



LẬP TRÌNH Java

BÀI 1: NHỮNG KHÁI NIỆM JAVA

❑ Kết thúc bài học này bạn có khả năng

- ❖ Hiểu ngôn ngữ lập trình Java
- ❖ Biết cách thiết lập môi trường cho ứng dụng java
- ❖ Nắm cấu trúc chương trình Java
- ❖ Sử dụng công cụ NetBean
- ❖ Biết cách nhập dữ liệu từ bàn phím
- ❖ Biết cách xuất dữ liệu ra màn hình
- ❖ Biết cách thực hiện các phép toán số học
- ❖ Biết cách sử dụng các hàm toán học

❑ Java là ngôn ngữ lập trình
có các đặc điểm sau

- ❖ Hướng đối tượng
- ❖ Chạy trên mọi nền tảng
- ❖ Bảo mật cao
- ❖ Mạnh mẽ
- ❖ Phân tán
- ❖ Đa luồng xử lý
- ❖ ...



Write once, run anywhere

LỊCH SỬ PHÁT TRIỂN CỦA JAVA

1991

Ra đời với tên gọi Oak bởi Sun Microsystem

1995

Đổi tên thành Java

2010

Oracle mua lại

- ❑ Học Java có thể làm ra những sản phẩm gì?



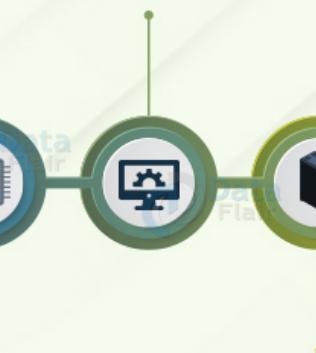
Applications of Java



Applications for
Mobiles



Web
Applications



Web
Servers



Scientific
Applications



Business
Applications



Desktop
GUI

Embedded
Systems

Application
Servers

Application for
Enterprises

Big Data
Technologies

VAI TRÒ CỦA LẬP TRÌNH JAVA 1

Lập trình Java 1 đóng vai trò như là ngôn ngữ lập trình nền tảng cho những môn học nào trong hệ thống nghề nghiệp của SV CNTT

Lập trình Java 1

Lập trình Java 2

Lập trình Java
3,4,5

Lập trình Android

Lập trình Game

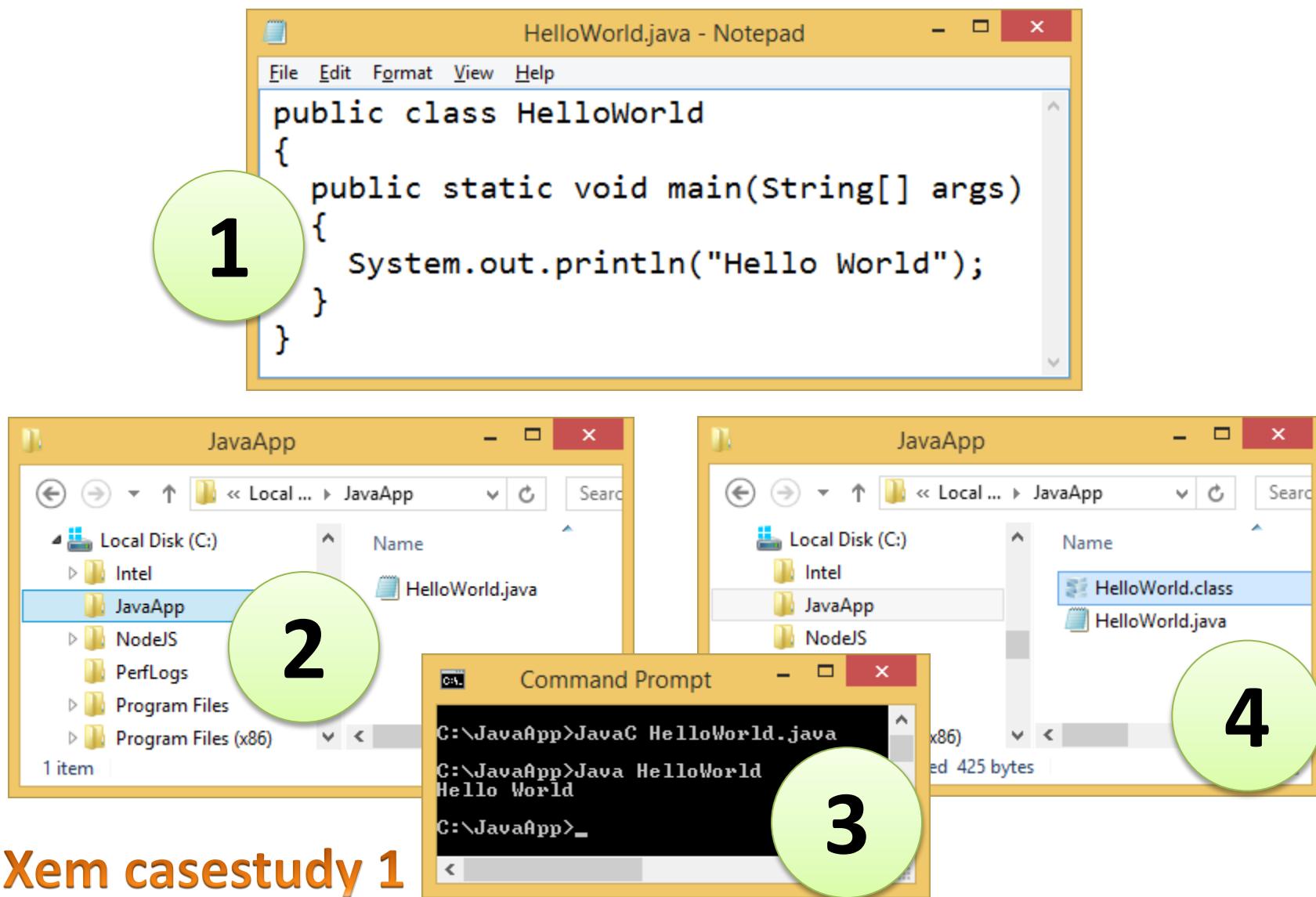
JavaScript

HTML5 & CSS3

Lập trình PHP



CHƯƠNG TRÌNH JAVA



Xem casestudy 1

CẤU TRÚC CHƯƠNG TRÌNH JAVA

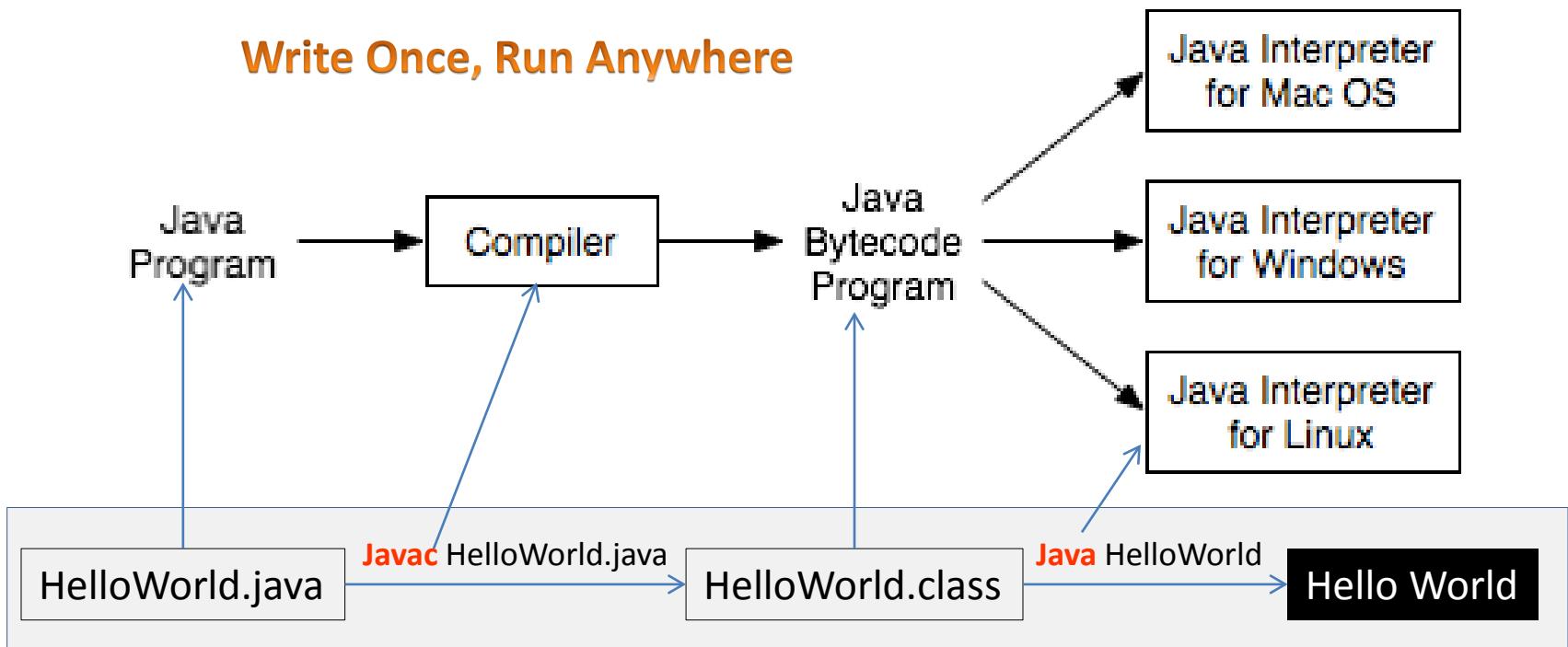
```
package it.com;  
public class Program{  
    public static void main(String[] args){  
        // mã thực thi  
    }  
}
```

Save as

Program.java

- ❑ it.com : tên gói chứa lớp
 - ❖ Sử dụng ký tự thường và dấu chấm. Có thể xem package như folder còn class như file.
- ❑ Program: tên lớp
 - ❖ Phải giống tên file java. Viết hoa ký tự đầu của mỗi từ
- ❑ main(): phương thức bắt đầu chạy
 - ❖ Lớp có thể có nhiều phương thức nhưng main() được gọi tự động khi ứng dụng chạy

- ❑ Khác với ngôn ngữ lập trình khác, thay vì biên dịch mã nguồn thành mã máy, Java được thiết kế biên dịch mã nguồn thành bytecode
- ❑ Bytecode sau đó được môi trường thực thi chạy



JDK – JAVA DEVELOPMENT KIT

- ❑ JDK và các công cụ (javac, java)
- ❑ Cấu hình JDK (path, classpath)

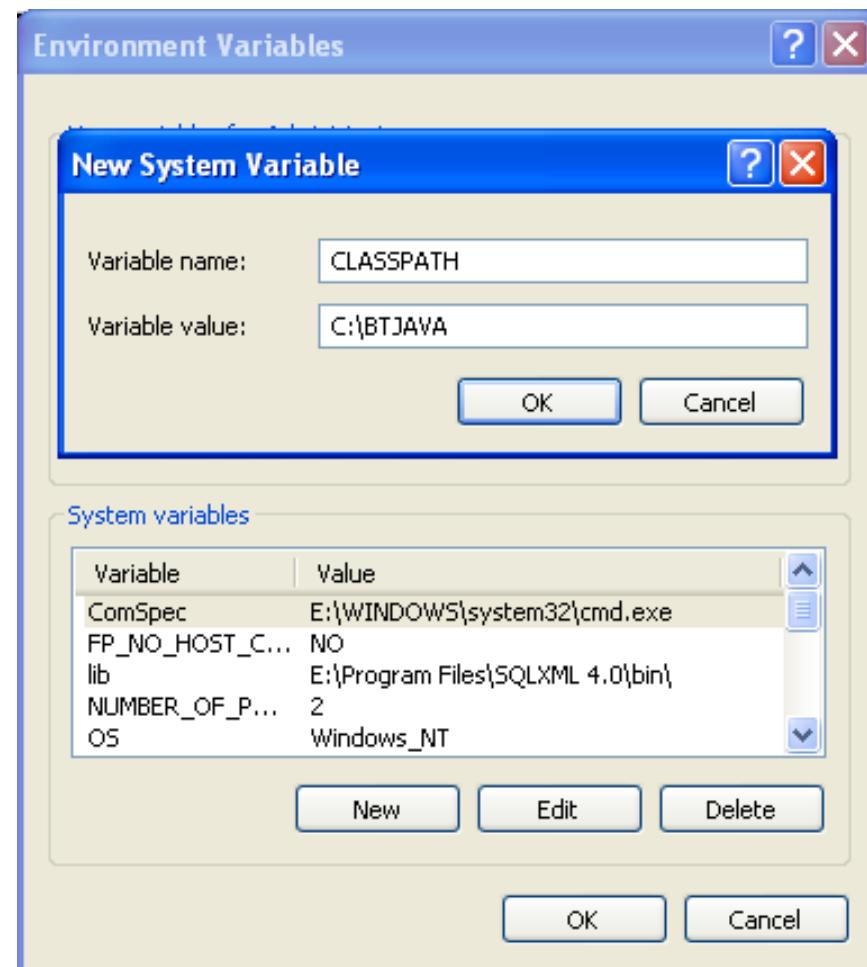
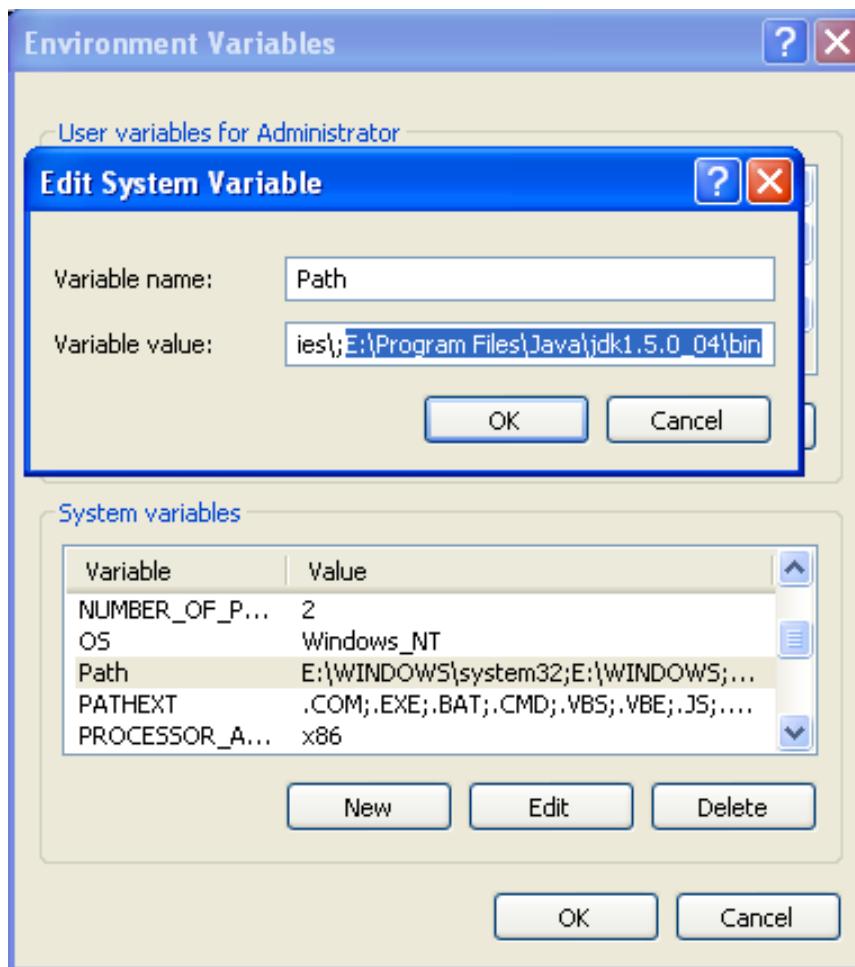


THIẾT LẬP MÔI TRƯỜNG JAVA TRÊN WINDOWS

PATH

Xem casestudy 2

CLASSPATH



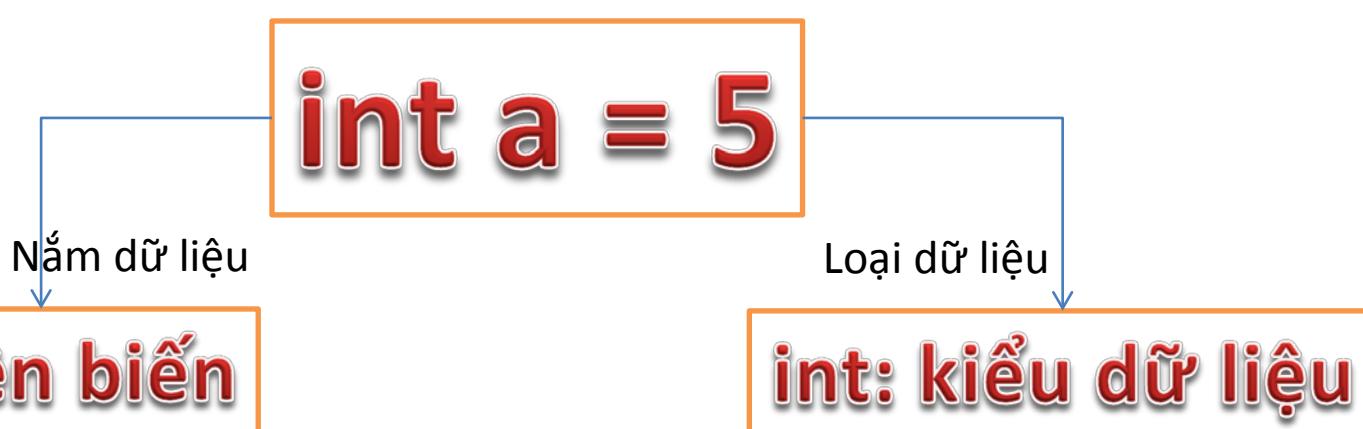
GIỚI THIỆU JAVA IDE

Hỗ trợ việc phát triển và triển khai ứng dụng dễ dàng hơn



```
public class MyClass{  
    public static void main(String[] args){  
        int a = 5;  
        int b = 7;  
        int c = a + b;  
        System.out.println("Tổng: " + c);  
    }  
}
```

- Đoạn mã trên gán các giá trị 5 cho a, 7 cho b và tổng a + b cho c sau đó xuất tổng ra màn hình
- a, b và c gọi là biến số nguyên
- Biến là thành phần nắm giữ dữ liệu được chương trình sử dụng trong các biểu thức tính toán
- Mỗi biến có kiểu dữ liệu riêng



- ☐ Biến là thành phần nắm giữ dữ liệu được chương trình sử dụng trong các biểu thức tính toán
(biến a nắm giữ số 5)

- ☐ int: Số nguyên
- ☐ double : số thực
- ☐ String: Chuỗi
- ☐ ...



❑ Cú pháp

<kiểu dữ liệu> <tên biến> [=giá trị khởi đầu];

❑ Ví dụ:

int a; // khai báo biến không khởi đầu giá trị

double b = 5; // khai báo biến có khởi đầu giá trị

❑ Khai báo nhiều biến cùng kiểu

int a, b=5, c;

❑ Gán giá trị cho biến

c = 9;

a = 15;

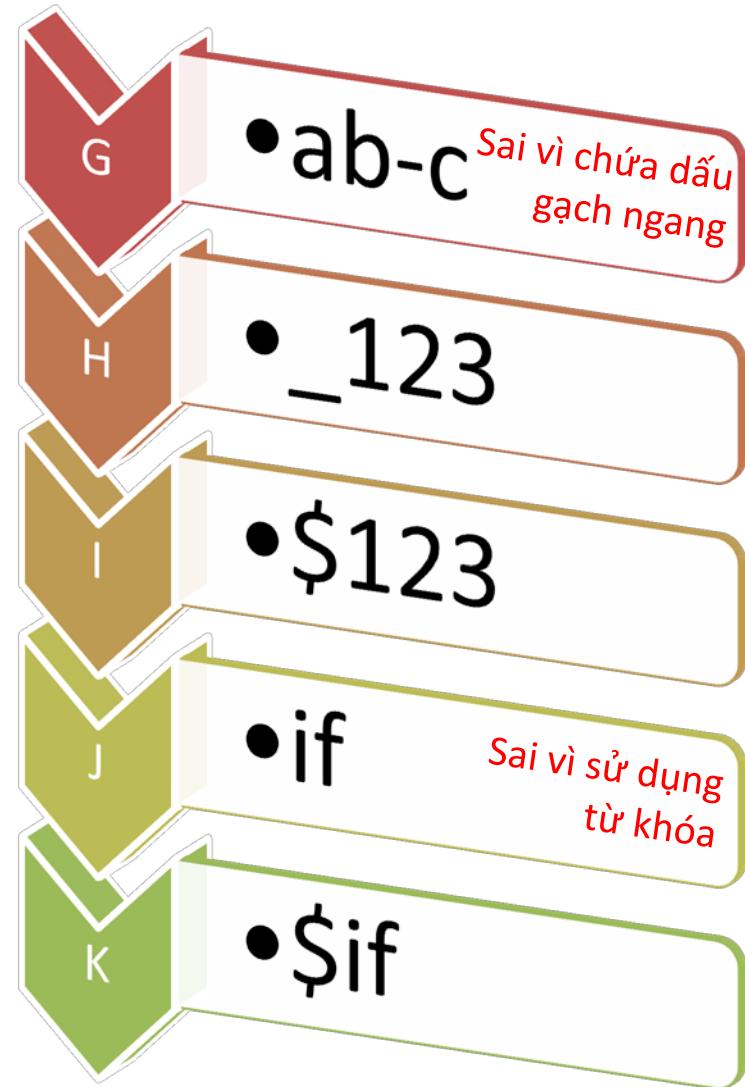
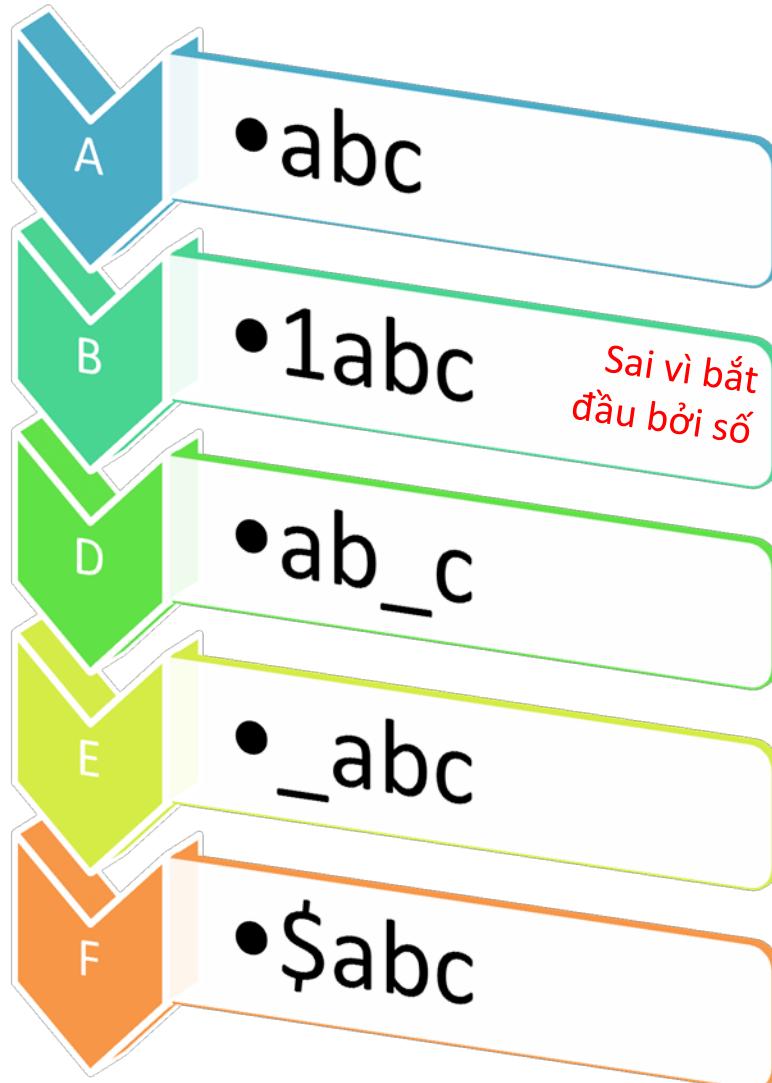


- ☐ Sử dụng ký tự alphabet, số, \$ hoặc gạch dưới (_).
Tên có phân biệt HOA/thường
- ☐ Không ~~bắt đầu bởi số~~, không dùng ~~từ khóa~~

* *Từ khóa là các từ được sử dụng để xây dựng ra ngôn ngữ lập trình java*

abstract	assert	boolean	break	byte	case
catch	char	class	const	continue	default
do	double	else	enum	extends	final
finally	float	for	goto	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				

TÊN BIẾN NÀO SAU ĐÂY KHÔNG HỢP LỆ



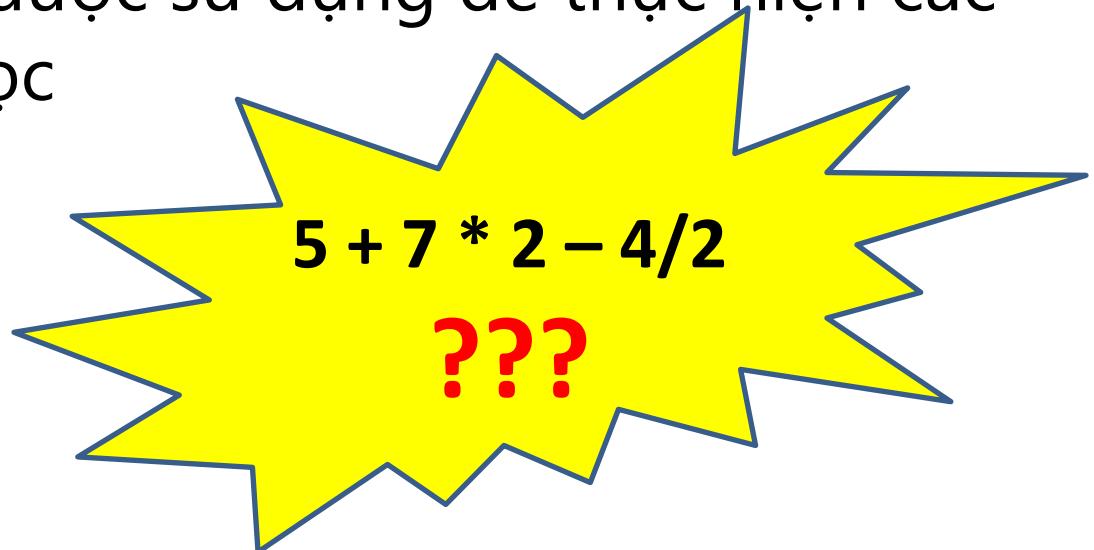


Toán tử	Diễn giải	Ví dụ
+	Phép cộng	int a = 5 + 7
-	Phép trừ	int b = 9 - 6
*	Phép nhân	double c = 9.5 * 2
/	Phép chia	double d = 3.5 / 5

❑ Toán tử số học được sử dụng để thực hiện các phép toán số học

❑ Thứ tự ưu tiên

1. Nhân và chia
2. Cộng và trừ
3. Trái sang phải





CÁC HÀM XUẤT RA MÀN HÌNH

- ❑ System.out.print(): Xuất xong không xuống dòng
- ❑ System.out.println(): Xuất xong có xuống dòng
- ❑ System.out.printf(): Xuất có định dạng, các ký tự
định dạng

- %d: số nguyên
- %f: số thực
 - ✓ Mặc định là 6 số lẻ
 - ✓ %.3f định dạng 3 số lẻ
- %s: chuỗi

- ❑ Ví dụ

```
System.out.print("FPT ");  
System.out.println("Polytechnic");  
System.out.printf("Đào tạo %d nghề", 12);
```

FPT Polytechnic
Đào tạo 12 nghề



Khai báo 2 biến hoten và tuoi
Sử dụng cả 3 hàm trên để xuất dòng sau
<<hoten>> năm nay <<tuoi>> tuoi

- ❑ `java.util.Scanner` cho phép nhận dữ liệu từ bàn phím một cách đơn giản
- ❑ Tạo đối tượng Scanner
 - ❖ `Scanner scanner = new Scanner(System.in)`
- ❑ Các phương thức thường dùng
 - ❖ `scanner.nextLine()`
 - Nhận 1 dòng nhập từ bàn phím
 - ❖ `scanner.nextInt()`
 - Nhận 1 số nguyên nhập từ bàn phím
 - ❖ `scanner.nextDouble()`
 - Nhận 1 số thực nhập từ bàn phím



DEMO

Khai báo 2 biến hoten và tuoi
Nhập họ tên và tuổi từ bàn phím
Xuất ra dòng theo định dạng sau
<<hoten>> năm nay <<tuoi>> tuoi

- ❑ Java cung cấp các hàm tiện ích giúp chúng ta thực hiện các phép tính khó một cách dễ dàng như:
 - ❖ Làm tròn số
 - ❖ Tính căn bậc 2
 - ❖ Tính lũy thừa
 - ❖ ...
- ❑ Ví dụ sau đây tính căn bậc 2 của 7
 - ❖ double a = **Math.sqrt(7)**
- ❑ Ngoài Math.sqrt() còn rất nhiều hàm khác được trình bày ở slide sau.

