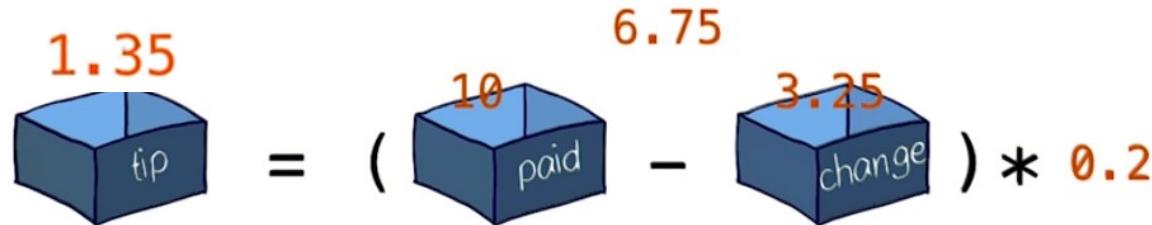
- Cho biết toán tử nào thực hiện trước
- Được xác định bởi các dấu ngoặc đơn hoặc theo ngầm định như sau (ưu tiên từ trên xuống thực hiện trước):

1. Toán tử đứng sau: [] .  
(params) x++ x--
2. Toán tử một ngôi: ++x --x +x -  
x ~ !
3. Toán tử khởi tạo, toán tử  
chuyển kiểu: new (type)x
4. Nhân, chia: \* / %
5. Cộng, trừ: + -
6. Dịch bit: << >> >>> (unsigned  
shift)
7. So sánh: < > <= >= instanceof
8. So sánh bằng == !=

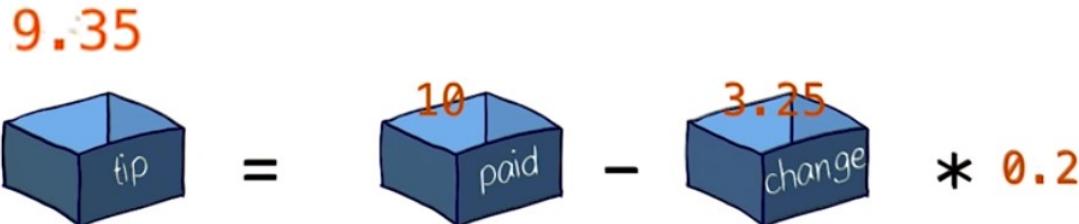
9. Toán tử bit AND: &
10. Toán tử bit OR: ^
11. Toán tử bit XOR: |
12. Toán tử logic AND: &&
13. Toán tử logic OR: ||
14. Toán tử điều kiện: (ternary)  
?:
15. Toán tử gán: = \*= /= %= += -=  
>>= <<= >>>= &= ^= |=

## 5.2. Thứ tự ưu tiên của toán tử (2)

```
double paid = 10;
double change = 3.25;
double tip = (paid-change)*0.2;
```

$$1.35 = (10 - 3.25) * 0.2$$


```
double tip = paid-change *0.2;
```

$$9.35 = 10 - 3.25 * 0.2$$


# 6. Chuyển đổi kiểu dữ liệu

*Tham khảo Lesson 1 - Session 19*

# 6.1. Chuyển đổi Casting

- Java là ngôn ngữ định kiểu chặc
  - Gán sai kiểu giá trị cho một biến có thể dẫn đến các lỗi biên dịch hoặc các ngoại lệ của JVM
- JVM có thể ngầm định chuyển từ một kiểu dữ liệu hẹp sang một kiểu rộng hơn (widening)
  - byte → short → int → long → float → double
  - không xảy ra mất mát thông tin

```
int a, b;
short c;
a = b + c; // ok
```

```
double f;
long g;
f = g;
g = f; //error
```

## 6.1. Chuyển đổi Casting (2)

- Để chuyển từ một kiểu rộng hơn sang một kiểu dữ liệu hẹp hơn (narrowing), cần phải chuyển kiểu rõ ràng (explicit cast).
  - Có thể làm mất thông tin giá trị được chuyển đổi, làm giảm độ chính xác
  - Phải dùng cú pháp chuyển/ép kiểu
  - Lưu ý: ép kiểu từ **short** về **char**, từ **byte** về **char** và ngược lại đều phải ép kiểu tường minh

```
int d;
short e;
e = (short)d;
```

```
int i = (int) 25.123;
```

Cú pháp chuyển kiểu (kiểu\_mới) biến\_cần\_chuyển

## 6.1. Chuyển đổi Casting (3)

- Chuyển từ xâu → số
  - chuyển đổi từ object → kiểu dữ liệu nguyên thủy, dùng hàm/phương thức của lớp tương ứng

```
String str1 = "12";
int i = Integer.parseInt(str1); //→ i =12
String str2 = "45.67";
double d= Double.parseDouble(str2); //→ d=45.67
```

## 6.2. Ví dụ

```
long p = (long) 12345.56; // ok, p == 12345
int g = p; // không hợp lệ (long -> int)
char c = 't';
int j = c; // hợp lệ, ngầm chuyển kiểu
short k = c; // không hợp lệ
short k = (short) c; // ok, ép kiểu tường minh
float f = 12.35; /* không hợp lệ, vì
float f = 0.0; gán giá trị double
float f = 0; cho biến kiểu float */
long l = 9999999999999; /* không hợp lệ, 999999999999
 có kiểu int nhưng giá trị
 ngoài khoảng của kiểu int*/
short k = 99999999; // không hợp lệ (int -> short)
```

## 6.2. Ví dụ (2)

- long l = 9999999999999L;
- int i = (int) l; //i=1316134911
- long l = 9999999999999L;
- int i = (int) l; //i =276447231
- int i = 9999999;
- short j = (short) i; //j =-7937
- int i = 999999;
- short j = (short) i; //j=-27009

**Câu hỏi:** Giải thích giá trị  
của i, j sau ép kiểu ?  
Gợi ý: chuyển đổi sang mã  
nhị phân

## 6.3. Chuyển kiểu trên toán tử “Numeric Promotions”

```
double div = 5/2; = 2.0 = 2
double accurate = 5/2.0; = 2.5
double div2 = 24/5; = 4.8 = 4
double accurate2 = 24/5.0; = 4.8
```

?

- Thực hiện trên các toán tử \*, /, %, + số, - số, toán tử so sánh, ...
- Nếu 1 trong hai toán hạng là kiểu double → toán tử kia được chuyển về kiểu double, kết quả là kiểu double
  - Nếu không, nếu 1 trong hai toán hạng là kiểu float → toán tử kia được chuyển về kiểu float, kết quả là kiểu float
  - Nếu không, nếu 1 trong hai toán hạng là kiểu long → toán tử kia được chuyển về kiểu long, kết quả là kiểu long
  - Nếu không, cả 2 toán tử được chuyển về kiểu int, kết quả là kiểu int

## 6.3. Chuyển kiểu trên toán tử (2) “Numeric Promotions”

- Kết hợp Numeric promotion và Casting

```
int x = 5;
int y = 2;
double div = x/y;
System.out.println(div);
double accurate = (double)x/y;
System.out.println(accurate);
```

Print output:

2  
2.5

## 6.3. Chuyển kiểu trên toán tử (3) “Numeric Promotions”

```
short i = 6, j=7;
i = i + j; // lỗi
i += j; // ok
```

- i, j khai báo kiểu short
- i + j : hiện tượng chuyển kiểu trên toán tử cộng '+', ép i, j về kiểu integer => gán i là short = i+j là int sẽ gây lỗi
- Toán tử gán cộng '+=' : kiểu trả về là kiểu của toán hạng bên trái => ok

```
short l, m = 5;
int n = 6;
l = (short)n + m; // lỗi
l = (short)(n + m); // ok
```

- Tương tự, hiện tượng chuyển kiểu trên toán tử cộng '+', ép (short)n và m về kiểu integer => lỗi

# 6.4. “Numeric Promotions” hay “Casting” ?

- `int i = 10/3;` // no
- `float f0 = 10;`
- `float f1 = (float) 10/3;` // casting rồi promotion
- `float f2 = 10/3;` // casting
- `float f3 = f0/3;` // promotion
- `System.out.println(i);` //3
- `System.out.println(f1);` //3.3333333
- `System.out.println(f2);` //3.0
- `System.out.println(f3);` //3.333333

# 7. Cấu trúc điều khiển

*Tham khảo Lesson 2 – Session 1..16*

# 7. Cấu trúc điều khiển

- Là các cấu trúc lệnh nhằm chỉ định cho chương trình thực hiện các câu lệnh/đoạn lệnh khác nhau, tùy theo từng điều kiện nào đó.
- Gồm các kiểu cấu trúc điều khiển:
  - Câu lệnh điều kiện
    - Lệnh if – else,
    - Lệnh switch – case
  - Câu lệnh lặp
    - Vòng lặp for
    - Vòng lặp while
    - Vòng lặp do – while

# 7.1. Lệnh if

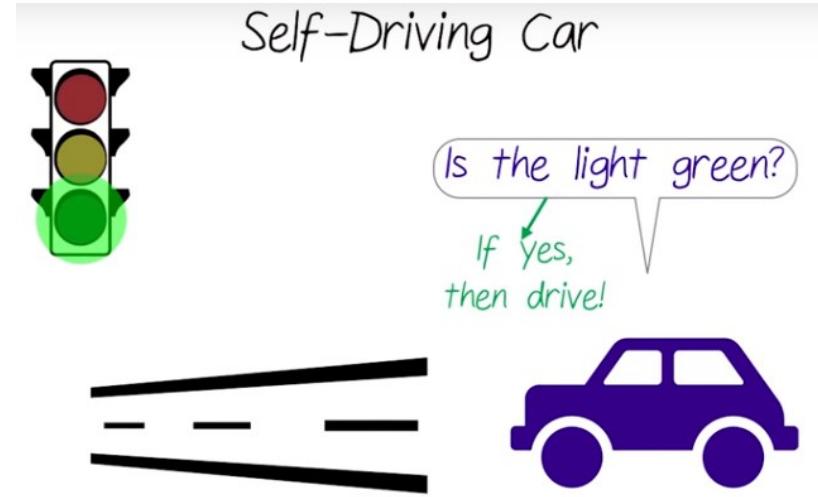
- Cú pháp

```
if (dieu_kien) {
 cac_cau_lenh;
}
```

- Nếu biểu thức điều kiện dieu\_kien (có kiểu boolean) nhận giá trị true thì thực hiện khối lệnh cac\_cau\_lenh;

```
boolean isLightGreen = ? ; //true or false

true
if(isLightGreen) {
 // traffic light is green
 System.out.println("Drive!");
}
```



```
boolean isLightGreen = ? ; //true or false

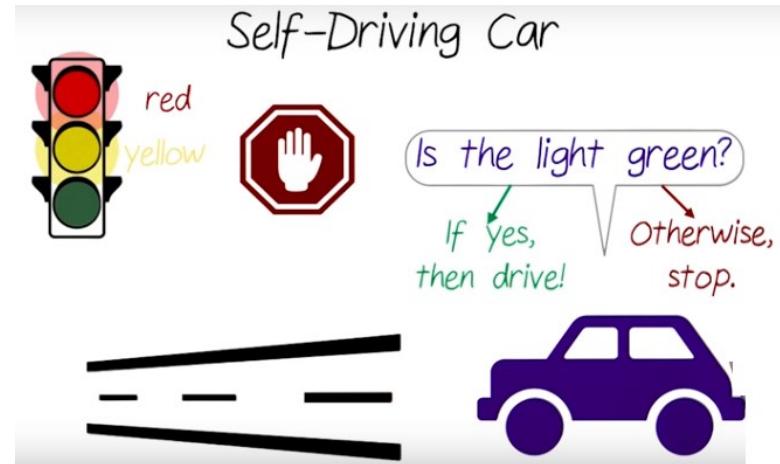
false
if(isLightGreen) {
 // traffic light is green
 System.out.println("Drive!");
}
skip down here
```

## 7.2. Lệnh if - else

- Cú pháp

```
if (dieu_kien) {
 cac_cau_lenh_true;
}
else {
 cac_cau_lenh_false;
}
```

- Nếu biểu thức điều kiện (kiểu boolean) nhận giá trị true thì thực hiện khối lệnh cac\_cau\_lenh\_true, là false thì thực hiện khối lệnh cac\_cau\_lenh\_false.



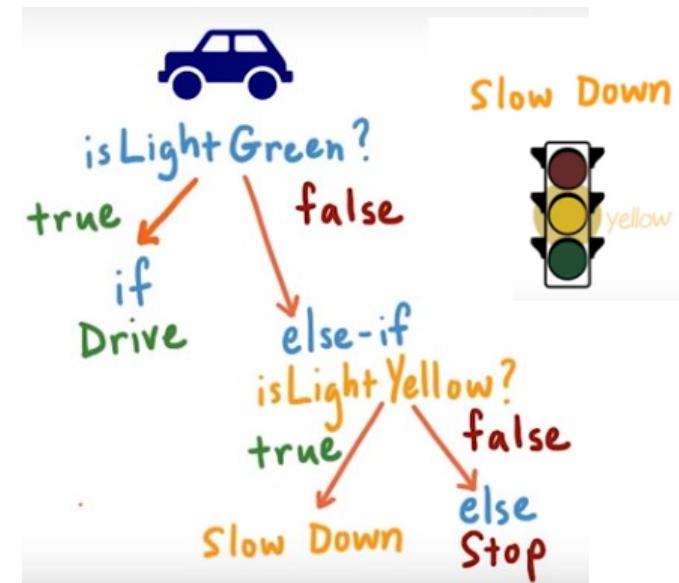
```
boolean isLightGreen = ? //true or false

if(isLightGreen) {
 //traffic light is green
 System.out.println("Drive!");
} else {
 //light is NOT green
 System.out.println("Stop.");
}
```

## 7.3. Lệnh else-if

- Cú pháp

```
if (dieu_kien_1) {
 cac_cau_lenh_1;
}
else if (dieu_kien_2) {
 cac_cau_lenh_2;
} else if (dieu_kien_3) {
 cac_cau_lenh_3;
} else {
 cac_cau_lenh_n;
}
```



```
boolean isLightGreen = false; //true or false
boolean isLightYellow = false; //true or false

if(isLightGreen) {
 //traffic light is green
 System.out.println("Drive!");
} else if(isLightYellow) {
 //light is NOT green but is yellow
 System.out.println("Slow down.");
} else {
 //light is neither green nor yellow
 System.out.println("Stop.");
}
```

- Có thể có nhiều else-if, chỉ có 1 else tối đa

## 7.4. Biểu thức điều kiện

- Toán tử so sánh

| <u>Expression</u>        | <u>Value</u> |
|--------------------------|--------------|
| <code>int x = 10;</code> |              |
| <code>3 &lt; 5</code>    | true         |
| <code>3 &gt; 5</code>    | false        |
| <code>7 &lt;= 6</code>   | false        |
| <code>x &gt;= 10</code>  | true         |
| <code>x == 9</code>      | false        |
| <i>equality check</i>    |              |
| <code>x != 9</code>      | true.        |
| <i>NOT equal</i>         |              |

# 7.4. Biểu thức điều kiện (2)

## • Toán tử logic

Three main logical operators:

1) AND     $3 < 5 \ \&\& \ 2 > 15 \longrightarrow \text{false}$   
            true                          false

2) OR       $3 < 5 \ \|\ 2 > 15 \longrightarrow \text{true}$

3) NOT       $!(3 < 5) \longrightarrow \text{false}$   
            NOT true

turns a value into its opposite

Using multiple logical operators

$\&\&$  will evaluate first, then  $\|$

false  $\&\&$  true  $\|$  true  $\longrightarrow \text{true}$

false  $\&\&($  true  $\|$  true  $) \longrightarrow \text{false}$

Museum discount cases

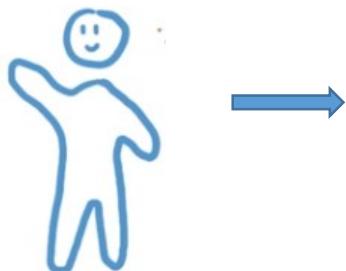
1.  $\text{age} \leq 15$

2.  $\text{age} > 60$

3.  $\text{isStudent}$

ticket

$= \$5$



if( $\text{age} \leq 15 \ \| \ \text{age} > 60 \ \| \ \text{isStudent}$ ) {

    ticket = \$5

}

# Ví dụ - Kiểm tra số chẵn – lẽ

```
class CheckNumber
{
 public static void main(String args[])
 {
 int num =10;
 if (num %2 == 0)
 System.out.println (num+ "la so chan");
 else
 System.out.println (num + "la so le");
 }
}
```

Kết quả: “10 is an even number” được hiển thị.

