

**BỘ MÔN CÔNG NGHỆ PHẦN MỀM**  
**VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**  
**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

# **LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG**

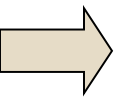
## **Bài 04. Các kỹ thuật xây dựng lớp và sử dụng đối tượng**

Nguyễn Thị Thu Trang  
[trangntt-fit@mail.hut.edu.vn](mailto:trangntt-fit@mail.hut.edu.vn)

# Mục tiêu bài học

- Nêu được bản chất, vai trò và biết sử dụng kỹ thuật chồng phương thức, chồng phương thức khởi tạo
- Thành viên đối tượng, thành viên lớp
- Hiểu về cách thức quản lý bộ nhớ và đối tượng trong Java
- Nắm về cách thức truyền tham số phương thức
- Biết cách sử dụng package, một số lớp tiện ích trong Java: Wrapper class, Math, System, String vs. StringBuffer

# Nội dung



1. Chồng phương thức
2. Thành viên ĐT và thành viên lớp
3. Quản lý bộ nhớ trong Java
4. Truyền tham số cho phương thức
5. Một số lớp tiện ích trong Java

# Nhắc lại về phương thức

- Mỗi phương thức phải có một chữ ký riêng
- Chữ ký của phương thức bao gồm:
  - Tên phương thức
  - Số lượng các đối số và kiểu của chúng

The diagram illustrates the components of a method signature in a Java code snippet. The code is: `public void credit(double amount) {`  
...  
`}`  
Annotations with arrows point to specific parts of the signature:   
- 'method name' points to 'credit' (highlighted in light blue).  
- 'argument type' points to 'double' (highlighted in light blue).  
- 'signature' points to the entire 'credit(double amount)' part, which is enclosed in a brown rectangular box.

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

# 1.1. Chồng phương thức

- Chồng phương thức (Method Overloading): Các phương thức **trong cùng một lớp** có thể trùng tên nhưng **chữ ký phải khác nhau**:
  - Số lượng tham số khác nhau
  - Nếu cùng số lượng tham số thì kiểu dữ liệu các tham số phải khác nhau
- Mục đích:
  - Tên trùng nhau để mô tả bản chất công việc
  - Thuận tiện cho lập trình vì không cần phải nhớ quá nhiều tên phương thức mà chỉ cần nhớ một tên và lựa chọn các tham số cho phù hợp.

## 1.1. Chồng phương thức (2)

- Ví dụ 1:
  - Phương thức `println()` trong `System.out.println()` có 10 khai báo với các tham số khác nhau: `boolean`, `char[]`, `char`, `double`, `float`, `int`, `long`, `Object`, `String`, và một không có tham số.
  - Không cần sử dụng các tên khác nhau (chẳng hạn `"printString"` hoặc `"printDouble"`) cho mỗi kiểu dữ liệu muốn hiển thị.

# 1.1. Chồng phương thức (3)

- Ví dụ 2:

```
class MyDate {  
    int year, month, day;  
    public boolean setMonth(int m) { ...}  
    public boolean setMonth(String s) { ...}  
}  
  
public class Test{  
    public static void main(String args[]){  
        MyDate d = new MyDate();  
        d.setMonth(9);  
        d.setMonth("September");  
    }  
}
```

# Một số chú ý với chồng phương thức

- Các phương thức chỉ được xem xét là chồng khi chúng thuộc cùng một lớp
- Chỉ nên sử dụng kỹ thuật này với các phương thức có cùng mục đích, chức năng; tránh lạm dụng
- Khi dịch, trình dịch căn cứ vào số lượng hoặc kiểu dữ liệu của tham số để quyết định gọi phương thức nào phù hợp.
  - ◇ Nếu không chọn được hoặc chọn được nhiều hơn 1 phương thức thì sẽ báo lỗi.



# Thảo luận

- Cho phương thức sau đây:  
**public double test(String a, int b)**
- Hãy chọn ra các phương thức chồng cho phương thức trên:
  1. **void test(String b, int a)**
  2. **public double test(String a)**
  3. **private int test(int b, String a)**
  4. **private int test(String a, int b)**
  5. **double test(double a, int b)**
  6. **double test(int b)**
  7. **public double test(String a, long b)**

# Thảo luận


```
void prt(String s) { System.out.println(s); }  
void f1(char x) { prt("f1(char)"); }  
void f1(byte x) { prt("f1(byte)"); }  
void f1(short x) { prt("f1(short)"); }  
void f1(int x) { prt("f1(int)"); }  
void f1(long x) { prt("f1(long)"); }  
void f1(float x) { prt("f1(float)"); }  
void f1(double x) { prt("f1(double)"); }
```

- Điều gì xảy ra nếu thực hiện:

- `f1(5);`
- `char x='a'; f1(x);`
- `byte y=0; f1(y);`
- `float z = 0; f1(z);...`



5 → int



```
f1(int)  
f1(char)  
f1(byte)  
f1(float)
```

# Thảo luận

```
void prt(String s) { System.out.println(s); }  
void f2(short x) { prt("f3(short)"); }  
void f2(int x) { prt("f3(int)"); }  
void f2(long x) { prt("f5(long)"); }  
void f2(float x) { prt("f5(float)"); }
```

- Điều gì xảy ra nếu thực hiện:

- `f2(5);`
- `char x='a'; f2(x);`
- `byte y=0; f2(y);`
- `float z = 0; f2(z);`



```
f3(int)  
f3(int)  
f3(short)  
f5(float)
```

- Điều gì xảy ra nếu gọi `f2(5.5)?`

Error: cannot find symbol: method f2(double)

## 1.2. Chồng phương thức khởi tạo

- Trong nhiều tình huống khác nhau cần khởi tạo đối tượng theo nhiều cách khác nhau
- → Cần xây dựng các phương thức khởi tạo khác nhau cho đối tượng theo nguyên lý chồng phương thức (constructor overloading).

# Ví dụ

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

public class Test{
    public static void main(String args[]){
        BankAccount acc1 = new BankAccount();
        BankAccount acc2 =
            new BankAccount("Thuy", 100);
    }
}
```

## 1.3. Từ khóa this

- Nhắc lại: Tự tham chiếu đến đối tượng hiện tại, sử dụng bên trong lớp tương ứng với đối tượng muốn tham chiếu.
- Sử dụng thuộc tính hoặc phương thức của đối tượng thông qua toán tử “.”, ví dụ:

```
public class BankAccount{  
    private String owner;  
    public void setOwner(String owner){  
        this.owner = owner;  
    }  
    public BankAccount() { this.setOwner("noname"); }  
    ...  
}
```

- Gọi đến phương thức khởi tạo khác của lớp:
  - `this(danh_sach_tham_so); //neu co tham so`

- Ví dụ

```
public class Ship {  
    private double x=0.0, y=0.0  
    private double speed=1.0, direction=0.0;  
    public String name;  
  
    public Ship(String name) {  
        this.name = name;  
    }  
    public Ship(String name, double x, double y) {  
        this(name); this.x = x; this.y = y;  
    }  
    public Ship(String name, double x, double y,  
        double speed, double direction) {  
        this(name, x, y);  
        this.speed = speed;  
        this.direction = direction;  
    }  
    //continue...
```