CÁC HÀM TOÁN HỌC

Hàm	Diễn giải	Ví dụ
Math.min(a, b)	Lấy số nhỏ nhất của 2 số a và b	x=Math.min(5, 3.5) => x=3.5
Math.max(a, b)	Lấy số lớn nhất của 2 số a và b	x=Math.max(5, 3.5) => x=5
Math.pow(a, n)	Tính a^n (a lũy thừa n)	x=Math.pow(5, 3) => x=75
Math.sqrt(a)	Tính \sqrt{a} (căn bậc 2 của a)	x=Math.sqrt(16) => x=4
Math.abs(a)	Lấy giá trị tuyệt đối của a	x=Math.abs(-5) => x=5
Math.ceil(a)	Lấy số nguyên trên của a	x=Math.ceil(3.5) => x=4
Math.floor(a)	Lấy số nguyên dưới của a	x=Math.floor(3.5) => x=3
Math.round(a)	Làm tròn số của a	x=Math.round(3.5) => x=4
Math.random()	Sinh số ngẫu nhiên từ 0 đến 1	x=Math.random() => x=0..1

Thảo luận: Làm tròn một số thực với 2 số lẻ



TỔNG KẾT NỘI DUNG BÀI HỌC

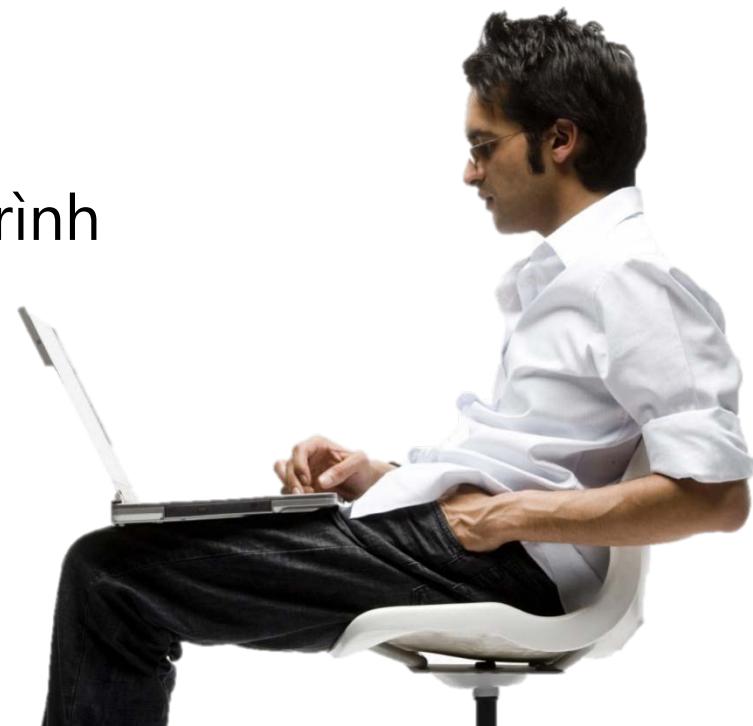
- ❑ Giới thiệu Java
- ❑ Thiết lập môi trường làm việc (JDK) và IDE
- ❑ Biến và quy tắc đặt tên biến
- ❑ Toán tử số học
- ❑ Xuất ra màn hình
- ❑ Nhập từ bàn phím
- ❑ Sử dụng các hàm toán học



BÀI 2: KIỂU, TOÁN TỬ, LỆNH IF, SWITCH

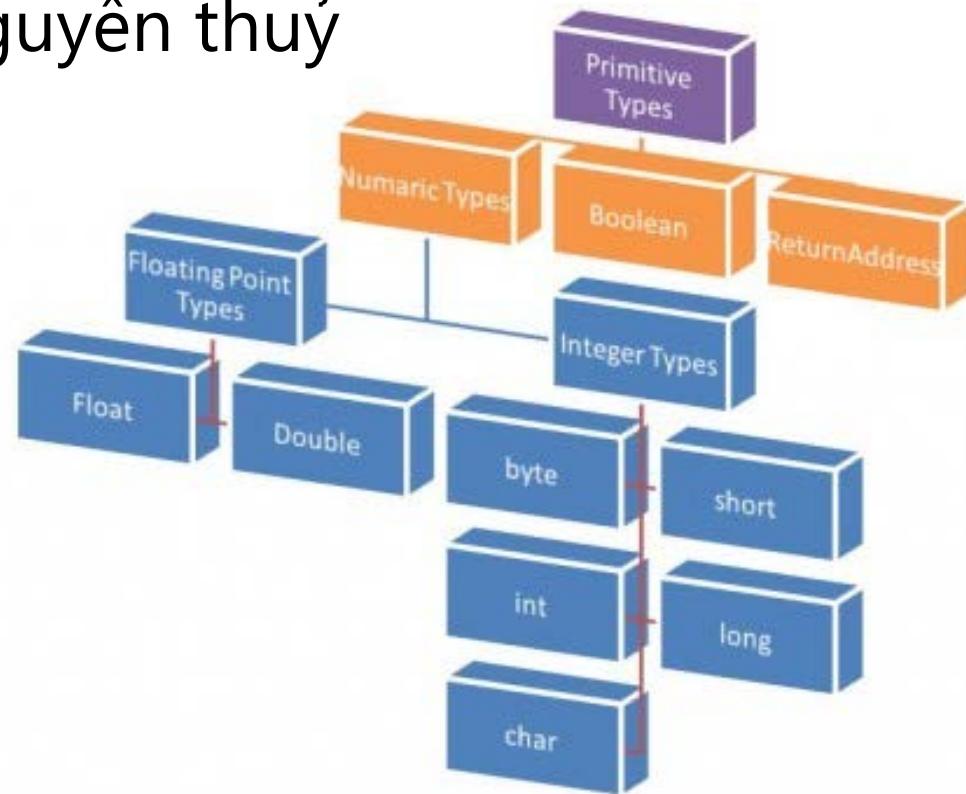
❑ Kết thúc bài học này bạn có khả năng

- ❖ Hiểu rõ và sử dụng kiểu nguyên thủy, lớp bao
- ❖ Chuyển đổi chuỗi sang kiểu nguyên thủy
- ❖ Sử dụng lệnh try...catch để bắt lỗi chuyển kiểu
- ❖ Hiểu và sử dụng toán tử, xây dựng biểu thức
- ❖ Sử dụng lệnh if
- ❖ Sử dụng lệnh switch case
- ❖ Biết cách tổ chức một chương trình



KIỂU DỮ LIỆU NGUYÊN THỦY

- ❑ Kiểu dữ liệu nguyên thủy là kiểu được giữ lại từ ngôn ngữ C (ngôn ngữ gốc của Java)
- ❑ Có 8 kiểu dữ liệu nguyên thuỷ
- ❑ Ví dụ
 - ❖ `int a = 8;`
 - ❖ `double b;`



KIỂU NGUYÊN THỦY

Kiểu	Mặc định	Bit	Khả năng lưu trữ	
			Giá trị nhỏ nhất	Giá trị lớn nhất
byte	0	8	-2^7	$+2^7-1$
short	0	16	-2^{15}	$+2^{15}-1$
int	0	32	-2^{31}	$+2^{31}-1$
long	0L	64	-2^{63}	$+2^{63}-1$
float	0.0F	32	$-3.40292347 \times 10^{38}$	$+3.40292347 \times 10^{38}$
double	0.0	64	$-1.79769313486231570 \times 10^{308}$	$+1.79769313486231570 \times 10^{308}$
char	'\u0000'	16	'\u0000'	'\uFFFF'
boolean	false	1	false	true

Giá trị mặc định là giá trị sẽ được gán cho biến khi khai báo không khởi đầu giá trị cho biến



GIÁ TRỊ HẰNG (LITERAL)

- Giá trị hằng là dữ liệu có kiểu là một trong các kiểu nguyên thuỷ

Kiểu int

```
int i = 3;
```

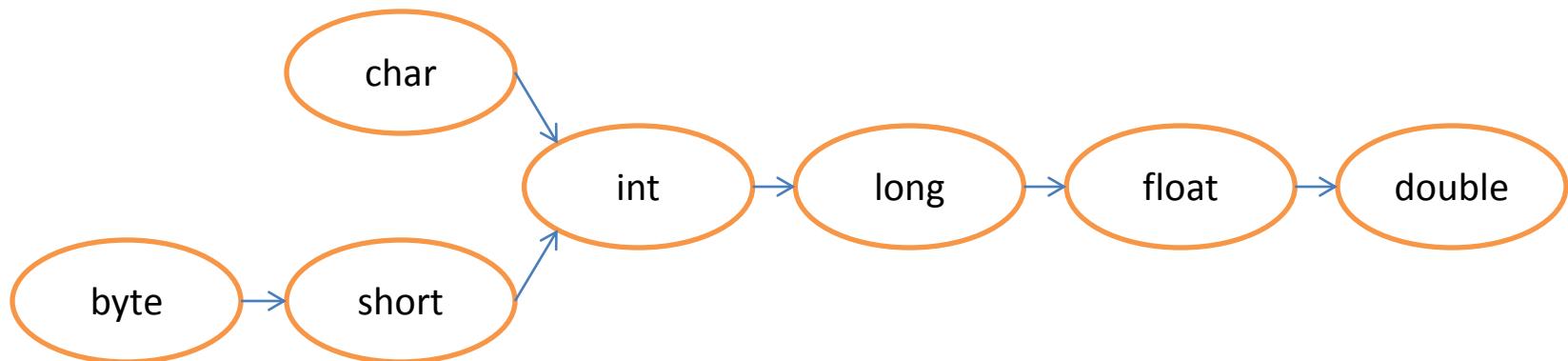
Kiểu long

```
long l = 12L;
```

Kiểu float

```
float = 10.19F;
```

- Đối với kiểu nguyên thủy, ép kiểu tự động xảy ra theo chiều mũi tên



- Ví dụ

```
int a = 5;
```

```
double b = 9.4;
```

```
b = a; //ép kiểu tự động
```

```
a = (int)b; //ép kiểu tương minh phần thập phân sẽ bị bỏ
```

CHUYỂN CHUỖI SANG KIỂU NGUYÊN THỦY

☐ Xét biểu thức 1

```
String a = "3";
```

```
String b = "4";
```

```
String c = a + b;
```

=> c là **"34"**

☐ Xét biểu thức 2

```
int a = Integer.parseInt("3");
```

```
int b = Integer.parseInt("4");
```

```
int c = a + b;
```

=> c là **7**

Chuỗi => Nguyên thủy

byte Byte.parseByte(String)

short Short.parseShort(String)

int Integer.parseInt(String)

long Long.parseLong(String)

float Float.parseFloat(String)

double Double.parseDouble(String)

boolean Boolean.parseBoolean(String)

SỬ DỤNG TRY...CATCH ĐỂ KIỂM LỖI

❑ Xét trường hợp

int a = **scanner.nextInt();**

hoặc

int a = **Integer.parseInt(s);**

❑ Điều gì sẽ xảy ra khi người dùng **nhập không phải số** hoặc chuỗi **s không phải là chuỗi chứa số**

❑ Hãy sử dụng lệnh **try...catch** để kiểm soát các lỗi trên

```
try{  
    int a = scanner.nextInt();  
    System.out.println("Bạn đã nhập đúng");  
}  
catch (Exception ex){  
    System.out.println("Vui lòng nhập số!");  
}
```



LỚP BAO KIỂU NGUYÊN THỦY (WRAPPER)

- ❑ Tương ứng với mỗi kiểu nguyên thủy Java định nghĩa một lớp bao để bao giá trị của kiểu nguyên thủy tương ứng gọi là lớp bao kiểu nguyên thủy
- ❑ Rất nhiều hàm trong Java chỉ làm việc với đối tượng mà không làm việc với kiểu nguyên thủy

Nguyên Thủy Lớp bao

byte	↔	Byte
short	↔	Short
int	↔	Integer
long	↔	Long
float	↔	Float
double	↔	Double
char	↔	Character
boolean	↔	Boolean

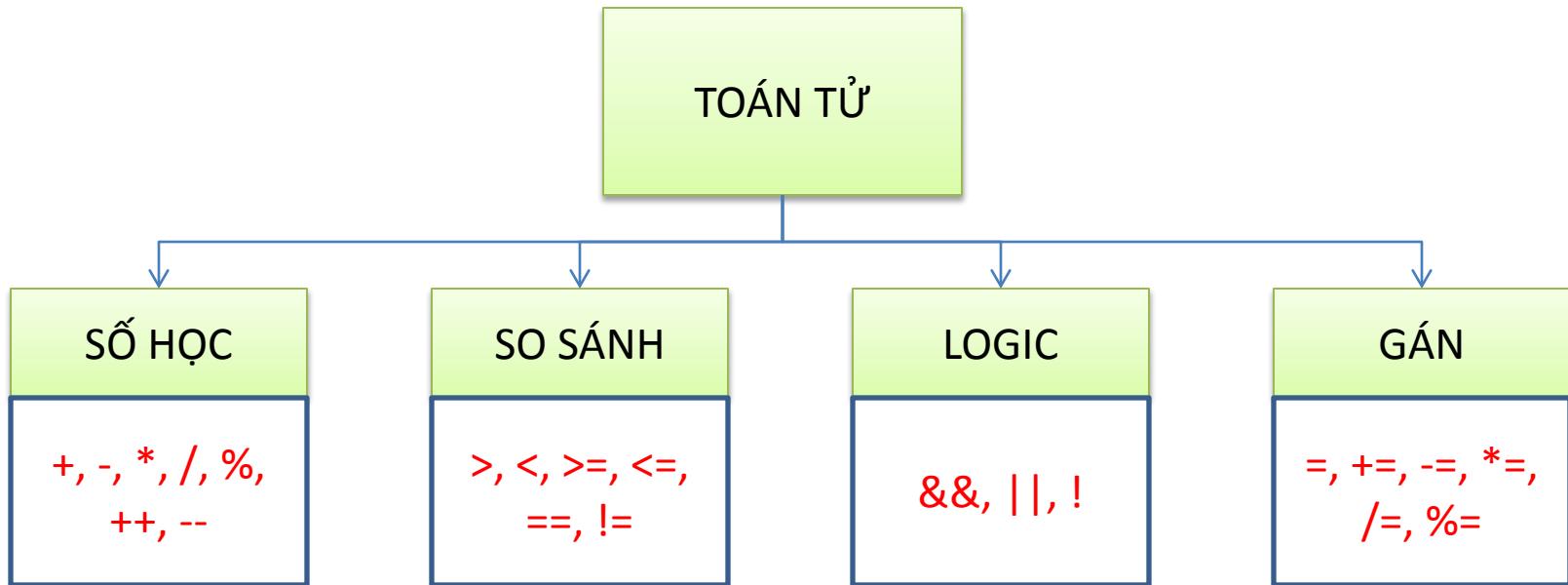


BAO (BOXING)/MỞ BAO(UNBOXING)

- ❑ Boxing là việc tạo đối tượng từ lớp bao để bọc giá trị nguyên thủy.
 - ❖ Có 3 cách để bao giá trị nguyên thủy sau
 - Integer a = Integer.valueOf(5) // bao tường minh
 - Integer a = new Integer(5) // bao tường minh
 - Integer a = 5 // bao ngầm định
- ❑ Unboxing là việc mở lấy giá trị nguyên thủy từ đối tượng của lớp bao
 - ❖ Có 2 cách mở bao để lấy giá trị nguyên thủy sau
 - int b = a.intValue() // mở bao tường minh
 - int b = a; // mở bao ngầm định

BOXING/UNBOXING

Boxing	Unboxing	Ví dụ
Byte.valueOf(byte)	<<Byte>>.byteValue()	long a = 5L;
Short.valueOf(short)	<<Short>>.shortValue()	Long b = Long.valueOf(a); long c = b.longValue();
Integer.valueOf(int)	<<Integer>>.intValue()	
Long.valueOf(long)	<<Long>>.longValue()	
Float.valueOf(float)	<<Float>>.floatValue()	
Double.valueOf(double)	<<Double>>.doubleValue()	
Boolean.valueOf(boolean)	<<Boolean>>.booleanValue()	



Biểu thức là sự kết hợp giữa toán tử và toán hạng. Kết quả của biểu thức là một giá trị.

Giá trị của các biểu thức sau?

int x = 11 % 4;

boolean a = 9 < 2 && true || 4 > 3;

☐ Toán tử số học là các phép toán thao tác trên các số nguyên và số thực

+	Tính tổng của 2 số
-	Tính hiệu của 2 số
*	Tính tích của 2 số
/	Tích thương của 2 số
%	Thực hiện chia có dư của 2 số
++	Tăng giá trị của biến lên 1 đơn vị
--	Giảm giá trị của biến xuống 1 đơn vị

- ❑ Toán tử so sánh là các phép toán so sánh hai toán hạng

==	So sánh bằng
>	So sánh lớn hơn
>=	So sánh lớn hơn hoặc bằng
<	So sánh nhỏ hơn
<=	So sánh nhỏ hơn hoặc bằng
!=	So sánh khác

- ❑ Toán tử logic là các phép toán thao tác trên các toán hạng logic

&&	Trả về giá trị true khi tất cả biểu thức tham gia biểu thức có giá trị true
	Trả về giá trị true khi có 1 biểu thức tham gia biểu thức có giá trị là true
!	Lấy giá trị phủ định của biểu thức



- ❑ Toán tử điều kiện là toán tử 3 ngôi duy nhất trong ngôn ngữ Java
- ❑ Cú pháp:
<điều kiện> ? <giá trị đúng> : <giá trị sai>
- ❑ Diễn giải:
 - ❖ Nếu biểu thức **<điều kiện>** có giá trị là true thì kết quả của biểu thức là **<giá trị đúng>**, ngược lại là **<giá trị sai>**
- ❑ Ví dụ: tìm số lớn nhất của 2 số a và b
int a = 1, b = 9;
int max = a > b ? a : b;

Tìm số lớn nhất trong 3 số a, b và c?

❑ Cú pháp

if(<<điều kiện>>)

{

 << Công việc >>

}

❑ Diễn giải:

- ❖ Nếu **điều kiện** có giá trị true thì **công việc** được thực hiện

❑ Ví dụ:

```
double diem = 4;  
if (diem >= 5) {  
    System.out.println("Đậu");  
}
```

❑ Diễn giải:

- ❖ Đoạn mã trên không xuất gì ra màn hình cả vì biểu thức điều kiện **diem >= 5** có giá trị false



Nhập số từ bàn phím.

Nếu số dương thì tính và xuất căn bậc 2
của số đó ra màn hình



❑ Cú pháp

```
if (<<điều kiện>>)
```

```
{
```

```
    << công việc 1 >>
```

```
}
```

```
else
```

```
{
```

```
    << công việc 2 >>
```

```
}
```

❑ Diễn giải

- ❖ Nếu **điều kiện** có giá trị true thì **công việc 1** được thực hiện, ngược lại **công việc 2** được thực hiện



❑ Ví dụ

```
double diem = 4;  
if (diem < 5) {  
    System.out.println("Rớt");  
}  
else {  
    System.out.println("Đậu");  
}
```

❑ Diễn giải:

- ❖ Đoạn mã trên xuất chữ “Rớt” ra màn hình vì điều kiện **diem < 5** có giá trị là **true**.



DEMΩ

Nhập số từ bàn phím.

Nếu số dương thì tính và xuất căn bậc 2
của số đó ra màn hình, ngược lại thì
thông báo lỗi



❑ Cú pháp

```
if (<<điều kiện 1>>){  
    << công việc 1 >>  
}  
  
else if (<<điều kiện 2>>){  
    << công việc 2 >>  
}  
  
...  
  
else {  
    << công việc N+1 >>  
}
```

❑ Diễn giải

- ❖ Chương trình sẽ kiểm tra từ **điều kiện 1 đến N** nếu gặp **điều kiện i** đầu tiên có giá trị true thì sẽ thực hiện **công việc i**, ngược lại sẽ thực hiện **công việc N+1**



❑ Ví dụ

```
double delta = Math.pow(b, 2) - 4 * a * c;  
if(delta < 0) {  
    System.out.println("Vô nghiệm");  
}  
else if(delta == 0) {  
    System.out.println("Nghiệm kép");  
}  
else {  
    System.out.println("2 nghiệm");  
}
```

❑ Diễn giải

- ❖ Đoạn mã trên biện luận và giải phương trình bậc 2



Tính thuế thu nhập mô tả slide sau

- ❑ Viết chương trình tính thuế thu nhập. Giả sử thu nhập gồm lương và thưởng
- ❑ Thuế thu nhập được tính như sau
 - ❖ Dưới 9 triệu: không đóng thuế
 - ❖ Từ 9 đến 15 triệu: thuế 10%
 - ❖ Từ 15 đến 30 triệu: 15%
 - ❖ Trên 30 triệu: 20%

❑ Cú pháp

```
switch (<<biểu thức>>)
{
```

```
    case <<giá trị 1>>:
```

```
        // Công việc 1
```

```
        break;
```

```
    case <<giá trị 2>>:
```

```
        // Công việc 2
```

```
        break;
```

```
    ...
```

```
    default:
```

```
        // Công việc N+1
```

```
        break;
```

```
}
```

❑ Diễn giải

- ❖ So sánh giá trị của biểu thức switch với giá trị của các case. Nếu bằng với giá trị của case nào thì sẽ thực hiện công việc của case đó, ngược lại sẽ thực hiện công việc của default.
- ❖ Nếu công việc của case không chứa lệnh break thì case tiếp sau sẽ được thực hiện
- ❖ default là tùy chọn

VÍ DỤ LỆNH SWITCH

```
double a = 5, b = 7, c = -1;  
char op = '+';  
switch(op){  
    case '+':  
        c = a + b;  
        break;  
    case '-':  
        c = a - b;  
        break;  
    case 'x':  
    case ':':  
        System.out.println("Đang xây dựng");  
        break;  
    default:  
        System.out.println("Vui lòng chọn +, -, x và :");  
        break;  
}
```

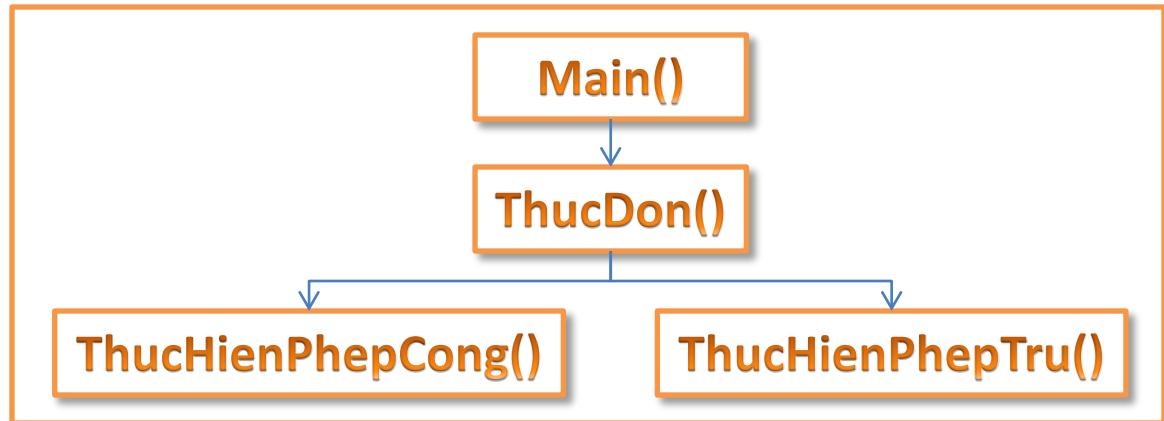
Không có **break**



Nhập tháng và năm từ bàn phím.
Xuất số ngày của tháng đã nhập.

