Bài 5

Một số kỹ thuật Java nâng cao

Nội dung

1. Lớp bao
2. Các hàm toán học
3. Các kỹ thuật thao tác với chuỗi
4. Quản lý bộ nhớ trong Java
5. So sánh đối tượng
6. Truyền tham số cho phương thức

1

Lớp bao

Wrapper class

Lớp bao

- Các kiểu dữ liệu nguyên thủy không có các phương thức liên quan đến nó.
  - Mỗi kiểu dữ liệu nguyên thủy có một lớp tương ứng gọi là lớp bao (wrapper class)
  - Các lớp bao sẽ "gói" dữ liệu nguyên thủy và cung cấp các phương thức thích hợp cho dữ liệu đó.
  - Mỗi đối tượng của lớp bao đơn giản là lưu trữ một biến đơn và đưa ra các phương thức để xử lý nó.
  - Các lớp bao là một phần của Java API

Các lớp bao

Kiểu dữ liệu nguyên thủy	*Lớp bao*
boolean	java.lang.Boolean
byte	java.lang.Byte
char	java.lang.Char
double	java.lang.Double
float	java.lang.Float
int	java.lang.Integer
long	java.lang.Long
short	java.lang.Short

Lớp bao

- Các lớp bao là không thay đổi được (immutable)
  - Sau khi đã được gán một giá trị, thể hiện của lớp đó không được phép thay đổi giá trị nữa.
- Các lớp bao là final
  - Không thể kế thừa từ các lớp bao
- Tất cả các phương thức của các lớp bao là static
- Tất cả các lớp bao trừ Boolean và Character là kế thừa từ lớp Number
  - Boolean và Character kế thừa trực tiếp từ lớp Object

Khởi tạo đối tượng lớp bao

- Tất cả các lớp bao cung cấp 2 phương thức khởi tạo (trừ lớp Character):
  - Một phương thức lấy tham số là kiểu dữ liệu nguyên thủy của giá trị cần khởi tạo
  - Một phương thức lấy biểu diễn kiểu String của giá trị đó

```
Float wfloat = new Float("12.34f");
Float yfloat = new Float(12.34f);
Boolean wbool = new Boolean("false");
Boolean ybool =new Boolean(false);
Character c1 = new Character('c');
```

Chuyển đổi từ String sang các đối tượng lớp bao

- Cách khác để tạo các đối tượng của lớp bao: sử dụng các phương thức static
  - `valueOf(String s)`
  - `valueOf(String s, int radix)`
  - Trả về các đối tượng có giá trị bằng tham số được đưa vào

```
Integer i2 = Integer.valueOf("101011", 2);
// converts 101011 to 43 and assigns the
// value 43 to the Integer object i2

Float f2 = Float.valueOf("3.14f");
// assigns 3.14 to the Float object f2
```

Trả về biến của kiểu dữ liệu nguyên thủy

- Để trả về các kiểu dữ liệu nguyên thủy: sử dụng phương thức `typeValue()`
  - Không có tham số
  - Mỗi kiểu số có 6 phương thức typeValue() tương ứng
  - Ví dụ

```
// make a new wrapper object
Integer i2 = new Integer(42);
// convert i2's value to a byte primitive
byte b = i2.byteValue();
// another of Integer's xxxValue methods
short s = i2.shortValue();
// yet another of Integer's xxxValue methods
double d = i2.doubleValue();
```

Chuyển đổi từ String sang các kiểu dữ liệu nguyên thủy

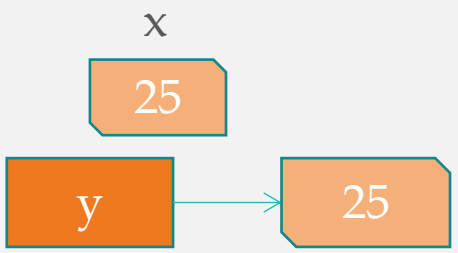
- Dùng các phương thức static của lớp bao
  `static <type> parseType(String s)`
- Ví dụ

```
String s = "123";
//assign an int value of 123 to the int variable i
int i = Integer.parseInt(s);
//assign an short value of 123 to the short variable j
short j = Short.parseShort(s)
```