```
//(cont.)
private double degreeToRadian(double degrees) {
    return(degrees * Math.PI / 180.0);
}
public void move() {
    move(1);
}
public void move(int steps) {
    double angle = degreesToRadians(direction);
    x = x + (double)steps*speed*Math.cos(angle);
    y = y + (double)steps*speed*Math.sin(angle);
}
public void printLocation() {
    System.out.println(name + " is at ("
                        + x + "," + y + ").");
}
} //end of Ship class
```



# Nội dung

1. Chồng phương thức
- 2. Thành viên ĐT và thành viên lớp
3. Quản lý bộ nhớ trong Java
4. Truyền tham số cho phương thức
5. Một số lớp tiện ích trong Java

## Thành viên đối tượng (Instance member)

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

## vs. Thành viên lớp (Class member)

Thuộc tính/phương thức có thể được truy cập thông qua **lớp**

Các đối tượng có **chung 1 bản sao** của 1 thuộc tính **lớp**

**Giá trị** của 1 thuộc tính lớp của các **đối tượng khác nhau là giống nhau.**

## 2.1. Thành viên static

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

- Ví dụ:

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

# Ví dụ lớp JOptionPane trong javax.swing

- Thuộc tính

Field Summary	
static int	<a href="#">CANCEL_OPTION</a> Return value from class method if CANCEL is chosen.
static int	<a href="#">CLOSED_OPTION</a> Return value from class method if user closes window CANCEL_OPTION or NO_OPTION.
static int	<a href="#">DEFAULT_OPTION</a> Type used for showConfirmDialog.
static int	<a href="#">ERROR_MESSAGE</a> Used for error messages.

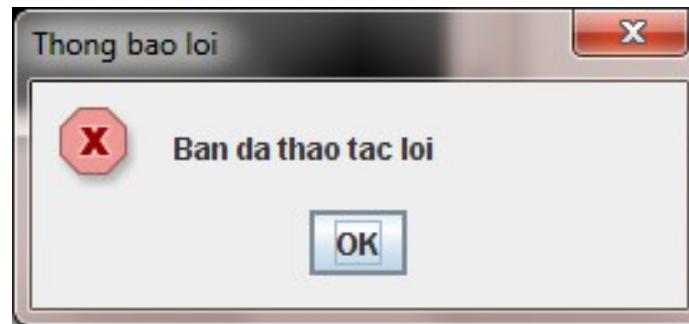
static int	<a href="#">WARNING_MESSAGE</a> Used for warning messages.
static int	<a href="#">YES_NO_CANCEL_OPTION</a> Type used for showConfirmDialog.
static int	<a href="#">YES_NO_OPTION</a> Type used for showConfirmDialog.
static int	<a href="#">YES_OPTION</a> Return value from class method if YES is chosen.

- Phương thức:

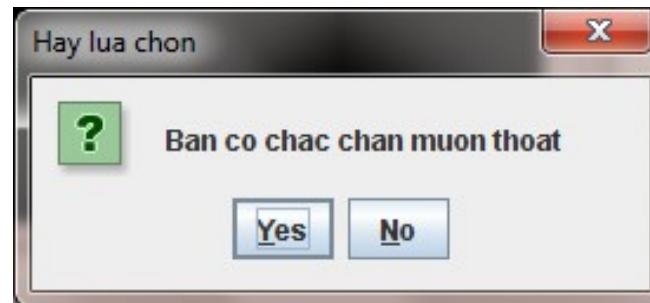
static void	<a href="#">showMessageDialog</a> ( <a href="#">Component</a> parentComponent, <a href="#">Object</a> message) Brings up an information-message dialog titled "Message".
static void	<a href="#">showMessageDialog</a> ( <a href="#">Component</a> parentComponent, <a href="#">Object</a> message,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

## Ví dụ - sử dụng thuộc tính và phương thức static lớp JOptionPane

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



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



## Ví dụ - sử dụng thuộc tính và phương thức static lớp JOptionPane (2)

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



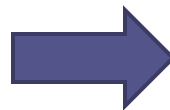
## 2.1. Thành viên static (2)

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

# Ví dụ 1

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

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



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



## Ví dụ 2

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

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

## 2.2. Thành viên hằng

- Một thuộc tính/phương thức không thể thay đổi giá trị/nội dung trong quá trình sử dụng.
- Cú pháp khai báo:

```
chi_dinh_truy_cap final kieu_du_lieu  
TEN_HANG = gia_tri;
```

- Ví dụ:

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

## 2.2. Thành viên hằng (2)

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

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

javax.swing

Class JOptionPane

**ERROR\_MESSAGE**