TỔ CHỨC CHƯƠNG TRÌNH

```
package com.fpoly;  
  
import java.util.Scanner;  
  
public class ChuongTrinh {  
    public static void main(String[] args) {  
        thucDon();  
    }  
  
    public static void thucDon() {  
        public static void thucHienPhepCong() {}  
  
        public static void thucHienPhepTru() {}  
    }  
}
```



Hiển thị thực đơn chính
của chương trình

THIẾT KẾ THỰC ĐƠN

```
System.out.println(">> MÁY TÍNH CÁ NHÂN <<");  
System.out.println("-----+");  
System.out.println("| 1. Cộng |");  
System.out.println("| 2. Trừ |");  
System.out.println("| 3. Kết thúc |");  
System.out.println("-----+");  
System.out.println(" >> Chọn chức năng? ");
```

```
Scanner scanner = new Scanner(System.in);  
int answer = scanner.nextInt();  
if(answer == 1){  
    thucHienPhepCong();  
}  
else if(answer == 2){  
    thucHienPhepTru();  
}  
else if(answer == 3){  
    System.exit(0);  
}
```

Gọi phương thức thực hiện phép cộng

Gọi phương thức thực hiện phép trừ

Thoát ứng dụng



Tổ chức chương trình trên
bằng cách đổi if...else sang switch...case

TỔNG KẾT NỘI DUNG BÀI HỌC

- Kiểu nguyên thủy
- Qui luật ép kiểu nguyên thủy
- Lớp bao giá trị kiểu nguyên thủy
- Boxing/Unboxing
- Chuyển đổi kiểu dữ liệu
- Toán tử và biểu thức
- Lệnh if
- Lệnh switch case
- Tổ chức chương trình



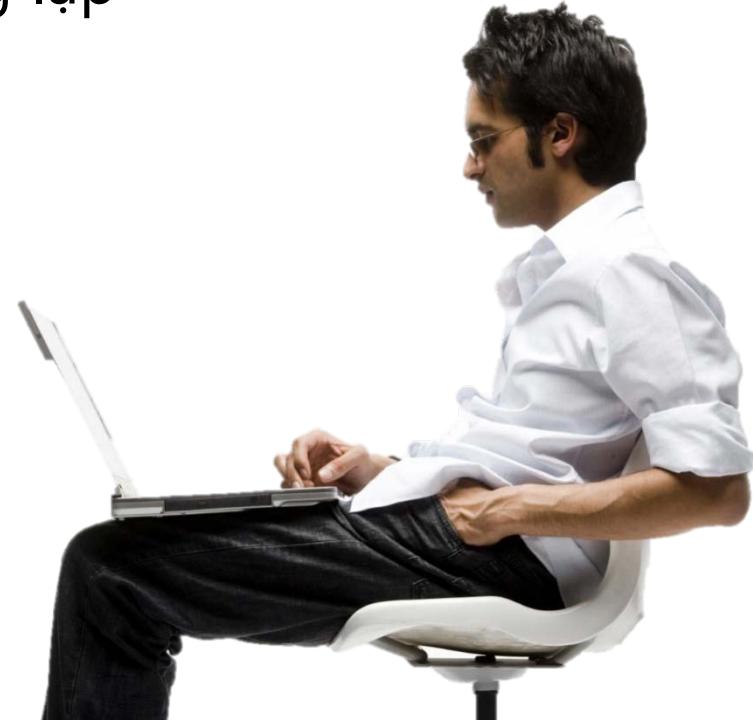
LẬP TRÌNH Java™

BÀI 3: MẢNG VÀ LỆNH LẤP

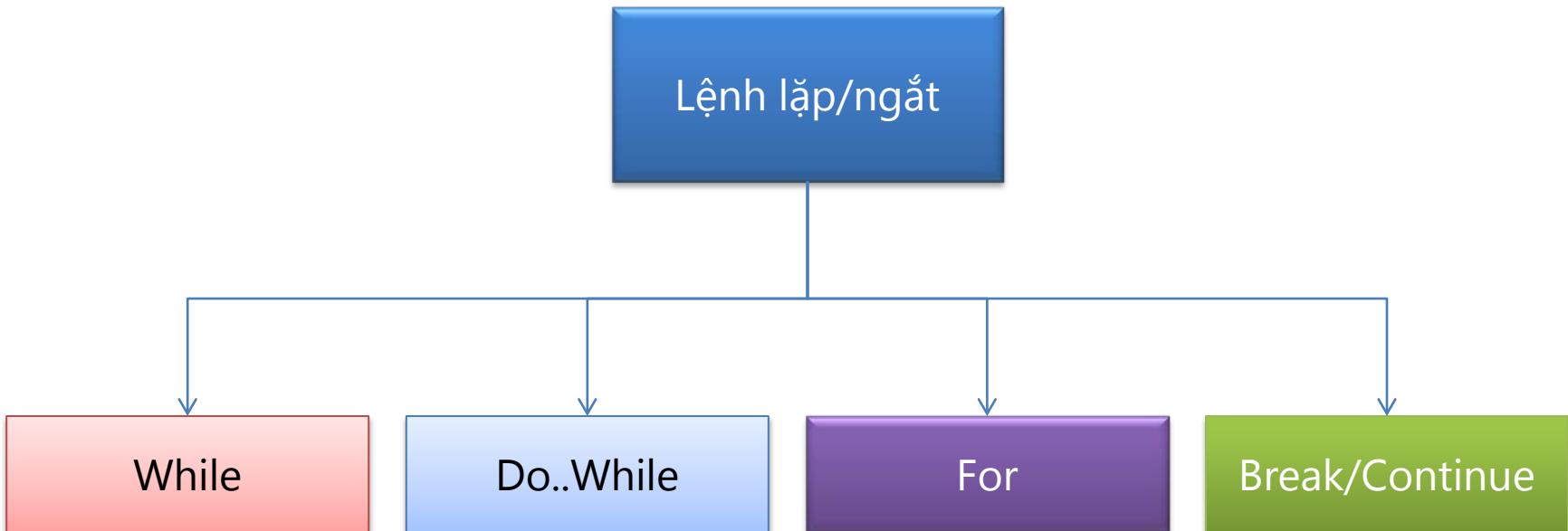


❑ Kết thúc bài học này bạn có khả năng

- ❖ Hiểu cấu trúc lệnh lặp và sử dụng các lệnh lặp
 - While
 - Do...while
 - For
- ❖ Hiểu và áp dụng lệnh ngắt vòng lặp
 - Break
 - Continue
- ❖ Hiểu và sử dụng mảng



LỆNH LẶP & NGẮT





❑ Cú pháp

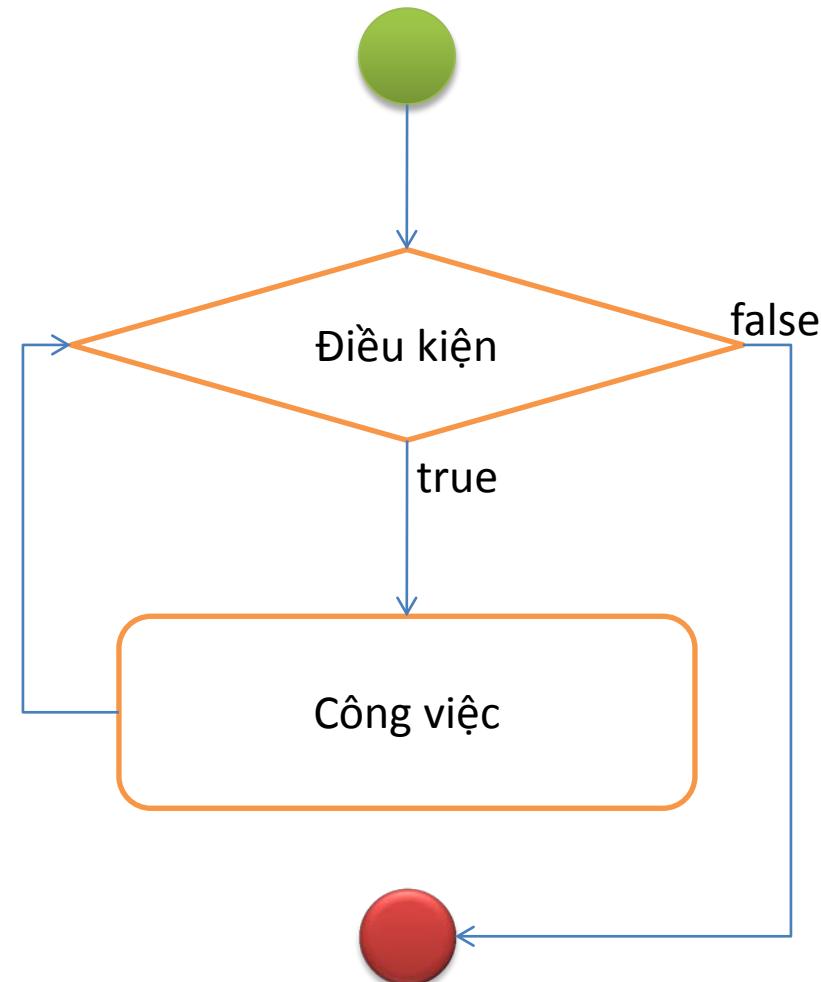
```
while (<<điều kiện>>) {
```

```
    // công việc
```

```
}
```

❑ Diễn giải:

- ❖ Thực hiện công việc trong khi biểu thức điều kiện có giá trị là true.

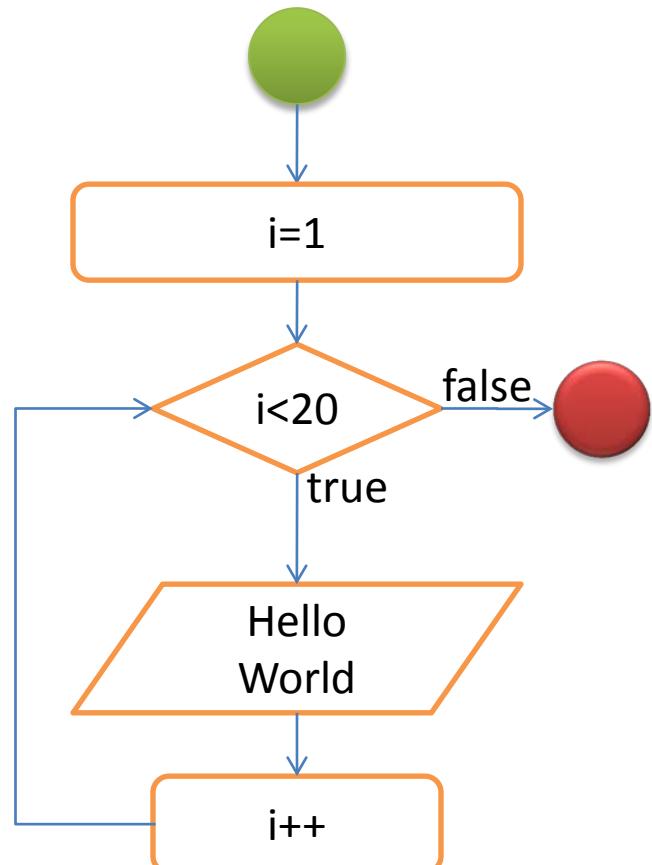


❑ Ví dụ

```
int i = 1;  
while (i < 20) {  
    System.out.println("Hello World !");  
    i++;  
}
```

❑ Diễn giải:

- ❖ Đoạn mã trên xuất 19 dòng Hello World ra màn hình





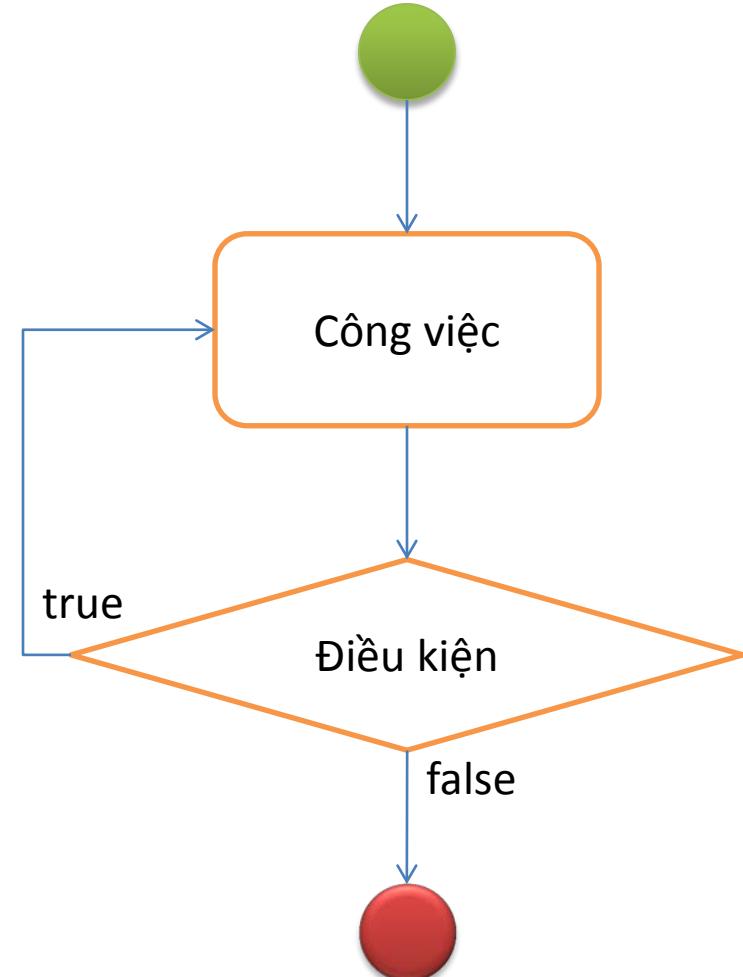
LỆNH LẶP DO...WHILE

❑ Cú pháp:

```
do {  
    // công việc  
}  
  
while (<<điều kiện>>);
```

❑ Diễn giải:

- ❖ Tương tự lệnh lặp while chỉ khác ở chỗ điều kiện được kiểm tra sau, nghĩa là công việc được thực hiện ít nhất 1 lần.



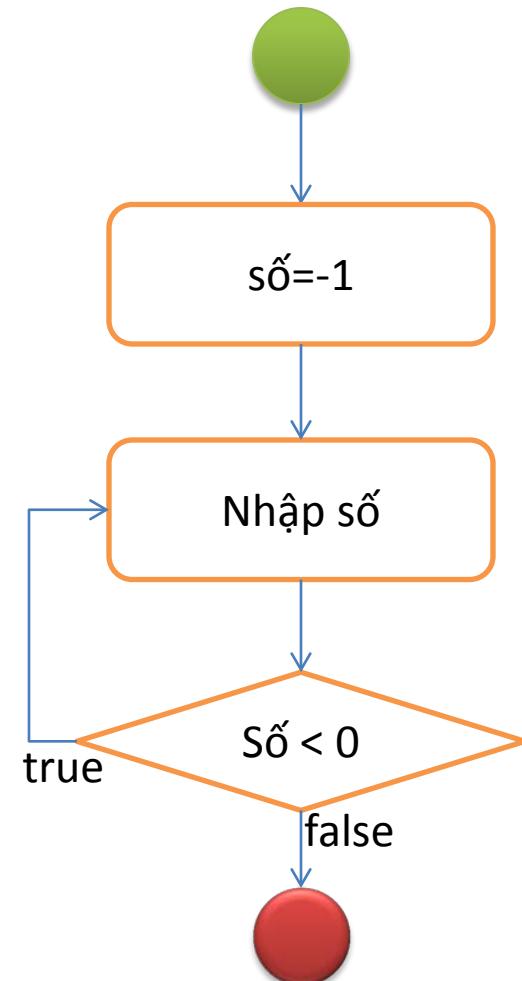
LỆNH LẶP DO...WHILE

❑ Ví dụ

```
int so = -1;  
do {  
    so = scanner.nextDouble();  
}  
while (so < 0);
```

❑ Diễn giải:

- ❖ Đoạn mã trên chỉ cho phép nhập số nguyên dương từ bàn phím.





❑ Cú pháp

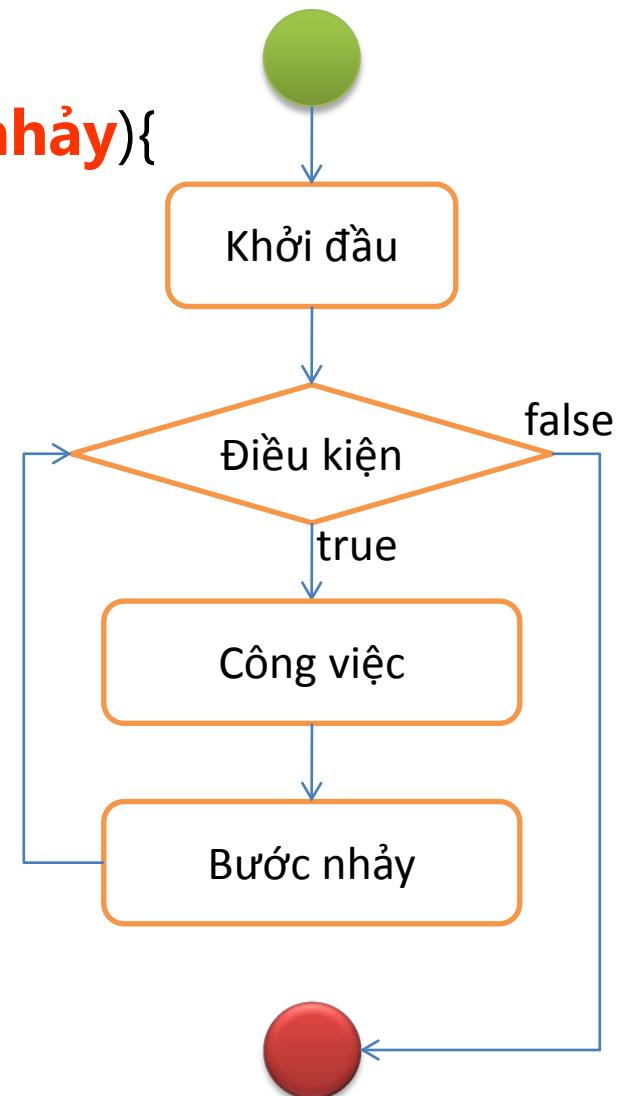
```
for (khởi đầu ; điều kiện; bước nhảy){
```

```
    // công việc
```

```
}
```

❑ Diễn giải

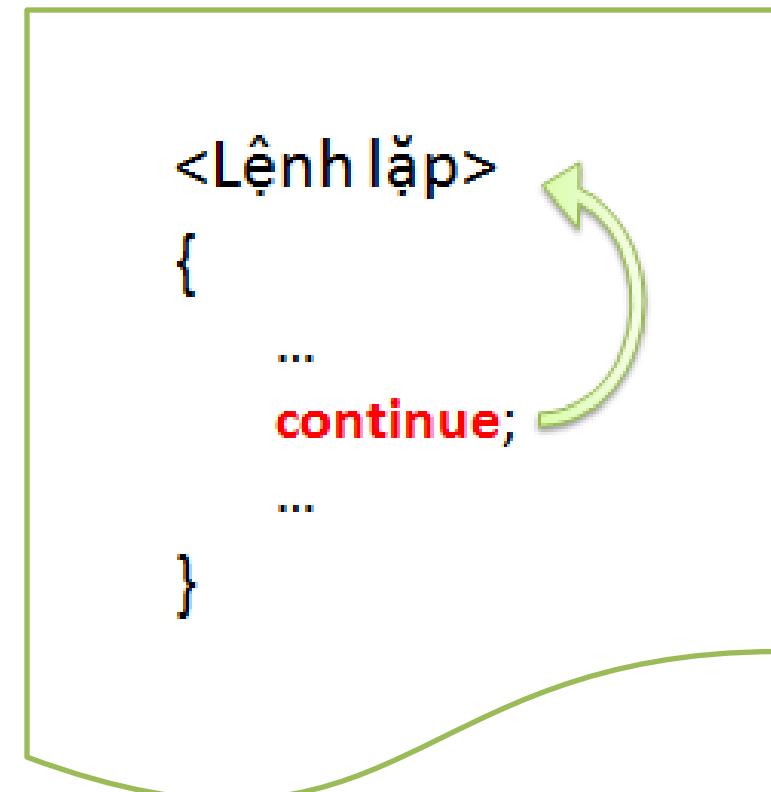
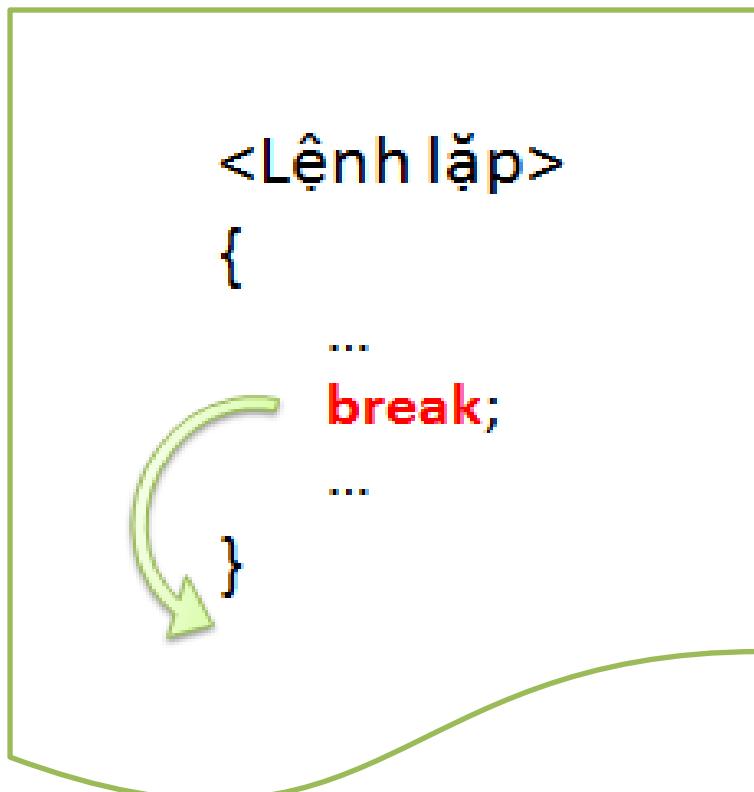
- ❖ B1: Thực hiện <<khởi đầu>>
- ❖ B2: Kiểm tra <<điều kiện>>
 - True: B3
 - False: kết thúc
- ❖ B3: Thực hiện <<công việc>>
- ❖ B4: Thực hiện <<bước nhảy>>
- ❖ B5: Trở lại B2





LỆNH BREAK & CONTINUE

- ❑ **break** dùng để ngắt lệnh lặp
- ❑ **continue** dùng để thực hiện lần lặp tiếp theo ngay lập tức



❑ Ví dụ:

```
int diem = 0;  
while(true){  
    diem = scanner.nextInt();  
    if(diem >= 0 && diem <=10){  
        break;  
    }  
    System.out.println("Điểm phải từ 0 đến 10");  
}
```

❑ Diễn giải:

- ❖ Nhập điểm hợp lệ (từ 0 đến 10)



MÀNG LÀ GÌ

- ☐ Mảng là cấu trúc lưu trữ nhiều phần tử có cùng kiểu dữ liệu



- Để truy xuất các phần tử cần biết chỉ số (index).
Chỉ số được đánh từ 0.

- # Các thao tác mảng

- ❖ Khai báo
 - ❖ Truy xuất (đọc/ghi) phần tử
 - ❖ Lấy số phần tử
 - ❖ Duyệt mảng
 - ❖ Sắp xếp các phần tử mảng



❑ Khai báo không khởi tạo

- ❖ `int[] a; // mảng số nguyên chưa biết số phần tử`
- ❖ `int b[]; // mảng số nguyên chưa biết số phần tử`
- ❖ `String[] c = new String[5]; // mảng chứa 5 chuỗi`

❑ Khai báo có khởi tạo

- ❖ `double[] d1 = new double[]{2, 3, 4, 5, 6}; // mảng số thực, 5 phần tử, đã được khởi tạo`
- ❖ `double[] d2 = {2, 3, 4, 5, 6}; // mảng số thực, 5 phần tử, đã được khởi tạo`



TRUY XUẤT CÁC PHẦN TỬ

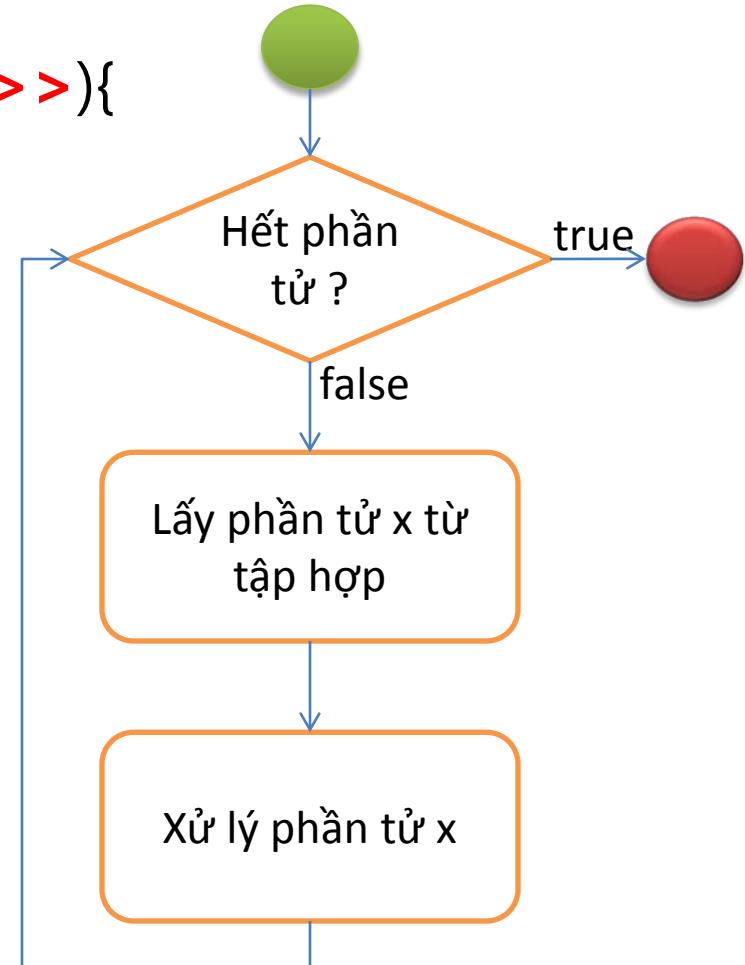
- ❑ Sử dụng chỉ số (**index**) để phân biệt các phần tử.
Chỉ số mảng tính từ 0.
 - ❖ int a[] = {4, 3, 5, 7};
 - ❖ a[**2**] = a[**1**] * 4; // $45 * 4 = 180$
 - ❖ Sau phép gán này mảng là {4, 3, **12**, 7};
- ❑ Sử dụng thuộc tính **length** để lấy số phần tử của mảng
 - ❖ a.length có giá trị là 9

❑ Cú pháp

```
for (<<kiểu>> x : <<tập hợp>>){  
    // Xử lý phần tử x  
}
```

❑ Diễn giải:

- ❖ For each được sử dụng để duyệt tập hợp. Mỗi lần lấy 1 phần tử từ tập hợp và xử lý phần tử đó.