Phân biệt kiểu dữ liệu nguyên thủy và lớp bao

- Ví dụ

```
int x = 25;
Integer y = new Integer(25);

int z = x + y; // ERROR
int z = x + y.intValue(); // OK!
```



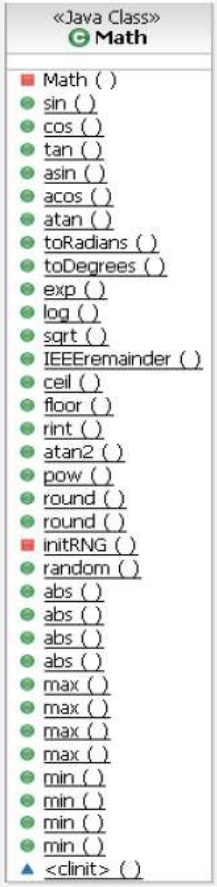
2

Các hàm toán học

Lớp Math

Lớp Math

- java.lang.Math cung cấp các thành phần static:
  - Các hằng toán học:
    + Math.E
    + Math.PI
  - Các hàm toán học:
    + max, min...
    + abs, floor, ceil...
    + sqrt, pow, log, exp...
    + cos, sin, tan, acos, asin, atan...
    + random



Lớp Math

- Hầu hết các hàm nhận tham số kiểu double và giá trị trả về cũng có kiểu double
  - Ví dụ :

$$e^{\sqrt{2\pi}}$$

```
Math.pow(Math.E, Math.sqrt(2.0*Math.PI))
```

hoặc

```
Math.exp(Math.sqrt(2.0*Math.PI))
```

3

Các kỹ thuật thao tác với chuỗi

String và StringBuffer

Xâu (String)

- Kiểu String là một lớp và không phải là kiểu dữ liệu nguyên thủy
- Một String được tạo thành từ một dãy các ký tự nằm trong dấu nháy kép:

```
String a = "A String";
String b = "";
```

- Đối tượng String có thể khởi tạo theo nhiều cách:

```
String c = new String();
String d = new String("Another String");
String e = String.valueOf(1.23);
String f = null;
```

Ghép xâu

- Toán tử + có thể nối các String:

```
String a = "This" + " is a " + "String";
//a = “This is a String”
```

- Các kiểu dữ liệu cơ bản sử dụng trong lời gọi println() được chuyển đổi tự động sang kiểu String

```
System.out.println("answer = " + 1 + 2 + 3);
System.out.println("answer = " + (1 + 2 + 3));
```

Các phương thức của xâu

```
String name = "Joe Smith";
name.toLowerCase();          // "joe smith"
name.toUpperCase();          // "JOE SMITH"
"Joe Smith ".trim();         // "Joe Smith"
"Joe Smith".indexOf('e');    // 2
"Joe Smith".length();        // 9
"Joe Smith".charAt(5);       // 'm'
"Joe Smith".substring(5);    // "mith"
"Joe Smith".substring(2,5); // "e S"
```

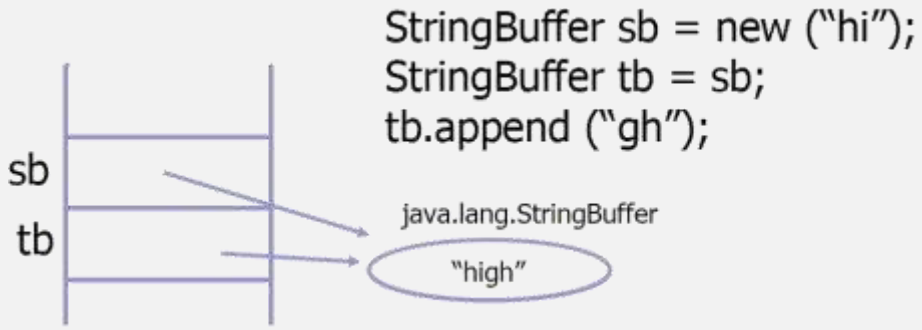
StringBuffer

- String là kiểu bất biến
  - Đối tượng không thay đổi giá trị sau khi được tạo ra → Các xâu của lớp String được thiết kế để không thay đổi giá trị.
  - Khi các xâu được ghép nối với nhau một đối tượng mới được tạo ra để lưu trữ kết quả → Ghép nối xâu thông thường rất tốn kém về bộ nhớ.

```
String s = "";
s += "Hello";
s += " ";
s += "World";
s += "!";
System.out.println(s);
```