```
public static final int ERROR_MESSAGE
```

# Nội dung

1. Chồng phương thức
2. Thành viên ĐT và thành viên lớp
- 3. Quản lý bộ nhớ trong Java
4. Truyền tham số cho phương thức
5. Một số lớp tiện ích trong Java

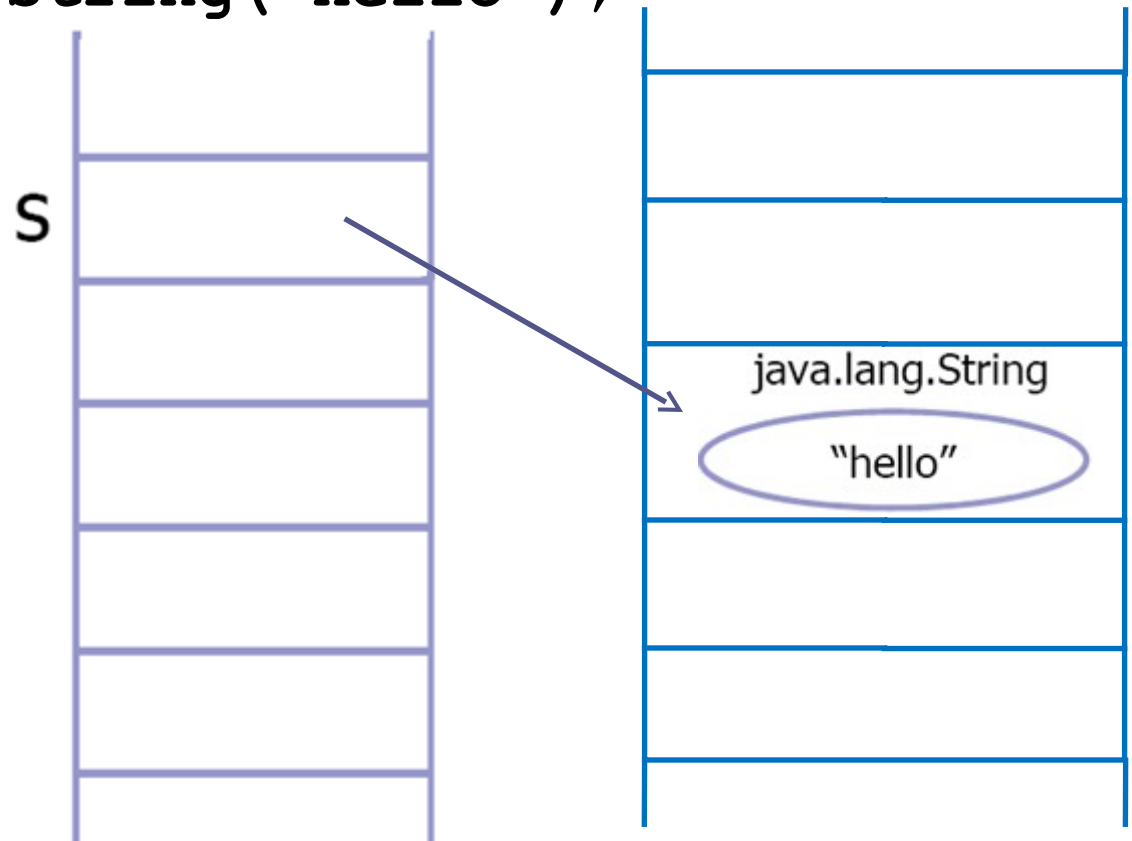
### 3. Quản lý bộ nhớ trong Java

- 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 với lập trình viên.
- Lập trình viên không cần quan tâm đến việc ghi dấu các phần bộ nhớ đã cấp phát trong heap để giải phóng sau này.

## 3.1. Bộ nhớ Heap

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

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

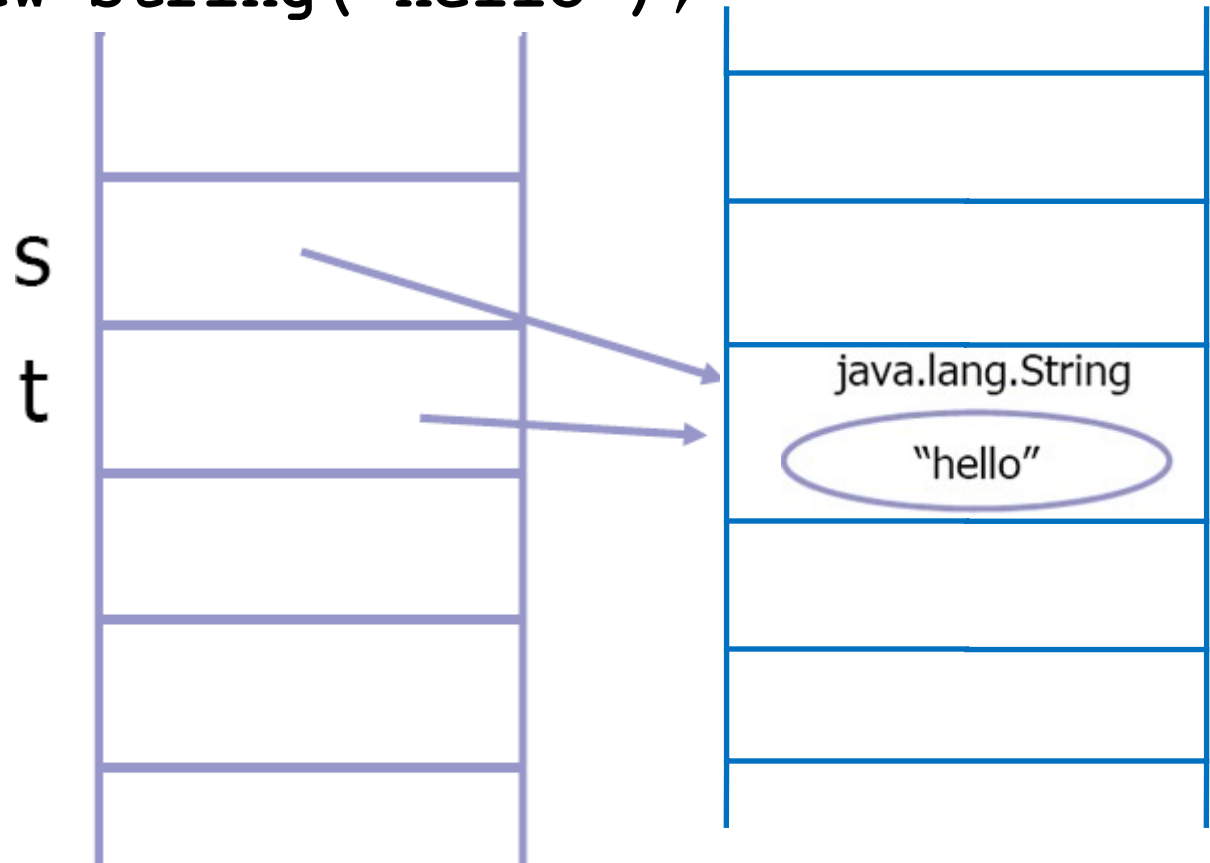


**Bộ nhớ Heap**

## 3.1. Bộ nhớ Heap

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

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

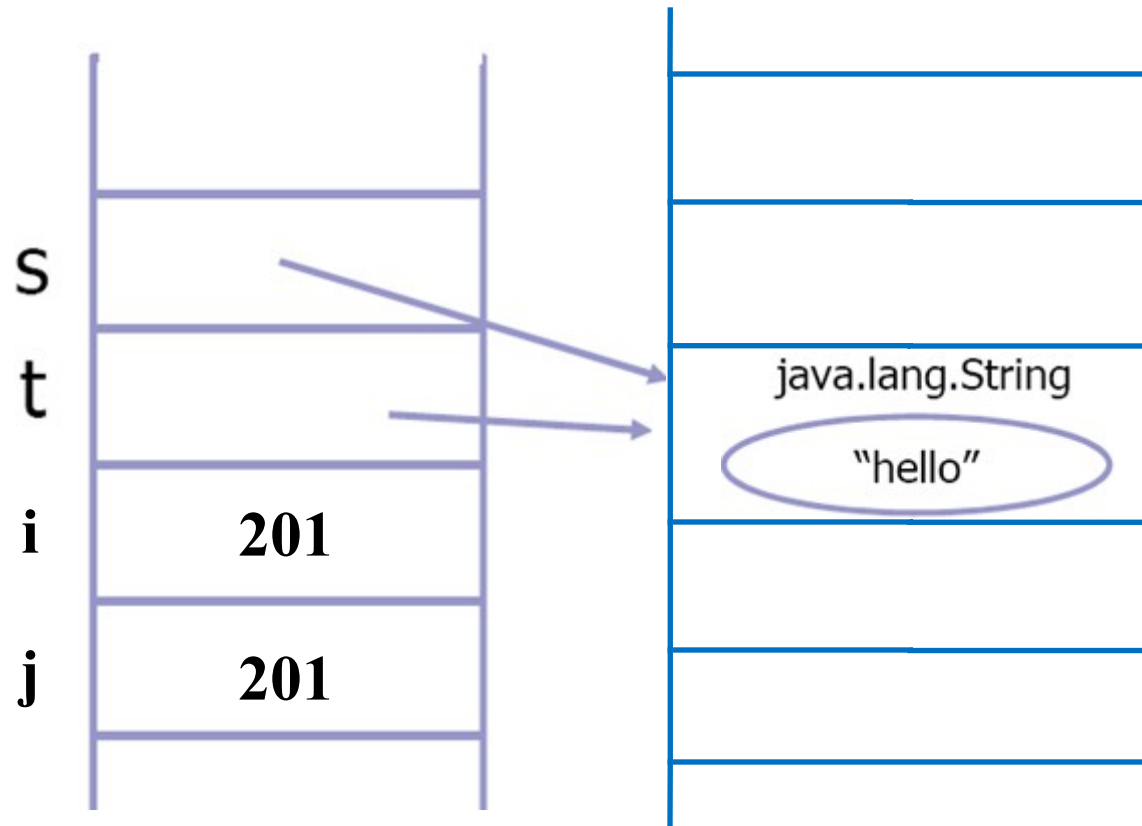


**Bộ nhớ Heap**

## 3.2. Bộ nhớ Stack

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

- Giá trị cục bộ trong bộ nhớ Stack được sử dụng như con trỏ tham chiếu tới Heap
- Giá trị của dữ liệu nguyên thủy được ghi trực tiếp trong Stack



**Bộ nhớ Stack**

**Bộ nhớ Heap**



## 3.3. Bộ thu gom rác (Garbage Collector)

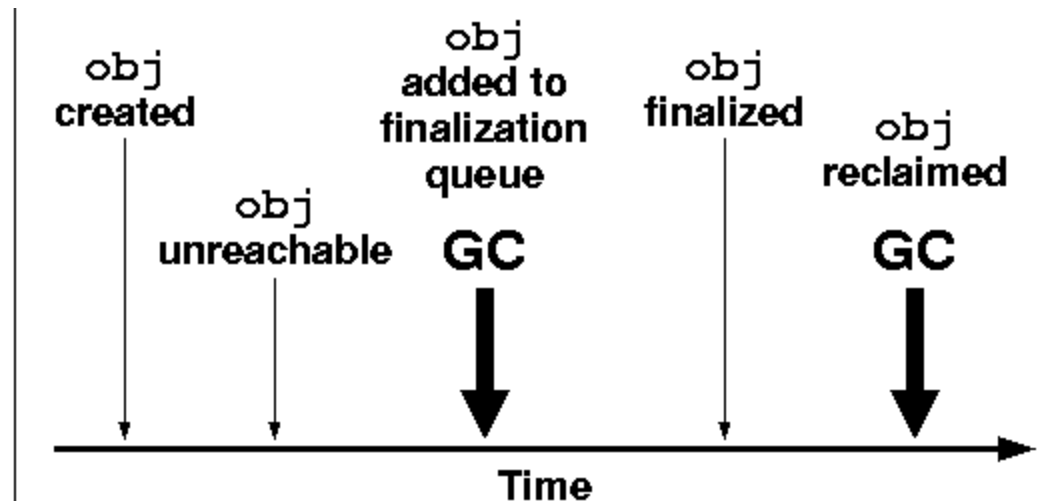
- Một tiến trình chạy ngầm gọi đến bộ “thu gom rác” để phục hồi lại phần bộ nhớ mà các đối tượng không tham chiếu đến (tái định vị)
- Các đối tượng không có tham chiếu đến được gán null.
- Bộ thu gom rác định kỳ quét qua danh sách các đối tượng của JVM và phục hồi các tài nguyên của các đối tượng không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

## 3.3. Bộ thu gom rác (2)

- 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.
- Có thể coi là hàm hủy (destructor) của lớp mặc dù Java không có khái niệm này.



## 3.4. 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
```