- 2 vòng lặp thường được sử dụng để duyệt mảng là for và for-each.

```
int[] a = {4, 3, 5, 9};  
for(int i=0; i<a.length; i++){  
    System.out.println(a[i]);  
}
```

for(;;)

for-each

```
int[] a = {4, 3, 5, 9};  
for (int x : a){  
    System.out.println(x);  
}
```

❑ Ví dụ sau tính tổng các số chẵn của mảng.

- ❖ Lấy từng phần tử từ mảng với for-each
- ❖ Nếu là số chẵn thì cộng vào tổng

```
int[] a = {9, 3, 8, 7, 3, 9, 4, 2};

double tong = 0;
for(int x : a){
    if(x % 2 == 0){
        tong += x;
    }
}

System.out.print("Tổng: " + tong);
```



DEM

Nhập mảng số nguyên

+ Tính và xuất trung bình cộng

+ Xuất lập phương các phần tử

THAO TÁC MẢNG NÂNG CAO

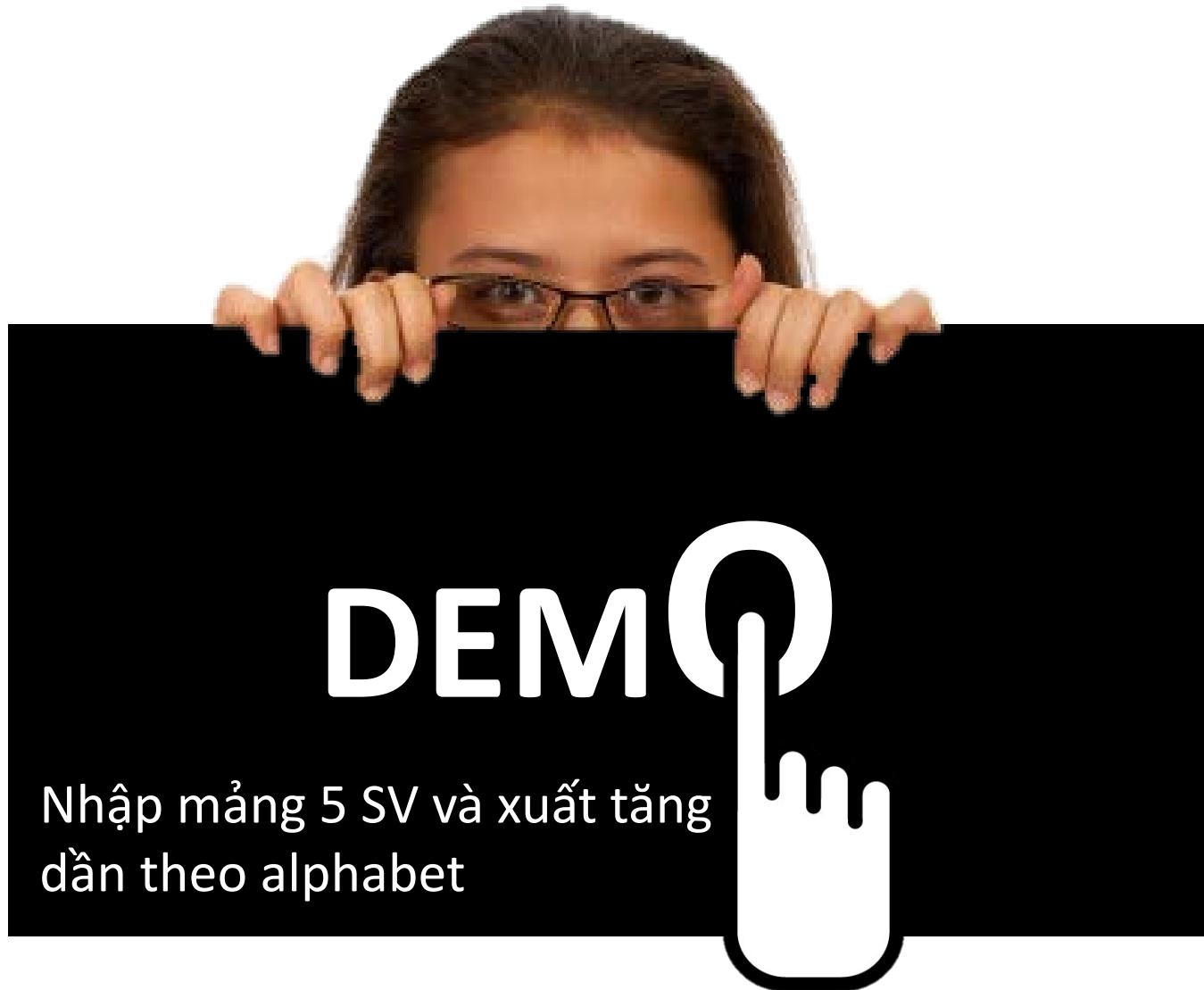
```
int[] a = {9, 3, 8, 7, 3, 9, 4, 2};  
  
System.out.println("Mảng gốc: " + Arrays.toString(a));  
[9, 3, 8, 7, 3, 9, 4, 2]  
Arrays.sort(a);  
System.out.println("Sau sort: " + Arrays.toString(a));  
[2, 3, 3, 4, 7, 8, 9, 9]  
int i = Arrays.binarySearch(a, 8);  
System.out.println("Vị trí của 8 là " + i);  
Vị trí của 8 là 5  
Arrays.fill(a, 0);  
System.out.println("Sau fill: " + Arrays.toString(a));  
[0, 0, 0, 0, 0, 0, 0, 0]
```

Mảng gốc: [9, 3, 8, 7, 3, 9, 4, 2]
Sau sort: [2, 3, 3, 4, 7, 8, 9, 9]
Vị trí của 8 là 5
Sau fill: [0, 0, 0, 0, 0, 0, 0, 0]

THAO TÁC MẢNG

```
Int[] a = {1, 9, 2, 8, 3, 7, 4, 6, 5};
```

Phương thức	Mô tả/ví dụ
<T> List<T> asList (T... a)	Chuyển một mảng sang List với kiểu tương ứng. Ví dụ: List<Integer> b = Arrays.asList(a);
int binarySearch (Object[] a, Object key)	Tìm vị trí xuất hiện đầu tiên của một phần tử trong mảng. Ví dụ: int i = Arrays.binarySearch(a, 8);
void sort (Object[] a)	Sắp xếp các phần tử theo thứ tự tăng dần. Ví dụ: Arrays.sort(a);
String toString (Object[] a)	Chuyển mảng thành chuỗi được bọc giữ cặp dấu [] và các phần tử mảng cách nhau dấu phẩy. Ví dụ: String s = Arrays.toString(a);
void fill (Object[] a, Object val)	Gán 1 giá trị cho tất cả các phần tử mảng. Ví dụ: Arrays.fill(a, 9);



Nhập mảng 5 SV và xuất tăng
dần theo alphabet

- ❑ Arrays.sort(mảng) không thể thực hiện
 - ❖ Sắp xếp giảm
 - ❖ Các kiểu không so sánh được
- ❑ Giải pháp: tự xây dựng thuật toán sắp xếp

```
int a[] = {8,2,6,2,9,1,5};  
for(int i=0; i<a.length-1; i++){  
    for(int j=i+1; j<a.length; j++){  
        if(a[i] > a[j]){  
            int temp = a[i],  
                a[i] = a[j],  
                a[j] = temp;  
        }  
    }  
}
```

Nếu thay đổi toán tử so sánh thành $<$ thì thuật toán trở thành sắp xếp tăng dần.



Nhập 2 mảng họ tên và điểm.
Xuất 2 mảng giảm theo điểm



TỔNG KẾT NỘI DUNG BÀI HỌC

❑ Loop

- ❖ While
- ❖ Do...while
- ❖ For(;điều kiện;)
- ❖ For(phần tử: tập hợp)

❑ Ngắt

- ❖ Break
- ❖ Continue

❑ Mảng



LẬP TRÌNH Java™

BÀI 4: LỚP VÀ ĐỐI TƯỢNG

❑ Kết thúc bài học này bạn có khả năng

- ❖ Hiểu rõ khái niệm đối tượng và lớp
- ❖ Mô hình hóa lớp và đối tượng
- ❖ Định nghĩa được lớp và tạo đối tượng
- ❖ Định nghĩa các trường, phương thức
- ❖ Định nghĩa và sử dụng hàm tạo
- ❖ Hiểu và sử package
- ❖ Sử dụng thành thạo các đặc tả truy xuất
- ❖ Hiểu được tính che dấu (encapsulation)

KHÁI NIỆM VỀ ĐỐI TƯỢNG

- ❑ Biểu diễn đối tượng trong thế giới thực
- ❑ Mỗi đối tượng được đặc trưng bởi các thuộc tính và các hành vi riêng của nó





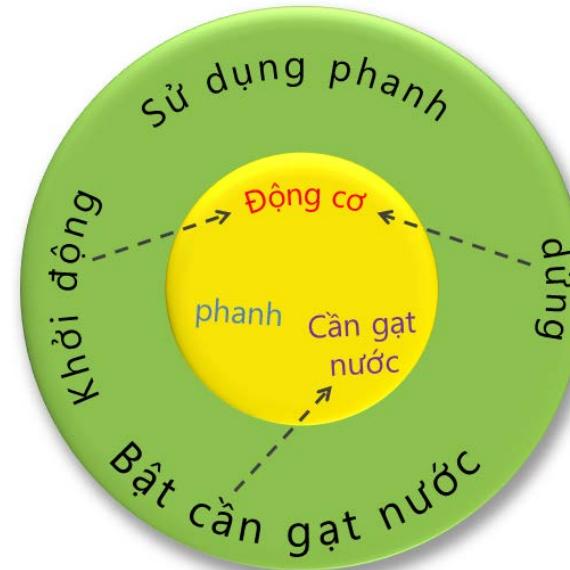
❑ Đặc điểm

- Hãng sản xuất
- Model
- Năm
- Màu



❑ Hành vi (Ô tô có thể làm gì?)

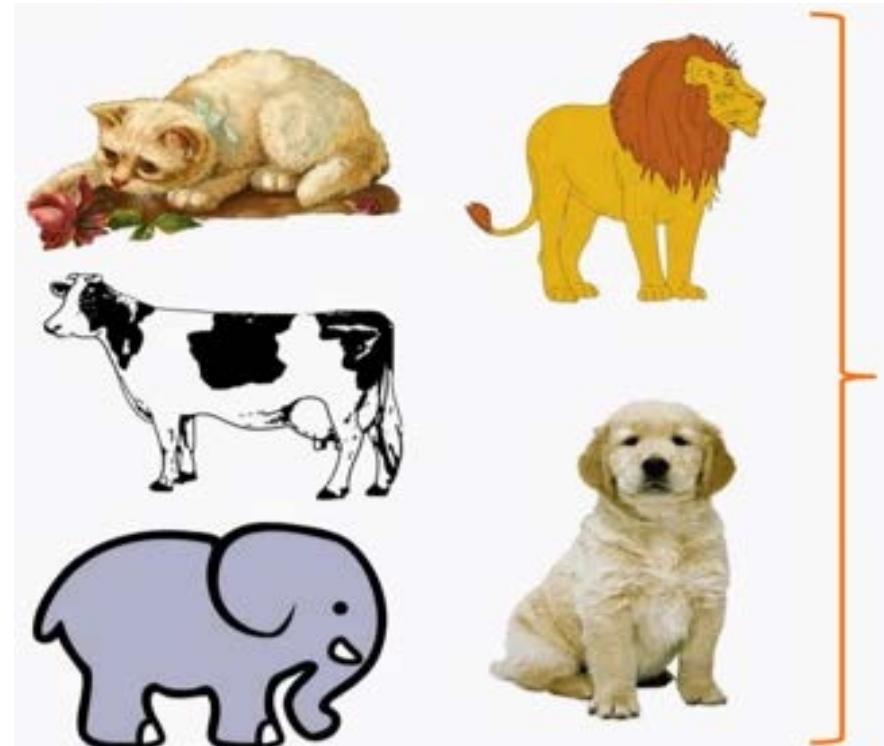
- Khởi động
- Dừng
- Phanh
- Bật cần gạt nước



CLASS LÀ GÌ?

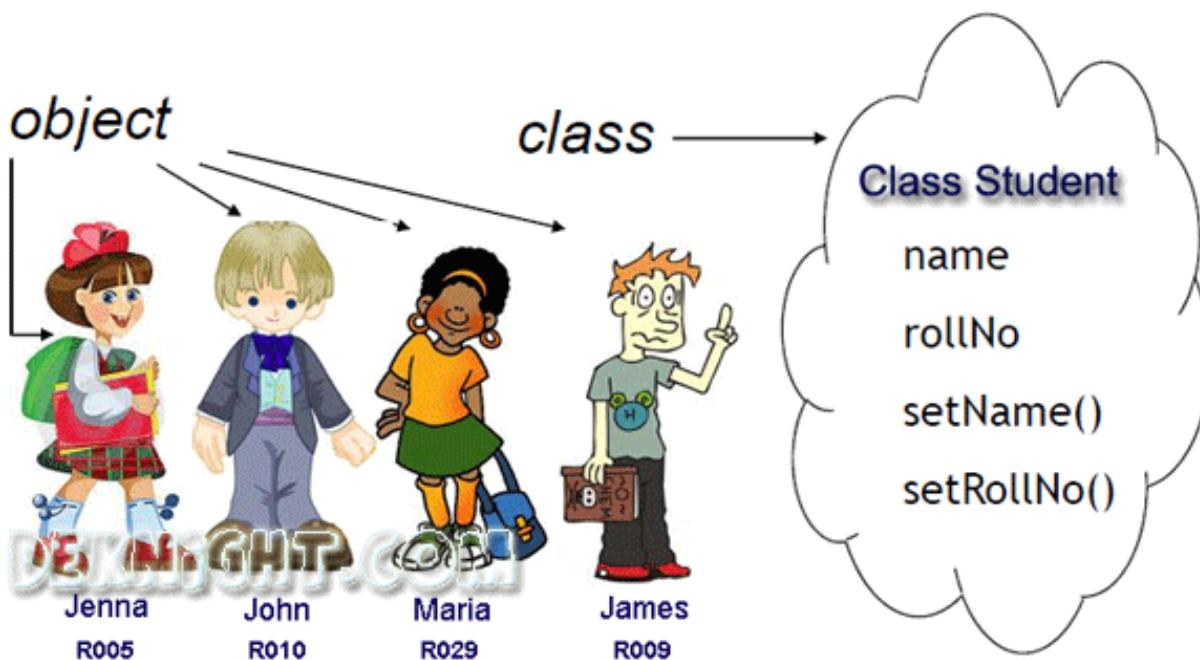


Nhóm các **Xe ô-tô**



Nhóm các **Động vật**

- ❑ Lớp là một khuôn mẫu được sử dụng để mô tả các đối tượng cùng loại.
- ❑ Lớp bao gồm các thuộc tính (trường dữ liệu) và các phương thức (hàm thành viên)





☐ Thuộc tính (field)

- ❖ Hãng sản xuất
- ❖ Model
- ❖ Năm
- ❖ Màu

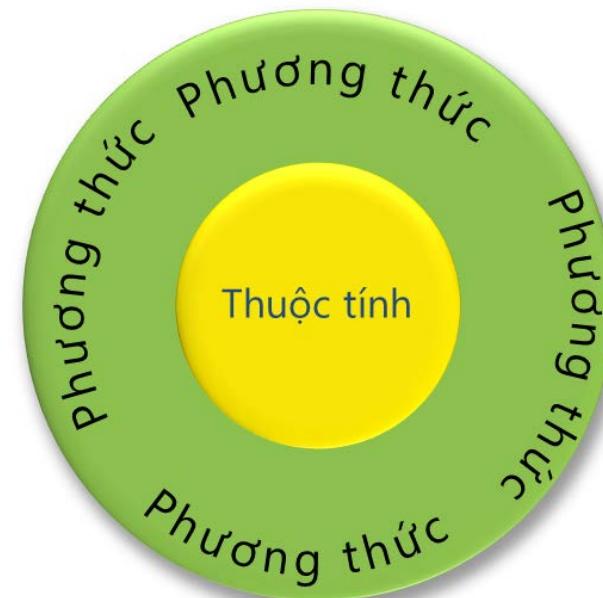
Danh từ



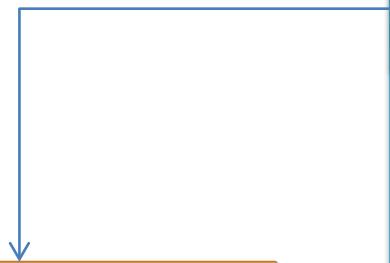
☐ Phương thức (method)

- ❖ Khởi động()
- ❖ Dừng()
- ❖ Phanh()
- ❖ Bật cần gạt nước()

Động từ



MÔ HÌNH LỚP VÀ ĐỐI TƯỢNG



Ô tô của Dũng

Thuộc tính

- Năm = 2010
- Nhà SX=Honda
- Model = Accord
- Màu = Xanh

Phương thức

- Khởi động
- Dừng
- Phanh

Ô tô

Thuộc tính

- Năm
- Nhà sản xuất
- Model
- Màu

Phương thức

- Khởi động
- Dừng
- Phanh



Ô tô của Mai

Thuộc tính

- Năm = 2012
- Nhà SX=BMW
- Model = CS30
- Màu = Bạc

Phương thức

- Khởi động
- Dừng
- Phanh



TÍNH TRỪU TƯỢNG (ABSTRACTION)

- ❑ Abstraction là công việc lựa chọn các thuộc tính và hành vi của thực thể vừa đủ để mô tả thực thể đó trong một bối cảnh cụ thể mà không phải liệt kê tất cả các thuộc tính, hành vi của thực thể có.
- ❑ Ví dụ: Mô tả một sinh viên ngành CNTT có rất nhiều thuộc tính và hành vi. Ở đây chúng ta chỉ sử dụng mã, họ và tên, điểm, ngành mà thôi, không cần thiết phải mô tả ~~cao, nặng, hát, cười, nhảy cò cò...~~

```
class <<ClassName>>
```

```
{
```

```
    <<type>> <<field1>>;
```

```
...
```

```
    <<type>> <<fieldN>>;
```

Khai báo các trường

```
<<type>> <<method1>>([parameters]) {
```

```
    // body of method
```

```
}
```

```
...
```

```
    <<type>> <<methodN>>([parameters]) {
```

```
    // body of method
```

```
}
```

```
}
```

VÍ DỤ ĐỊNH NGHĨA LỚP

```
public class Employee{  
    public String fullname;  
    public double salary;  
  
    public void input(){  
        Scanner scanner = new Scanner(System.in);  
        System.out.print(" > Full Name: ");  
        this.fullname = scanner.nextLine();  
  
        System.out.print(" > Salary: ");  
        this.salary = scanner.nextDouble();  
    }  
  
    public void output(){  
        System.out.println(this.fullname);  
        System.out.println(this.salary);  
    }  
}
```

Trường



Phương thức

Lớp Employee có 2 thuộc tính là fullname và salary và 2 phương thức là input() và output()



- ❑ Đoạn mã sau sử dụng lớp Employee để tạo một nhân viên sau đó gọi các phương thức của lớp.

```
public static void main(String[] args) {  
  
    Employee emp = new Employee();  
    emp.input();  
    emp.output();  
}
```



- ❑ Chú ý:
 - ❖ Toán tử **new** được sử dụng để tạo đối tượng
 - ❖ Biến emp chứa tham chiếu tới đối tượng
 - ❖ Sử dụng dấu chấm (.) để truy xuất các thành viên của lớp (trường và phương thức).



Tạo lớp mô tả sinh viên bao gồm họ
tên, điểm và các phương thức nhập,
xuất và xếp loại học lực

- ❑ Phương thức là một mô-đun mã thực hiện một công việc cụ thể nào đó
 - ❖ Trong lớp Employee có 2 phương thức là input() và output()
- ❑ Phương thức có thể có một hoặc nhiều tham số
- ❑ Phương thức có thể có kiểu trả về hoặc void (không trả về gì cả)
- ❑ Cú pháp

```
<<kiểu trả về>> <<tên phương thức>> ([danh sách tham số])  
{  
    // thân phương thức  
}
```

VÍ DỤ PHƯƠNG THỨC

```
public class Employee{  
    public String fullname;  
    public double salary;
```

```
    public void input(){...}  
    public void output(){...}
```

Kiểu trả về là **void** nên thân phương thức
không chứa lệnh return giá trị

```
    public void setInfo(String fullname, double salary) {  
        this.fullname = fullname;  
        this.salary = salary;  
    }
```

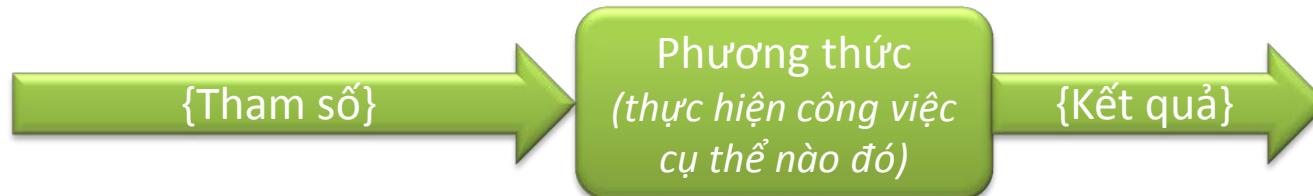
Kiểu trả về là **double** nên thân phương
thức phải chứa lệnh **return số thực**

```
    public double incomeTax(){  
        if(this.salary < 5000000){  
            return 0;  
        }  
        double tax = (this.salary - 5000000) * 10/100;  
        return tax;  
    }
```

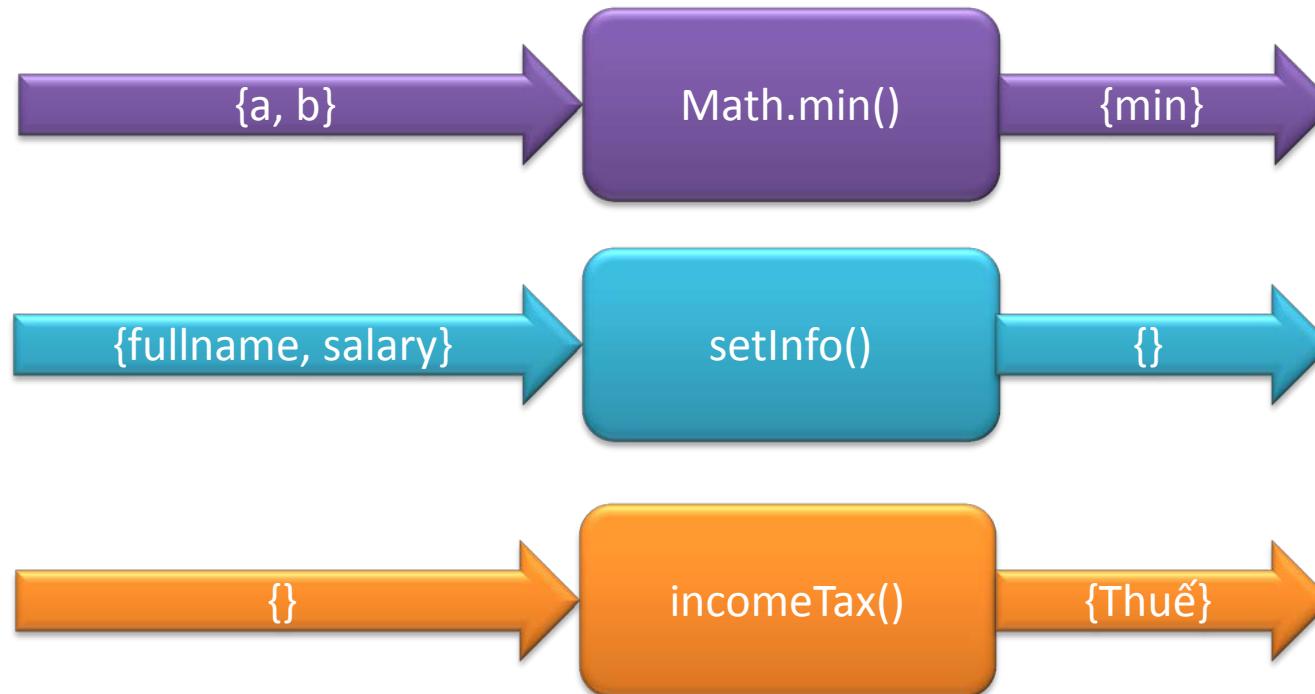
MÔ HÌNH PHƯƠNG THỨC



❑ Mô hình



❑ Ví dụ





NẠP CHỒNG PHƯƠNG THỨC (OVERLOADING)

- ❑ Trong một lớp có thể có nhiều phương thức cùng tên nhưng khác nhau về tham số (kiểu, số lượng và thứ tự)

```
public class MyClass{  
    void method(){...}  
    void method(int x){...}  
    void method(float x){...}  
    void method(int x, double y){...}  
}
```

- ❑ Trong lớp MyClass có 4 phương thức cùng tên là method nhưng khác nhau về tham số



VÍ DỤ NẠP CHỒNG PHƯƠNG THỨC

- ❑ Xét trường hợp overload sau

```
class MayTinh{  
    int tong(int a, int b){return a + b;}  
    int tong(int a, int b, int c){return a + b + c;}  
}
```

- ❑ Với lớp trên, bạn có thể sử dụng để tính tổng 2 hoặc 3 số nguyên.

```
MayTinh mt = new MayTinh();  
int t1 = mt.tong(5, 7);  
int t2 = mt.tong(5, 7, 9);
```

HÀM TẠO (CONSTRUCTOR)

- ❑ Hàm tạo là một phương thức đặc biệt được sử dụng để tạo đối tượng.
- ❑ Đặc điểm của hàm tạo
 - ❖ Tên trùng với tên lớp
 - ❖ Không trả lại giá trị

❑ Ví dụ

```
public class ChuNhat{  
    double dai, rong;  
    public ChuNhat(double dai, double rong){  
        this.dai = dai;  
        this.rong = rong;  
    }  
}
```

Lớp

```
ChuNhat cn1 = new ChuNhat(20, 15);  
ChuNhat cn2 = new ChuNhat(50, 25);
```

Đối tượng

HÀM TẠO (CONSTRUCTOR)

- ❑ Trong một lớp có thể định nghĩa nhiều hàm tạo khác tham số, mỗi hàm tạo cung cấp 1 cách tạo đối tượng.
- ❑ Nếu không khai báo hàm tạo thì Java tự động cung cấp hàm tạo mặc định (không tham số)

```
public class ChuNhat{  
    double dai, rong;  
    public ChuNhat(double dai, double rong){  
        this.dai = dai;  
        this.rong = rong;  
    }  
    public ChuNhat(double canh){  
        this.dai = canh;  
        this.rong = canh;  
    }  
}
```

```
ChuNhat cn = new ChuNhat(20, 15);  
ChuNhat vu= new ChuNhat(30);
```

- ❑ **this** được sử dụng để đại diện cho đối tượng hiện tại.
- ❑ **this** được sử dụng trong lớp để tham chiếu tới các thành viên của lớp (field và method)
- ❑ Sử dụng **this.field** để phân biệt field với các biến cục bộ hoặc tham số của phương thức

```
public class MyClass{  
    int field;  
    void method(int field){  
        this.field = field;  
    }  
}
```

Trường

Tham số

SinhVien

+ hoTen: String
+ diemTB: double

+ xepLoai(): String
+ xuat(): void
+ nhap(): void

+ SinhVien()
+ SinhVien(hoTen, diemTB)



Xây dựng lớp mô tả sinh viên như mô hình trên.
Trong đó nhap() cho phép nhập họ tên và điểm
từ bàn phím; xuat() cho phép xuất họ tên, điểm
và học lực ra màn hình; xepLoai() dựa vào điểm
để xếp loại học lực

Sử dụng 2 hàm tạo để tạo 2 đối tượng sinh viên



- ❑ Package được sử dụng để chia các class và interface thành từng gói khác nhau.
 - ❖ Việc làm này tương tự quản lý file trên ổ đĩa trong đó class (file) và package (folder)
- ❑ Ví dụ sau tạo lớp MyClass thuộc gói com.poly

```
package com.poly;  
public class MyClass{...}
```

- ❑ Trong Java có rất nhiều gói được phân theo chức năng
 - ❖ java.util: chứa các lớp tiện ích
 - ❖ java.io: chứa các lớp vào/ra dữ liệu
 - ❖ java.lang: chứa các lớp thường dùng...