StringBuffer

- Trong trường hợp phải làm việc với các xâu biến đổi → Sử dụng StringBuffer
  - Dự đoán các ký tự trong xâu có thể thay đổi.
  - Khi xử lý các xâu một cách linh động, ví dụ như đọc dữ liệu text từ một tệp tin.
- Cung cấp các cơ chế hiệu quả hơn cho việc xây dựng, ghép nối các xâu:
  - Việc ghép nối xâu thường được các trình biên dịch chuyển sang thực thi trong lớp StringBuffer



Các phương thức của StringBuffer

```
StringBuffer buffer = new StringBuffer(15);
buffer.append("This is ") ;
buffer.append("String") ;
buffer.insert(7," a") ;
buffer.append('.');
System.out.println(buffer.length());     // 17
System.out.println(buffer.capacity());   // 32
String output = buffer.toString() ;
System.out.println(output); // "This is a String."
```

4

Quản lý bộ nhớ trong Java

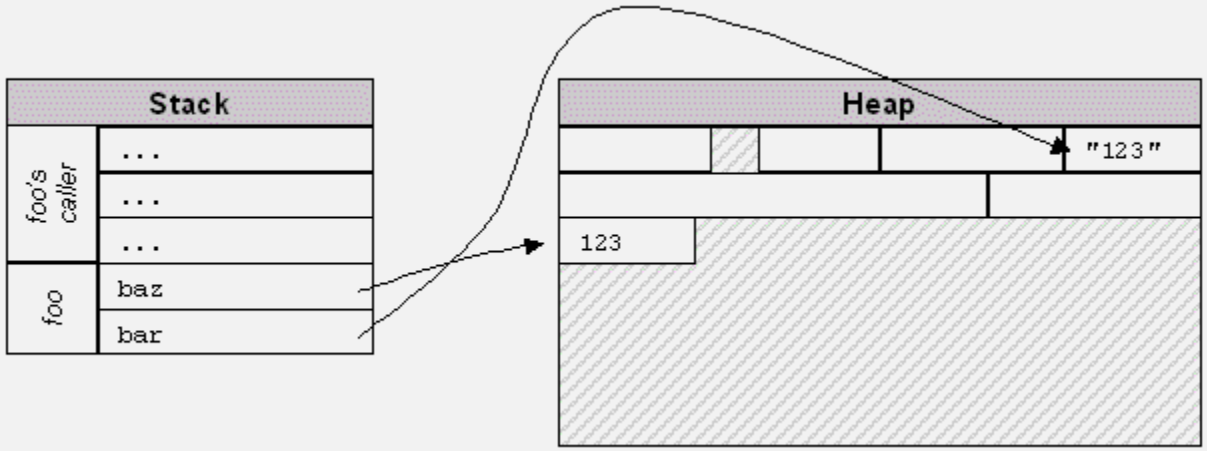
Định vị, tái định vị và quản lý bộ nhớ

Quản lý bộ nhớ trong Java

- Java không sử dụng con trỏ nên các địa chỉ bộ nhớ không thể bị ghi đè lên một cách ngẫu nhiên hoặc cố ý.
- Các vấn đề định vị và tái định vị bộ nhớ, quản lý bộ nhớ do JVM kiểm soát, hoàn toàn trong suốt (transparent) với lập trình viên.
- Lập trình viên không cần quan tâm đến việc ghi dấu các phần bộ nhớ đã cấp phát tđể giải phóng sau này.