## 7.5. Lệnh switch - case

- Kiểm tra một biến đơn (kiểu int hoặc Enum) với nhiều giá trị khác nhau và thực hiện lệnh tương ứng



```
int passcode = ? ;
String coffeeType;
switch(passcode) {
 case 555: coffeeType = "Espresso";
 break;
 case 312: coffeeType = "Vanilla latte";
 break;
 case 629: coffeeType = "Drip coffee";
 break;
 default : coffeeType = "Unknown";
 break;
}
```

```
System.out.println(coffeeType);
```

break;

Breaks out of each case

```
int passcode = ? ;
String coffeeType;
if (passcode == 555) {
 //espresso passcode
 coffeeType = "Espresso";
} else if (passcode == 312) {
 //vanilla passcode
 coffeeType = "Vanilla latte";
} else if (passcode == 629) {
 //drip coffee code
 coffeeType = "Drip coffee";
} else{
 //unknown passcode
 coffeeType = "Unknown";
}
```

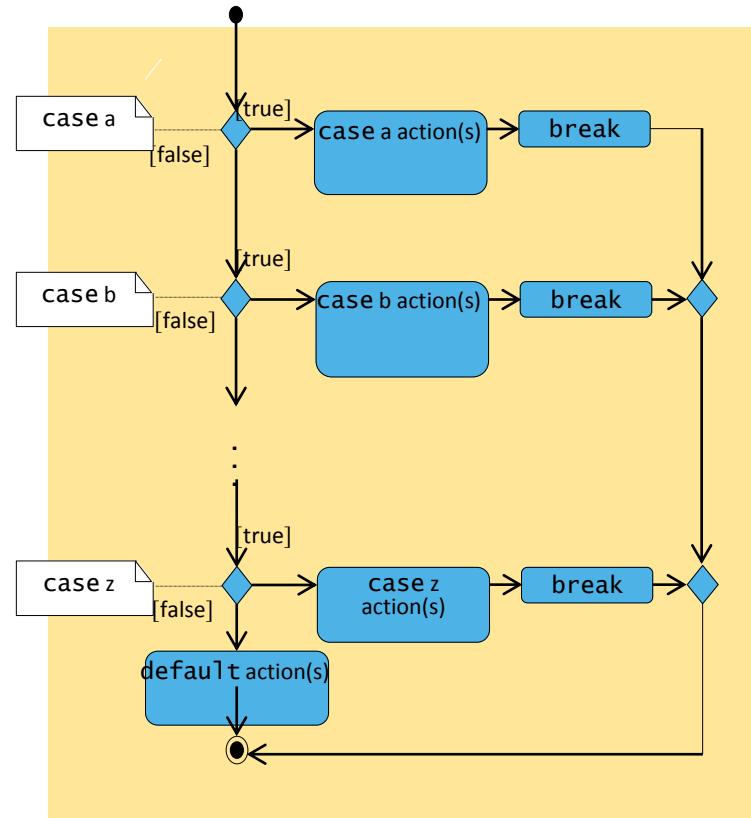
```
System.out.println(coffeeType);
```

# 7.5. Lệnh switch - case (2)

- Cú pháp:

```
switch(expression) {
 case x:
 // code block
 break;
 case y:
 // code block
 break;
 default:
 // code block
}
```

- break: Thoát khỏi lệnh switch-case
- Các case không được lặp lại
- default kiểm soát các giá trị nằm ngoài các giá trị case



# Ví dụ - Lệnh switch - case

```
public class Test {
 public static void main(String
args[]) {
 int i = 2;

 switch (i) {
 case 1:
 System.out.println("1");
 case 2:
 System.out.println("2");
 case 3:
 System.out.println("3");
 }
 }
}
```

```
switch (day) {
 case 0:
 case 1:
 rule = "weekend";
 break;
 case 2:
 case 3:
 case 4:
 case 5:
 case 6:
 rule = "weekday";
 break;
 default:
 rule = "error";
}
```

Viết thay lệnh switch-case bằng lệnh if-else ?

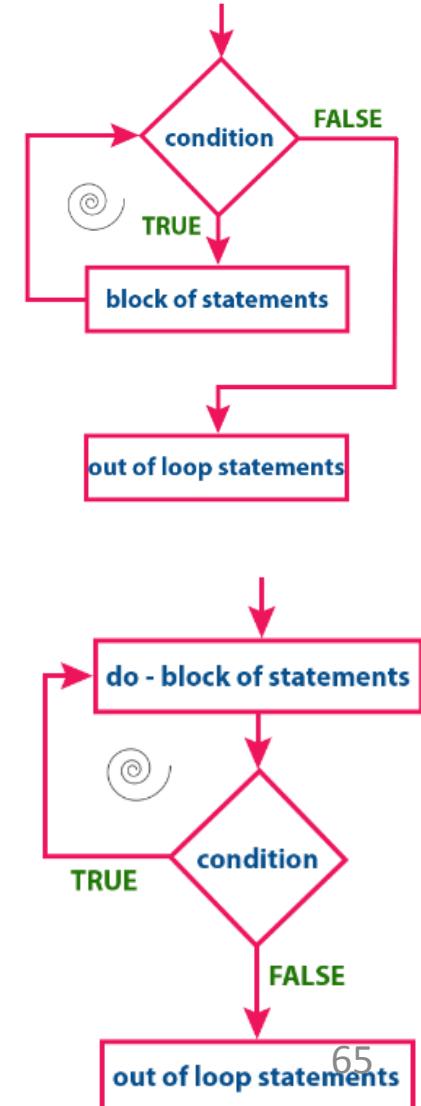
# 7.6. Vòng lặp while và do while

- Thực hiện một câu lệnh hoặc một khối lệnh khi điều kiện vẫn nhận giá trị true

```
while (condition) {
 // code block to be executed
}
```

---

```
do {
 // code block to be executed
}
while (condition);
```



- `while()` thực hiện 0 hoặc nhiều lần
- `do...while()` thực hiện ít nhất một lần

# Ví dụ - Vòng lặp while

```
int numOfWarnings = 5;
int i = 1; ← (1) loop counter
while (i <= numOfWarnings) { ← (2) loop condition
 System.out.println("Warning");
 i++; ← (3) loop increment
}
```

Print output

```
Warning
Warning
Warning
Warning
Warning
```

| i | i <= numOfWarnings |
|---|--------------------|
| 1 | 1 <= 5 (true)      |
| 2 | 2 <= 5 (true)      |
| 3 | 3 <= 5 (true)      |
| 4 | 4 <= 5 (true)      |
| 5 | 5 <= 5 (true)      |
| 6 | 6 <= 5 (false)     |

Chú ý: Tránh vòng lặp vô tận ! (luôn có lệnh thay đổi biến đếm vòng lặp)

# Ví dụ - Vòng lặp while (2)

```
class WhileDemo{
 public static void main(String args[]){
 int a = 5,fact = 1;
 while (a >= 1){
 fact *=a;
 a--;
 }
 System.out.println("The Factorial of 5 is "+fact);
 }
}
```

Kết quả: "The factorial of 5 is 120" được hiển thị.

Viết thay lệnh while bằng lệnh do-while ?

## 7.7. Vòng lặp for

- Cú pháp:

```
for (start_expr; test_expr; increment_expr) {
 // code to execute repeatedly
}
```

- Ví dụ:

| <i>loop counter</i><br>(1) | <i>loop condition</i><br>(2)          | <i>loop increment</i><br>(3) |
|----------------------------|---------------------------------------|------------------------------|
| <b>for(int i = 1;</b>      | <b>i &lt;= numOfWorkWarnings ;</b>    | <b>i++) {</b>                |
|                            | <b>System.out.println("Warning");</b> |                              |
| <b>}</b>                   |                                       |                              |

- 3 biểu thức (1) (2) (3) đều có thể vắng mặt (thay bằng lệnh tương ứng trong khối lệnh)
- Có thể khai báo biến trong câu lệnh for
  - Thường sử dụng để khai báo một biến đếm
  - Thường khai báo trong biểu thức “start”
  - Phạm vi của biến giới hạn trong vòng lặp

# Ví dụ - Vòng lặp for

```
class ForDemo
{
 public static void main(String args[])
 {
 int i=1, sum=0;
 for (i=1;i<=10;i+=2)
 sum+=i;
 System.out.println ("Sum of first five
 odd numbers is " + sum);
 }
}
```

Kết quả: “Sum of first five odd numbers is 25” được hiển thị.

## 7.8. Sử dụng for và while

- Các câu lệnh for và while cung cấp chức năng tương đương nhau
- Các cấu trúc lặp thường được sử dụng trong các tình huống khác nhau
  - while được sử dụng cho lặp từ đầu đến cuối, chưa xác định số lần lặp.
  - for được sử dụng để lặp với số vòng lặp xác định.

```
int sum = 0;
int index = 1;
while (index <= 10)
{
 sum += index;
 index++;
}
```

```
int sum = 0;
for (int index = 1; index <= 10; index++)
{
 sum += index;
}
```

## 7.8. Sử dụng for và while (2)

- Các lệnh thay đổi cấu trúc điều khiển:
- break
  - Ngoài việc được sử dụng để thoát ra ngoài câu lệnh switch còn được dùng để kết thúc sớm vòng lặp for, while hoặc do...while
  - Có hai dạng:
    - Có gắn nhãn: Tiếp tục thực hiện câu lệnh tiếp theo sau vòng lặp được gắn nhãn
    - Không gắn nhãn: Thực hiện câu lệnh tiếp theo bên ngoài vòng lặp
- continue
  - Được sử dụng cho vòng lặp for, while hoặc do...while nhằm bỏ qua các câu lệnh còn lại của lần lặp hiện thời và chuyển sang thực hiện lần lặp tiếp theo.

# Ví dụ - break và continue

```
public int myMethod(int x) {
 int sum = 0;
 outer: for (int i=0; i<x; i++) {
 inner: for (int j=i; j<x; j++) {
 sum++;
 if (j==1) continue;
 if (j==2) continue outer;
 if (i==3) break;
 if (j==4) break outer;
 }
 }
 return sum;
}
```

## 7.8. Sử dụng for và while (3)

- Các biến được khai báo trong vòng lặp hoặc khối lệnh thì chỉ có thể truy cập trong vòng lặp hoặc khối lệnh đó

```
int a = 1;
for (int b = 0; b < 3; b++) {
 int c = 1;
 for (int d = 0; d < 3; d++) {
 if (c < 3) c++;
 }
 System.out.print(c);
 System.out.println(b); abc
}
a = c; // ERROR! c is out of scope a
```

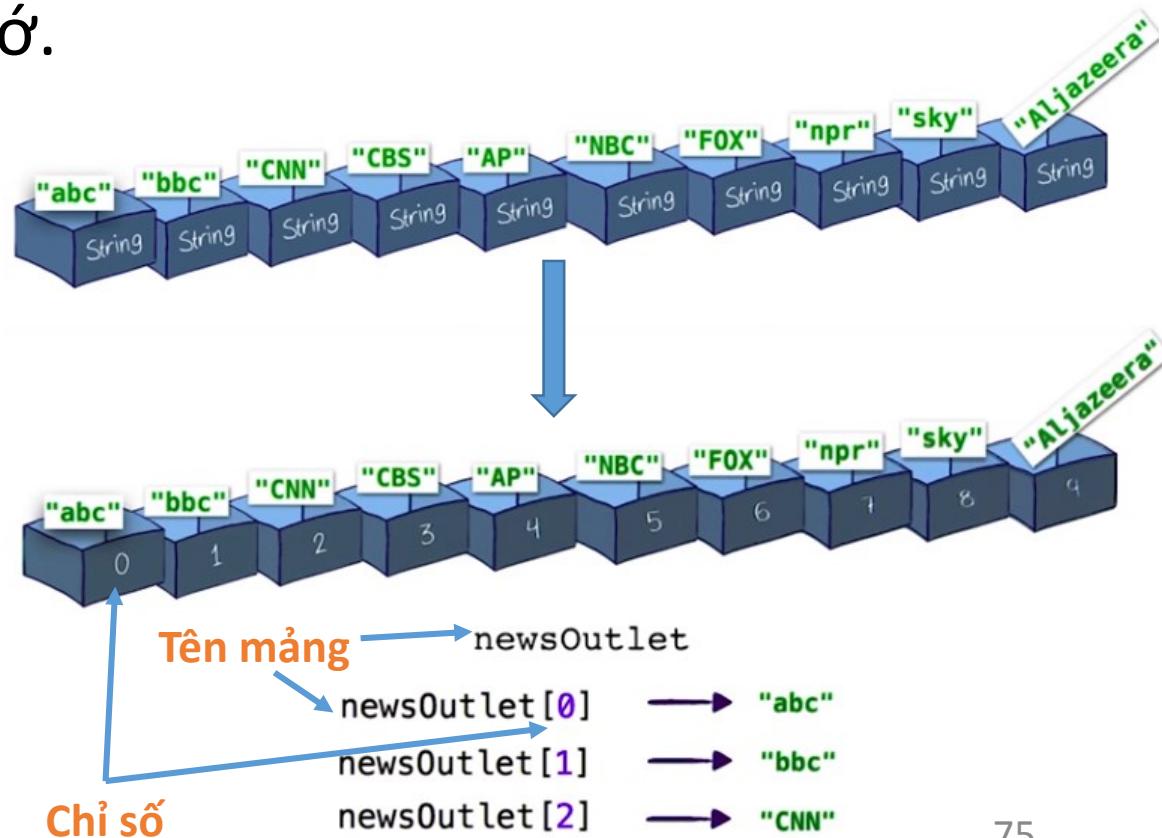
# 8. Mảng

*Tham khảo Lesson 2 – Session 21..31*

## 5.1. Khái niệm Mảng (array)

- Dùng để lưu một *tập hợp hữu hạn* các phần tử *cùng kiểu* (nguyên thuỷ hoặc đối tượng), *liền kề nhau* trong bộ nhớ.