## 3.4. So sánh đối tượng (2)

- 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
```

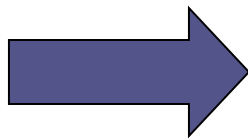
```
Employee a = new Employee(1);  
Employee b = a;  
if (a==b) ... // true
```

## 3.4. So sánh đối tượng (3)

- Phương thức equals
  - Đối với kiểu dữ liệu nguyên thủy ◇ Không tồn tại.
  - Đối với các đối tượng: Bất kỳ đối tượng nào cũng có phương thức này
    - So sánh giá trị của đối tượng

## Ví dụ == và equals - Lớp Integer

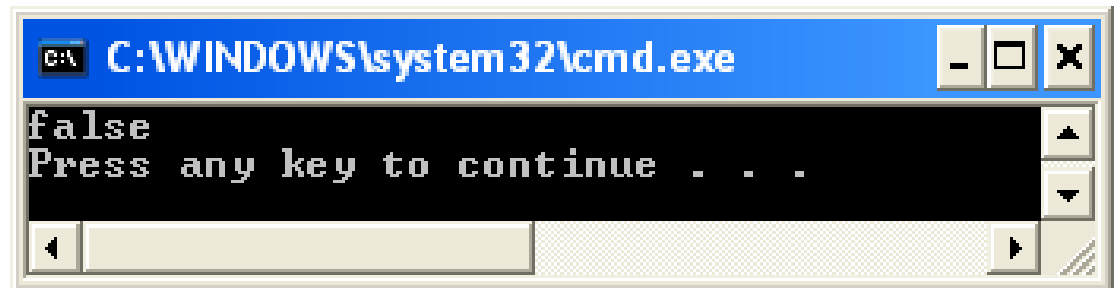
```
public class Equivalence {  
    public static void main(String[] args) {  
        Integer n1 = new Integer(47);  
        Integer n2 = new Integer(47);  
        System.out.println(n1 == n2);  
        System.out.println(n1.equals(n2));  
    }  
}
```

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe'. The window contains the following text: 'false', 'true', and 'Press any key to continue . . .'. The output 'false' corresponds to the 'n1 == n2' comparison, and 'true' corresponds to the 'n1.equals(n2)' comparison. The prompt is waiting for a key press to continue.

```
C:\Windows\system32\cmd.exe  
false  
true  
Press any key to continue . . .
```

## Ví dụ 3 - 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));  
    }  
}
```





# Nội dung

1. Chồng phương thức
2. Thành viên ĐT và thành viên lớp
3. Quản lý bộ nhớ trong Java
- 4. Truyền tham số cho phương thức
5. Một số lớp tiện ích trong Java

## 4. Truyền tham số cho phương thức

- Có thể sử dụng bất kỳ kiểu dữ liệu nào cho tham số của phương thức hoặc constructor
  - Kiểu dữ liệu nguyên thủy
  - Kiểu dữ liệu tham chiếu: mảng và đối tượng
- Ví dụ

```
public Polygon polygonFrom(Point[] corners) {  
    // method body goes here  
}
```

## 4. Truyền tham số cho phương thức (2)


- Java truyền mọi tham số cho phương thức dưới dạng giá trị (pass-by-value): 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

## 4.1. Với kiểu dữ liệu tham trị

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

```
public void method1() {  
    int a = 0;  
    System.out.println(a); // outputs 0  
    method2(a);  
    System.out.println(a); // outputs 0  
}
```

```
void method2(int a) {  
    a = a + 1;  
}
```

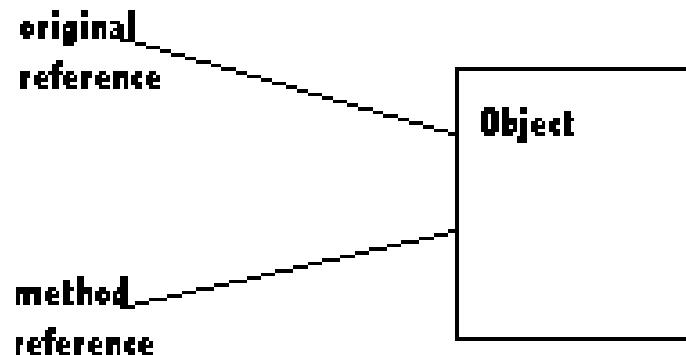


- Phương thức swap này có hoạt động đúng không?