Các loại bộ nhớ trong Java

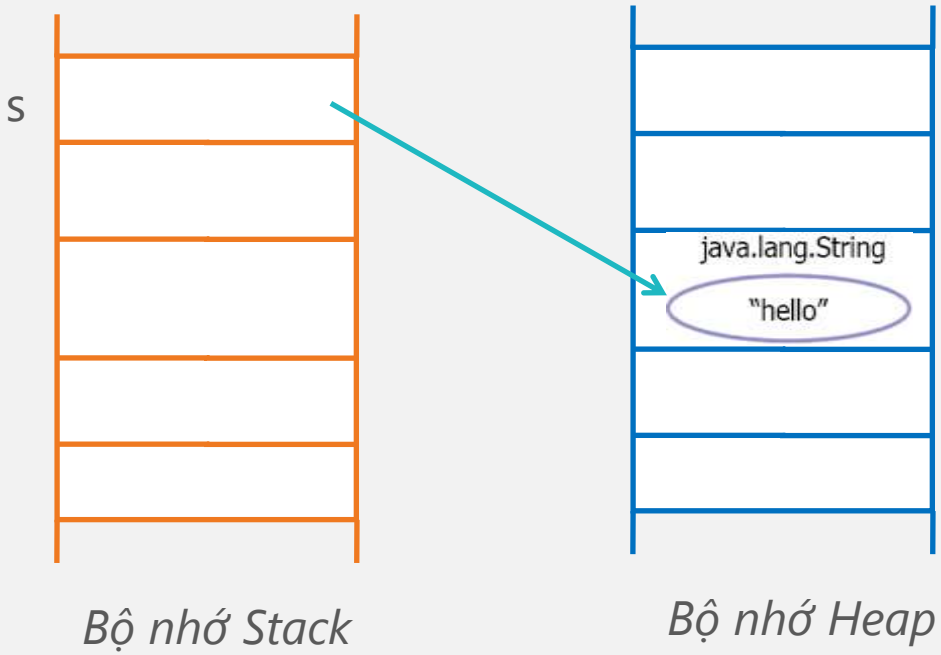
- Trong Java có hai loại bộ nhớ chính
  - Bộ nhớ heap: lưu trữ các dữ liệu được cấp phát cho các tham chiếu
  - Bộ nhớ stack: lưu trữ các tham chiếu (~địa chỉ các con trỏ) và các dữ liệu nguyên thủy