- ❑ Lệnh import được sử dụng để chỉ ra lớp đã được định nghĩa trong một package
- ❑ Các lớp trong gói java.lang và các lớp cùng định nghĩa trong cùng một gói với lớp sử dụng sẽ được import ngầm định

```
package com.polyhcm;
import com.poly.MyClass;
import java.util.Scanner;
public class HelloWorld{
    public static void main(String[] args){
        MyClass obj = new MyClass();
        Scanner scanner = new Scanner(System.in);
    }
}
```

❑ Đặc tả truy xuất được sử dụng để định nghĩa khả năng cho phép truy xuất đến các thành viên của lớp. Trong java có 4 đặc tả khác nhau:

❖ **private**: chỉ được phép sử dụng nội bộ trong class

❖ **public**: công khai hoàn toàn

❖ **{default}**:

➢ Là public đối với các lớp truy xuất cùng gói

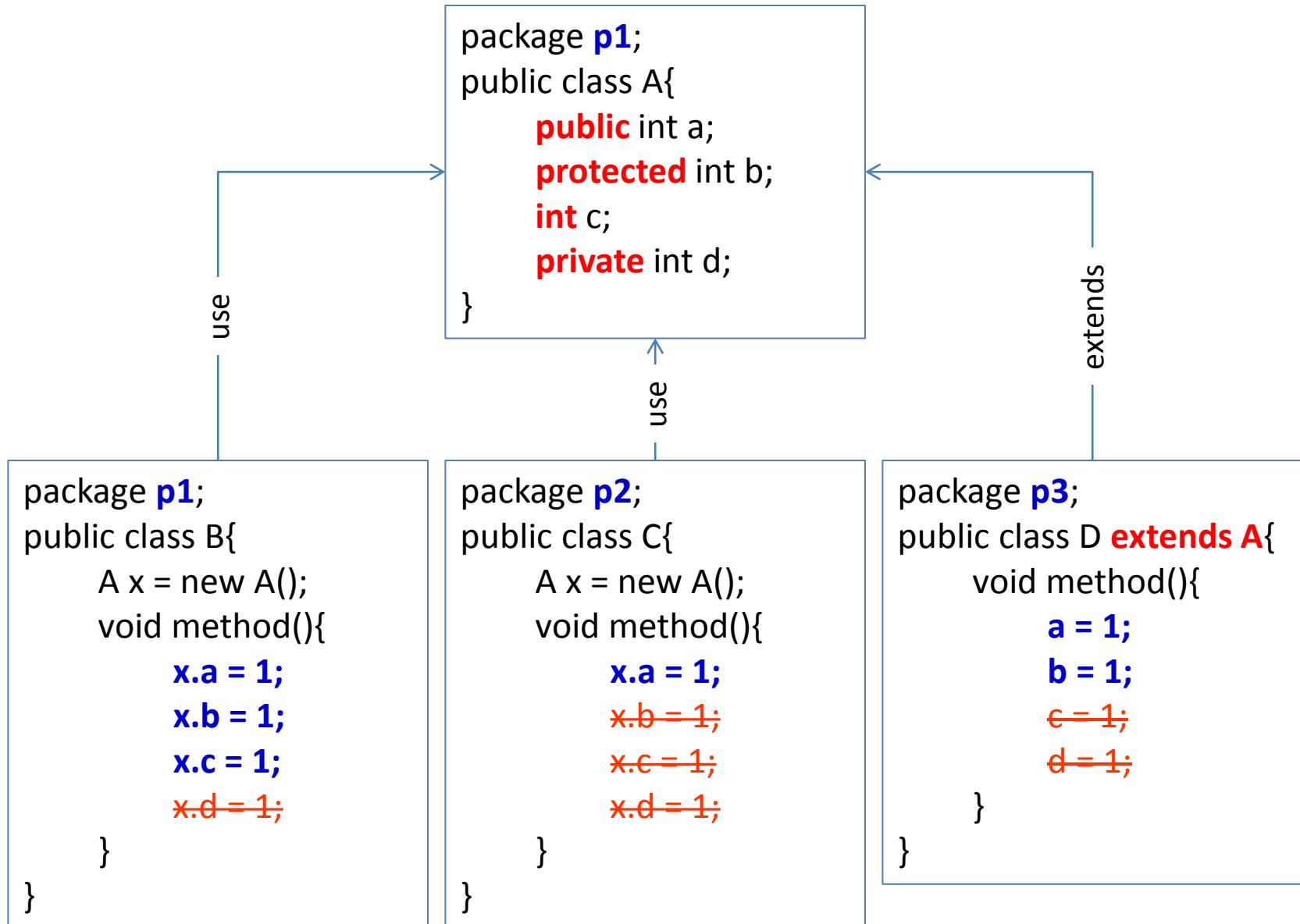
➢ Là private với các lớp truy xuất khác gói.

❖ **protected**: tương tự {default} nhưng cho phép kế thừa dù lớp con và cha khác gói.

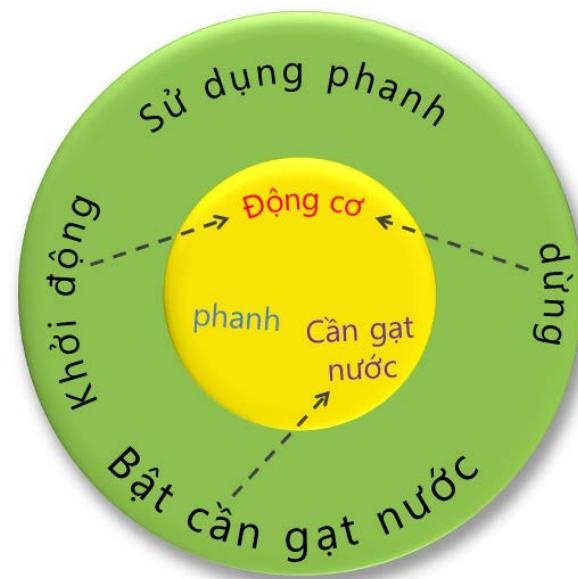
❑ Mức độ che dấu tăng dần theo chiều mũi tên

public → **protected** → **{default}** → **private**

ĐẶC TẢ TRUY XUẤT



- ❑ Encapsulation là tính che dấu trong hướng đối tượng.
 - ❖ Nên che dấu các trường dữ liệu
 - ❖ Sử dụng phương thức để truy xuất các trường dữ liệu
- ❑ Mục đích của che dấu
 - ❖ Bảo vệ dữ liệu
 - ❖ Tăng cường khả năng mở rộng



- ❑ Giả sử định nghĩa lớp SinhVien và công khai hoTen và điểm như sau

```
public class SinhVien{  
    public String hoTen;  
    public double diem;  
}
```

```
public class MyClass{  
    public static void main(String[] args){  
        SinhVien sv = new SinhVien();  
        sv.hoTen = "Nguyễn Văn Tèo";  
        sv.diem = 20.5;  
    }  
}
```

- ❑ Khi sử dụng người dùng có thể gán dữ liệu cho các trường một cách tùy tiện
- ❑ Điều gì sẽ xảy ra nếu điểm hợp lệ chỉ từ 0 đến 10

- ❑ Để che dấu thông tin, sử dụng private cho các trường dữ liệu.

```
private double diem;
```

- ❑ Bổ sung các phương thức getter và setter để đọc ghi các trường đã che dấu

```
public void setDiem(double diem){
```

```
    this.diem = diem;
```

```
}
```

```
public String getDiem(){
```

```
    return this.diem;
```

```
}
```

ENCAPSULATION

```
public class SinhVien{  
    private String hoTen;  
    private double diem;  
    public void setHoTen(String hoTen){  
        this.hoTen = hoTen;  
    }  
    public String getHoTen(){  
        return this.hoTen;  
    }  
    public void setDiem(double diem){  
        if(diem < 0 || > 10){  
            System.out.println("Điểm không hợp lệ");  
        }  
        else{  
            this.diem = diem;  
        }  
    }  
    public String getDiem(){  
        return this.diem;  
    }  
}
```

☐ Chỉ cần thêm mã vào phương thức setDiem() để có những xử lý khi dữ liệu không hợp lệ

```
public class MyClass{  
    public static void main(String[] args){  
        SinhVien sv = new SinhVien();  
        sv.setHoTen("Nguyễn Văn Tèo");  
        sv.setDiem(20);  
    }  
}
```



QUI TẮC ĐẶT TÊN TRONG JAVA

- ❑ Tên (class, field, method, package, interface, variable) được đặt theo qui ước (mềm) như sau:
 - ❖ Tên package: toàn bộ ký tự thường và dấu chấm
 - java.util, com.poly
 - ❖ Tên class, interface: Các từ phải viết hoa ký tự đầu
 - class Employee{}, class SinhVien{}, class HinhChuNhat()
 - ❖ Tên field, method, variable: Các từ phải viết hoa ký tự đầu ngoại trừ từ đầu tiên phải viết thường
 - hoTen, diem, fullName, mark
 - setHoTen(), input(), setDiem()
- ❑ Tên class, field và variable sử dụng danh từ
- ❑ Tên phương thức sử dụng động từ

TỔNG KẾT NỘI DUNG BÀI HỌC

- Khái niệm về đối tượng
- Khái niệm lớp
- Mô hình đối tượng và lớp
- Định nghĩa lớp
- Tạo đối tượng
- Định nghĩa phương thức
- Nạp chồng phương thức
- Hàm tạo
- Package
- Đặc tả truy xuất
- Encapsulation
- Qui ước đặt tên





LẬP TRÌNH Java™

BÀI 5: ARRAYLIST

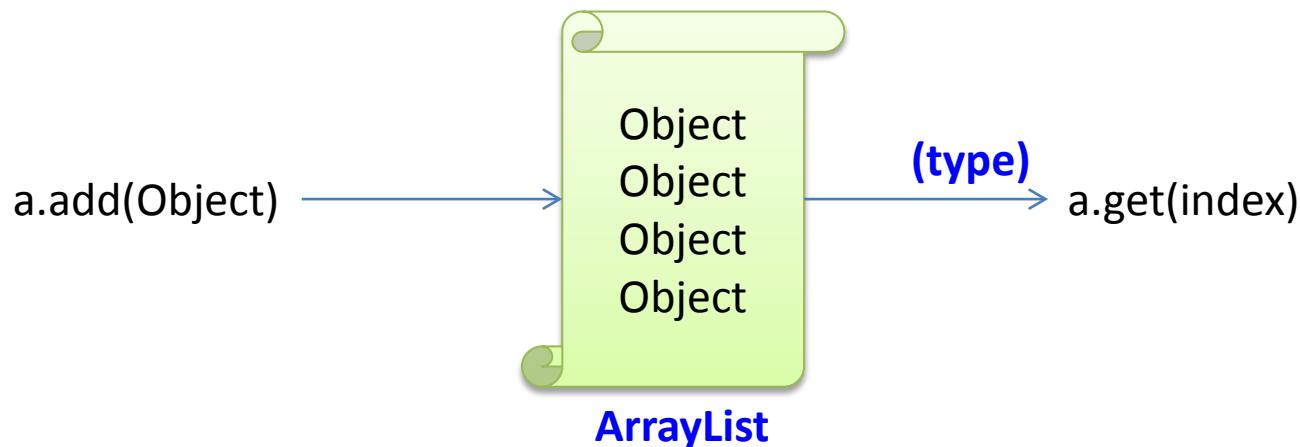
- ❑ Kết thúc bài học này bạn có khả năng
 - ❖ Hiểu và ứng dụng ArrayList
 - ❖ Hiểu và ứng dụng các hàm tiện ích của Collections



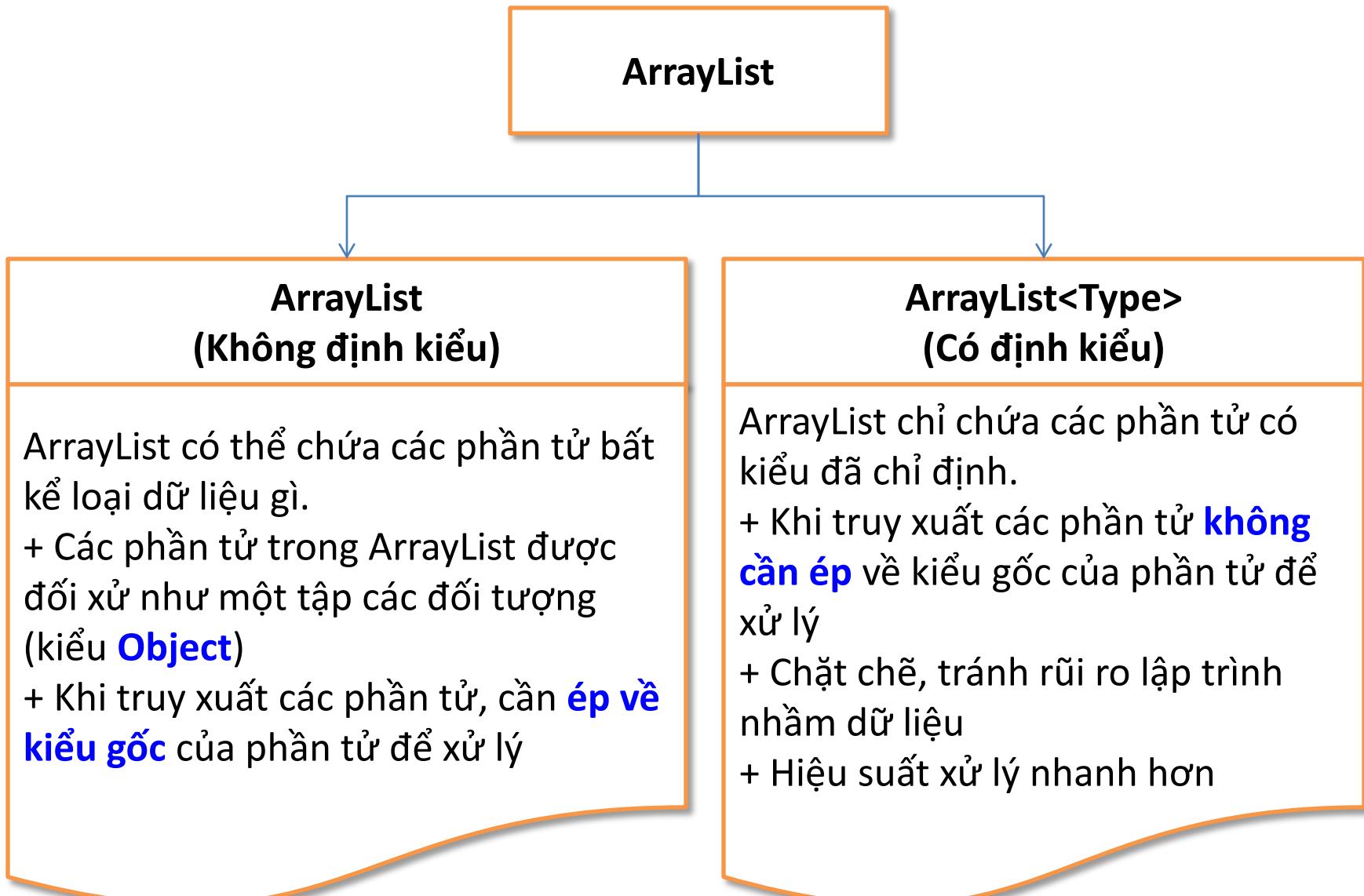
- ❑ Mảng có số phần tử cố định. Vì vậy có các nhược điểm sau:
 - ❖ Không thể bổ sung thêm hoặc xóa bớt các phần tử.
 - ❖ Lãng phí bộ nhớ
 - Nếu khai báo mảng với kích thước lớn để nắm giữ một vài phần tử.
 - Khai báo mảng với kích thước nhỏ thì không đủ chứa
- ❑ ArrayList giúp khắc phục nhược điểm nêu trên của mảng.
 - ❖ ArrayList có thể được xem như mảng động, có thể thêm bớt các phần tử một cách mềm dẻo.
- ❑ ArrayList còn cho phép thực hiện các phép toán tập hợp như hợp, giao, hiệu...

```
ArrayList a = new ArrayList();
a.add("Cường");
a.add(true);
a.add(1);
a.add(2.5)
Integer x = (Integer)a.get(2);
```

- + Khi add thêm số nguyên thủy thì tự động chuyển sang đối tượng kiểu **wrapper**
- + Khi truy xuất các phần tử, cần **ép về kiểu gốc** của phần tử để xử lý



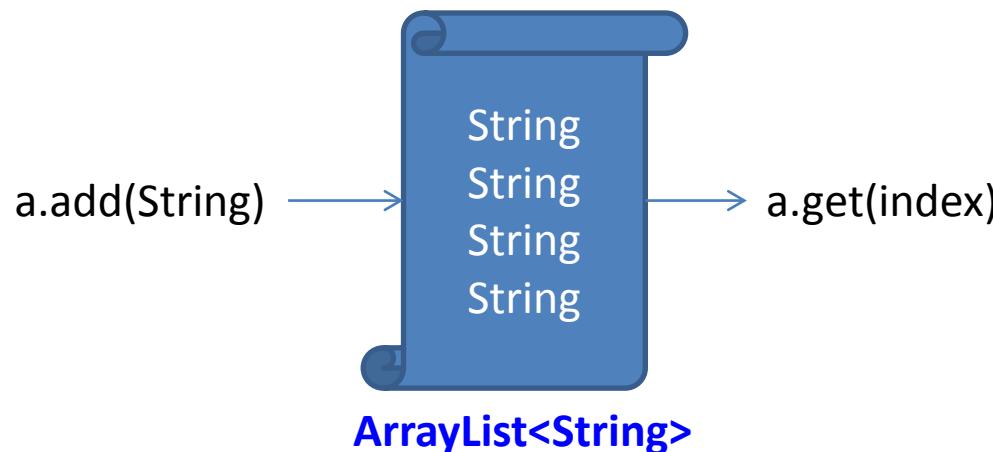
ARRAYLIST ĐỊNH KIỂU



ARRAYLIST<TYPE> ĐỊNH KIỂU

```
ArrayList<String> a = new ArrayList<String>();  
a.add("Cường");  
a.add("Tuấn");  
a.add("Phương");  
a.add("Hạnh")  
String s = a.get(2);
```

+ Khi truy xuất các phần tử **không cần ép** về kiểu gốc của phần tử để xử lý



Chú ý: <Type> là kiểu dữ liệu không phải kiểu nguyên thủy (phải sử dụng wrapper)

THAO TÁC THƯỜNG DÙNG

PHƯƠNG THỨC	MÔ TẢ
boolean add(Object)	Thêm vào cuối
void add(int index, Object elem)	Chèn thêm phần tử vào vị trí
boolean remove(Object)	Xóa phần tử
Object remove(int index)	Xóa và nhận phần tử tại vị trí
void clear()	Xóa sạch
Object set(int index, Object elem)	Thay đổi phần tử tại vị trí
Object get(int index)	Truy xuất phần tử tại vị trí
int size()	Số phần tử
boolean contains(Object)	Kiểm tra sự tồn tại
boolean isEmpty()	Kiểm tra rỗng
int indexOf(Object elem)	Tìm vị trí phần tử

THAO TÁC ARRAYLIST

```
ArrayList<String> a = new ArrayList<String>();
```

```
a.add("Cường");    ← [Cường]
```

```
a.add("Tuấn");    ← [Cường, Tuấn]
```

```
a.add("Phương");   ← [Cường, Tuấn, Phương]
```

```
a.add("Hồng");    ← [Cường, Tuấn, Phương, Hồng]
```

```
a.add(1, "Hạnh");   ← [Cường, Hạnh, Tuấn, Phương, Hồng]
```

```
a.set(0, "Tèo");    ← [Tèo, Hạnh, Tuấn, Phương, Hồng]
```

```
a.remove(3)        ← [Tèo, Hạnh, Tuấn, Hồng]
```

```
ArrayList<String> a = new ArrayList<String>();  
a.add("Cường");  
a.add("Tuấn");  
a.add("Phương");  
a.add("Hồng");  
a.add(1, "Hạnh");  
a.set(0, "Tèo");  
a.remove(3);  
a.remove("Phương");  
int x = a.size() – a.indexOf("Hồng");
```

1. Biến x có giá trị bằng bao nhiêu?
A. 0
B. 1
C. 2
D. 3
E. 4

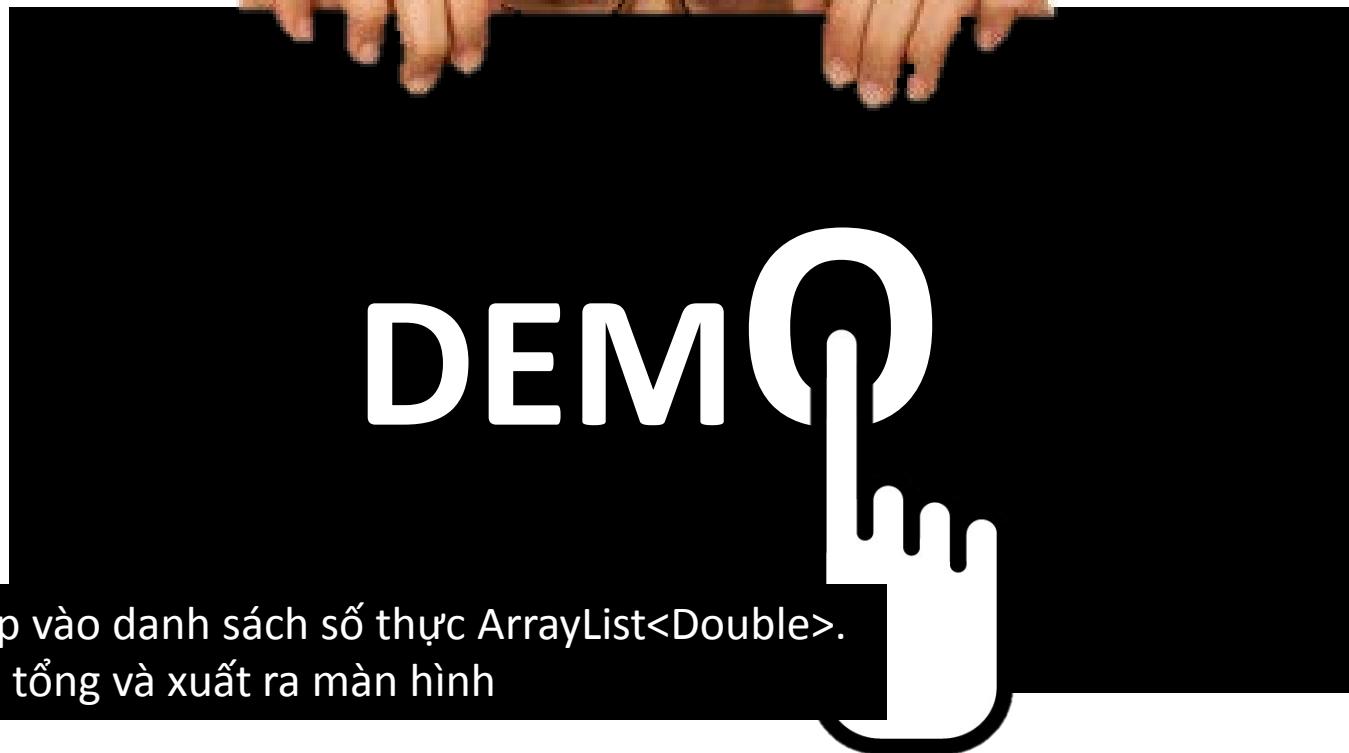
2. Nếu thay ~~a.indexOf("Hồng")~~ bằng
a.indexOf("Phương") thì kết quả x có giá trị là bao nhiêu