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

## 4.2. 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)

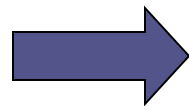


- Sau khi truyền cho phương thức, đối tượng có ít nhất 2 tham chiếu

# 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);  
    }  
}
```

```
public class Test {  
    public static void tricky(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(); tricky(pnt1, pnt2);  
        pnt1.printPoint(); pnt2.printPoint();  
    }  
}
```



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

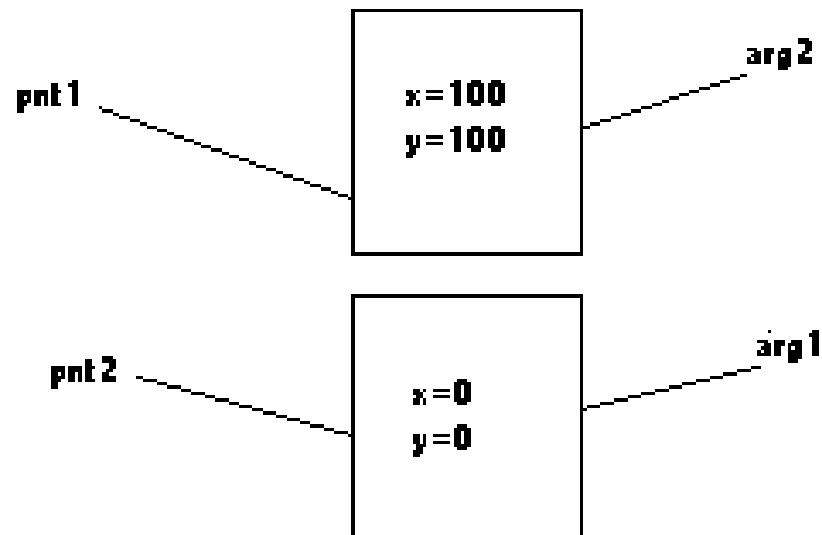
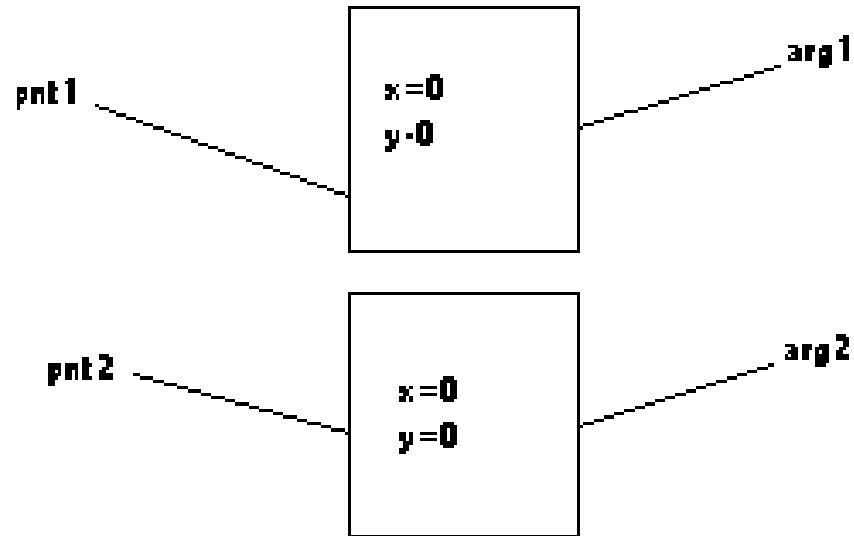
```
X: 0.0 Y: 0.0
```

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

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

```
Press any key to continue . . . _
```

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





## 4.3. Truyền số lượng tham số tùy ý

- Được gọi là *varargs*. Cú pháp:
  - `ten_phuong_thuc(Kieu_dl... ten_tham_so)`
- Ví dụ 1:
  - Khai báo:  

```
public PrintStream printf(String format,  
                           Object... args)
```
  - Sử dụng
    - `System.out.printf ("%s: %d, %s\n",  
 name, idnum, address);`
    - `System.out.printf ("%s: %d, %s, %s, %s\n",  
 name, idnum, address, phone, email);`

- Ví dụ 2

```
public Polygon polygonFrom(Point... corners) {  
    int numberOfSides = corners.length;  
    double squareOfSide1, lengthOfSide1;  
    squareOfSide1 = (corners[1].x - corners[0].x)  
        * (corners[1].x - corners[0].x)  
        + (corners[1].y - corners[0].y)  
            * (corners[1].y - corners[0].y) ;  
    lengthOfSide1 = Math.sqrt(squareOfSide1) ;  
    //create & return a polygon connecting the Points  
}
```

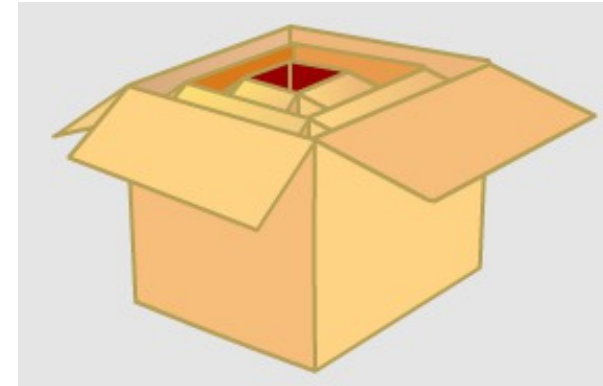
- Nhận xét

- **corners** được coi như một mảng
- Phương thức có thể được gọi bằng cách truyền một mảng hoặc một loạt các tham số truyền vào

# Nội dung

1. Chồng phương thức
2. Thành viên ĐT và thành viên lớp
3. Quản lý bộ nhớ trong Java
4. Truyền tham số cho phương thức
5. Một số lớp tiện ích trong Java

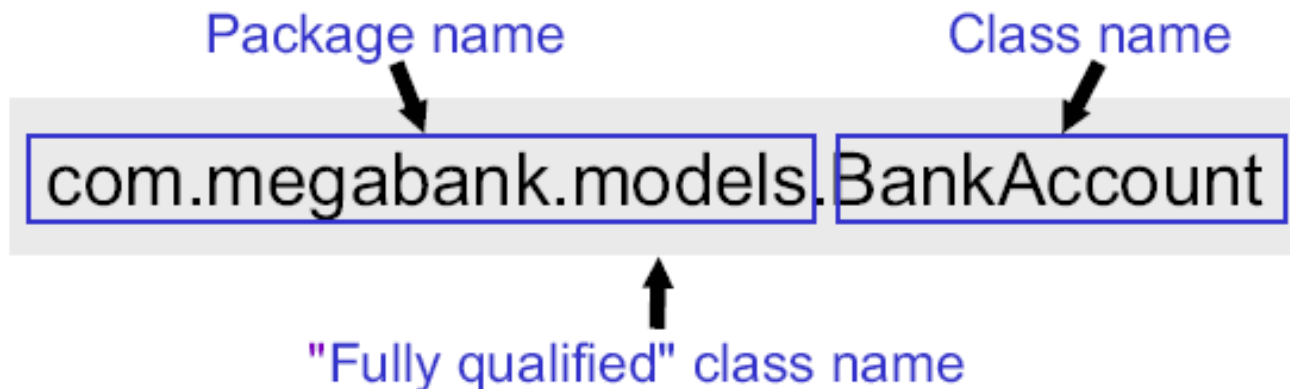
## 5.1. Package trong Java



- Package giống như thư mục giúp:
  - Tổ chức và xác định vị trí lớp dễ dàng và sử dụng các lớp một cách phù hợp.
  - Tránh cho việc đặt tên lớp bị xung đột (trùng tên)
    - Các package khác nhau có thể chứa các lớp có cùng tên
  - Bảo vệ các lớp, dữ liệu và phương thức ở mức rộng hơn so với mối quan hệ giữa các lớp.
- Một package cũng có thể chứa các package khác

## 5.1. Package trong Java (2)

- Tên đầy đủ của lớp bao gồm tên gói và tên lớp:



## a. Tham chiếu giữa các lớp

- Trong cùng 1 package: Sử dụng tên lớp
- Khác package: Phải cung cấp tên đầy đủ cho các lớp được định nghĩa trong package khác.
- Ví dụ:

```
public class HelloNameDialog{  
    public static void main(String[] args){  
        String result;  
        result = javax.swing.JOptionPane.showInputDialog  
            ("Hay nhập ten ban:");  
        javax.swing.JOptionPane.showMessageDialog(null,  
            "Xin chao "+ result + "!");  
    }  
}
```

## a. Tham chiếu giữa các lớp (2)

- Lệnh **import**:
  - Sử dụng lệnh **import** để khai báo các package hoặc các lớp để khi sử dụng không cần nêu tên đầy đủ.
  - Ví dụ:

```
import javax.swing.JOptionPane;  
public class HelloNameDialog{  
    public static void main(String[] args){  
        String result;  
        result = JOptionPane.showInputDialog  
                                ("Hay nhập ten ban:");  
        JOptionPane.showMessageDialog(null,  
                                "Xin chào "+ result + "!");  
        System.exit(0);  
    }  
}
```

## b. Các package trong Java

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



## b. Các package trong Java (2)

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

## b. Các package trong Java (3)

- Các package cơ bản trong Java

- **java.math**

- Cung cấp các lớp thực thi các phép toán với số nguyên và các phép toán thập phân

- **java.sql**

- Cung cấp các API cho phép truy nhập và xử lý dữ liệu được lưu trữ trong một nguồn dữ liệu (thường sử dụng cơ sở dữ liệu quan hệ)

- **javax.swing**

- Cung cấp các lớp và giao diện cho phép tạo ra các ứng dụng đồ họa.

- ...

## 5.2. Các lớp bao (Wrapper class)

- 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:
  - 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

## 5.2. Các lớp bao (2)

Primitive Type	Wrapper Class
boolean	Boolean
byte	Byte
char	Character
double	Double
float	Float
int	Integer
long	Long
short	Short

## a. Chuyển đổi kiểu dữ liệu

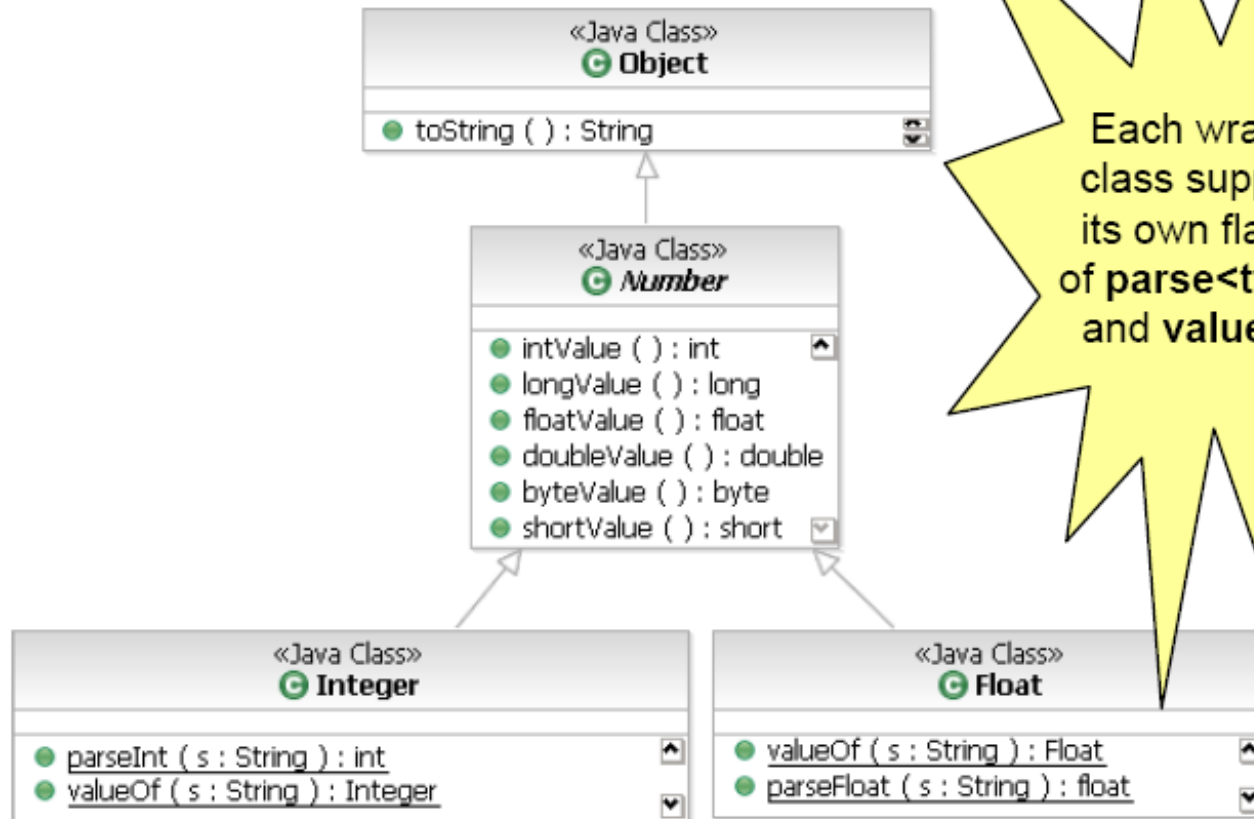
- Sử dụng **toString()** để chuyển các giá trị số thành chuỗi.
- Sử dụng **<type>Value()** để chuyển từ đối tượng của lớp bao thành giá trị nguyên thủy của đối tượng tương ứng

```
Float objF = new Float("4.67");  
float f = objF.floatValue(); // f=4.67F  
int i = objF.intValue(); //i=4
```

- Sử dụng **parse<type>()** và **valueOf()** để chuyển chuỗi thành các giá trị số.