- Mỗi mảng có 1 tên gọi
- Các phần tử được đánh số thứ tự, bắt đầu từ 0
- Truy cập từng phần tử trong mảng, dùng toán tử lấy chỉ số []
- Mỗi phần tử lúc này được xác định như 1 biến đơn



## 5.2. Khai báo và khởi tạo mảng

- Khai báo trước khi sử dụng, kích thước của một mảng sau khai báo sẽ không thể thay đổi
  - Cú pháp:
    - `kieu_dulieu[] ten_mang = new kieu_dulieu[KichThuoc_MANG];`
    - `kieu_dulieu ten_mang[] = new kieu_dulieu[KichThuoc_MANG];`
    - `kieu_dl[] ten_mang = {ds_gia_tri_cac_ptu};`
  - Nếu không khởi tạo → tất cả các phần tử của mảng nhận giá trị mặc định tùy thuộc vào kiểu dữ liệu.

```
int [] numbers ={12,1,777,3,4,0,0,121,1,-4,0,-100,2}; // Khai báo kèm khởi tạo giá trị ban đầu
```



## 5.2. Khai báo và khởi tạo mảng (2)

| Cách khai báo                  | Mô tả                                                                                     | Cú pháp                                              | Ví dụ                                                                                          |
|--------------------------------|-------------------------------------------------------------------------------------------|------------------------------------------------------|------------------------------------------------------------------------------------------------|
| Chỉ đơn thuần khai báo mảng    | Chỉ đơn thuần khai báo mảng                                                               | Datatype identifier[]                                | <b>char ch[ ];</b><br>khai báo mảng ký tự có tên ch                                            |
| Khai báo và tạo mảng           | Khai báo và cấp phát bộ nhớ cho các phần tử mảng sử dụng toán tử “new”                    | Datatype identifier[] = new datatype [size]          | <b>char ch[] = new char [10];</b><br>Khai báo một mảng ch và lưu trữ 10 ký tự                  |
| Khai báo, kiến tạo và khởi tạo | Khai báo mảng, cấp phát bộ nhớ cho nó và gán các giá trị ban đầu cho các phần tử của mảng | Datatype identifier[] = {value1, value2 ... valueN}; | <b>char ch [] = { 'A' , 'B' , 'C' , 'D' };</b><br>khai báo mảng ch và lưu 4 chữ cái kiểu ký tự |

## 5.3. Làm việc với mảng

- Dùng thuộc tính `.length` để lấy kích thước của một mảng
- Lưu ý không truy cập vào các chỉ số không thuộc mảng, ví dụ chỉ số âm, chỉ số  $\geq$  kích thước mảng.
- Duyệt tất cả các phần tử trong mảng: dùng vòng lặp.



```
int size = temperatures.length; ← 4
System.out.println(temperatures[10]); Error!
ArrayIndexOutOfBoundsException
```

```
for(int i=0; i<size; i++){
 // truy cập temperatures[i];
}
```

loop counter: (0, 1, 2 ...)

# Ví dụ về mảng

```
int MAX = 5;
boolean bit[] = new boolean[MAX];
float[] value = new float[2*3];
int[] number = {10, 9, 8, 7, 6};
System.out.println(bit[0]); // "false"
System.out.println(value[3]); // "0.0"
System.out.println(number[1]); // "9"
```

## 5.4. Mảng 2 chiều

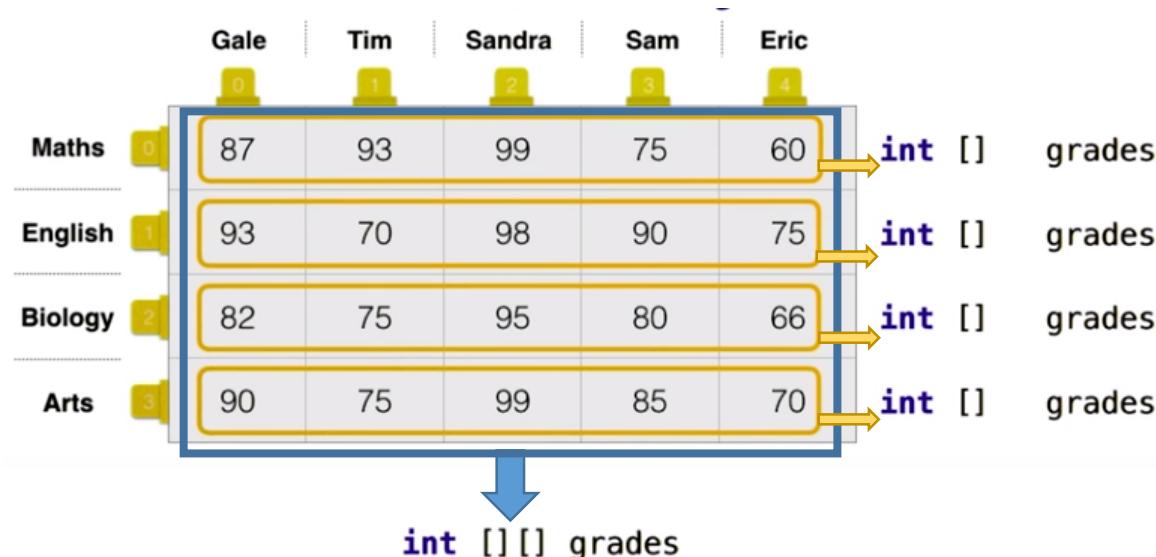
- Có thể khai báo và sử dụng một mảng nhiều chiều.
- Khi mảng 2 chiều, giống một bảng với các dòng và cột.

|         | Gale | Tim | Sandra | Sam | Eric |
|---------|------|-----|--------|-----|------|
| Maths   | 87   | 93  | 99     | 75  | 60   |
| English | 93   | 70  | 98     | 90  | 75   |
| Biology | 82   | 75  | 95     | 80  | 66   |
| Arts    | 90   | 75  | 99     | 85  | 70   |

| 2D Array |      |     |        |     |      |    |
|----------|------|-----|--------|-----|------|----|
|          | Gale | Tim | Sandra | Sam | Eric |    |
| Maths    | 0    | 87  | 93     | 99  | 75   | 60 |
| English  | 1    | 93  | 70     | 98  | 90   | 75 |
| Biology  | 2    | 82  | 75     | 95  | 80   | 66 |
| Arts     | 3    | 90  | 75     | 99  | 85   | 70 |

## 5.4. Mảng 2 chiều (2)

- Mảng 2 chiều: coi như 1 mảng của các phần tử A, mỗi phần tử A lại là 1 mảng các phần tử B.
- Khai báo mảng 2 chiều `kieu_dulieu[][] ten_mang;`



## 5.4. Mảng 2 chiều (3)

- Truy cập phần tử trong mảng:

ten\_mang [chi\_so\_hang] [chi\_so\_cot]

- Duyệt tất cả các phần tử trong mảng:
  - Dùng vòng lặp lồng nhau

|         | Gale | Tim | Sandra | Sam | Eric |
|---------|------|-----|--------|-----|------|
| Maths   | 87   | 93  | 99     | 75  | 60   |
| English | 93   | 70  | 98     | 90  | 75   |
| Biology | 82   | 75  | 95     | 80  | 66   |
| Arts    | 90   | 75  | 99     | 85  | 70   |

grades[2][1] = 75  
array #2 (Biology)  
item #1 (Tim)

```
for(int i=0; i<4; i++){
 for (int j=0; j<5; j++){
 // truy cập grades[i][j];
 }
}
```

# Bài tập – Nội dung 1

- **Bài tập 1:** Viết chương trình tráo đổi ngẫu nhiên vị trí một dãy số cho trước  
Để lấy một số int ngẫu nhiên từ 0 đến n-1 ta dùng lệnh  
`int i = Random.nextInt(n);`
- **Bài tập 2:** Viết chương trình sắp xếp một dãy số theo thứ tự tăng dần, dãy số được nhập từ bàn phím.
- **Bài tập 3:** Viết chương trình nhập chiều cao h từ bàn phím, sau đó hiển thị các tam giác hình sao có chiều cao h như dưới đây. Chú ý có kiểm tra điều kiện của h:  $2 \leq h \leq 10$ . Nếu h nằm ngoài đoạn trên, yêu cầu người dùng nhập lại.
- **Bài tập 4:** Nhập vào kích thước ô vuông  $n*n$ , kiểm tra  $3 \leq n \leq 8$ . Hiển thị ra màn hình kết quả như ví dụ sau.

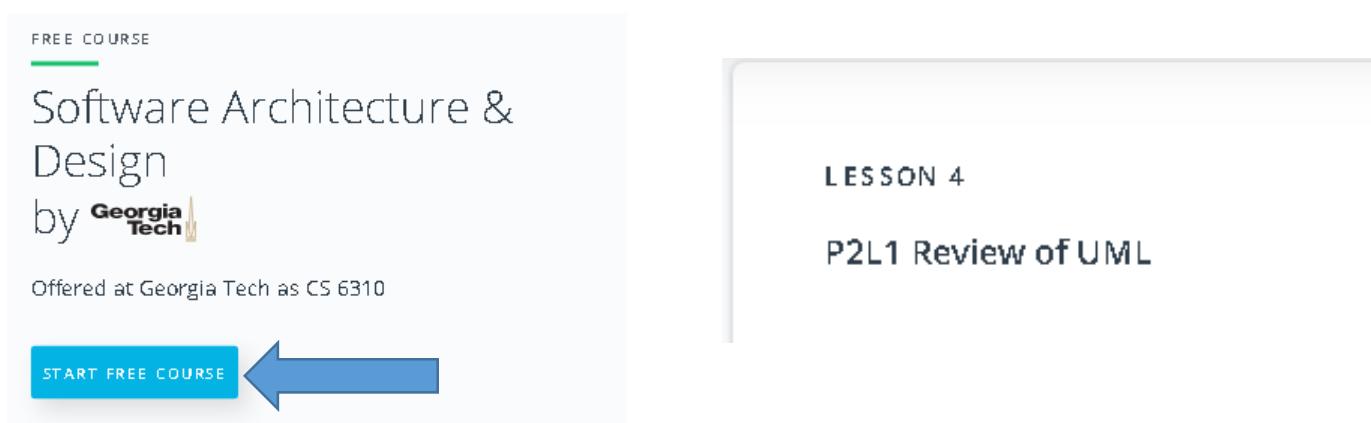
|    |    |    |   |
|----|----|----|---|
| 1  | 2  | 3  | 4 |
| 12 | 13 | 14 | 5 |
| 11 | 16 | 15 | 6 |
| 10 | 9  | 8  | 7 |

# Nội dung 2: Giới thiệu về UML

# Bài giảng e-learning đi kèm

- Đăng ký vào trang <https://www.udacity.com>
- Theo dõi bài giảng có tên “Software Architecture & Design”  
<https://www.udacity.com/course/software-architecture-design--ud821>

Tập trung vào Lesson 4, các bài khác: tham khảo thêm



The image shows a screenshot of a course landing page. At the top left, it says "FREE COURSE". Below that is the course title "Software Architecture & Design" followed by "by Georgia Tech". Underneath the title, it says "Offered at Georgia Tech as CS 6310". At the bottom left is a blue button labeled "START FREE COURSE". To the right of this, there is a thumbnail for "LESSON 4 P2L1 Review of UML". A large blue arrow points from the "START FREE COURSE" button towards the "P2L1 Review of UML" thumbnail.

# Bài giảng e-learning đi kèm (2)

## Cấu trúc bài giảng e-learning

The screenshot shows a user interface for an e-learning platform. On the left, there is a dark sidebar menu titled "Lesson 4: P2L1 Review of UML". The menu includes sections for "SEARCH" and "RESOURCES", and a "CONCEPTS" section with five items: 1. Diagrams (marked with a checkmark), 2. OMT, 3. UML, 4. Diagram Types, and 5. Quiz: Diagram Quiz. A blue arrow points downwards from the top of the sidebar towards the "CONCEPTS" list. On the right, the main content area features a video player. The video shows a man speaking, with a "UML" logo and the word "Diagrams" displayed below him. A subtitle at the bottom of the video frame reads: "way of doing that is with diagrams. This course focused on UML, using UML and". Below the video player, there is a control bar with icons for play, volume, and progress (0:10 / 1:06). A blue arrow points upwards from the bottom of the video player towards the "SEND FEEDBACK" button in the top right corner.