- Duyệt theo **chỉ số** với for hoặc sử dụng **for-each**.
Với ArrayList for-each thường được sử dụng hơn

```
ArrayList<Integer> a = new ArrayList<Integer>();  
a.add(5);  
a.add(9);  
a.add(4);  
a.add(8)
```

```
for(int i=0;i<a.size();i++){  
    Integer x = a.get(i);  
    <<xử lý x>>  
}
```

```
for(Integer x : a){  
    <<xử lý x>>  
}
```



☐ Sử dụng **ArrayList<SVPoly>** để nắm giữ danh sách sinh viên. Thông tin mỗi sinh viên gồm họ tên và điểm trung bình. Viết chương trình thực hiện việc quản lý như menu sau:

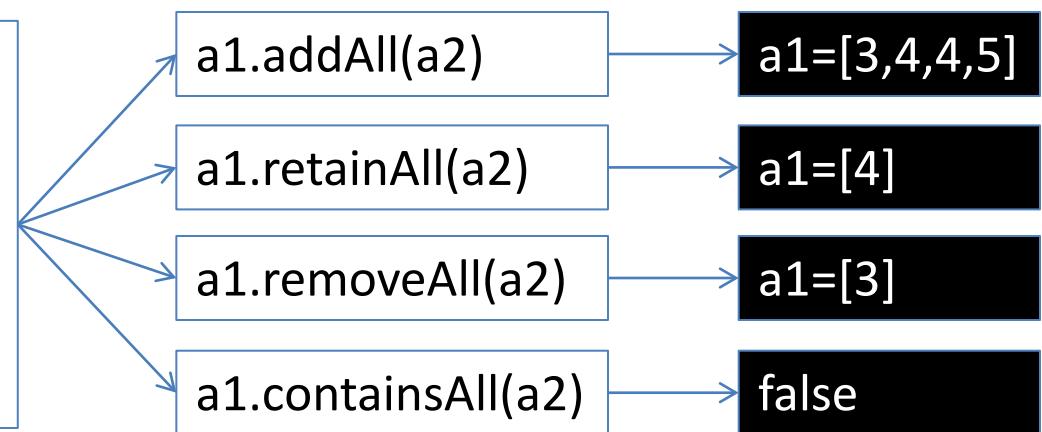
1. Nhập danh sách sinh viên
2. Xuất danh sách sinh viên đã nhập
3. Xuất danh sách sinh viên theo khoảng điểm
4. Tìm sinh viên theo họ tên
5. Tìm và sửa sinh viên theo họ tên
6. Tìm và xóa theo họ tên
7. Kết thúc

```
public class SVPoly{  
    public String hoTen;  
    public Double diemTB;  
}
```

THAO TÁC TẬP HỢP

PHƯƠNG THỨC	MÔ TẢ
addAll(Collection)	Hợp 2 tập hợp
removeAll(Collection)	Hiệu 2 tập hợp
retainAll(Collection)	Giao 2 tập hợp
boolean containsAll(Collection)	Kiểm tra sự tồn tại
toArray(T[])	Chuyển đổi sang mảng

```
ArrayList a1 = new ArrayList();
a1.add(3);
a1.add(4);
ArrayList a2 = new ArrayList();
a2.add(4);
a2.add(5);
```





THAO TÁC ARRAYLIST NÂNG CAO

- ☐ Lớp tiện ích **Collections** cung cấp các hàm tiện ích hỗ trợ việc xử lý ArrayList

PHƯƠNG THỨC	MÔ TẢ
int binarySearch (List list, Object key)	Tìm kiếm theo thuật toán chia đôi
void fill (List list, Object value)	Gán giá trị cho tất cả phần tử
void shuffle (List list)	Hoán vị ngẫu nhiên
void sort (List list)	Sắp xếp tăng dần
void reverse (List list)	Đảo ngược
void rotate (List list, int distance)	Xoay vòng
void swap(List list, int i, int j)	Tráo đổi

THAO TÁC ARRAYLIST NÂNG CAO

```
ArrayList<Integer> a = new ArrayList<Integer>();
```

```
a.add(3);
```

```
a.add(9);
```

```
a.add(8);
```

```
a.add(2);
```

← [3, 9, 8, 2]

```
Collections.swap(a, 0, 2);
```

← [8, 9, 3, 2]

```
Collections.shuffle(a);
```

← [X, X, X, X]

```
Collections.sort(a);
```

← [2, 3, 8, 9]

```
Collections.reverse(a);
```

← [9, 8, 3, 2]



Nhập danh sách 5 câu hỏi. Tráo ngẫu
nhiên và xuất danh sách câu hỏi đã tráo

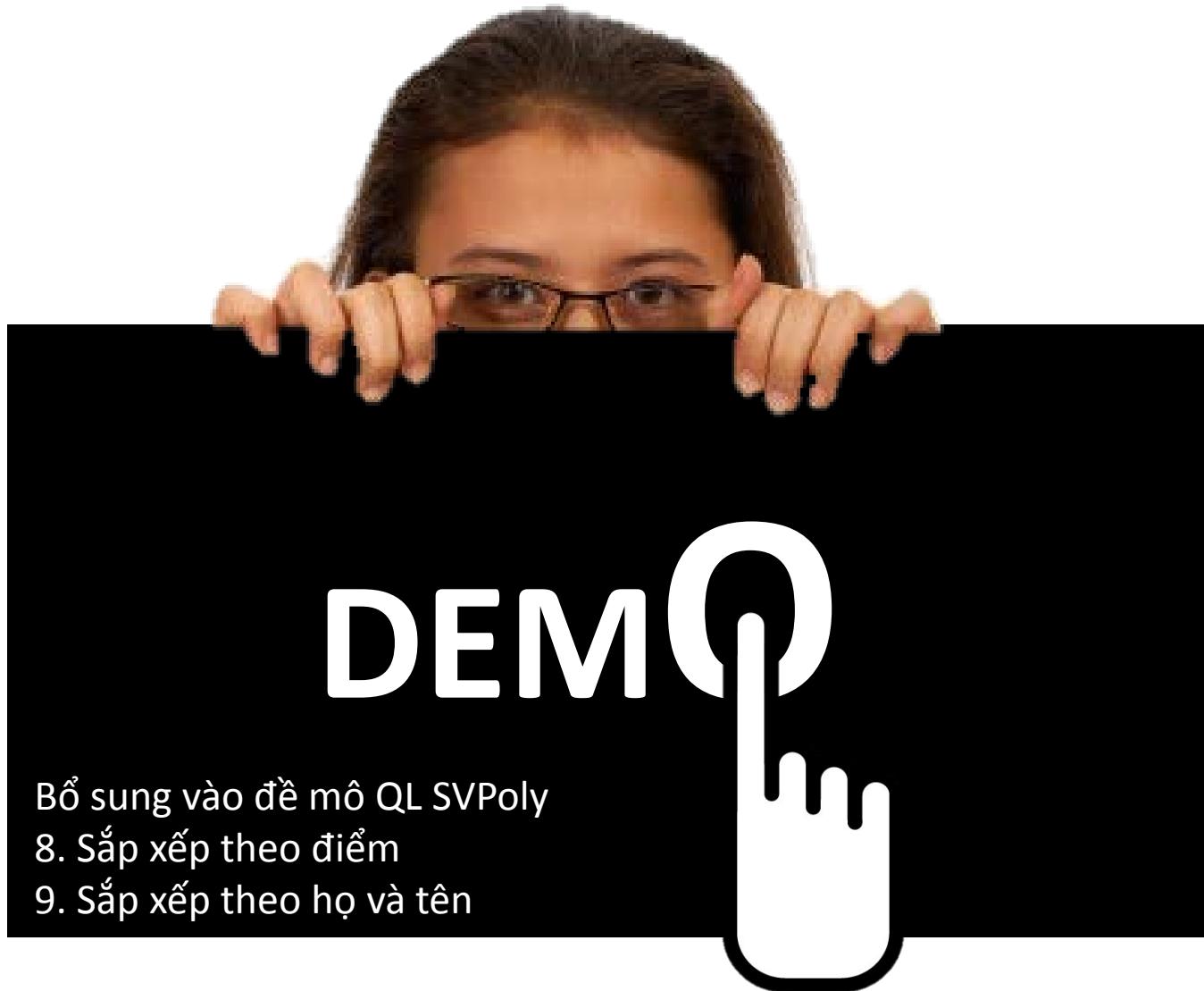
- ❑ Có 2 cách sử dụng Collections.**sort()** để sắp xếp ArrayList<Object>
- ❑ Cách 1: Collections.**sort(ArrayList)** đối với các phần tử có khả năng so sánh (Integer, Double, String...)
- ❑ Cách 2: Collections.**sort(ArrayList, Comparator)** bổ sung tiêu chí so sánh cho các phần tử. Cách này thường áp dụng cho các lớp do người dùng định nghĩa (NhanVien, SinhVienPoly...)

- ❑ Tiêu chí so sánh được chỉ ra để thực hiện việc sắp xếp. Trong bài này tiêu chí so sánh 2 SVPoly là so sánh theo điểm.

```
ArrayList<SVPoly> list = new ArrayList<SVPoly>();
Comparator<SVPoly> comp = new Comparator<SVPoly>() {
    @Override
    public int compare(SVPoly o1, SVPoly o2) {
        return o1.diemTB.compareTo(o2.diemTB);
    }
};
Collections.sort(list, comp);
```

Kết quả của compare() được sử dụng để sắp xếp o1 và o2. Có 3 trường hợp xảy ra:

- ✓ = 0: o1 = o2
- ✓ > 0: o1 > o2
- ✓ < 0: o1 < o2





TỔNG KẾT NỘI DUNG BÀI HỌC

- ❑ Giới thiệu ArrayList
- ❑ ArrayList có định kiểu
- ❑ Thao tác ArrayList
- ❑ Lớp tiện ích Collections



BÀI 5: CHUỖI VÀ BIỂU THỨC CHÍNH QUI

- ❑ Kết thúc bài học này bạn có khả năng
 - ❖ Hiểu và sử dụng chuỗi
 - ❖ Hiểu và sử dụng biểu thức chính qui

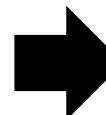


- ❑ String là xâu các ký tự.
 - ❖ String s = "Hello World";
- ❑ String là một class được xây dựng sẵn trong Java. String có rất nhiều phương thức giúp xử lý chuỗi một cách thuận tiện và hiệu quả.
- ❑ String là kiểu dữ liệu được sử dụng nhiều nhất trong lập trình



Ký tự	Hiển thị
\t	Ký tự tab
\r	Về đầu dòng
\n	Xuống dòng
\\"	\
\”	“

```
System.out.print("\t+ Họ và tên: Tuấn\r\n\t+ Tuổi: 40");
```



+ Họ và tên: Tuấn
+ Tuổi: 40

- So sánh
- Tìm vị trí của chuỗi con
- Lấy chuỗi con
- Tách và hợp chuỗi
- Chuyển đổi hoa thường
- Lấy độ dài
- ...

```
String fullname = "Nguyễn Văn Tèo";  
String first = fullname.substring(0, 6);
```

↓
Nguyễn

Phương thức	Mô tả
toLowerCase ()	Đổi in thường
toUpperCase ()	Đổi in hoa
trim()	Cắt các ký tự trắng 2 đầu chuỗi
length()	Lấy độ dài chuỗi
substring()	Lấy chuỗi con
charAt (index)	Lấy ký tự tại vị trí
replaceAll(find, replace)	Tìm kiếm và thay thế tất cả
split(separator)	Tách chuỗi thành mảng

Phương thức	Mô tả
equals()	So sánh bằng có phân biệt hoa/thường
equalsIgnoreCase()	So sánh bằng không phân biệt hoa/thường
contains()	Kiểm tra có chứa hay không
startsWith()	Kiểm tra có bắt đầu bởi hay không
endsWith ()	Kiểm tra có kết thúc bởi hay không
matches ()	So khớp với hay không?
indexOf()	Tìm vị trí xuất hiện đầu tiên của chuỗi con
lastIndexOf()	Tìm vị trí xuất hiện cuối cùng của chuỗi con

- ❑ Đăng nhập hợp lệ khi mã tài khoản là “hello” và mật khẩu trên 6 ký tự
- ❑ Thực hiện:
 - ❖ Nhập username và password từ bàn phím
 - ❖ Sử dụng equalsIgnoreCase() để so sánh username và length() để lấy độ dài mật khẩu

```
if(username.equalsIgnoreCase("hello") && password.length() > 6){  
    ...  
}  
else{  
    ...  
}
```



❑ Quản lý sinh viên

- ❖ Nhập mảng họ tên sinh viên
- ❖ Xuất họ và tên (IN HOA) những sinh viên tên Tuấn hoặc họ Nguyễn
- ❖ Xuất tên những sinh viên có tên lót là Mỹ

❑ Thự hiện

- ❖ **fullname.toUpperCase()**: đổi IN HOA
- ❖ **fullname.startsWith("Nguyễn ")**: họ Nguyễn
- ❖ **fullname.endsWith(" Tuấn")**: tên Tuấn
- ❖ **fullname.contains(" Mỹ ")**: lót Mỹ
- ❖ **fullname.lastIndexOf(" ")**: Lấy vị trí trắng cuối cùng
- ❖ **fullname.substring(lastIndex + 1)**: Lấy tên

- ❑ Tìm kiếm và thay thế chuỗi
- ❑ Thực hiện theo hướng dẫn sau
 - ❖ Nhập chuỗi nội dung, tìm kiếm và thay thế từ bàn phím
 - String content = scanner.nextLine()
 - String find = scanner.nextLine()
 - String replace = scanner.nextLine()
 - ❖ Thực hiện tìm và thay
 - String result = content.replaceAll(find, replace)



- ❑ Nhập chuỗi chứa dãy số phân cách bởi dấu phẩy và xuất các số chẵn
- ❑ Thực hiện
 - ❖ Sử dụng split() để tách chuỗi thành mảng bởi ký tự phân cách là dấu phẩy
 - ❖ Duyệt mảng, đổi sang số nguyên và kiểm tra số chẵn

```
String[] daySo = chuoi.split(",")
for(String so : daySo){
    int x = Integer.parseInt(so);
    if(x % 2 == 0){
        Số chẵn
    }
}
```

Bạn có biết các chuỗi sau đây biểu diễn những gì hay không?

- ❖ teo@fpt.edu.vn
- ❖ 54-P6-6661
- ❖ 54-P6-666.01
- ❖ 0913745789
- ❖ 192.168.11.200

1. Bạn có biết tại sao bạn nhận ra chúng không?
2. Làm thế nào để máy tính cũng có thể nhận ra như bạn?

- ❑ Máy tính có thể nhận dạng như chúng ta nếu chúng ta cung cấp qui luật nhận dạng cho chúng. Biểu thức chính qui cung cấp qui luật nhận dạng chuỗi cho máy tính.
- ❑ Biểu thức chính qui là một chuỗi mẫu được sử dụng để qui định dạng thức của các chuỗi. Nếu một chuỗi nào đó phù hợp với mẫu dạng thức thì chuỗi đó được gọi là khớp (hay đối sánh).
- ❑ Ví dụ: **[0-9]{3,7}**: Biểu thức chính qui này so khớp các chuỗi từ 3 đến 7 ký tự số.
 - ❖ [0-9]: đại diện cho 1 ký tự số
 - ❖ {3,7}: đại diện cho số lần xuất hiện (ít nhất 3 nhiều nhất 7)

VÍ DỤ: BIỂU THỨC CHÍNH QUI

```
Scanner scanner = new Scanner(System.in);
System.out.print("Số mobile: ");
String mobile = scanner.nextLine();
String pattern = "0[0-9]{9,10}";
if(mobile.matches(pattern)){
    System.out.println("Bạn đã nhập đúng số mobile");
}
else{
    System.out.println("Bạn đã nhập không đúng số mobile");
}
```

Biểu thức
chính qui

Kiểm tra mobile có so
khớp với pattern không?

s.matches(regex)

XÂY DỰNG BIỂU THỨC CHÍNH QUI

Regular Expression

Ký tự đại diện	
[xyz]	đại diện một ký tự x, y hay z
[ad-f]	đại diện một ký tự a, d, e hay f
[^xyz]	đại diện ký tự không thuộc [xyz]
\d	tương đương [0-9]
\w	tương đương [0-9a-zA-Z_]
\D	tương đương [^\d]
\W	tương đương [^\w]
\s	đại diện ký tự trắng (\r\n\t\f)
.	đại diện ký tự bất kỳ
^	chỉ ra mẫu bắt đầu
\$	chỉ ra mẫu kết thúc
\\, \., \\$, \^	đại diện '\', '.', '\$' hay '^'

{M,N}	ít nhất M, nhiều nhất N lần
{N}	Đúng N lần
?	0-1
*	0-N
+	1-N
Không	1

[0-9]{3, 7}



❑ Số CMND

❖ [0-9]{9}

❑ Số điện thoại di động việt nam

❖ 0\d{9,10}

❑ Số xe máy sài gòn

❖ 5\d-[A-Z]\d-((\d{4})|(\d{3}\.\d{2}))

❑ Địa chỉ email

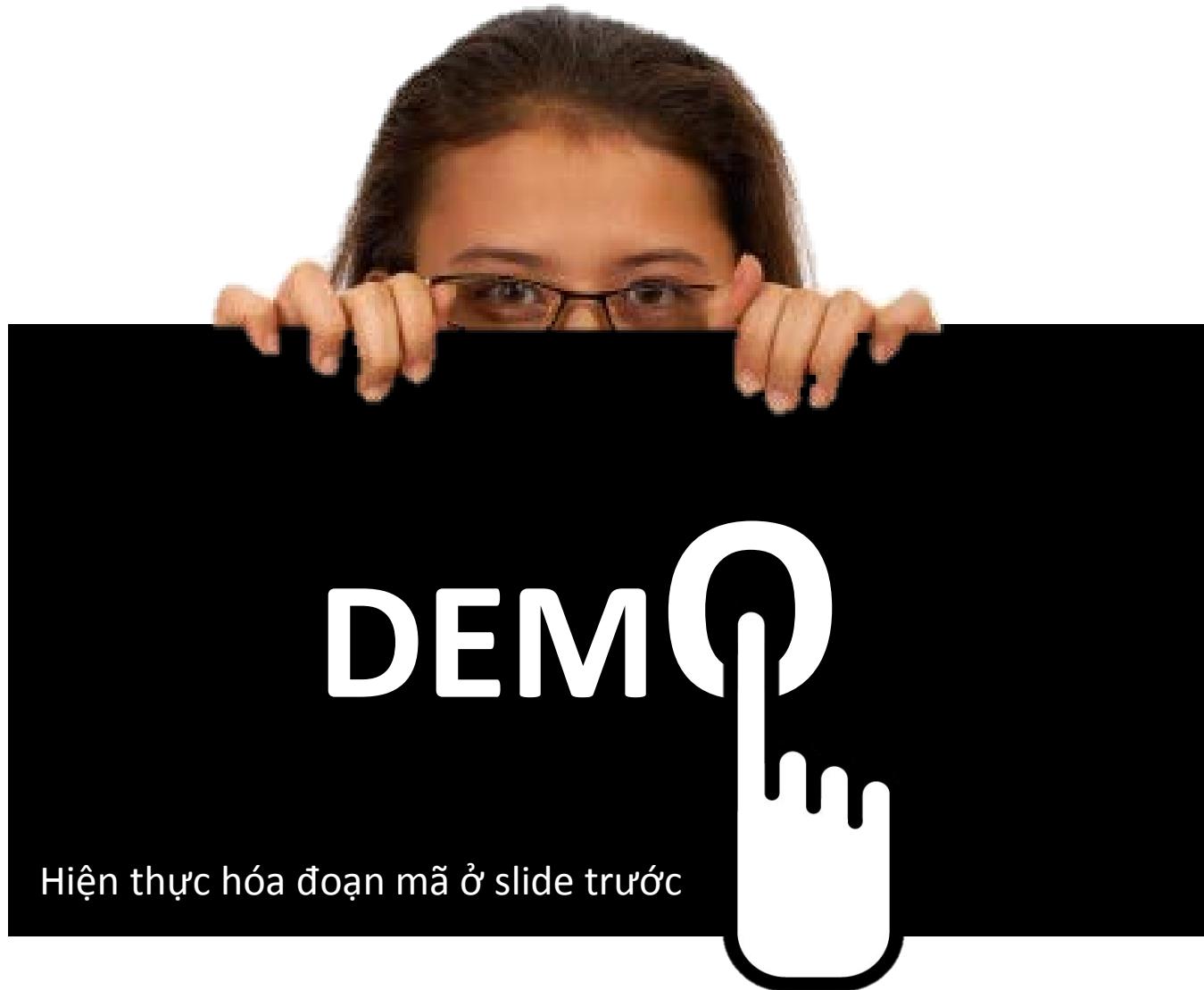
❖ \w+@\w+(\.\w){1,2}

VÍ DỤ VỀ REGEX

```
Scanner in = new Scanner(System.in);  
  
System.out.print("Email: ");  
String email = in.nextLine();  
  
System.out.print("Số điện thoại Huế: ");  
String phone = in.nextLine();  
  
String reEmail = "\\w+@\\w+\\.\\w+";  
if(!email.matches(reEmail)) {  
    System.out.println("Không đúng dạng email!");  
}  
  
String rePhone = "0543\\d{6}";  
if(!phone.matches(rePhone)) {  
    System.out.println("Không phải số điện thoại ở Huế!");  
}
```

Email đơn giản

Số điện thoại để bàn ở Huế



THỰC HÀNH - VALIDATION

- ☐ Nhập thông tin nhân viên từ bàn phím. Thông tin của mỗi nhân viên phải tuân theo các ràng buộc sau. Xuất thông báo lỗi và yêu cầu nhập lại

Thông tin	Kiểm soát	RegEx
Mã sinh viên	5 ký tự hoa	[A-Z]{5}
Mật khẩu	Ít nhất 6 ký tự	.{6,}
Họ và tên	Chỉ dùng alphabet và ký tự trắng	[a-zA-Z]+
Email	Đúng dạng email	\w+@\w+(\.\w+){1,2}
Điện thoại	Điện thoại Sài gòn	083\d{7}
Số xe máy	Số xe máy Sài gòn	5\d-[A-Z]-((\d{4}) (\d{3}).{2}))
Số CMND	10 chữ số	\d{10}
Website	Địa chỉ website	http://www\.\w+\.\w{2,4}



TỔNG KẾT NỘI DUNG BÀI HỌC

- ❑ Giới thiệu chuỗi (String)
- ❑ Ký tự đặc biệt
- ❑ Thao tác chuỗi
- ❑ Giới thiệu biểu thức chính qui (Regular Expression)
- ❑ Xây dựng biểu thức chính qui
- ❑ Ứng dụng biểu thức chính qui



LẬP TRÌNH Java™

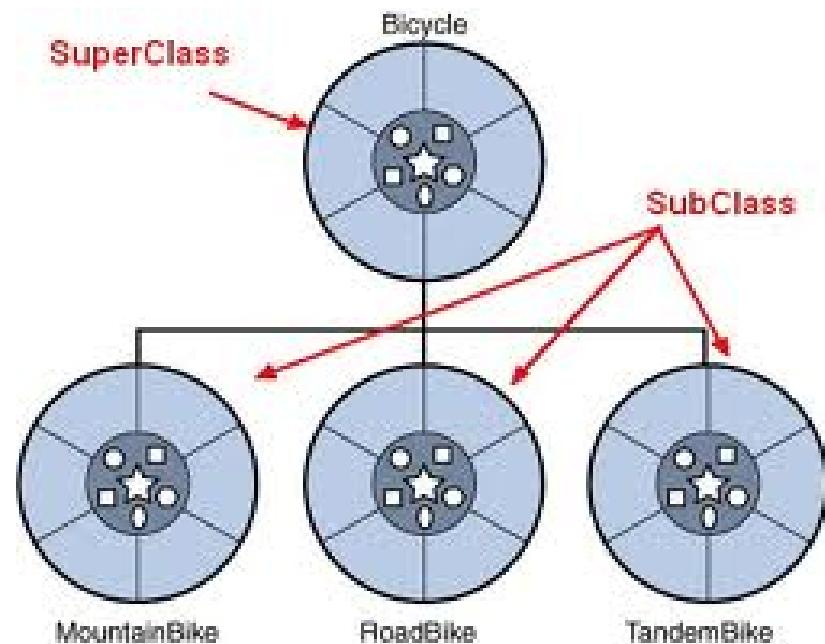
BÀI 7: KẾ THỪA

☐ Kết thúc bài học này bạn có khả năng

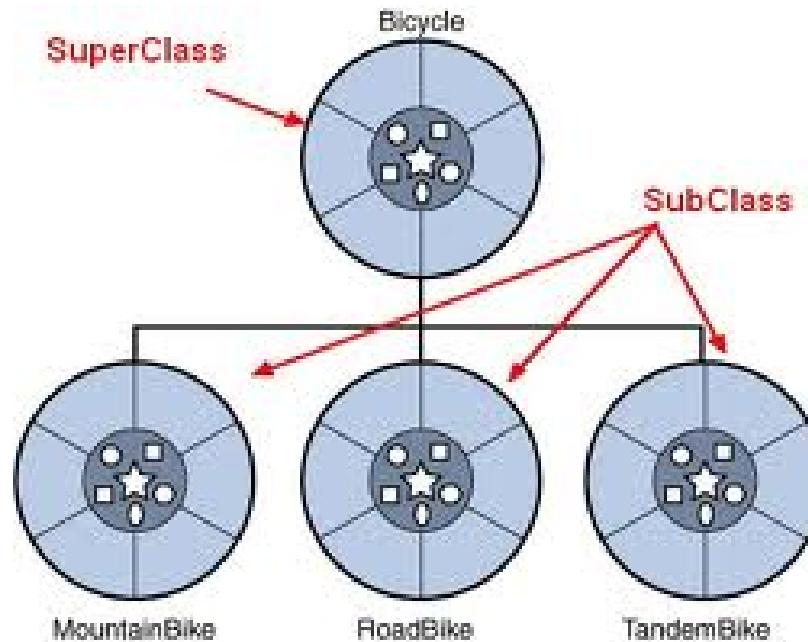
- ❖ Nắm vững sự phân cấp thừa kế
- ❖ Tái sử dụng các lớp sẵn có
- ❖ Biết cách ghi đè phương thức
- ❖ Nắm vững lớp và phương thức trừu tượng