```
int i = Integer.parseInt("123"); //i=123  
double d = Double.parseDouble("1.5"); // d=1.5  
Double objF2 = Double.valueOf("-36.12");  
long l = objF2.longValue(); // l=-36L
```

## a. Chuyển đổi kiểu dữ liệu (2)



Each wrapper class supports its own flavors of `parse<type>()` and `valueOf ()`

## b. Các hằng số

- **Boolean**

- Boolean FALSE
- Boolean TRUE

- **Byte**

- byte MIN\_VALUE
- byte MAX\_VALUE

- **Character**

- int MAX\_RADIX
- char MAX\_VALUE
- int MIN\_RADIX
- char MIN\_VALUE
- Unicode classification constants

- **Double**

- double MAX\_VALUE
- double MIN\_VALUE
- double NaN
- double NEGATIVE\_INFINITY
- double POSITIVE\_INFINITY

- **Float**

- float MAX\_VALUE
- float MIN\_VALUE
- float NaN
- float NEGATIVE\_INFINITY
- float POSITIVE\_INFINITY

- **Integer**

- int MIN\_VALUE
- int MAX\_VALUE

- **Long**

- long MIN\_VALUE
- long MAX\_VALUE

- **Short**

- short MIN\_VALUE
- short MAX\_VALUE

# Ví dụ

```
double d = (new Integer(Integer.MAX_VALUE)).  
            doubleValue();  
System.out.println(d); // 2.147483647E9  
  
String input = "test 1-2-3";  
int output = 0;  
for (int index=0;index<input.length();index++) {  
    char c = input.charAt(index);  
    if (Character.isDigit(c))  
        output = output * 10 + Character.digit(c, 10);  
}  
System.out.println(output); // 123
```



## 5.3. 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;
```

## a. 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));
```

◇ Hai câu lệnh trên có in ra cùng một kết quả?

## b. Các phương thức của chuỗi

```
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"
```

## c. So sánh hai chuỗi

- `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

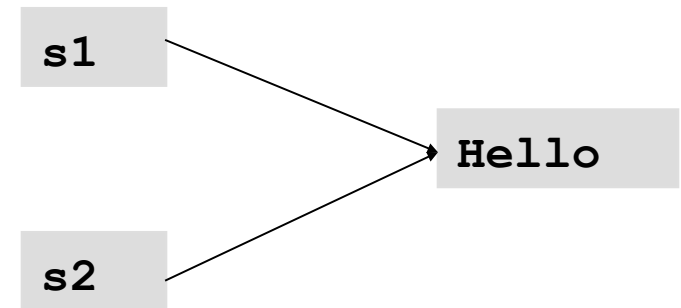
```
boolean same = "Joe".equalsIgnoreCase("joe");
```

- So sánh `oneString == anotherString` sẽ gây nhập nhằng

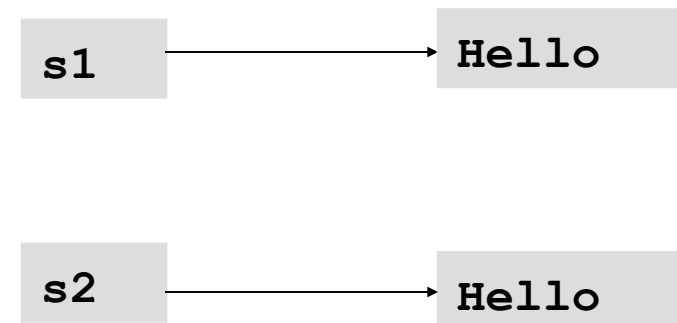
- So sánh 2 đối tượng

## c. So sánh hai chuỗi (2)

```
String s1 = new String("Hello");  
String s2 = s1;  
(s1==s2) trả về true
```



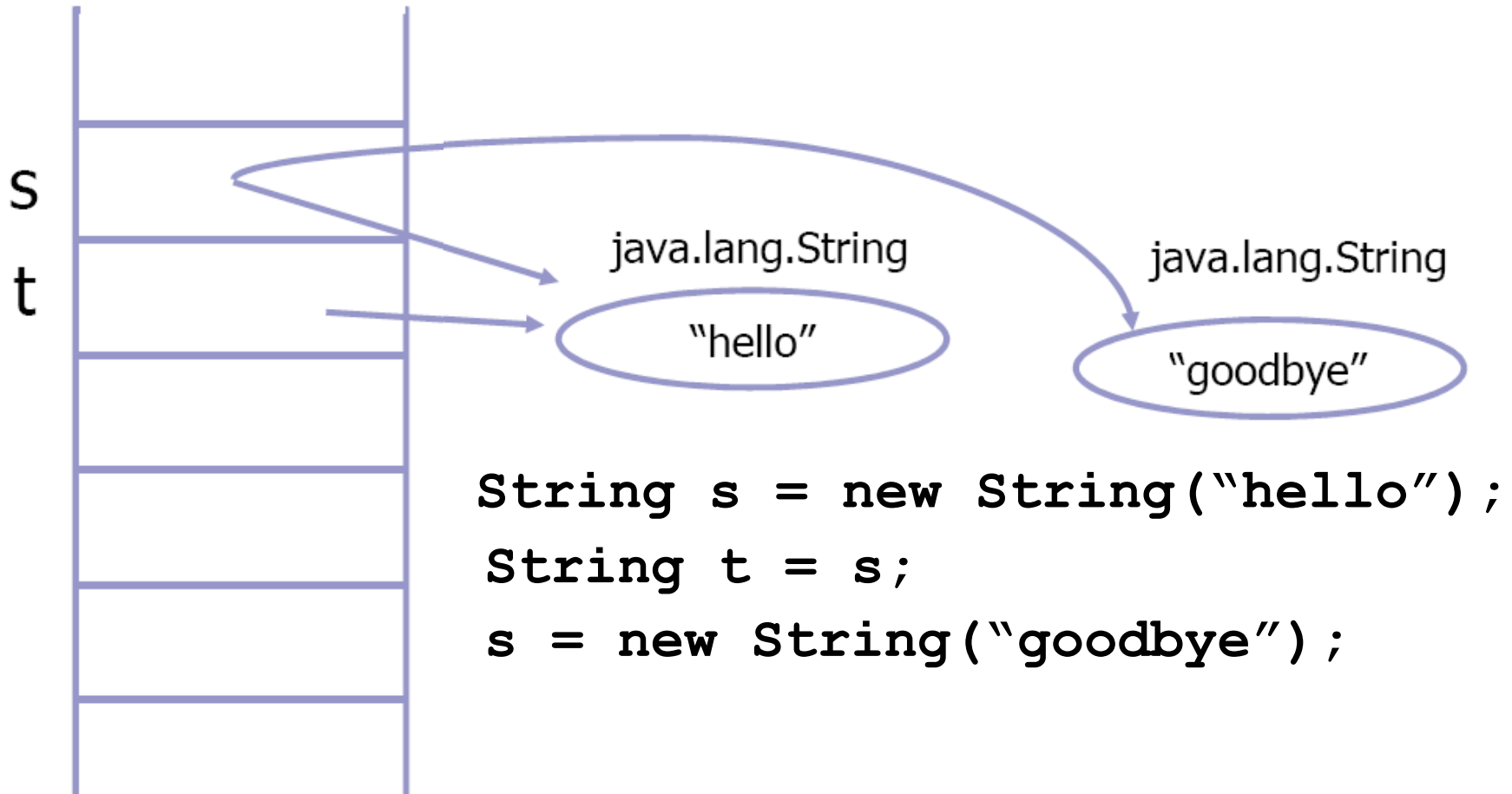
```
String s1 = new String("Hello");  
String s2 = new String("Hello");  
(s1==s2) trả về false
```



## 5.4. 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.
- StringBuffer là kiểu biến đổi:
  - Đối tượng có thể thay đổi giá trị sau khi được tạo ra

## 5.4. StringBuffer (2)



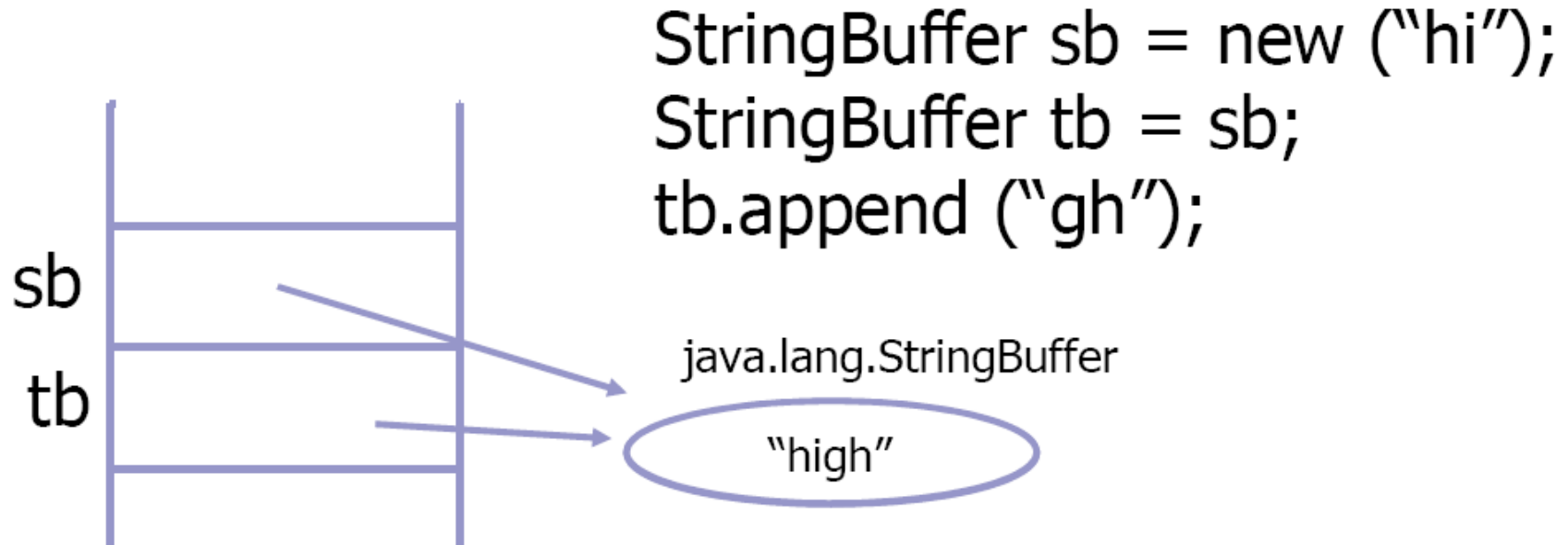
## 5.4. StringBuffer (3)

- **StringBuffer:**
  - Cung cấp các đối tượng xâu có thể thay đổi giá trị  
→ Sử dụng **StringBuffer** khi:
    - 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



## 5.4. StringBuffer (4)

- Tính biến đổi: Nếu một đối tượng bị biến đổi, thì tất cả các quan hệ với đối tượng sẽ nhận giá trị mới.



## 5.4. StringBuffer (5)

- Nếu tạo xâu thông qua vòng lặp thì sử dụng **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."
```