Chọn phụ đề nếu cần

# Bài giảng e-learning đi kèm (3)

- Bài giảng của Smartdraw
  - <https://www.smartdraw.com/uml-diagram/>
  - <https://www.smartdraw.com/use-case-diagram/>
  - <https://www.smartdraw.com/activity-diagram/>
  - <https://www.smartdraw.com/sequence-diagram/>
  - <https://www.smartdraw.com/class-diagram/>

# Mục tiêu bài học

## Làm quen với UML

- Giới thiệu ngôn ngữ mô hình hóa UML
- Lịch sử phát triển UML
- Giới thiệu các biểu đồ cơ bản
- Làm quen 4 biểu đồ thông dụng nhất
  - Biểu đồ Use case
  - Biểu đồ Hoạt động
  - Biểu đồ Lớp
  - Biểu đồ Tương tác

# Nội dung

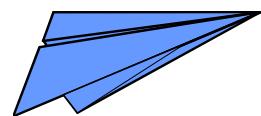
1. UML là gì ?
2. Các biểu đồ UML cơ bản
3. Giới thiệu biểu đồ Usecase
4. Giới thiệu biểu đồ Hoạt động
5. Giới thiệu biểu đồ Lớp
6. Giới thiệu biểu đồ Tương tác

# 1. UML là gì

*Tham khảo Lesson 4 – Session 1, 2, 3*

## 1.1. Tầm quan trọng của phân tích và thiết kế

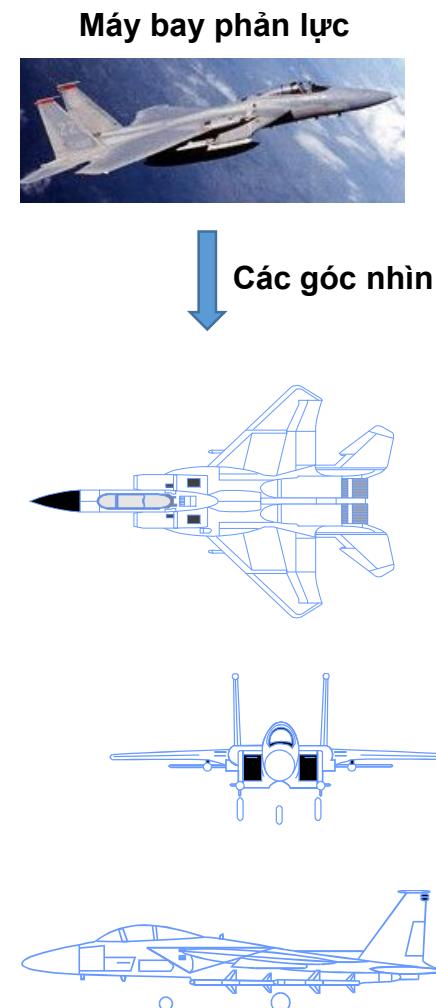
- Hướng tiếp cận không có phân tích – thiết kế:
  - Bắt đầu lập trình ngay khi có được yêu cầu
  - Mất rất nhiều thời gian và tạo đi tạo lại nhiều mã nguồn
  - Không có bất kỳ một kiến trúc nào
  - Phải chịu khổ với những lỗi phát sinh
- Hướng tiếp cận có phân tích – thiết kế:
  - Chuyển các yêu cầu của bài toán thành một bản thiết kế rõ ràng
  - Tập trung vào phân tích các YÊU CẦU và thiết kế các MÔ HÌNH cho hệ thống TRƯỚC khi lập trình



## 1.1. Tầm quan trọng của phân tích và thiết kế (2)

- **Ưu điểm của việc PTTK hệ thống:**

- Đơn giản hóa thế giới thực bằng các mô hình
- Mô tả đúng, đồng nhất cấu trúc, cách ứng xử của HT trong suốt quá trình xây dựng
- Đảm bảo mục đích và yêu cầu của HT được thỏa mãn trước khi xây dựng
- Cung cấp cho người dùng, khách hàng, kỹ sư phân tích, thiết kế, kỹ sư lập trình nhiều cái nhìn khác nhau về cùng một HT
- Ghi lại các quyết định của nhà phát triển để sử dụng sau này



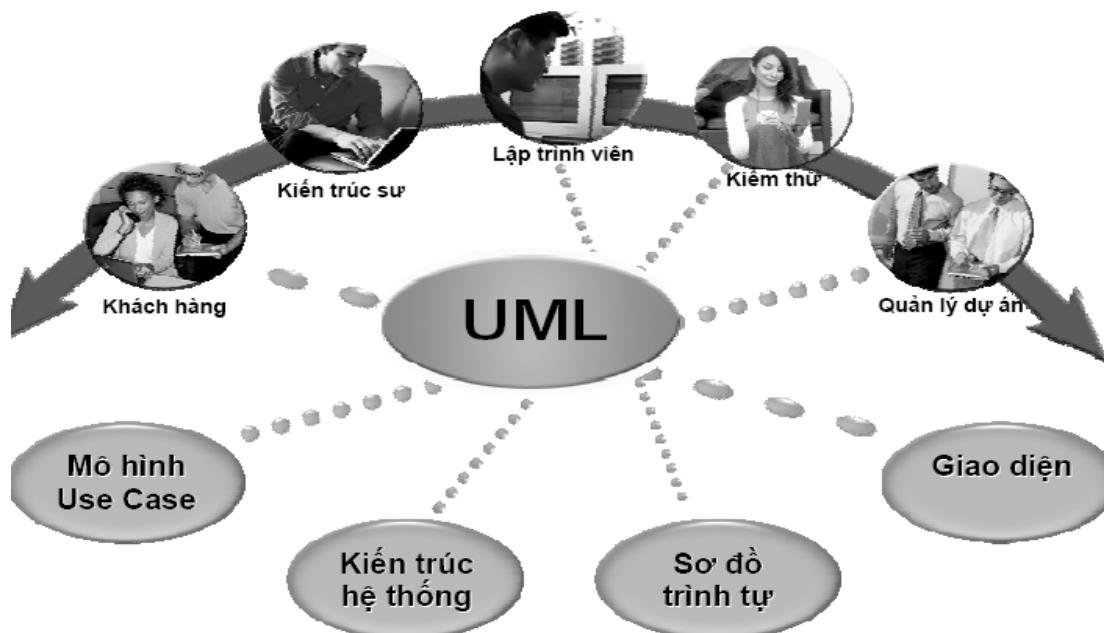


## 1.2. Khái niệm UML

- UML: viết tắt của “Unified Modeling Language” là một Ngôn ngữ mô hình hóa được thống nhất
- UML là ngôn ngữ trực quan để:
  - trực quan hóa (visualizing)
  - đặc tả (specifying)
  - xây dựng (constructing)
  - tài liệu hóa (documenting)
- các cấu phần của một hệ thống phần mềm
- Giúp công việc phát triển được xử lý nhất quán, giảm thiểu lỗi xảy ra
  - Giúp dễ hình dung hơn cấu trúc của hệ thống
  - Hiệu quả hơn trong việc liên lạc, trao đổi

## 1.2. Khái niệm UML (2)

- Thiết lập một phương thức thống nhất để xây dựng và “vẽ” ra các yêu cầu và thiết kế hướng đối tượng trong quá trình PTTK phần mềm



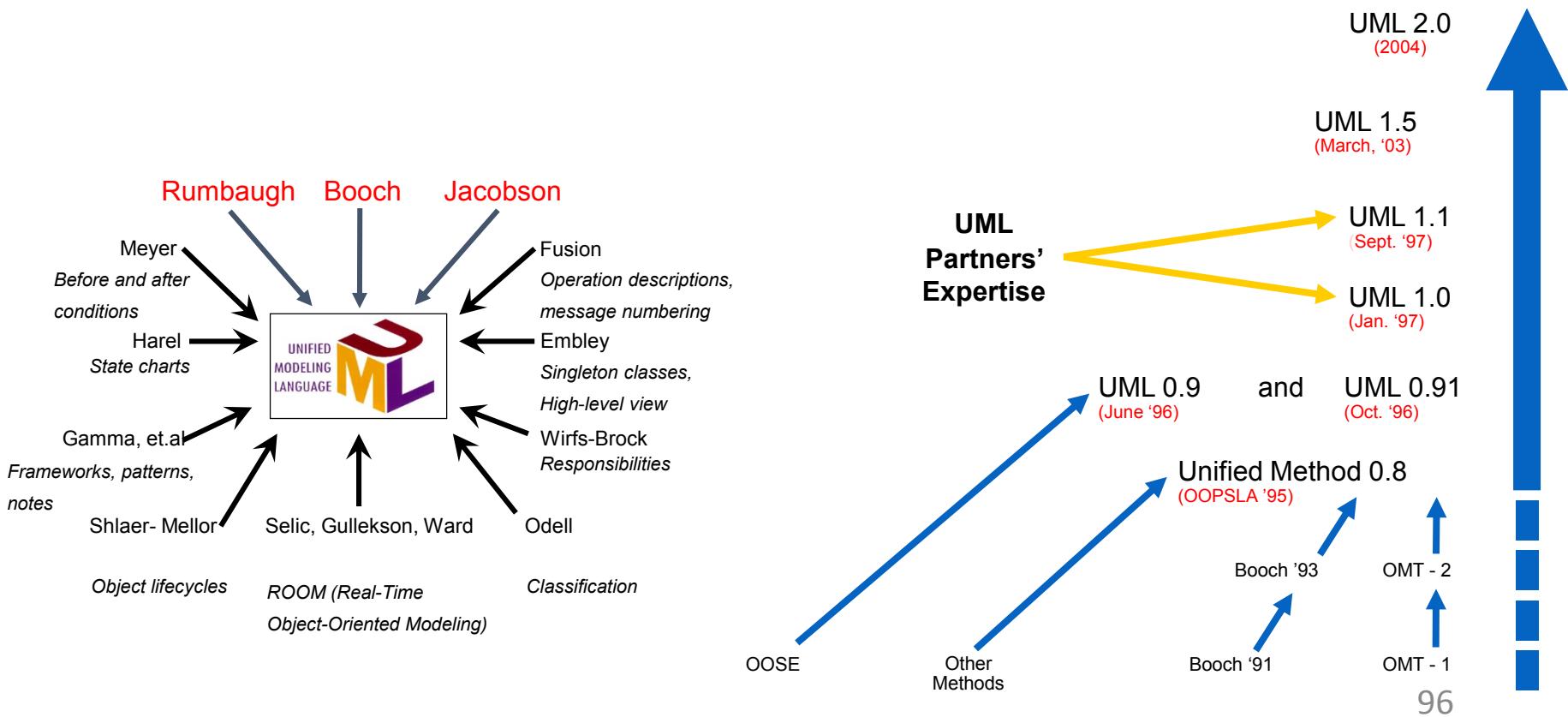
# 1.3. Lịch sử phát triển UML

- Vào năm 1994, có hơn 50 phương pháp mô hình hóa hướng đối tượng:
  - *Fusion, Shlaer-Mellor, ROOM, Class-Relation, Wirfs-Brock, Coad-Yourdon, MOSES, Syntropy, BOOM, OOSD, OSA, BON, Catalysis, COMMA, HOOD, Ooram, DOORS ...*
  - Mô tả về mô hình “Meta-models” tương đồng với nhau
  - Các ký pháp đồ họa khác nhau
  - Quy trình khác nhau hoặc không rõ ràng

→ Cần chuẩn hóa và thống nhất các phương pháp
- UML được 3 chuyên gia hướng đối tượng hợp nhất các kỹ thuật của họ vào năm 1994:
  - Booch91 (Grady Booch): Conception, Architecture
  - OOSE (Ivar Jacobson): Use cases
  - OMT (Jim Rumbaugh): Analysis

# 1.3. Lịch sử phát triển UML (2)

- UML là ngôn ngữ hợp nhất các mô hình khác nhau* Được công nhận là chuẩn chung vào năm 1997.



# 1.4. Làm việc với UML

- Các mô hình UML có thể kết nối trực tiếp với rất nhiều ngôn ngữ lập trình.
  - Ánh xạ sang Java, C++, Visual Basic...
  - Các bảng trong RDBMS hoặc kho lưu trữ trong OODBMS
  - Cho phép các kỹ nghệ xuôi (chuyển UML thành mã nguồn)
  - Cho phép kỹ nghệ ngược (xây dựng mô hình hệ thống từ mã nguồn)
- Các công cụ UML
  - Công cụ mã nguồn mở: EclipseUML, UmlDesigner, StarUML, Argo UML...
  - Công cụ thương mại: Enterprise Architect, IBM Rational Software Architect, Microsoft Visio, Visual Paradigm for UML, SmartDraw...

## 2. Giới thiệu các biểu đồ UML cơ bản

*Tham khảo Lesson 4*

## 2.1. Biểu đồ UML

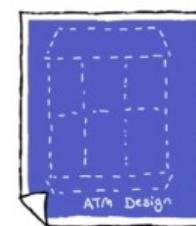
- Biểu đồ:
  - là các hình vẽ bao gồm các ký hiệu phần tử mô hình hóa
  - minh họa một thành phần cụ thể hay một khía cạnh cụ thể của hệ thống.
- Một mô hình hệ thống thường có nhiều loại biểu đồ, mỗi loại gồm nhiều biểu đồ khác nhau.
- Một biểu đồ là một thành phần của một hướng nhìn cụ thể
- Một số loại biểu đồ có thể là thành phần của nhiều hướng nhìn khác nhau
- UML thế hệ 2 có tới 13-14 loại biểu đồ. Trong một project, chỉ sử dụng những biểu đồ phù hợp nhất

## 2.1. Biểu đồ UML (2)

- Phân biệt:
  - **Biểu đồ cấu trúc:** mô tả thành phần tinh, luôn có của hệ thống và mối quan hệ giữa chúng
  - **Biểu đồ hành vi:** mô tả cách hoạt động của hệ thống.

### Diagram Types

Two Main Categories:



Structural



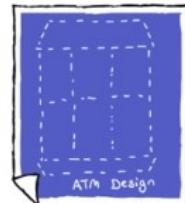
Behavioral

*what is contained in a system*

*what must happen in a system*

## 2.2. Các loại biểu đồ UML cơ bản

- Biểu đồ cấu trúc



Structural

- Biểu đồ cấu trúc tĩnh

- Biểu đồ lớp (Class Diagram)

Tham khảo Lesson 4

- Biểu đồ đối tượng (Object Diagram)

Session 6..9

- Biểu đồ gói (Package diagram)

Session 10

- Biểu đồ thực thi

- Biểu đồ thành phần (Component Diagram)

Session 12-13

- Biểu đồ triển khai (Deployment Diagram)

Session 14-15

- Biểu đồ cấu thành (Composite Diagram)

Session 11

- Biểu đồ profile (Profile Diagram)

Session 19

## 2.2. Các loại biểu đồ UML cơ bản (2)

- Biểu đồ hành vi

- Biểu đồ use case (Use Case Diagram)



Tham khảo Lesson 4

- Biểu đồ hoạt động (Activity Diagram)

Behavioral

Session 24..28

- Biểu đồ tương tác

- Biểu đồ tổng quát (Interaction overview diagram)

Session 37

- Biểu đồ trình tự (Sequence Diagram)

Session 31-32

- Biểu đồ giao tiếp/cộng tác (Communication/Collaboration Diagram)

Session 33-34

- Biểu đồ trạng thái (State Diagram)

Session 39-40

- Biểu đồ thời gian (Timing Diagram)

Session 38

### 3. Biểu đồ Use case

*Tham khảo Lesson 4 – Session 24..28, bài giảng Smartdraw*

<https://www.smartdraw.com/use-case-diagram/>

<https://www.youtube.com/watch?v=Omp4RbHbB0s>

### 3.1. Khái niệm use case

- Use case mô tả một chuỗi các hành động của người dùng cùng với chuỗi các đáp ứng/phản hồi của hệ thống để thực hiện 1 việc, 1 chức năng nào đó của hệ thống.
- Nội dung use case giống như kịch bản/đoạn hội thoại mô tả cách sử dụng hệ thống: luôn có cấu trúc “người dùng làm gì – hệ thống phản ứng như thế nào”.
- Cần mô tả kỹ về cách người dùng sử dụng hệ thống:
  - Người dùng làm theo kịch bản thì hệ thống phản ứng thế nào ?: kịch bản cơ sở
  - Người dùng làm khác kịch bản thì hệ thống phản ứng thế nào ?: kịch bản thay thế
  - ...
- Use case được mô tả bởi văn bản, hoặc bằng bảng

## 3.2. Mô tả một use case

Tên use case: “Buy a book”

Foster wants to buy a book. He goes to the Amazon web site and browses to find his book. He adds it to his shopping cart. He then goes to the check-out page where he provides his billing information and confirms his purchase.