Bộ nhớ Stack

- Bộ nhớ Heap sử dụng để ghi thông tin được tạo bởi toán tử new

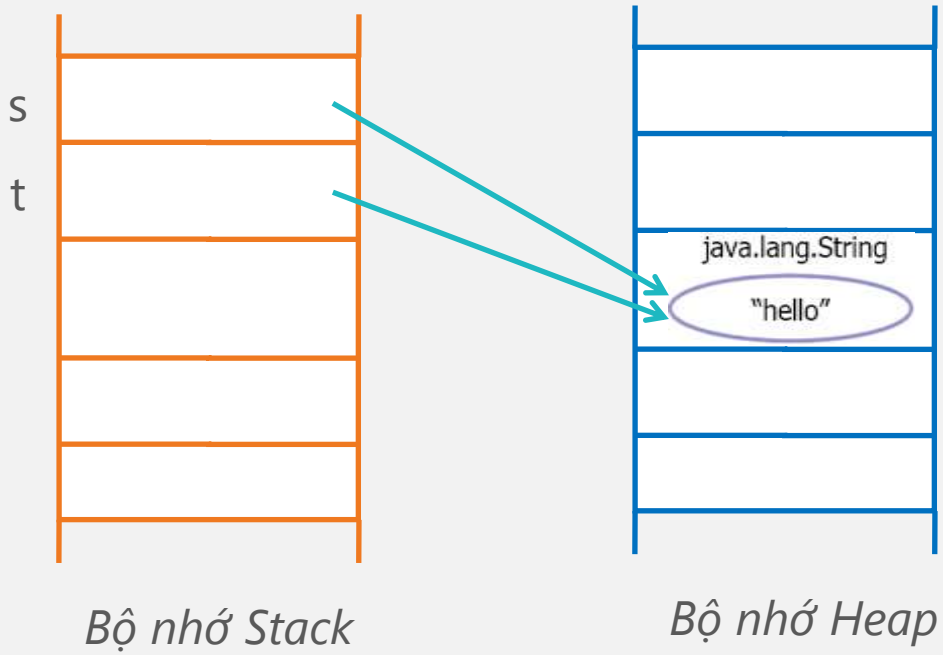
```
String s = new String("hello");
```



Bộ nhớ Heap

- Giá trị cục bộ trong bộ nhớ Stack được sử dụng như con trỏ tham chiếu tới Heap

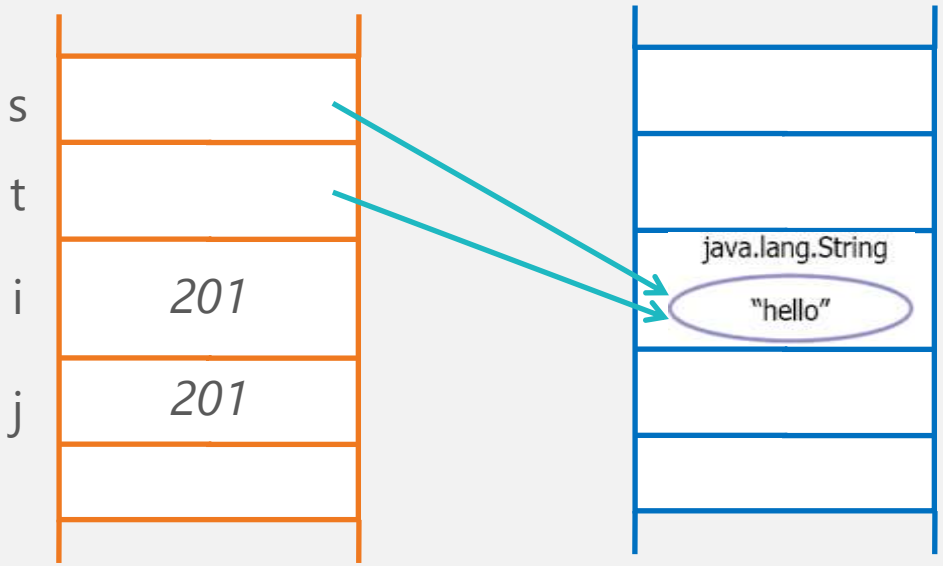
```
String s = new String("hello");
String t = s;
```



Bộ nhớ Heap

- Giá trị của dữ liệu nguyên thủy được ghi trực tiếp trong Stack

```
String s = new String("hello");
String t = s;
int i = 201;
int j = i;
```



Bộ thu gom rác

- Một tiến trình chạy ngầm gọi đến bộ "thu gom rác" (garbage collector) để phục hồi lại phần bộ nhớ mà các đối tượng không tham chiếu đến (tái định vị)
- Các đối tượng không có tham chiếu đến được gán null.
- Bộ thu gom rác định kỳ quét qua danh sách các đối tượng của JVM và phục hồi các tài nguyên của các đối tượng không có tham chiếu.

Bộ thu gom rác

- JVM quyết định khi nào thực hiện thu gom rác:
  - Thông thường sẽ thực thi khi thiếu bộ nhớ
  - Tại thời điểm không dự đoán trước
- Không thể ngăn quá trình thực hiện của bộ thu gom rác nhưng có thể yêu cầu thực hiện sớm hơn:

```
System.gc(); hoặc Runtime.gc();
```

Phương thức void finalize()

- Lớp nào cũng có phương thức finalize() – được thực thi ngay lập tức khi quá trình thu gom xảy ra
- Thường chỉ sử dụng cho các trường hợp đặc biệt để "tự dọn dẹp" các tài nguyên sử dụng khi đối tượng được gc giải phóng
  - Ví dụ cần đóng các socket, file,... nên được xử lý trong luồng chính trước khi các đối tượng bị ngắt bỏ tham chiếu.
- Tương tự với hàm hủy (destructor) của lớp trong C++