SỰ PHÂN CẤP THỪA KẾ

- ❑ Các lớp trong Java tồn tại trong một hệ thống thứ bậc phân cấp, gọi là cây thừa kế
- ❑ Lớp bậc trên gọi là lớp cha (super class) trong khi các lớp bậc dưới gọi là lớp con (sub class)
- ❑ Trong Java một lớp chỉ có một lớp cha duy nhất (đơn thừa kế)



PHÂN CẤP THỪA KẾ

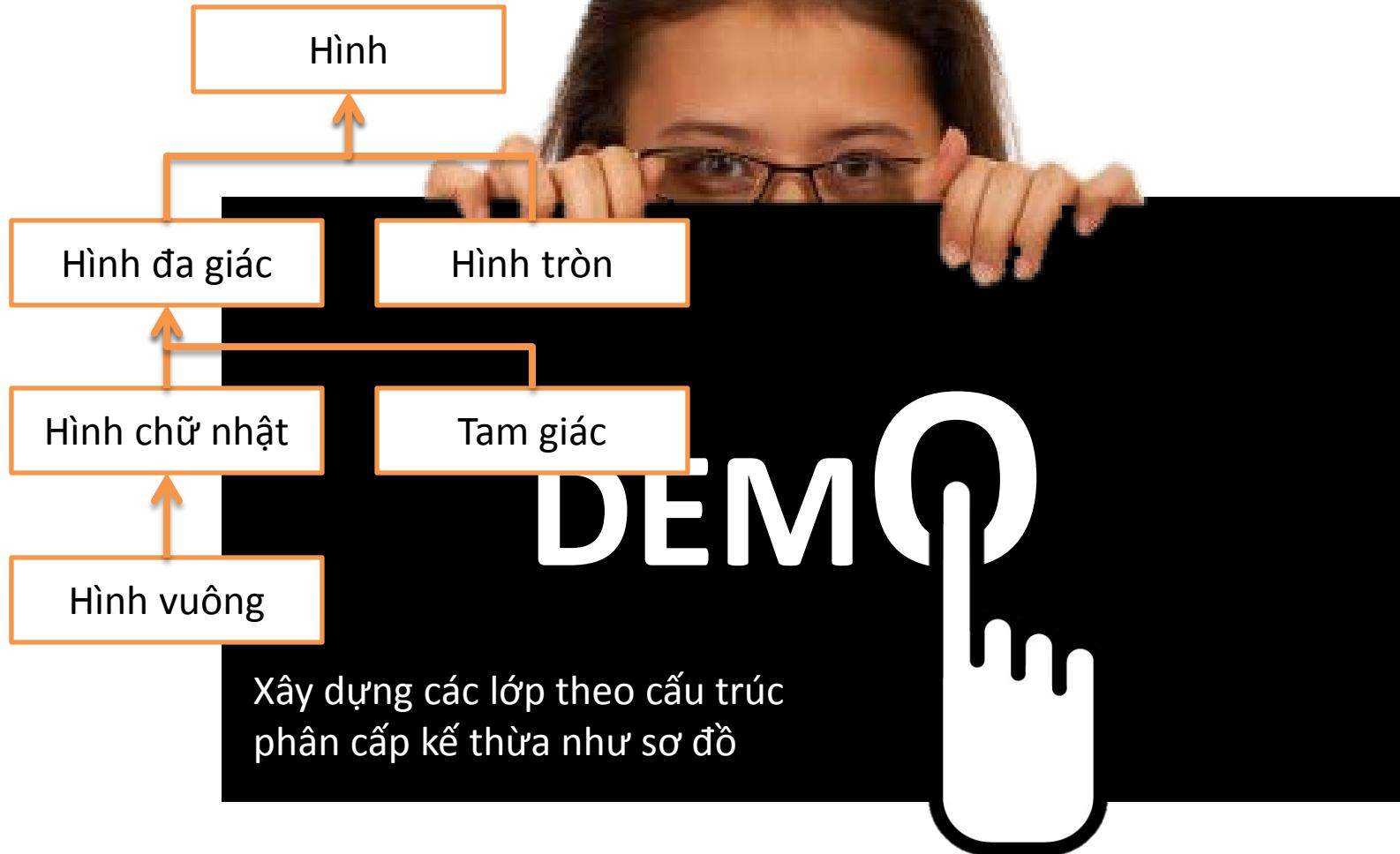


```
class Bicycle{...}
```

```
class MountainBike extends Bicycle{...}
```

```
class RoadBike extends Bicycle{...}
```

```
class TandemBike extends Bicycle{...}
```



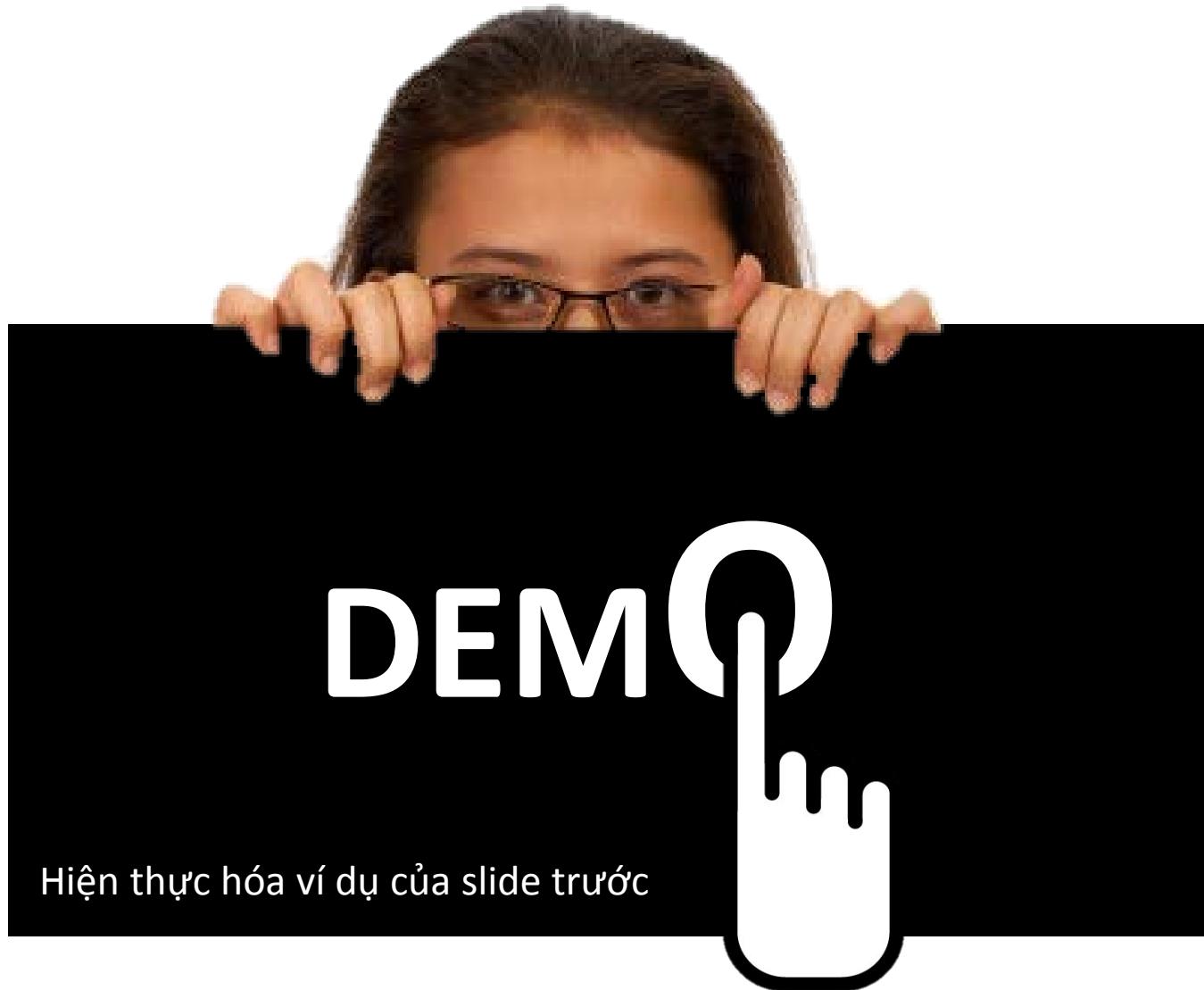


- ❑ Mục đích của thừa kế là tái sử dụng.
- ❑ Lớp con được phép sở hữu các tài sản (trường và phương thức) của lớp cha
 - ❖ Lớp con được phép sở hữu các tài sản public hoặc protected của lớp cha
 - ❖ Lớp con cũng được phép sở hữu các tài sản mặc định {default} của lớp cha nếu lớp con và lớp cha được định nghĩa cùng gói
 - ❖ Lớp con không thể truy cập thành viên private của lớp cha
- ❑ Lớp con không kế thừa các hàm tạo của lớp cha

```
package poly.ho;
public class NhanVien{
    public String hoTen;
    protected double luong;
    public NhanVien(String hoTen, double luong){...}
    void xuat(){...}
    private double thueThuNhap(){...}
}
```

```
package poly.hcm;
public class TruongPhong extends NhanVien{
    public double trachNhiem;
    public TruongPhong_(String hoTen, double luong, double trachNhiem){...}
    public void xuat(){
        // Mã ở đây có thể sử dụng những tài sản nào của lớp cha
    }
}
```

- A. super.hoTen
- B. super.luong
- C. super.xuat()
- D. super.thueThuNhap()



- ❑ Truy cập đến các thành viên của lớp cha bằng cách sử dụng từ khóa super
- ❑ Có thể sử dụng super để gọi hàm tạo của lớp cha

```
public class Parent{  
    public String name;  
    public void method(){}
}
```

```
public class Child extends Parent{  
    public String name;  
    public void method(){  
        this.name = super.name;  
        super.method()  
    }
}
```

SỬ DỤNG SUPER

```
package poly.ho;
public class NhanVien{
    public NhanVien(String hoTen, double luong){...}
    public void xuat(){...}
}
```

```
package poly.hcm;
public class TruongPhong extends NhanVien{
    public double trachNhiem;
    public TruongPhong (String hoTen, double luong, double trachNhiem){
        super(hoTen, luong);
        this.trachNhiem = trachNhiem
    }
    public void xuat(){
        super.xuat()
        System.out.println(trachNhiem)
    }
}
```

GHI ĐỀ PHƯƠNG THỨC (OVERRIDING)

- Overriding là trường hợp lớp con và lớp cha có phương thức cùng cú pháp.



- Lớp Parent và Child đều có phương thức method() cùng cú pháp nên method() trong Child sẽ đè lên method() trong Parent

Parent o = new Child();
o.method()

Mặc dù o có kiểu là Parent nhưng o.method()
thì method() của lớp Child sẽ chạy do bị đè

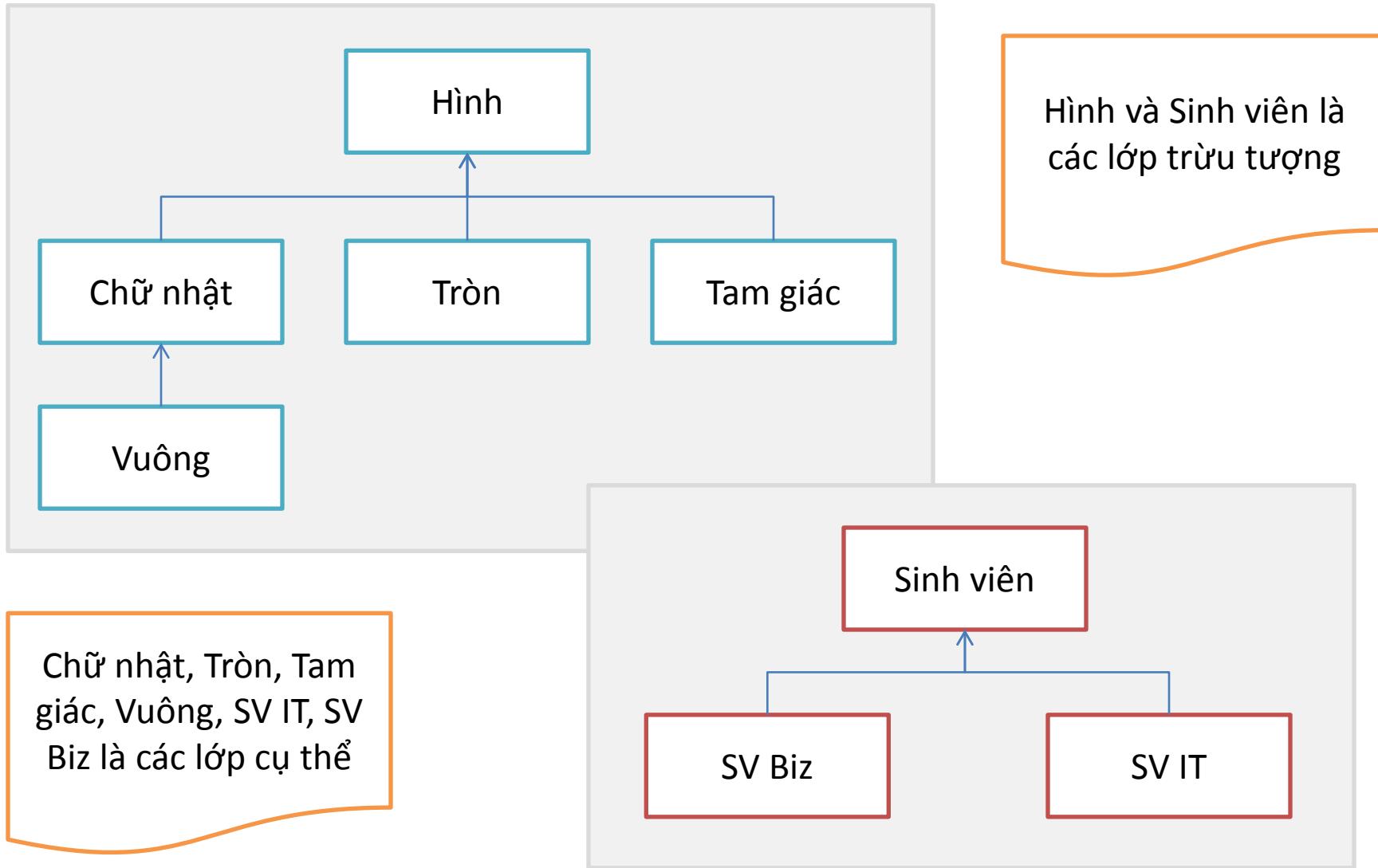
- Lớp con ghi đè phương thức của lớp cha thì sẽ che dấu phương thức của lớp cha
- Mục đích của ghi đè là để sửa lại phương thức của lớp cha trong lớp con
- Sử dụng từ khóa super để truy cập đến phương thức đã bị ghi đè của lớp cha.
- Đặc tả truy xuất của phương thức lớp con phải có độ công khai **bằng hoặc hơn** đặc tả truy xuất của phương thức lớp cha.





- ❑ Lớp trừu tượng là lớp có các hành vi chưa được xác định rõ
 - ❖ Ví dụ 1: Đã là hình thì chắc chắn là có diện tích và chu vi nhưng chưa xác định được cách tính mà phải là một hình cụ thể như chữ nhật, tròn, tam giác... mới có thể xác định cách tính
 - ❖ Ví dụ 2: Sinh viên thì chắc chắn có điểm trung bình nhưng chưa xác định được cách tính như thế nào mà phải là sinh viên của ngành nào mới biết được môn học và công thức tính điểm cụ thể.
- ❑ Vậy lớp hình và lớp sinh viên là các lớp trừu tượng vì phương thức tính chu vi, diện tích và tính điểm chưa thực hiện được.

LỚP TRỪ TƯỢNG



ĐỊNH NGHĨA LỚP TRƯỞNG

```
abstract public class MyClass{  
    abstract public type MyMethod();  
}
```

Sử dụng từ khóa
abstract để định
nghĩa lớp và
phương thức trùu
tương

```
abstract public class SinhVien{  
    abstract public double getDiemTB();  
}
```

```
abstract public class Hinh{  
    abstract public double getChuVi();  
    abstract public double getDienTich();  
}
```

ĐỊNH NGHĨA LỚP TRƯỞNG

```
abstract public class SinhVien{  
    public String hoTen;  
    abstract public double getDiemTB();  
}
```

```
public class SinhVienIT extends SinhVien{  
    public double diemJava;  
    public double diemCss;  
    @Override  
    public double getDiemTB(){  
        return (2 * diemJava + diemCss)/3;  
    }  
}
```

```
public class SinhVienBiz extends SinhVien {  
    public double keToan;  
    public double marketting;  
    public double banHang;  
    @Override  
    public double getDiemTB(){  
        return  
            (keToan + marketting + banHang)/3;  
    }  
}
```

ĐỊNH NGHĨA LỚP TRỪU TƯỢNG

- ❑ Từ khóa **abstract** được sử dụng để định nghĩa lớp và phương thức trừu tượng
- ❑ Phương thức trừu tượng là phương thức không có phần thân xử lý và được khai báo bằng từ khóa abstract.
- ❑ Lớp chứa phương thức trừu tượng thì lớp đó phải là lớp trừu tượng.
- ❑ Trong lớp trừu tượng có thể định nghĩa các phương thức cụ thể hoặc khai báo các trường
- ❑ Không thể sử dụng new để tạo đối tượng từ lớp trừu tượng.



TÍNH ĐA HÌNH (POLYMORPHISM)

- ❑ Overriding thực hiện tính đa hình trong lập trình hướng đối tượng (một hành vi được thể hiện với các hình thái khác nhau)
- ❑ Gọi phương thức bị ghi đè được quyết định lúc chạy chương trình (runtime) chứ không phải lúc biên dịch chương trình (compile time)



TÍNH ĐA HÌNH (POLYMORPHISM)

```
abstract public class DongVat{  
    abstract public void speak();  
}
```

```
DongVat cho = new Cho();  
DongVat meo= new Meo();  
DongVat vit = new Vit();  
  
cho.speak();  
meo.speak();  
vit.speak();
```

```
public class Cho extends DongVat{  
    public void speak(){  
        System.out.println("Woof");  
    }  
}
```

```
public class Meo extends DongVat{  
    public void speak(){  
        System.out.println("Meo");  
    }  
}
```

```
public class Vit extends DongVat{  
    public void speak(){  
        System.out.println("Quack");  
    }  
}
```

TỔNG KẾT NỘI DUNG BÀI HỌC

- Thừa kế
- Gọi hàm tạo của lớp cha
- Sử dụng super
- Ghi đè phương thức
- Lớp và phương thức trừu tượng



BÀI 8: KIẾN THỨC NÂNG CAO VỀ PHƯƠNG THỨC VÀ LỚP

❑ Kết thúc bài học này bạn có khả năng

- ❖ Hiểu sâu hơn về hàm tạo
- ❖ Phân biệt được tham biến và tham trị
- ❖ Sử dụng tham số biến đổi
- ❖ Biết cách sử dụng static, final
- ❖ Hiểu thuật toán đệ qui

VẤN ĐỀ VỀ CONSTRUCTOR

- 1) Nếu một lớp không định nghĩa constructor thì Java tự động cung cấp **constructor mặc định** (không tham số) cho lớp.
- 2) Trong một constructor muốn gọi constructor khác cùng lớp thì sử dụng **this(tham số)**, muốn gọi constructor của lớp cha thì sử dụng **super(tham số)**
- 3) Nếu trong constructor không gọi constructor khác thì nó tự gọi constructor không tham số của lớp cha **super()**
- 4) Lời gọi constructor (super() hoặc this()) khác phải là lệnh đầu tiên
- 5) Khi đã định nghĩa các constructor cho một lớp thì chỉ được phép sử dụng các constructor này để tạo đối tượng

❑ Hãy cho biết đoạn mã lệnh sau sai ở đâu? vì sao?

```
public class Parent{  
    public Parent(int x){}  
}
```

```
public class Child extends Parent{  
}
```



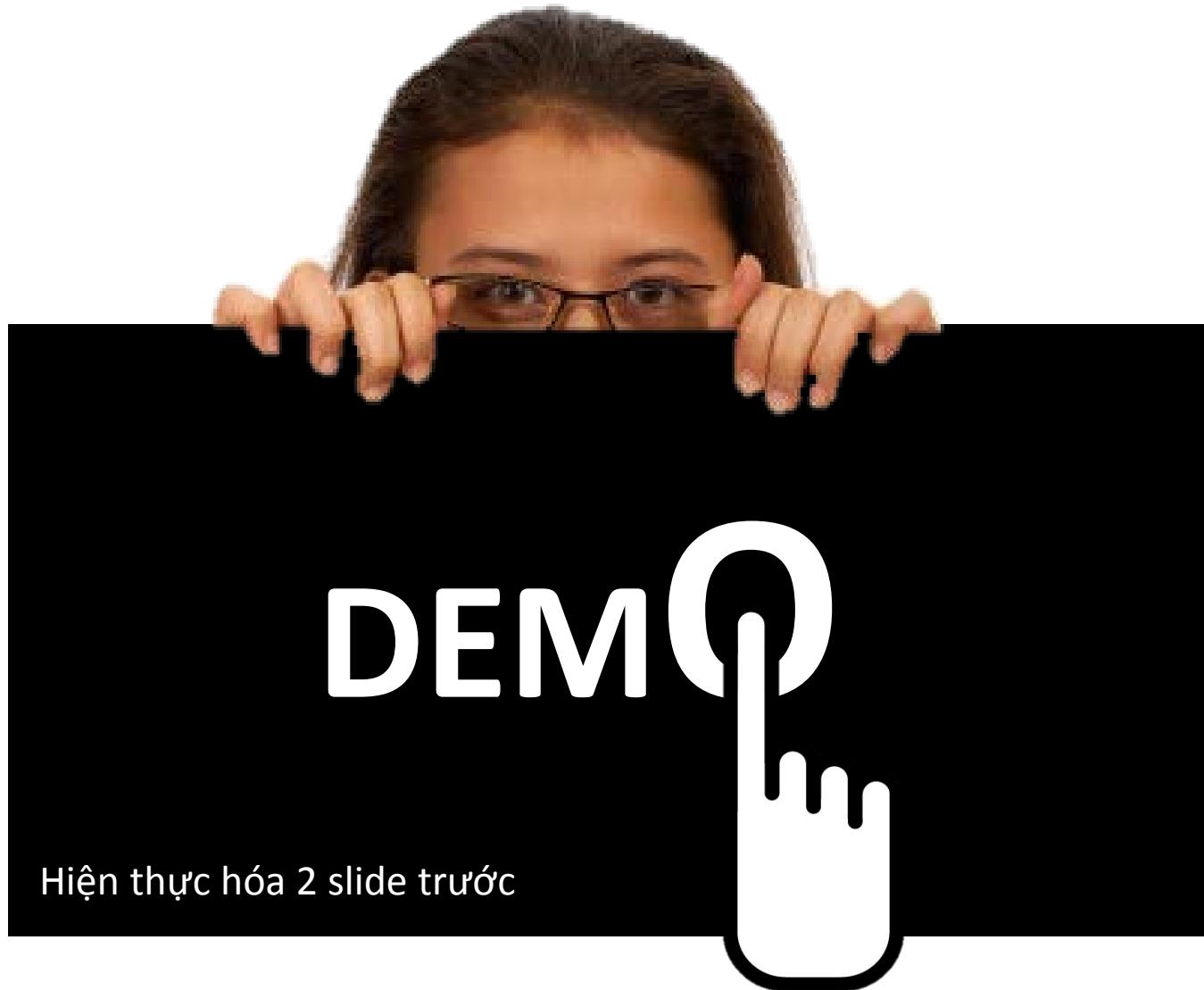
- Chiếu theo điều 1) và điều 3) slide trước ta có sơ đồ tương đương

```
public class Parent{  
    public Parent(int x){}  
}
```

```
public class Child extends Parent{  
    public Child(){  
        super()  
    }  
}
```

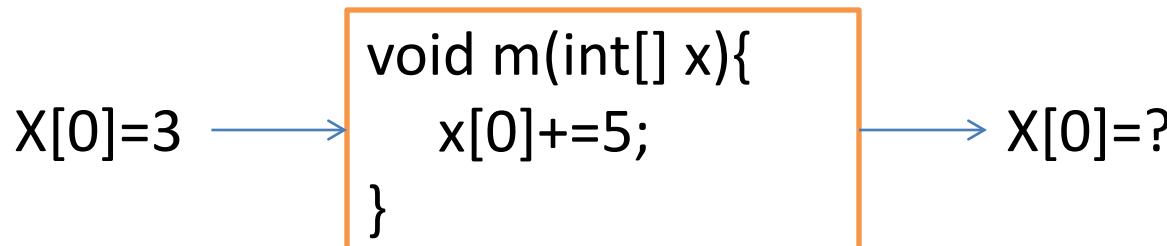
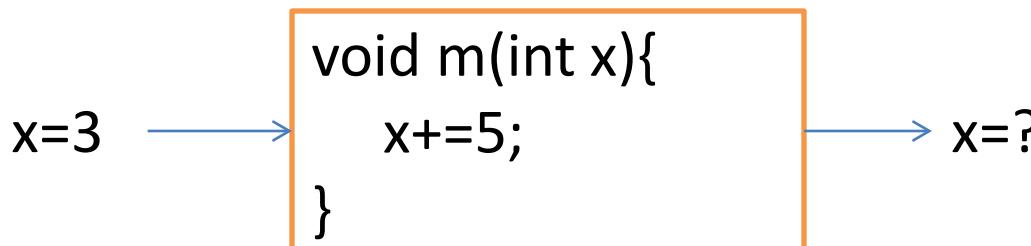


Chiếu theo điều 4 thì
Parent không có
constructor không tham
số nên gây lỗi lúc dịch

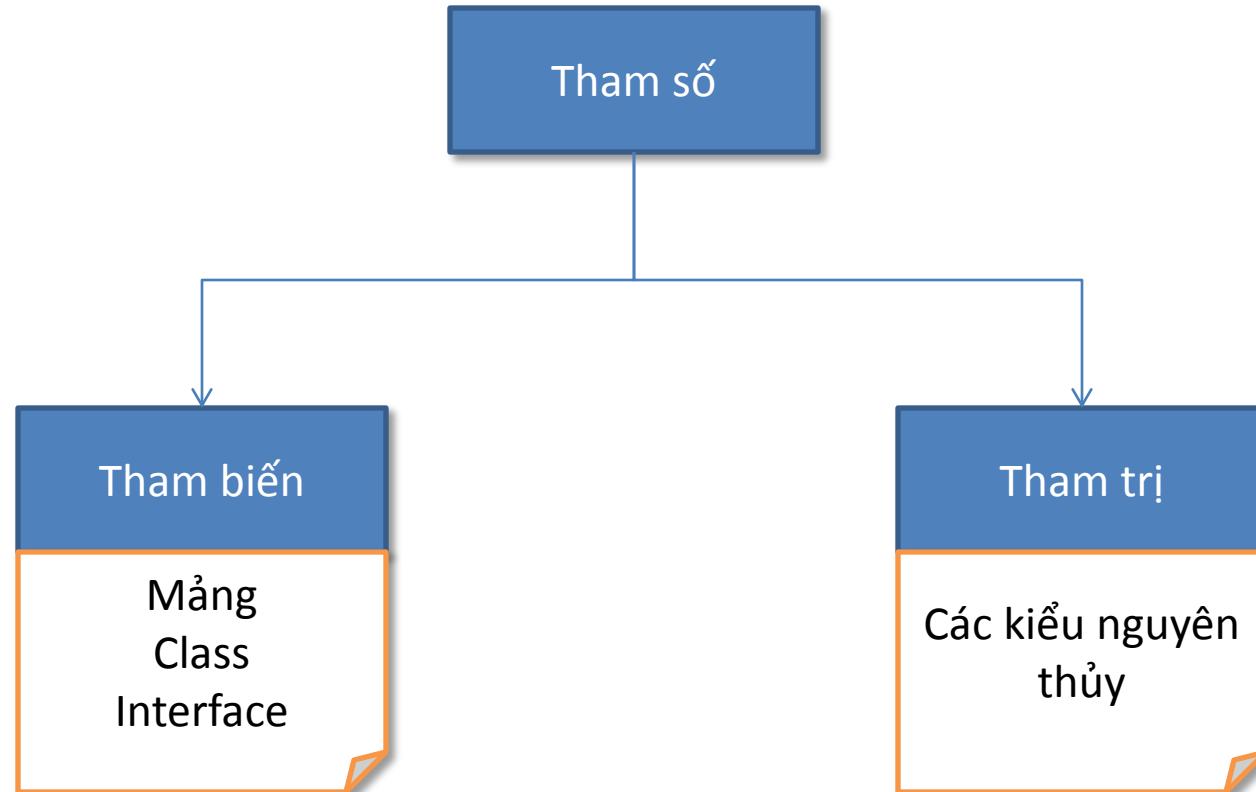


THAM SỐ PHƯƠNG THỨC

- Khi truyền tham số vào một phương thức, nếu phương thức có làm thay đổi giá trị của tham số thì giá trị của tham số sau khi gọi phương thức có bị thay đổi hay không?