Từng hành động trong use case:

- + Ai, làm gì, lên đối tượng nào
- + Hệ thống, làm gì, lên đối tượng nào

| Actor  | Action     | Object                |
|--------|------------|-----------------------|
| Foster | browses    | Amazon website        |
| Foster | selects    | book                  |
| Foster | adds       | book, shopping cart   |
| Foster | checks out |                       |
| Amazon | requests   | billing information   |
| Foster | provides   | name, address, credit |
| Amazon | verifies   | credit information    |
| Foster | confirms   | purchase              |

## 3.2. Mô tả một use case (2)

- Mô tả use case bằng khuôn mẫu

**USE CASE Id:** <NAME - should be the goal as a short active verb phrase>

**Context of use:** <a longer statement of the goal, if needed, its normal occurrence conditions>

**Scope:** <design scope, what system is being considered black-box under design>

**Level:** <one of: summary, user-goal, subfunction>

**Primary Actor:** <a role name for the primary actor, or description>

**Stakeholders & Interests:** <list of stakeholders and key interests in the use case>

**Precondition:** <what we expect is already the state of the world>

**Minimal Guarantees:** <how the interests are protected under all exits>

**Success Guarantees:** <the state of the world if goal succeeds>

**Trigger:** <what starts the use case, may be time event>

**Main Success Scenario:**

<put here the steps of the scenario from trigger to goal delivery and any cleanup after>

<step #> <action description>

**Extensions:**

<put here there extensions, one at a time, each referring to the step of the main scenario>

<step altered> <condition>: <action or sub use case>

<step altered> <condition>: <action or sub use case>

....

### 3.3. Biểu đồ Use case

- Mô tả tập hợp các use case của hệ thống
  - Mỗi use case mô tả một yêu cầu chức năng của hệ thống
- Các thành phần chính:
  - Tác nhân : Thường là người dùng, hệ thống ngoài, thiết bị ngoài, v.v. có tương tác với hệ thống.



- Các Use case



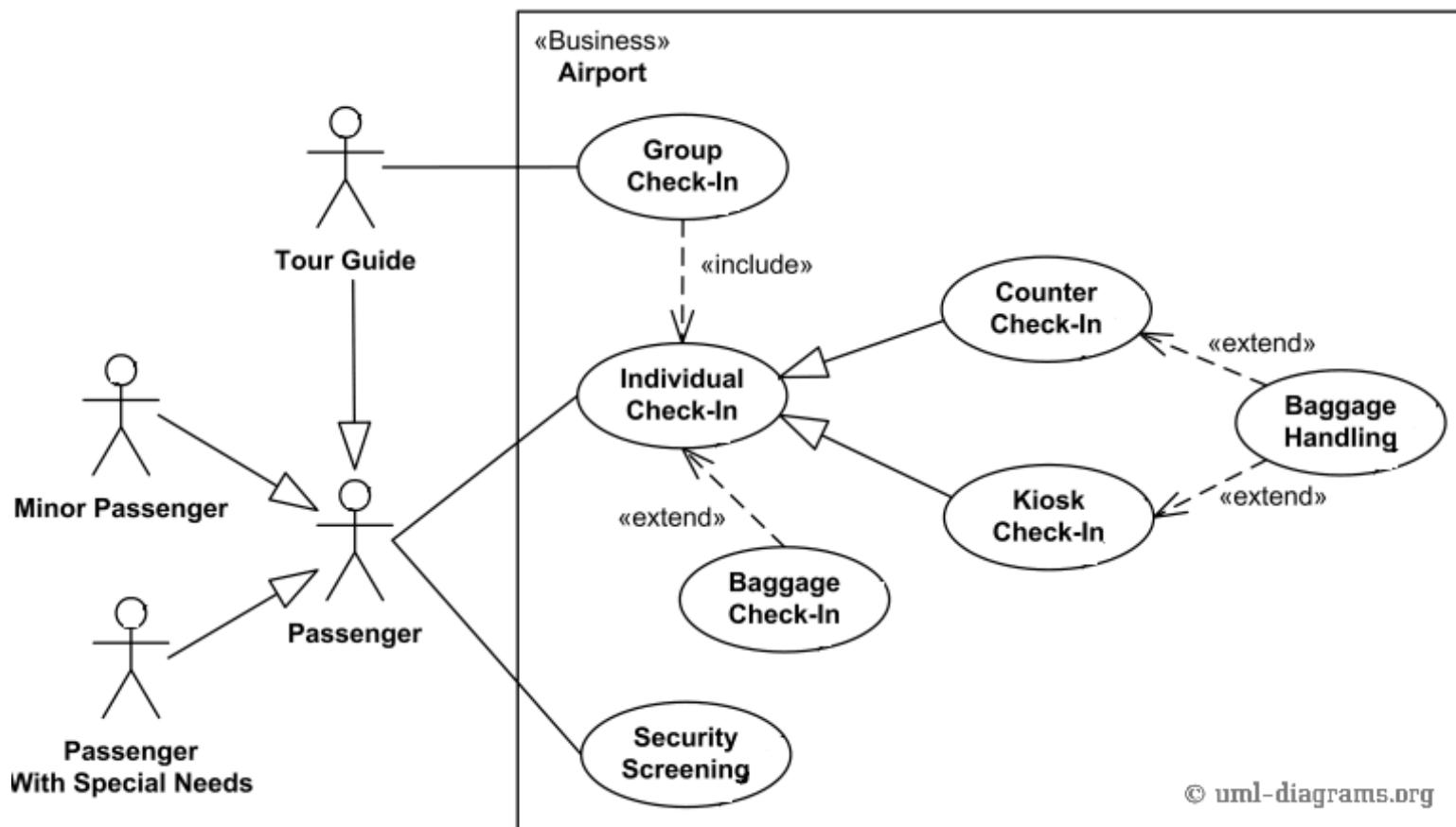
- Đường bao hệ thống



### 3.3. Biểu đồ Use case (2)

- Liên kết trong biểu đồ use case
    - Liên kết không có nhãn: liên kết tác nhân tham gia vào use case
    - Liên kết có nhãn <<extend>>: Use case B *có thể được sử dụng* trong use case A.
    - Liên kết có nhãn <<include>>: Use case B *luôn được sử dụng* trong use case A.
    - Liên kết tổng quát hóa: mô tả quan hệ khái quát hóa của tác nhân hoặc use case, A là khái quát hóa của B.
- 
- The diagram illustrates several types of associations in UML:
- Actor to Use Case:** An actor (stick figure) is connected to a use case (oval). This is a simple participation relationship.
  - Extend Association:** Use case A is connected to use case B via a dashed blue arrow labeled <<extend>>. This indicates that use case B can be used within the context of use case A.
  - Include Association:** Use case A is connected to use case B via a dashed blue arrow labeled <<include>>. This indicates that use case B is always used within the context of use case A.
  - Generalization Association:** Use case A is connected to use case B via a solid blue arrow pointing from A to B. This indicates that use case A generalizes use case B.

# Ví dụ - Biểu đồ Use case



### 3.4. Cách xây dựng BĐ use case

- Người PT phần mềm thống nhất với khách hàng về bản mô tả hệ thống chi tiết, các yêu cầu chức năng, phi chức năng, v.v.
- Xác định các use case từ bản mô tả hệ thống:
  - Mỗi use case mô tả một yêu cầu chức năng của hệ thống, chưa quan tâm tới việc chức năng đó được thực thi như thế nào.
  - Tên use case là một Cụm động từ, ví dụ “Mua hàng”
- Xác định các tác nhân bên ngoài có ảnh hưởng đến hệ thống
  - Tác nhân là nhóm người có 1 vai trò rõ ràng. Tên tác nhân là Danh từ.
  - Ví dụ “Người mua”, “Quản trị viên” chứ không phải tên riêng của người dùng “Bob”, “John” ...
- Hoàn thiện các liên kết, đường bao hệ thống.

# 4. Biểu đồ Hoạt động

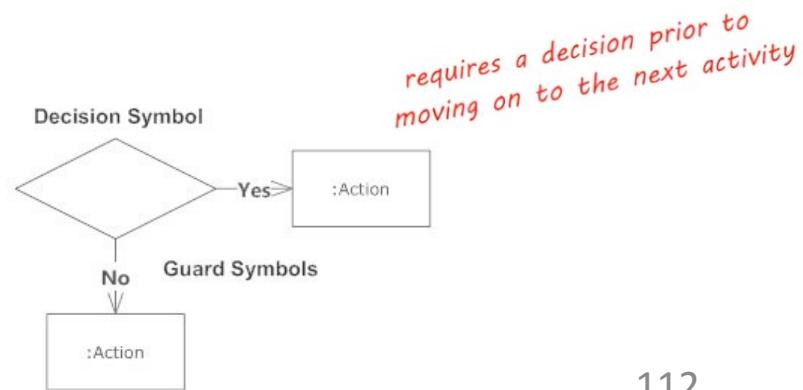
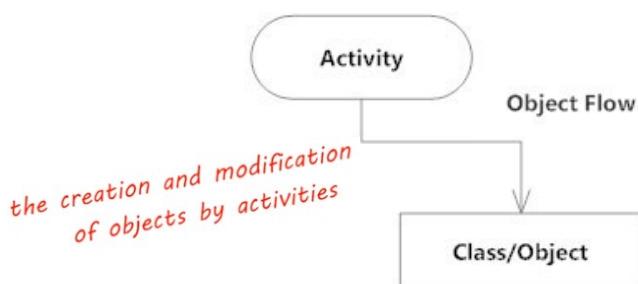
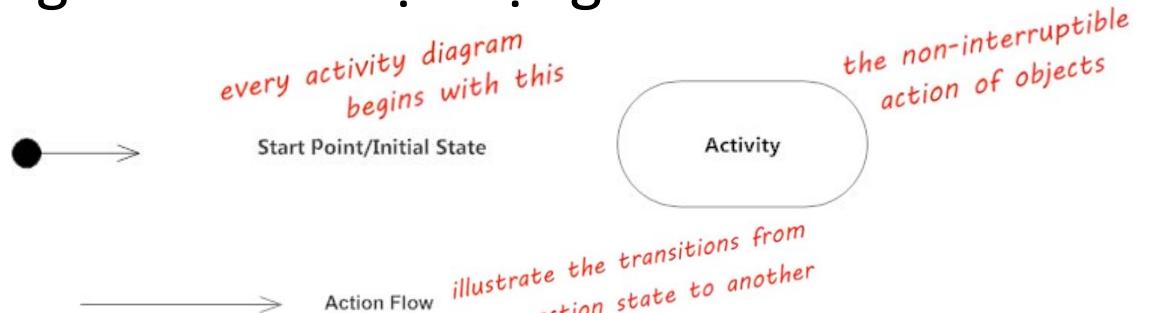
*Tham khảo Lesson 4 – Session 35-36, bài giảng Smartdraw*

<https://www.smartdraw.com/activity-diagram/>

[https://www.youtube.com/watch?v=Wf\\_xlagfHmg](https://www.youtube.com/watch?v=Wf_xlagfHmg)

## 4.1. Khái niệm BĐ hoạt động

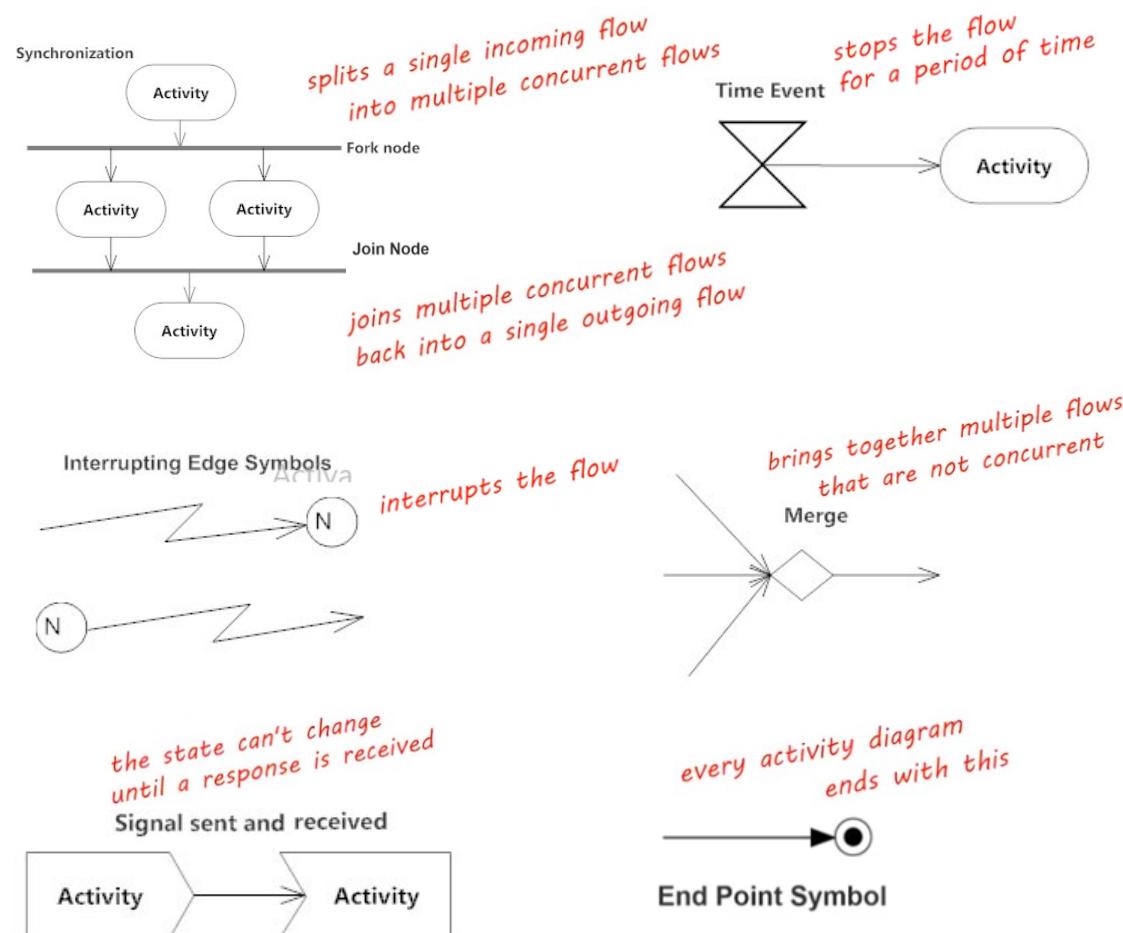
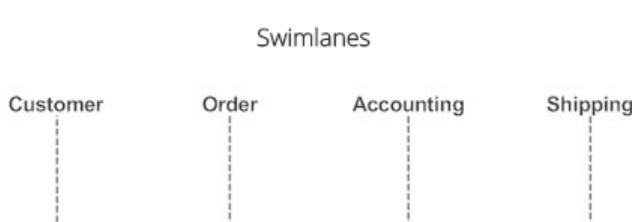
- Biểu đồ hoạt động được sử dụng để biểu diễn chuỗi các hoạt động hoặc luồng điều khiển có thứ tự của hệ thống, được thực hiện trong một use case.
- Các thành phần trong biểu đồ hoạt động
  - Điểm khởi đầu
  - Hoạt động
  - Luồng hành động
  - Luồng đối tượng
  - Quyết định rẽ nhánh