5

So sánh đối tượng

So sánh đối tượng

- Đối với các kiểu dữ liệu nguyên thủy, toán tử == kiểm tra xem chúng có giá trị bằng nhau hay không
- Ví dụ

```
int a = 1;
int b = 1;
if (a==b)... // true
```

So sánh đối tượng

- Đối với các đối tượng, toán tử == kiểm tra xem hai đối tượng có đồng nhất hay không, có cùng tham chiếu đến một đối tượng hay không.
- Ví dụ

```
Employee a = new Employee(1);
Employee b = new Employee(1);
if (a==b)... // false

Employee a = new Employee(1);
Employee b = a;
if (a==b)... // true
```

Phương thức equals

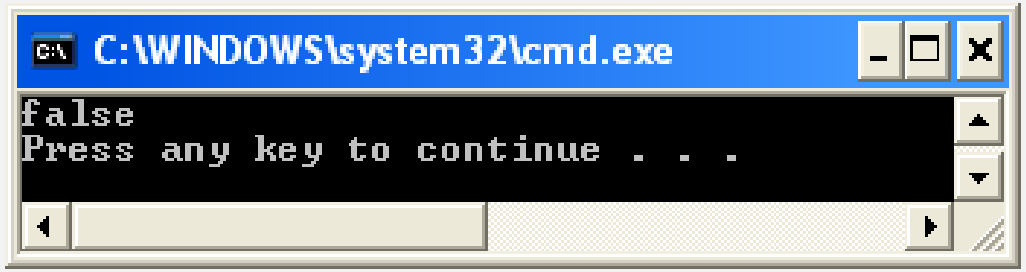
- Để so sánh giá trị của hai đối tượng → dùng phương thức equals
  - Tất cả các lớp có sẵn trong Java đều có phương thức này
- Không sử dụng được đối với các kiểu dữ liệu nguyên thủy

Ví dụ: equals trong Lớp Integer

```
public class Equivalence {
     public static void main(String[] args) {
             Integer n1 = new Integer(47);
             Integer n2 = new Integer(47);
             System.out.println(n1 == n2);
             System.out.println(n1.equals(n2));
     }
}
```

equals của lớp tự viết

```
class Value {
     int i;
}
public class EqualsMethod2 {
     public static void main(String[] args) {
     Value v1 = new Value();
     Value v2 = new Value();
     v1.i = v2.i = 100;
     System.out.println(v1.equals(v2));
     }
}
```



So sánh hai xâu

- oneString.equals(anotherString)
  - Kiểm tra tính tương đương
  - Trả về true hoặc false

```
String name = "Joe";
if ("Joe".equals(name))
     name += " Smith";
```

- oneString.equalsIgnoreCase(anotherString)
  - Kiểm tra KHÔNG xét đến ký tự hoa, thường

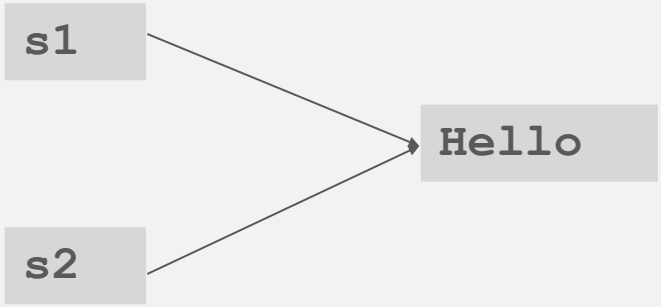
```
boolean same = "Joe".equalsIgnoreCase("joe");
```