## 5.5. Lớp Math

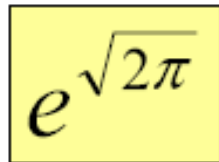
- java.lang.Math cung cấp các thành phần static:
  - Các hằng toán học:
    - Math.E
    - Math.PI
  - Các hàm toán học:
    - max, min...
    - abs, floor, ceil...
    - sqrt, pow, log, exp...
    - cos, sin, tan, acos, asin, atan...
    - random

«Java Class» G Math	
Math ( )	
sin ( )	
cos ( )	
tan ( )	
asin ( )	
acos ( )	
atan ( )	
toRadians ( )	
toDegrees ( )	
exp ( )	
log ( )	
sqrt ( )	
IEEEremainder ( )	
ceil ( )	
floor ( )	
rint ( )	
atan2 ( )	
pow ( )	
round ( )	
round ( )	
initRNG ( )	
random ( )	
abs ( )	
abs ( )	
abs ( )	
abs ( )	
abs ( )	
max ( )	
max ( )	
max ( )	
max ( )	
min ( )	
min ( )	
min ( )	
min ( )	
<clinit> ( )	

## 5.5. Lớp Math (2)

- 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))
```

```
Math ( )  
sin ( )  
cos ( )  
tan ( )  
asin ( )  
acos ( )  
atan ( )  
toRadians ( )  
toDegrees ( )  
exp ( )  
log ( )  
sqrt ( )  
IEEEremainder ( )  
ceil ( )  
floor ( )  
rint ( )  
atan2 ( )  
pow ( )  
round ( )  
round ( )  
initRNG ( )  
random ( )  
abs ( )  
abs ( )  
abs ( )  
abs ( )  
max ( )  
max ( )  
max ( )  
max ( )  
min ( )  
min ( )  
min ( )  
min ( )  
<clinit> ( )
```

## 5.6. Lớp System

- `java.lang.System` chứa nhiều hàm tiện ích hữu dụng
  - Kiểm soát vào ra (I/O) chuẩn
    - Các luồng `InputStream in`, `PrintStreams out` và `err` là các thuộc tính của lớp `System`.
    - Có thể thiết lập lại nhờ các hàm `setIn()`, `setOut()` và `setErr()`
  - **`arraycopy()`** : Sao chép mảng hoặc tập con với hiệu năng cao.

## 5.6. Lớp System (2)

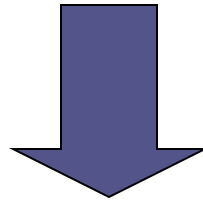
- **currentTimeMillis ()** : Trả về thời gian hiện tại theo millisecond
- **exit ()** : Kết thúc hoạt động của Java Virtual Machine
- **gc ()** : Yêu cầu bộ thu gom rác hoạt động
- Các phương thức liên quan đến thuộc tính của hệ thống:  
Lấy các thông tin thuộc tính như phiên bản của Java Runtime Environment version, thư mục cài đặt Java,...

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

## 5.6. Lớp System (3)

```
import java.util.Properties;
public class PropertiesTest {
    public static void main(String[] args) {
        System.out.println(
            System.getProperty("path.separator"));
        System.out.println(
            System.getProperty("file.separator"));
        System.out.println(
            System.getProperty("java.class.path"));
        System.out.println(
            System.getProperty("os.name"));
        System.out.println(
            System.getProperty("os.version"));
        System.out.println(System.getProperty("user.dir"));
        System.out.println(System.getProperty("user.home"));
        System.out.println(System.getProperty("user.name"));
    }
}
```

## 5.6. Lớp System (4)



```
C:\WINDOWS\system32\cmd.exe  
;  
\  
Windows XP  
5.1  
D:\FIT-HUT\Lectures\OOP\OOP-Java\Demo  
C:\Documents and Settings\Nguyen Thi Thu Trang  
Nguyen Thi Thu Trang  
Press any key to continue . . . _
```