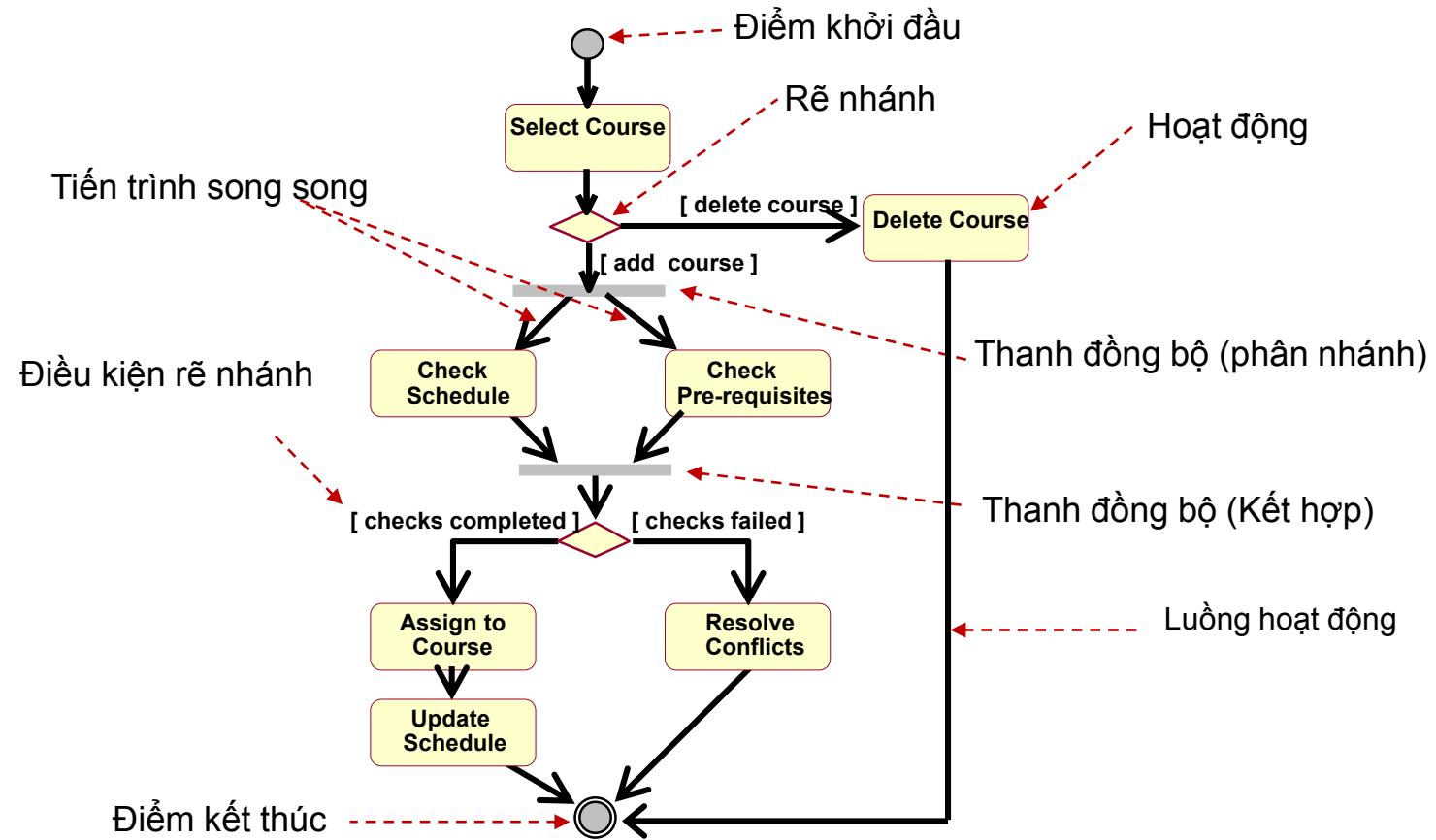
# 4.1. Khái niệm BĐ hoạt động (2)

- Các thành phần trong biểu đồ hoạt động (tiếp)

- Thanh đồng bộ
- Sự kiện thời gian
- Sự kiện gộp
- Gửi nhận tín hiệu
- Sự kiện ngắn
- Làn hoạt động
- Điểm kết thúc

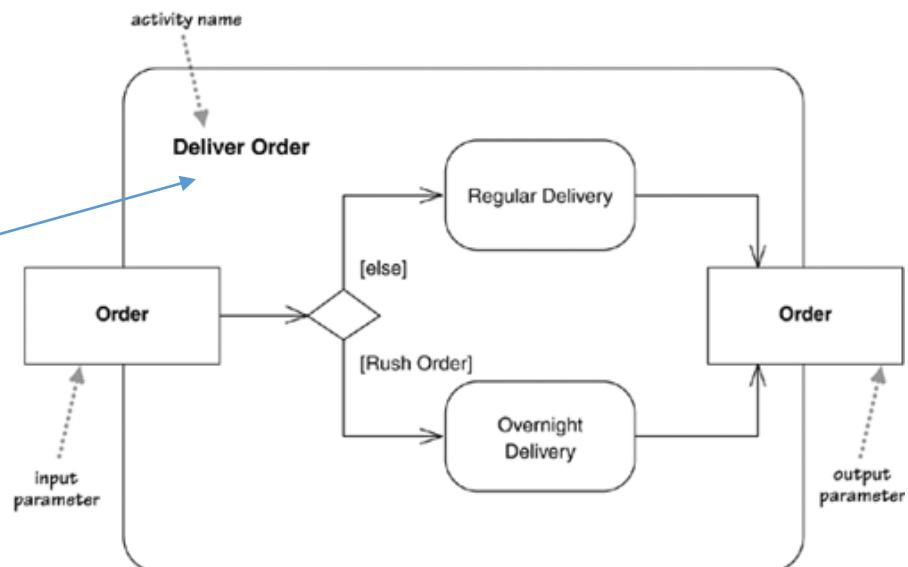
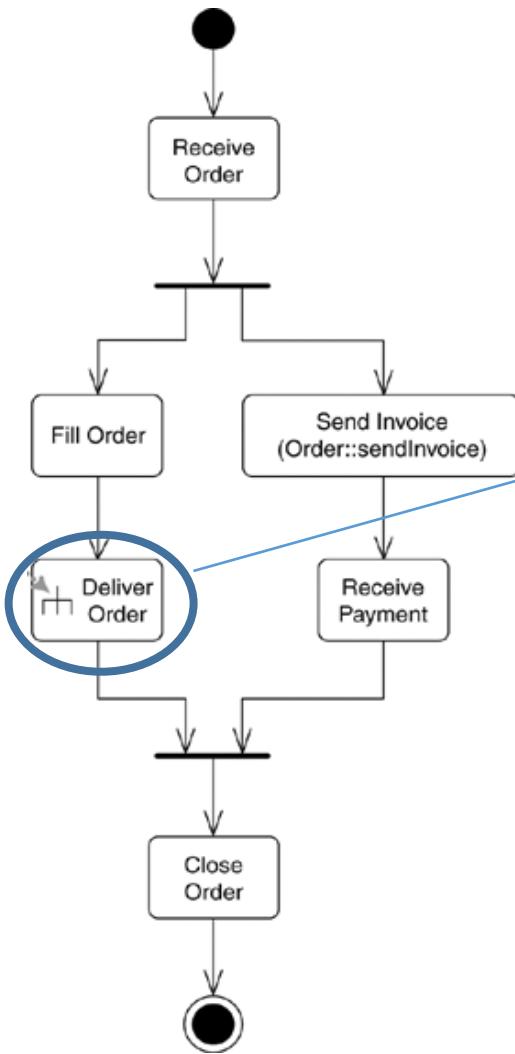


## 4.2. Ví dụ



## 4.2. Ví dụ (2)

Gọi một  
biểu đồ  
hoạt động  
khác



# 5. Biểu đồ lớp

*Tham khảo Lesson 4 – Session 6-9, bài giảng Smartdraw*

<https://www.smartdraw.com/class-diagram/>

[https://www.youtube.com/watch?time\\_continue=1&v=Wl0oyCeon2A](https://www.youtube.com/watch?time_continue=1&v=Wl0oyCeon2A)

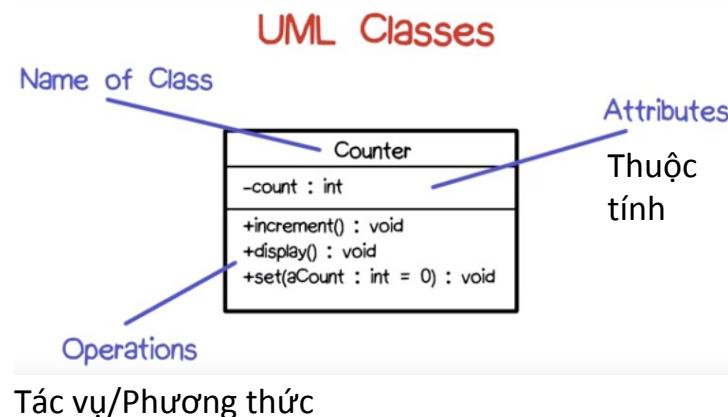
## 5.1. Khái niệm lớp

- Lớp là khái niệm mô tả 1 tập hợp các đối tượng mang cùng một ý nghĩa (bao gồm cả thuộc tính, hành vi/tác vụ, quan hệ).
  - Mỗi một đối tượng sẽ là một thực thể / cá thể rõ ràng của lớp.
    - Mỗi sinh viên là một đối tượng với
      - các thuộc tính: tên, tuổi, khoa, lớp, khoá,...
      - các tác vụ: học bài, lên lớp, làm bài kiểm tra,...
    - Mỗi chiếc điện thoại là một đối tượng với
      - các thuộc tính: số SIM, model, kích thước,...
      - các tác vụ: gọi số, nhắn tin, nghe cuộc gọi tới, từ chối cuộc gọi,...
- Mô hình hóa → Lớp Student
- Mô hình hóa → Lớp SmartPhone



## 5.2. Biểu đồ lớp

- Dạng biểu đồ phổ biến nhất, mô tả cấu trúc tinh của hệ thống
- Biểu diễn các lớp và mối quan hệ giữa các lớp



| Class Name   |
|--------------|
| attributes   |
| operations() |

| Class Name                                                            |
|-----------------------------------------------------------------------|
| - private attributes<br>+ public attributes<br># protected attributes |
| - private operation<br>+ public operation<br># protected operation    |

## 5.2. Biểu đồ lớp (2)

- Các ký hiệu

- Lớp: tên lớp, thuộc tính, tác vụ/phương thức
- Phạm vi truy cập: ai được truy cập các thông tin trong lớp
  - - : private: không được truy cập từ ngoài lớp
  - + : public: truy cập được từ bất kỳ lớp/phương thức ngoài lớp
  - # : protected: truy cập được từ các lớp thừa kế
- Quan hệ giữa các lớp:
  - Liên kết (Association): có kết nối, có sử dụng
    - Bội số quan hệ (Multiplicity)
    - Có hướng (navigability): chỉ rõ hướng liên kết
    - Ràng buộc (constraint): có điều kiện về liên kết

| Indicator | Meaning                 |
|-----------|-------------------------|
| 0..1      | Zero or one             |
| 1         | One only                |
| 0..*      | 0 or more               |
| 1..*      | *                       |
| n         | Only n (where n > 1)    |
| 0..n      | Zero to n (where n > 1) |
| 1..n      | One to n (where n > 1)  |

◀ association

1

\*

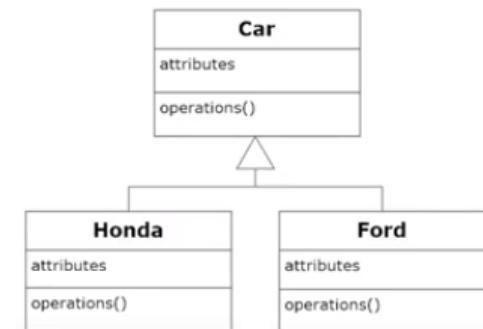
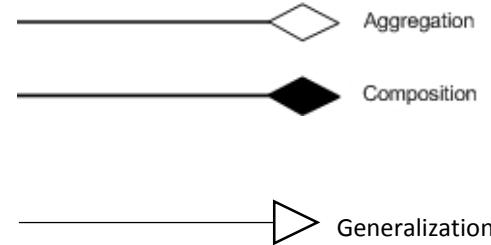
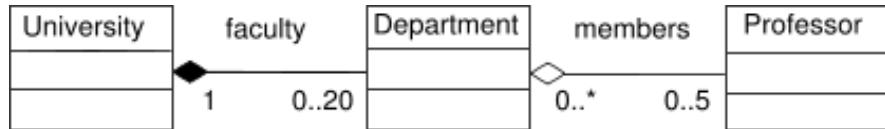
— {constraint} —

## 5.2. Biểu đồ lớp (3)

- Các ký hiệu (tiếp)

- Quan hệ giữa các lớp (tiếp):

- Kết tập (Aggregation): mô tả quan hệ toàn thể - bộ phận hay còn gọi là quan hệ “is a part of”.
    - Cấu thành (composition): là một dạng kết tập , bộ phận không thể tồn tại nếu toàn thể bị hủy bỏ.
    - Tổng quát hóa (generalization): khi một lớp là một dạng cụ thể của một lớp khác. Hay còn gọi là quan hệ “is a kind of”
      - Lớp con thừa hưởng các thuộc tính/phương thức public và protected của lớp mẹ



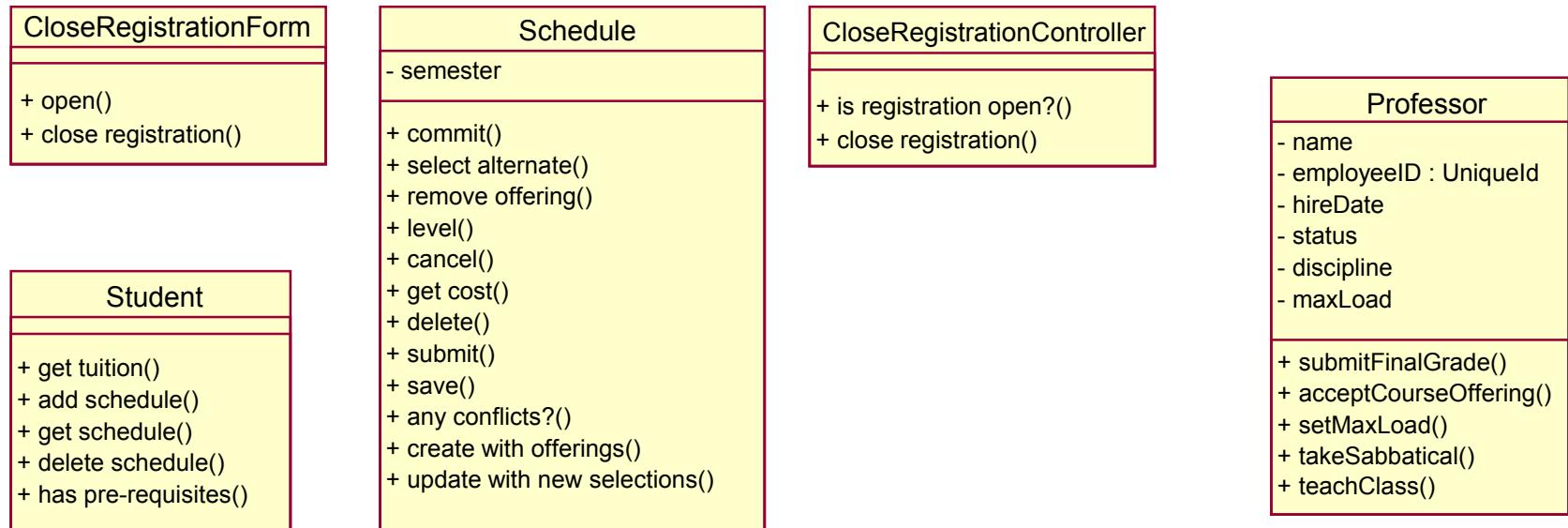
Trong quan hệ Agrregation, lớp toàn thể chỉ có thể truy cập vào các thành phần Public của lớp bộ phận.