- So sánh `oneString == anotherString` sẽ gây nhập nhằng với so sánh 2 đối tượng

So sánh hai xâu

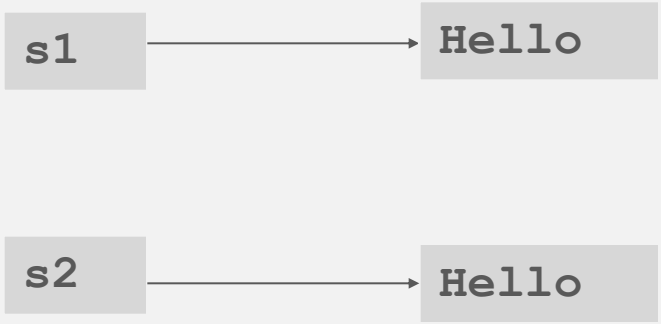
```
String s1 = new String(“Hello”);
String s2 = s1;
```

+ (s1==s2) trả về true




```
String s1 = new String(“Hello”);
String s2 = new String(“Hello”);
```

+ (s1==s2) trả về false



6

Truyền tham số cho phương thức

Truyền tham số

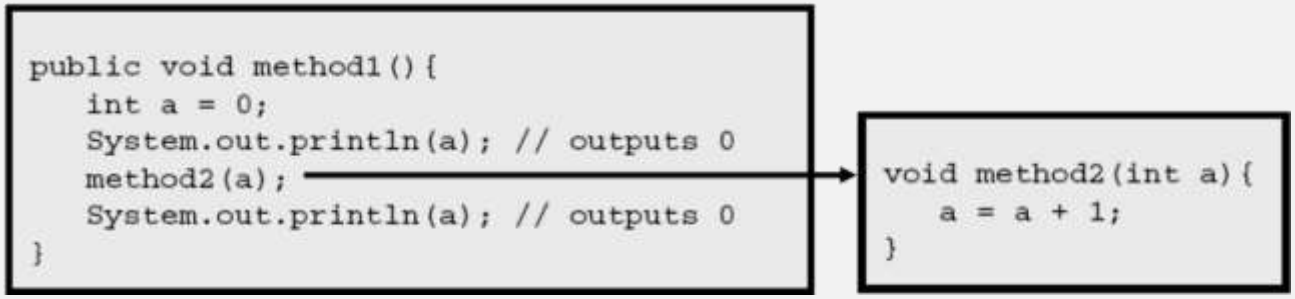
- Trong C/C++: có nhiều cách truyền tham số
  - Truyền theo tham trị (pass-by-value), hay còn gọi là truyền giá trị
  - Truyền theo tham chiếu (pass-by-reference), hay còn gọi là truyền địa chỉ
  - Truyền con trỏ
- Trong Java:
  - Chỉ có truyền theo tham trị (pass-by-value)

Truyền theo tham trị

- Truyền giá trị/bản sao của tham số thực
  - Với tham số có kiểu dữ liệu tham trị (kiểu dữ liệu nguyên thủy): Truyền giá trị/bản sao của các biến nguyên thủy truyền vào
  - Với tham số có kiểu dữ liệu tham chiếu (mảng và đối tượng): Truyền giá trị/bản sao của tham chiếu gốc truyền vào
- Thay đổi tham số hình thức không làm ảnh hưởng đến tham số thực

Với kiểu dữ liệu nguyên thủy

- Các giá trị nguyên thủy không thể thay đổi khi truyền như một tham số

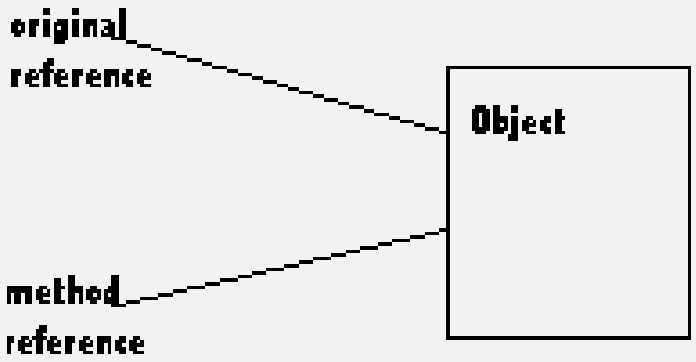


- Phương thức swap này có hoạt động đúng không?

```
public void swap(int var1, int var2) {
  int temp = var1;
  var1 = var2;
  var2 = temp;
}
```