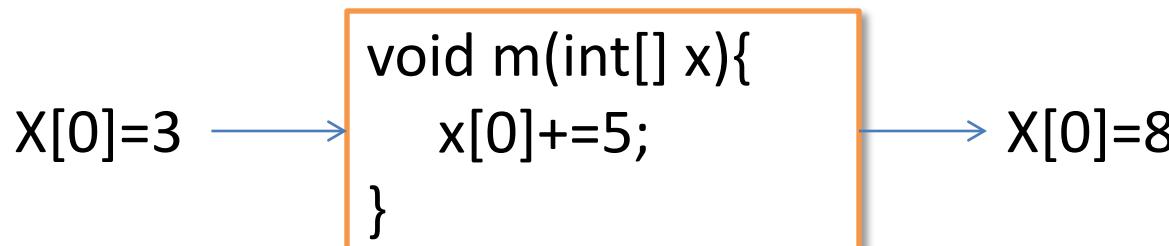
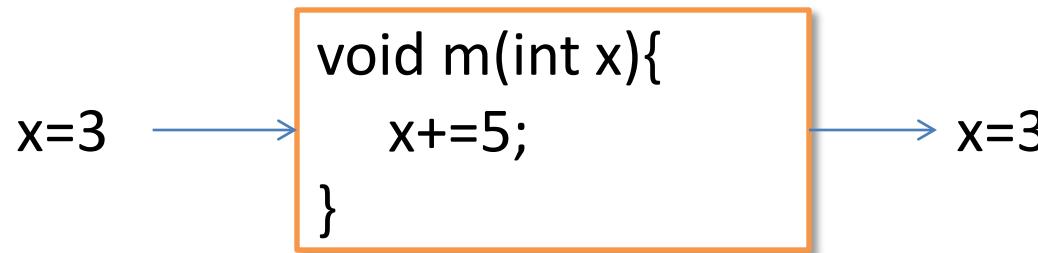
PHÂN LOẠI THAM SỐ



TRUYỀN THAM SỐ CHO PHƯƠNG THỨC

☐ Khi phương thức làm thay đổi giá trị của tham số thì

- ❖ Nếu là tham trị: giá trị của tham số sẽ không bị thay đổi
- ❖ Nếu là tham biến: giá trị của tham số sẽ bị thay đổi theo





1. Hiện thực hóa `2 m()` ở slide trước
2. Bổ sung thêm một phương thức nhận tham số là một đối tượng và phương thức làm thay đổi các trường dữ liệu của đối tượng tham số. Kiểm tra các trường dữ liệu có thay đổi hay không sau khi gọi phương thức

THAM SỐ BIẾN ĐỔI (VARARGS)

- Tham số biến đổi là tham số khi truyền vào phương thức với số lượng tùy ý (phải cùng kiểu).

```
void m(int...x){...}
```

Gọi phương thức

```
m(2,6,8)
```

```
m(2)
```

```
int[] x = {2,6,8}  
m(x)
```

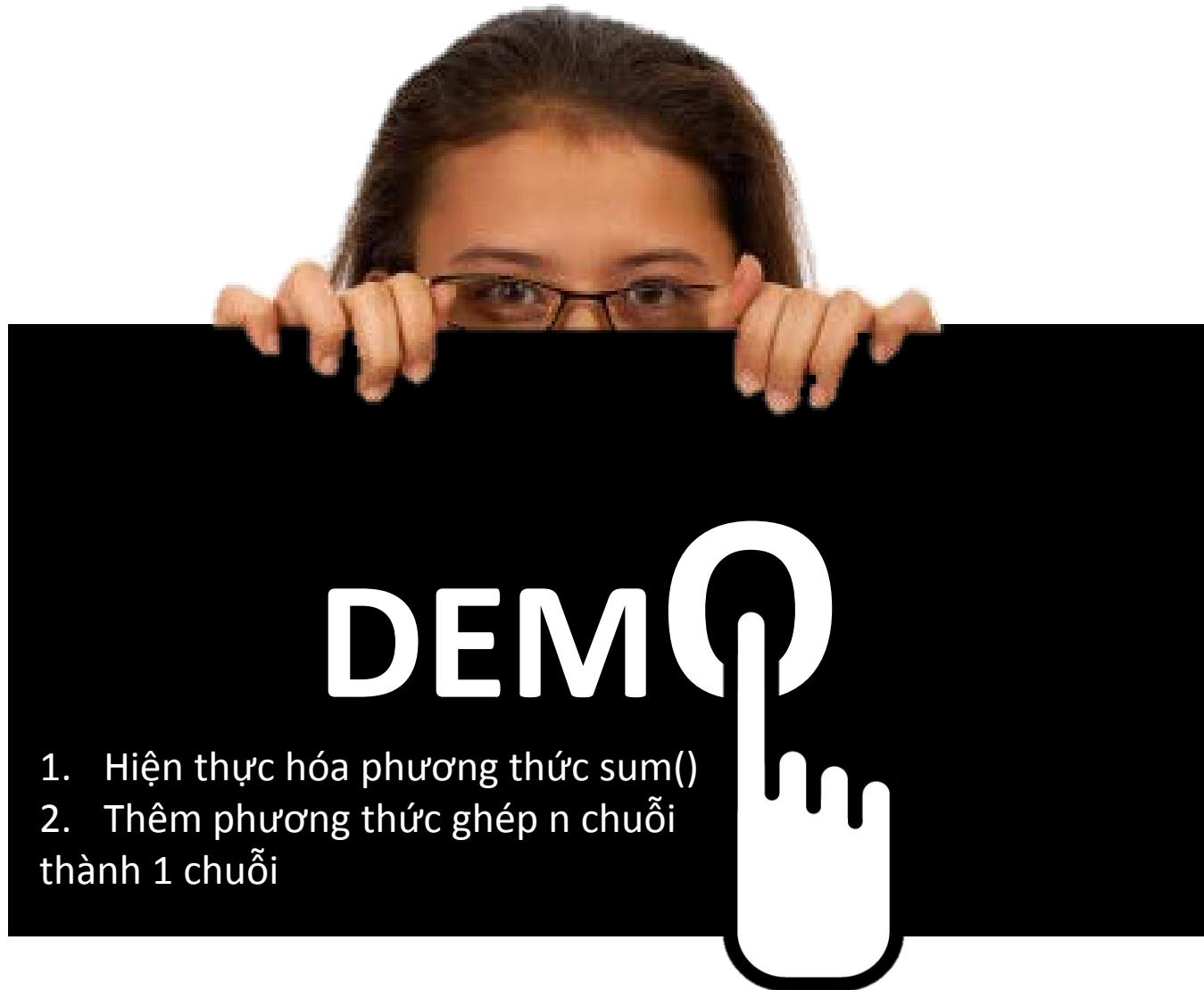
TRUYỀN THAM BIẾN ĐỔI (VARARGS)

- ❑ Bản chất của tham số biến đổi là mảng nhưng khi truyền tham số bạn có thể truyền vào nguyên mảng hoặc liệt kê các phần tử
- ❑ Trong một hàm, chỉ có thể khai báo **duy nhất** một tham số kiểu varargs và phải là tham số **cuối cùng**

```
int sum(int...x){  
    int s = 0;  
    for(int a : x){  
        s += a;  
    }  
    return s;  
}
```

```
int s1 = sum(2,7)
```

```
int s2 = sum(3,8,3,7,4)
```



1. Hiện thực hóa phương thức sum()
2. Thêm phương thức ghép n chuỗi thành 1 chuỗi

- ☐ Từ khóa static được sử dụng để định nghĩa cho khối và các thành viên tĩnh (**lớp nội, phương thức, trường**).

```
public class MyClass{  
    static public int X;  
    static{  
        X+=100;  
    }  
    static public void method(){  
        X+=200;  
    }  
    static class MyInnerClass{}  
}
```

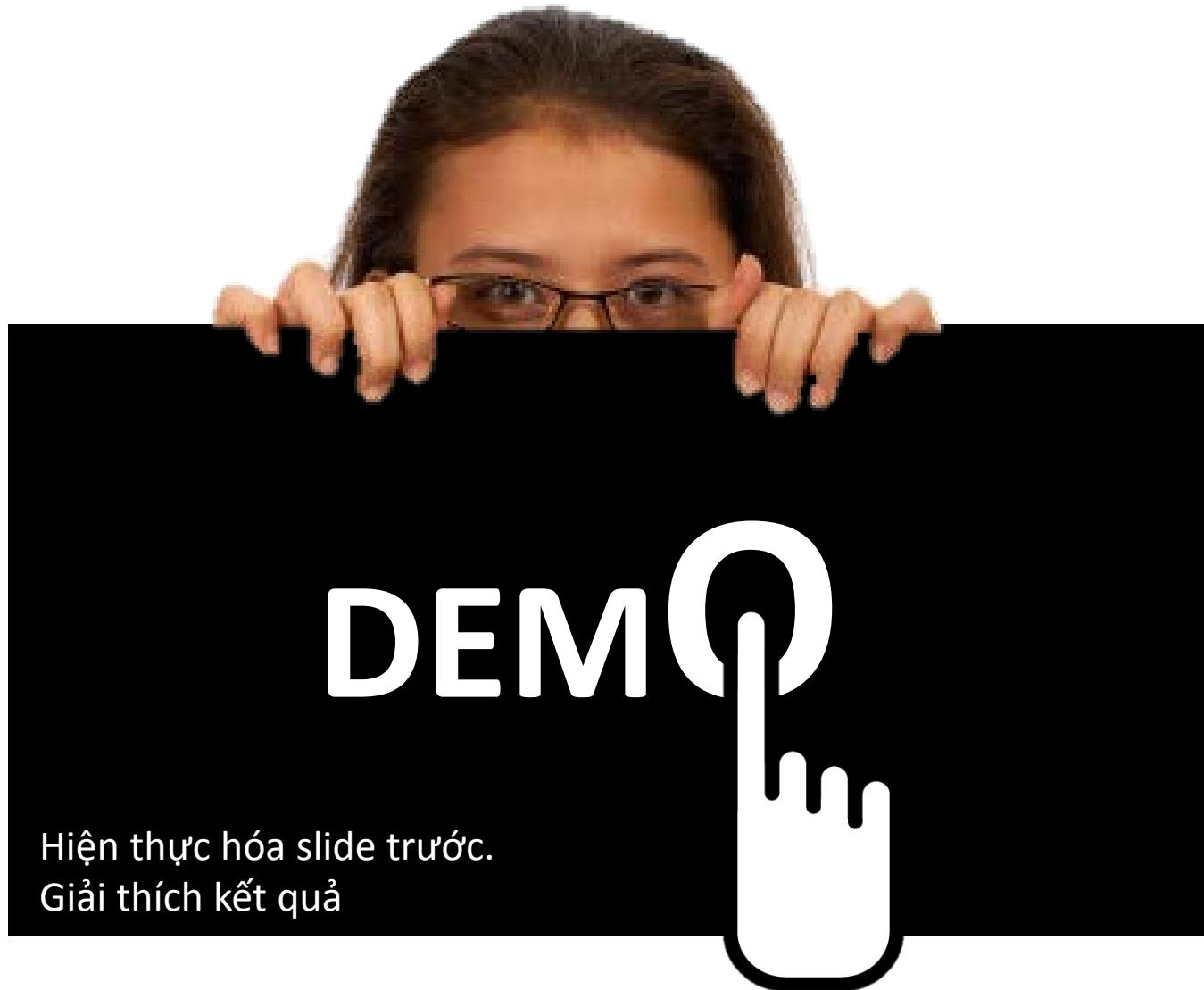
MyClass.X = 700;
MyClass.method()

- ❑ Khối static {} sẽ **chạy trước** khi tạo đối tượng hoặc truy xuất bất kỳ thành viên tĩnh khác
- ❑ Thành viên tĩnh của lớp được sử dụng **độc lập** với các đối tượng được tạo ra từ lớp đó.
- ❑ Có thể truy cập đến một thành viên tĩnh thông qua **tên lớp** mà không cần tham chiếu đến một đối tượng cụ thể
- ❑ Trường static là dữ liệu **dùng chung** cho tất cả các đối tượng được tạo ra từ lớp đó.
- ❑ Trong khối và phương thức tĩnh **chỉ được truy cập đến các thành viên tĩnh** khác mà không được phép truy cập đến thành viên thông thường của class

```
public class MyClass{  
    static public int X = 100;  
    static{  
        X+=100;  
    }  
    static public void method(){  
        X+=200;  
    }  
}
```

```
MyClass o = new MyClass();  
o.X += 300;  
MyClass.X += 500;  
MyClass.method()
```

MyClass.X, o.X có
giá trị là bao nhiêu



Hiện thực hóa slide trước.
Giải thích kết quả

❑ Trong Java có 3 loại hằng

- ❖ Lớp hằng là lớp không cho phép thừa kế
- ❖ Phương thức hằng là phương thức không cho phép ghi đè
- ❖ Biến hằng là biến không cho phép thay đổi giá trị

❑ Sử dụng từ khóa final để định nghĩa hằng

```
final public class MyFinalClass{...}
```

```
public class MyClass{  
    final public double PI = 3.14  
    final public void method(){...}  
}
```



CHỌN ĐOẠN MÃ ĐÚNG

A

```
final public class Parent{...}
```

```
public class Child extends Parent{  
...  
}
```

B

```
public class MyClass{  
final int PI = 3.14;  
public void method(){  
    PI = 3.1475;  
}  
}
```

C

```
public class Parent{  
    final public void method(){...}  
}
```

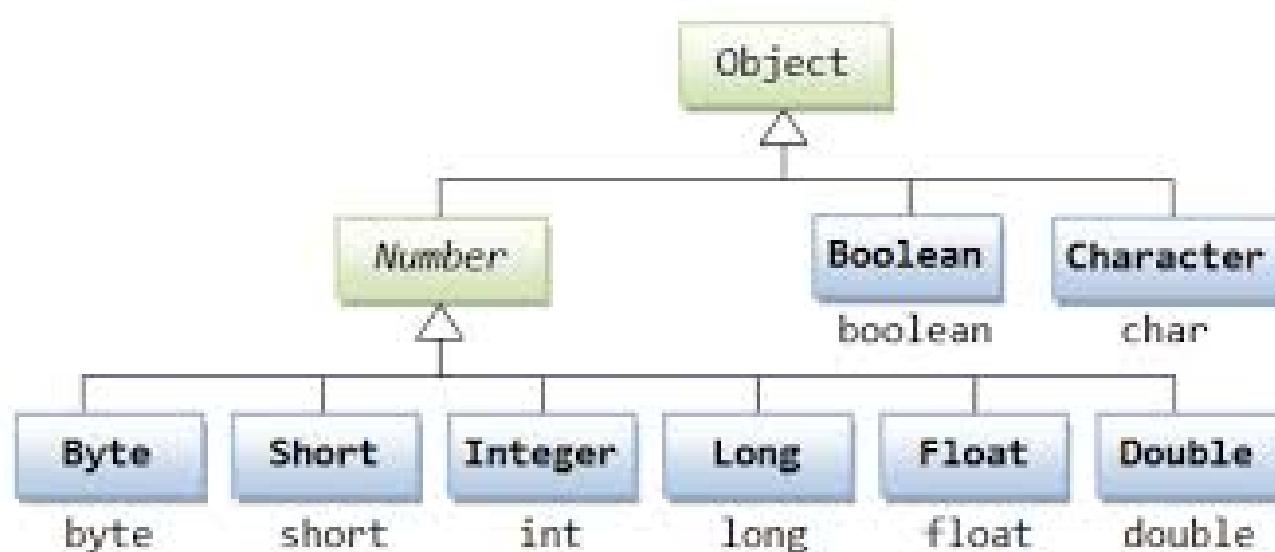
D

```
public class Parent{  
    public void method(){...}  
}
```

```
public class Child extends Parent{  
    public void method(){...}  
}
```

```
public class Child extends Parent{  
    public void method(){...}  
}
```

- ❑ Khi định nghĩa một lớp mà không kế thừa từ một lớp khác thì mặc định kế thừa lớp Object
- ❑ Như vậy mọi lớp đều có lớp cha chỉ duy nhất một lớp không có cha là Object



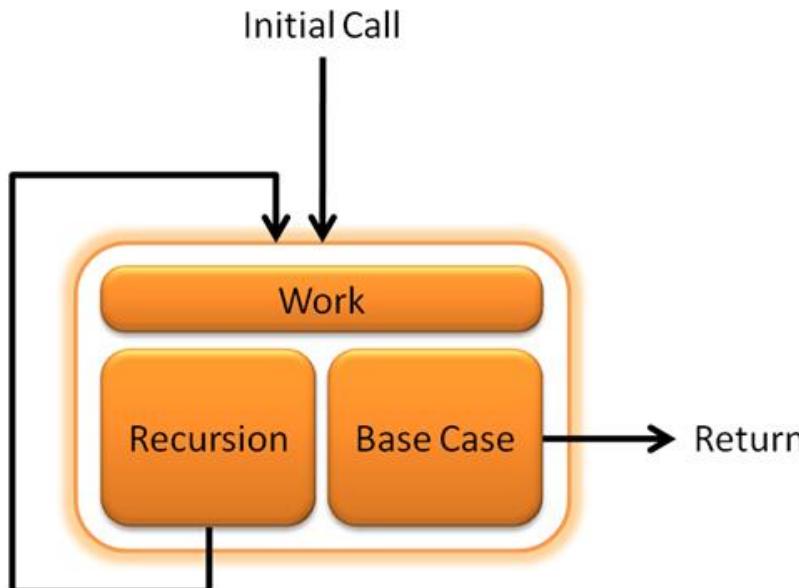
- ☐ Lớp nội là lớp được khai báo bên trong một lớp khác
- ☐ Có hai loại: lớp nội tĩnh và lớp nội thông thường
- ☐ Lớp bên trong chỉ có thể xác định trong phạm vi lớp ngoài cùng và có thể truy cập các thành viên của lớp bao nó

```
public class MyClass{  
    static public class MyInnerStaticClass{}  
    public class MyInnerClass{}  
}
```

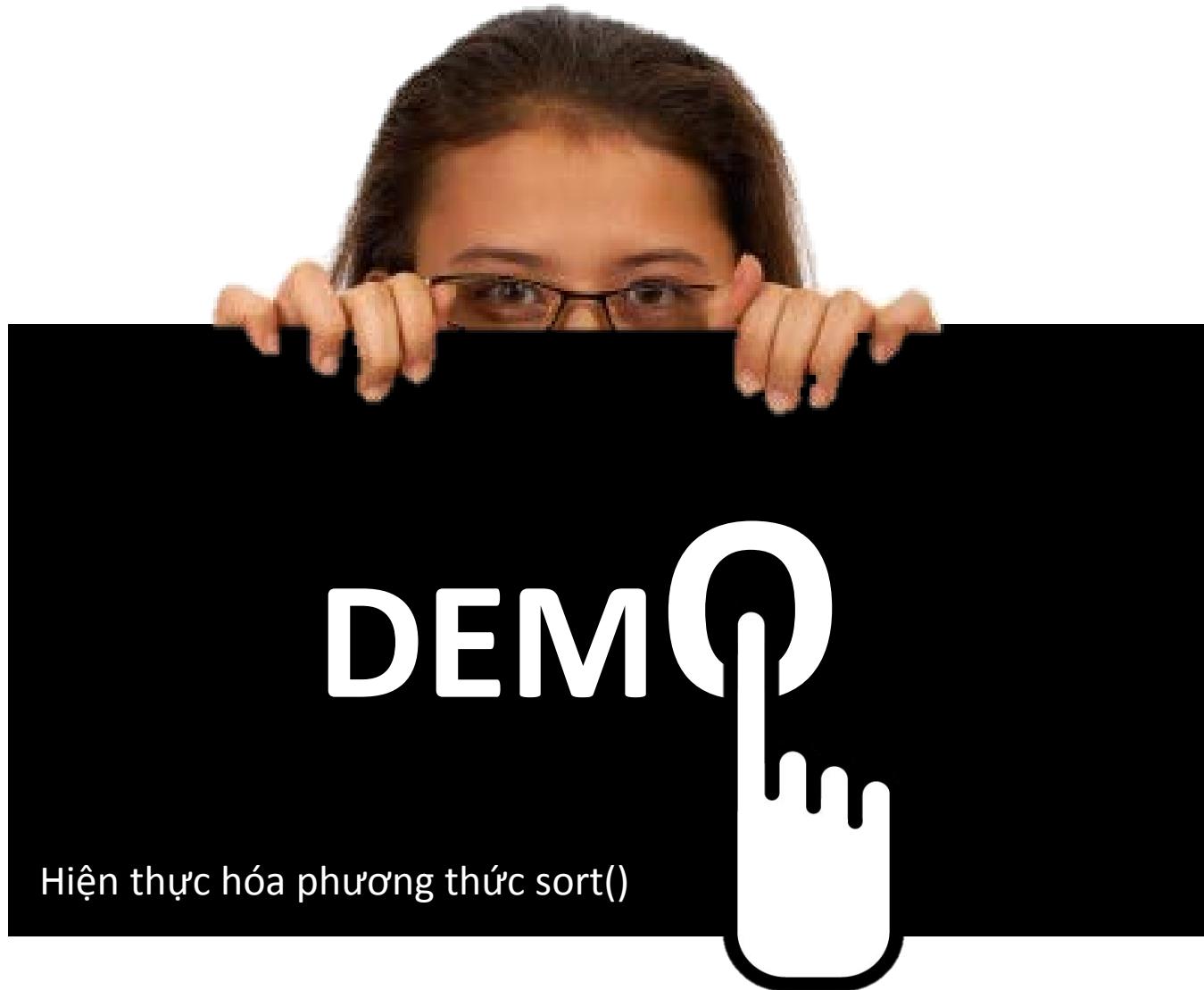
Sử dụng lớp nội

```
MyClass.MyInnerStaticClass x = new MyClass.MyInnerStaticClass();  
MyClass.MyInnerClass y = new MyClass().new MyInnerClass();
```

- ❑ Một phương thức gọi chính nó
- ❑ Phải có lệnh dừng đệ quy trong phương thức để tránh vòng lặp vô hạn
- ❑ Đệ qui dễ hiểu nhưng rất tốn tài nguyên



```
public void sort(int[] a, int i){  
    if(i >= a.length){  
        return;  
    }  
    for(int j = i + 1; j < a.length; j++){  
        if(a[i] < a[j]){  
            int tmp = a[i];  
            a[i] = a[j];  
            a[j] = tmp;  
        }  
    }  
    sort(a, i + 1);  
}
```



Hiện thực hóa phương thức sort()

TỔNG KẾT NỘI DUNG BÀI HỌC

- Tìm hiểu sâu về constructor
- Phân loại tham số
- Tham số biến đổi
- Sử dụng static
- Định nghĩa hằng
- Lớp nội
- Đệ quy