## 5.3. Cách phát hiện lớp

- Phân tích bản mô tả yêu cầu về hệ thống, phát hiện ra tập các đối tượng có thể có trong hệ thống (dựa trên các Danh từ, Ngữ danh từ).
- Phân tích bản mô tả use case của hệ thống, bổ sung các đối tượng phát sinh.
- Thống nhất các đối tượng trùng nhau
  - Ví dụ “shopping cart”, “shopping basket”, “shopping trolley” thành “shopping cart”
- Loại bỏ các đối tượng vô nghĩa, hoặc không đủ lớn thành một lớp
  - Ví dụ “Internet”, “Password”, “Title”, ...
- Giao diện giữa hệ thống và bên ngoài có thể coi là các đối tượng biên
  - Ví dụ: form đăng nhập, form đăng ký tín chỉ, v.v.
- Có thể hình thành các đối tượng điều khiển nếu cần
  - Ví dụ: đối tượng điều khiển đăng nhập RegistrationController v.v.
- Mô hình hóa các đối tượng thành các lớp, biểu diễn quan hệ giữa các lớp.

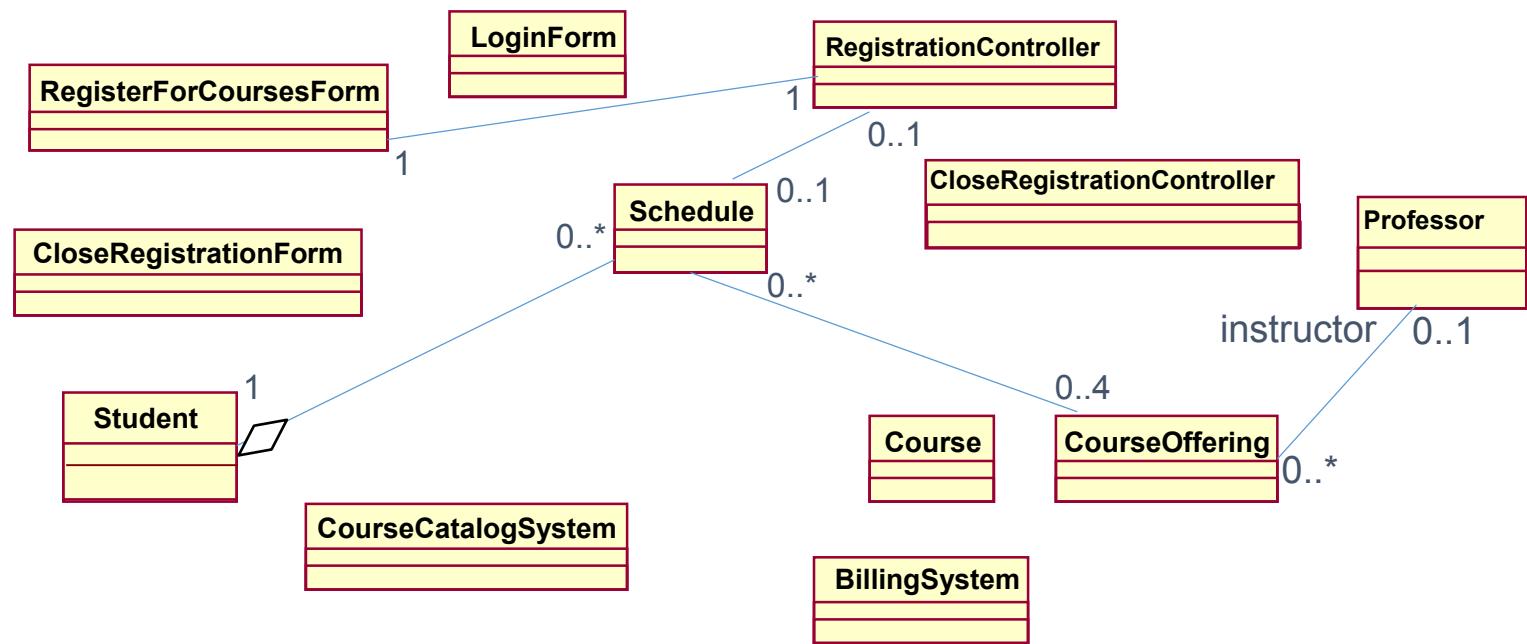
# Ví dụ: Hệ thống đăng ký khóa học

- Các lớp cơ bản



# Ví dụ: Hệ thống đăng ký khóa học (2)

- Biểu đồ lớp sơ lược

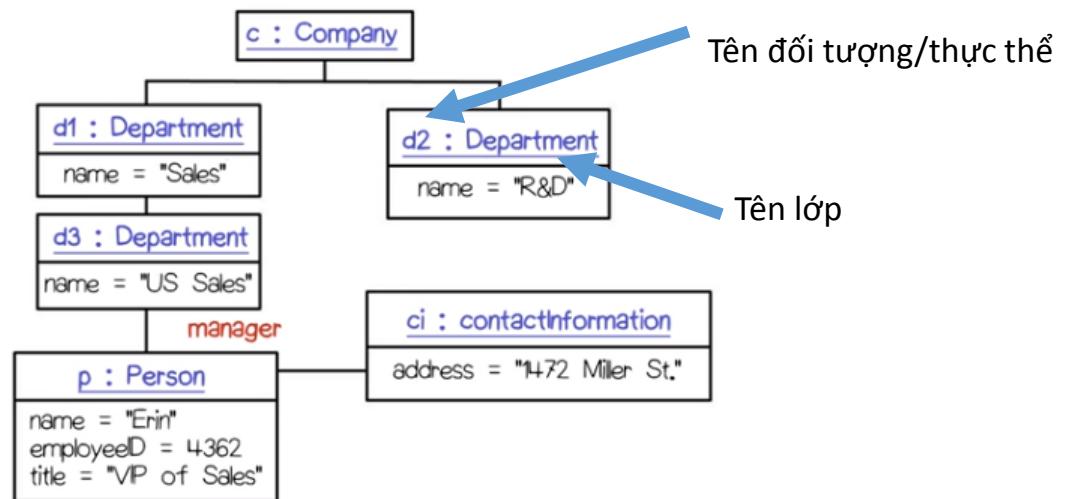


## 5.4. Biểu đồ đối tượng

- Giống biểu đồ lớp, thay vì biểu diễn lớp, biểu đồ đối tượng biểu diễn các thực thể thật của lớp và quan hệ giữa chúng

### Object Diagram

Conveys **objects** and **links** instead of **classes** and **relationships**



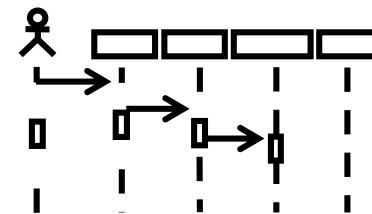
# 6. Biểu đồ tương tác

# 6. Biểu đồ tương tác

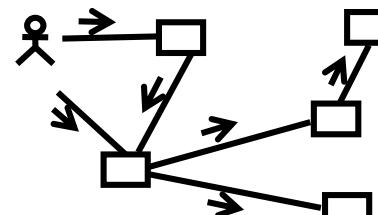
- Xác định hành vi của các đối tượng trong hệ thống, cách chúng tương tác với nhau để thực hiện 1 use case nào đó.
- Các đối tượng tương tác với nhau thông qua các thông điệp
- Thông điệp có thể hiểu như là một hành động, có thể gây ra sự thay đổi trạng thái của đối tượng.
- Thông điệp sẽ yêu cầu đối tượng nhận thông điệp thực hiện một phương thức nào đó
  - do đó phải bổ sung vào đối tượng nhận thông điệp phương thức xử lý thông điệp được gửi tới
- Khi phương thức được thực hiện xong thì đối tượng nhận sẽ trả quyền điều khiển lại cho đối tượng gọi, cùng với giá trị trả về

# 6. Biểu đồ tương tác (2)

- Biểu đồ tương tác mô tả tương tác giữa các đối tượng
  - Biểu đồ tuần tự (Sequence diagram)
    - Một cách nhìn hướng về trình tự thời gian tương tác giữa các đối tượng
  - Biểu đồ giao tiếp (Communication diagram)
    - Một cách nhìn thông điệp giữa các đối tượng, hướng về cấu trúc của quá trình truyền



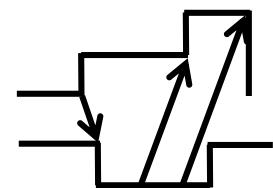
Biểu đồ tuần tự



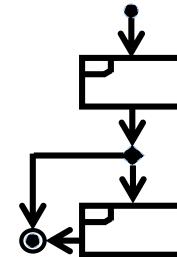
Biểu đồ giao tiếp

# 6. Biểu đồ tương tác (3)

- Biểu đồ tương tác mô tả tương tác giữa các đối tượng
  - Biểu đồ thời gian (Timing Diagram)
    - Một cách nhìn về sự ràng buộc thời gian của các thông điệp trong một tương tác.
    - Thường sử dụng trong các ứng dụng thời gian thực, vì trong các ứng dụng này yếu tố thời gian mang tính quyết định
  - Biểu đồ tương tác tổng quát (Interaction Overview Diagram)
    - Một cách nhìn tương tác ở mức cao bằng cách kết hợp các biểu đồ tương tác theo một trình tự logic nào đó.



Biểu đồ thời gian



Biểu đồ tương tác  
tổng quan

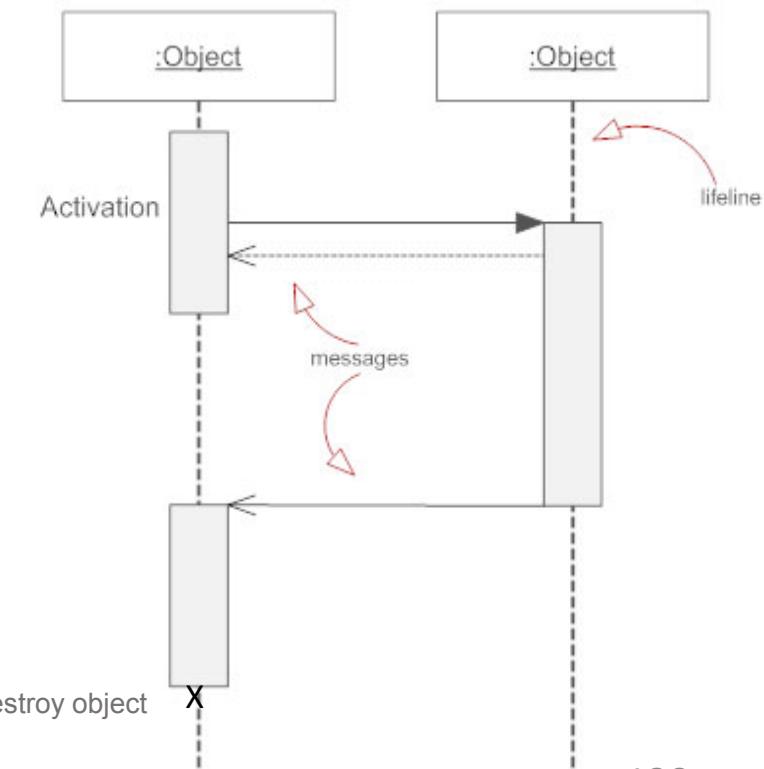
## 6.1. Biểu đồ tuần tự

*Tham khảo Lesson 4 – Session 31-32, bài giảng Smartdraw*

<https://www.smartdraw.com/sequence-diagram/>

## 6.1.1. Biểu đồ tuần tự

- Nhấn mạnh vào *trình tự* trao đổi thông điệp giữa các đối tượng theo thời gian trong một use case
- Các ký hiệu
  - Đối tượng (:Object)
  - Thông điệp (Message)
  - Vùng kích hoạt (Activation)
  - Đường sống (Lifeline)
  - Hủy đối tượng (Destroying)



## 6.1.1. Biểu đồ tuần tự (2)

- Các ký hiệu (tiếp)

- Các loại thông điệp:

- Thông điệp đồng bộ: yêu cầu chờ phản hồi



- Thông điệp không đồng bộ: không cần chờ phản hồi



- Thông điệp trả về



- Thông điệp tự gọi



- Thông điệp tạo đối tượng mới

- Thông điệp hủy đối tượng



- Thông điệp nhận được từ một đối tượng chưa xác định



- Thông điệp gửi tới một đối tượng chưa xác định



## 6.1.1. Biểu đồ tuần tự (3)

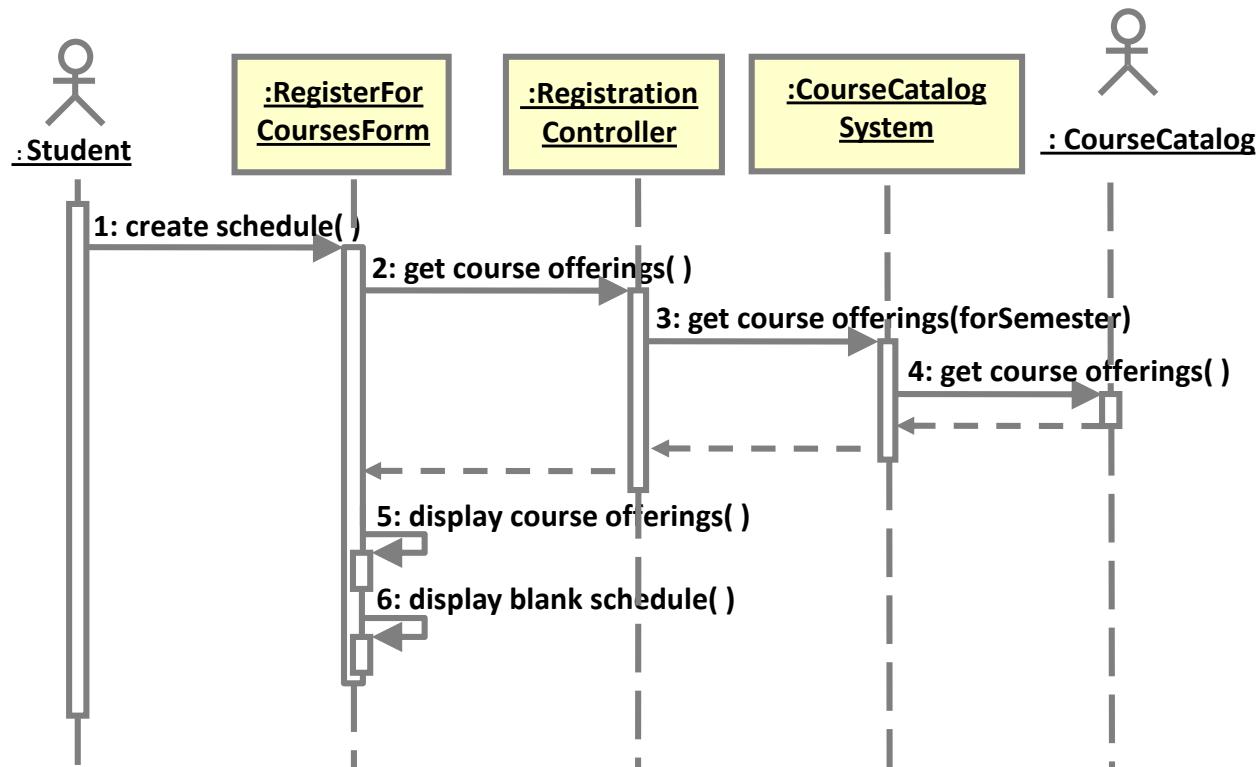
- Các ký hiệu (tiếp)
  - Khung tương tác (Interaction frame)

Toán tử

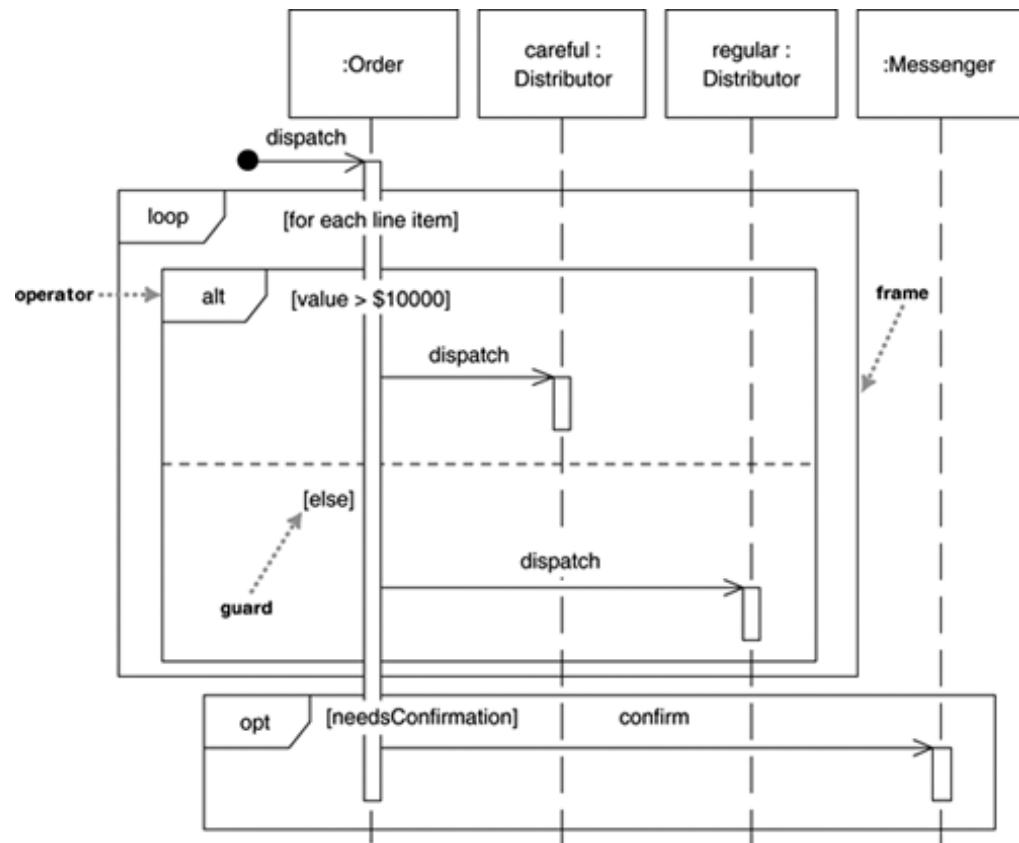
Tương tác

| Toán tử | Ý nghĩa                                                                                                                        |
|---------|--------------------------------------------------------------------------------------------------------------------------------|
| alt     | Khung lựa chọn nhiều, chỉ có lựa chọn có điều kiện đúng sẽ được thực hiện                                                      |
| opt     | Tùy chọn, chỉ thực hiện khi điều kiện thỏa mãn                                                                                 |
| par     | Song song, mỗi khung chạy song song                                                                                            |
| loop    | Lặp lại, khung có thể được thực hiện nhiều lần                                                                                 |
| region  | Vùng then chốt, tại một thời điểm chỉ có một luồng chạy nó                                                                     |
| ref     | Tham chiếu đến một tương tác khác trong biểu đồ khác, vẽ trùm trên các lifetime liên quan, có thể có tham số và giá trị trả về |
| sd      | Vẽ xung quanh 1 biểu đồ biểu đồ trình tự nếu cần                                                                               |

# Ví dụ - Biểu đồ tuần tự



# Ví dụ - Biểu đồ tuần tự (2)



```
procedure dispatch
foreach (lineitem)
 if (product.value>$10K)
 careful.dispatch
 else
 regular.dispatch
 end if
end for
if (needsConfirmation)
 messenger.confirm
end procedure
```

## 6.1.2. Lưu ý

- Sau khi vẽ xong biểu đồ tuần tự, cần kiểm tra trong biểu đồ lớp, mỗi thông điệp tới một đối tượng đã được biểu diễn thành một phương thức của lớp hay chưa? Nếu chưa, bổ sung phương thức vào lớp tương ứng.

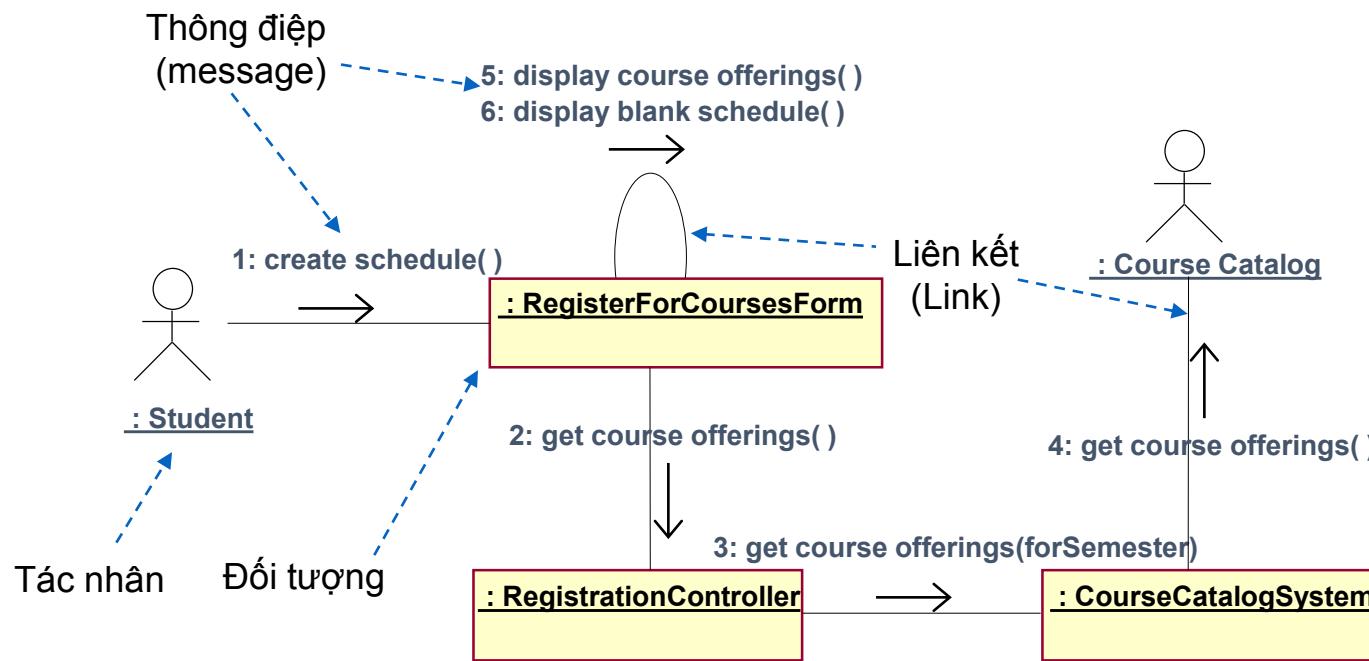
## 6.2. Biểu đồ giao tiếp

*Tham khảo Lesson 4 – Session 33-34*

## 6.2.1. Biểu đồ giao tiếp

- Biểu đồ giao tiếp nhấn mạnh vào việc tổ chức các đối tượng tham gia vào tương tác.
- Biểu đồ giao tiếp chỉ ra:
  - Các đối tượng tham gia vào tương tác.
  - Các liên kết giữa các đối tượng.
  - Các thông điệp trao đổi giữa các đối tượng.
- Các ký hiệu:
  - Tác nhân
  - Đối tượng
  - Thông điệp
  - Liên kết

## 6.2.1. Biểu đồ giao tiếp (2)



## 6.2.2. SD và CD – Giống nhau

- Tương đương về ngữ nghĩa
  - Cùng đưa ra thông tin về sự tương tác giữa các đối tượng qua các thông điệp
  - Có thể chuyển đổi giữa hai biểu đồ mà không mất mát thông tin
- Mô hình hóa phương diện động của hệ thống
- Mô hình hóa kịch bản use case.

### 6.2.3. SD và CD – Khác nhau

| <b>Biểu đồ tuần tự</b>                                                                                                                                                                                                                                                                                  | <b>Biểu đồ giao tiếp</b>                                                                                                                                                                                                                                                                                                                         |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"><li>– Chỉ ra thứ tự rõ ràng của các thông điệp</li><li>– Thể hiện tốt hơn luồng công việc</li><li>– Mô hình hóa trực quan hơn toàn bộ luồng thực thi (theo thời gian)</li><li>– Thể hiện tốt hơn đối với các đặc tả thời gian thực và các kịch bản phức tạp</li></ul> | <ul style="list-style-type: none"><li>– Chỉ ra mối quan hệ rõ ràng giữa các đối tượng</li><li>– Thể hiện tốt hơn quá trình giao tiếp</li><li>– Mô hình hóa trực quan hơn cho tất cả các ảnh hưởng của đối tượng</li><li>– Thể hiện rõ hơn hiệu quả của quá trình tương tác trên từng đối tượng, dễ hiểu hơn cho các buổi brainstorming</li></ul> |

# Bài tập làm quen UML

## Bài 1. Cho:

- Các tác nhân: Người mua, Hệ thống E-mail, Hệ thống cho vay và Hệ thống báo cáo tín dụng
- Các use case: Tìm người môi giới, Quản lý hồ sơ cá nhân, Tìm kiếm nhà và Yêu cầu vay
- Các mối liên kết:
  - Từ người mua tới Tìm người môi giới
  - Từ người mua tới Quản lý hồ sơ cá nhân
  - Từ người mua tới Tìm kiếm nhà
  - Từ người mua tới Yêu cầu vay
  - Quản lý hồ sơ cá nhân tới Hệ thống e-mail
  - Tìm kiếm nhà tới Hệ thống e-mail
  - Yêu cầu vay tới Hệ thống e-mail, Hệ thống cho vay
  - Yêu cầu vay tới Hệ thống báo cáo tín dụng

Hãy vẽ biểu đồ use case tương ứng.

# Bài tập làm quen UML

## Bài 2: Cho:

- Các trạng thái hành động:
  - Chọn hồ sơ
  - Tìm hồ sơ người mua
  - Tạo hồ sơ mới
  - Đăng nhập
- Luồng hoạt động:
  - Bắt đầu từ Chọn hồ sơ tới Tìm hồ sơ người mua rồi đi từ Tìm hồ sơ người mua đến Tạo hồ sơ mới nếu hồ sơ không tồn tại. Nếu hồ sơ tồn tại thì có thể Đăng nhập.

Hãy vẽ Biểu đồ hoạt động tương ứng.

# Bài tập làm quen UML

**Bài 3:** Một biểu đồ lớp gồm các lớp sau: Personal Planner Profile, Personal Planner Controller, Customer Profile, và Buyer Record. Các lớp có các quan hệ sau:

- Mỗi một đối tượng Personal Planner Profile có thể liên kết với tối đa một đối tượng Personal Planner Controller.
- Mỗi đối tượng Personal Planner Controller phải được liên kết với một đối tượng Personal Planner Profile.
- Một đối tượng Personal Planner Controller có thể được liên kết với tối đa một đối tượng Buyer Record và Customer Profile.
- Một thực thể của lớp Buyer Record có thể liên quan tới 0 hoặc 1 đối tượng Personal Planner Controller.
- Có 0 hoặc 1 đối tượng Personal Planner Controller được liên kết với mỗi thực thể Customer Profile.

Hãy vẽ biểu đồ lớp tương ứng

# Bài tập làm quen UML

**Bài 4:** Biểu đồ giao tiếp hoạt động như sau:

1. Người dùng Prospective Buyer bắt đầu biểu đồ giao tiếp bằng cách yêu cầu đối tượng Personal Planner Profile (PPF) cập nhật một hồ sơ.
2. Đối tượng PPF yêu cầu đối tượng Personal Planner Controller (PPC) cập nhật một hồ sơ
3. Đối tượng PPC gửi một thông điệp cho đối tượng Buyer Record yêu cầu tìm hồ sơ.
4. Sau khi tìm xong đối tượng PPF hiển thị hồ sơ.
5. Người dùng Prospective Buyer cập nhật thông tin trong hồ sơ và yêu cầu PPF lưu trữ thông tin vừa cập nhật.
6. Đối tượng PPF nhận thông tin mới và yêu cầu đối tượng PPC lưu trữ thông tin mới.
7. Đối tượng PPC yêu cầu đối tượng Buyer Record cập nhật lại bản ghi hồ sơ với những thông tin mà người dùng đã cung cấp.

Hãy vẽ biểu đồ tuần tự, biểu đồ giao tiếp tương ứng.