Với kiểu dữ liệu tham chiếu

- Thực ra là truyền bản sao của tham chiếu gốc, chứ không phải truyền tham chiếu gốc hoặc truyền đối tượng (pass the references by value, not the original reference or the object)



- Sau khi truyền cho phương thức, đối tượng có ít nhất 2 tham chiếu

Ví dụ

```
public class Point {
  private double x;
  private double y;
  public Point() { }
    public Point(double x, double y) {
      this.x = x; this.y = y;
    }
  public void setX(double x) { this.x = x; }
  public void setY(double y) { this.y = y; }
  public void printPoint() {
    System.out.println("X: " + x + " Y: " + y);
  }
}
```

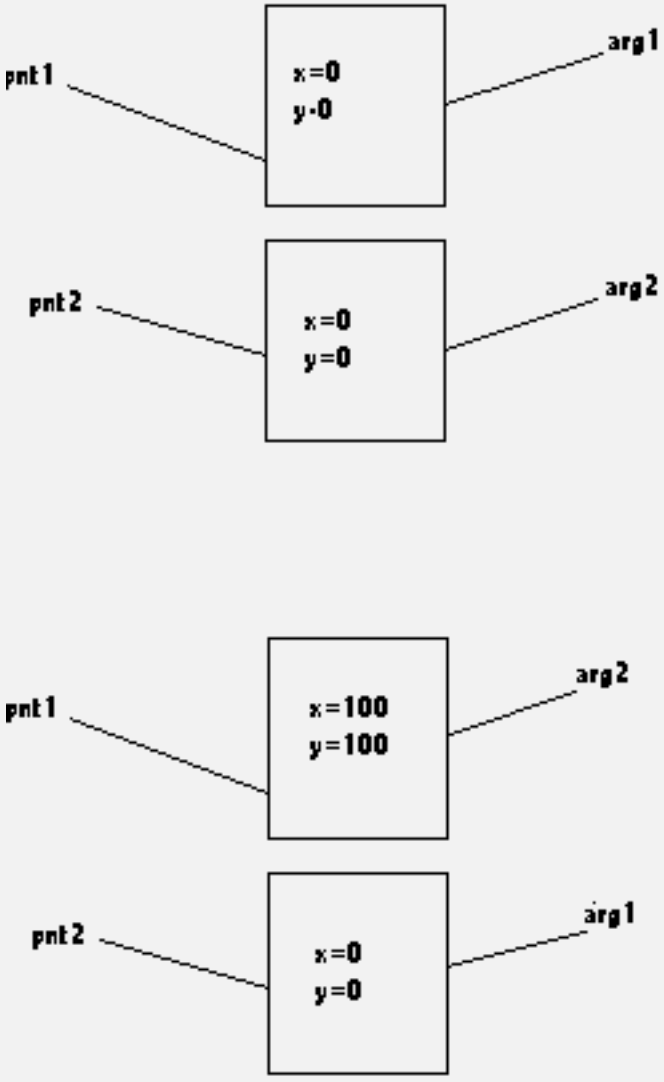
Ví dụ (tiếp)

```
public class Test {
  public static void swapDemo(Point arg1,
                              Point arg2) {
    arg1.setX(100); arg1.setY(100);
    Point temp = arg1;
    arg1 = arg2; arg2 = temp;
  }

  public static void main(String [] args) {
    Point pnt1 = new Point(0,0);
    Point pnt2 = new Point(0,0);
    pnt1.printPoint(); pnt2.printPoint();
    System.out.println();
    swapDemo(pnt1, pnt2);
    pnt1.printPoint(); pnt2.printPoint();
  }
}
```

```
X: 0.0 Y: 0.0
X: 0.0 Y: 0.0

X: 100.0 Y: 100.0
X: 0.0 Y: 0.0
```

Hoán đổi tham chiếu

- Chỉ có các tham chiếu của phương thức được tráo đổi, chứ không phải các tham chiếu gốc



Thank you!

Any questions?