



# Java

## Fundamentals

**[thangld@uit.edu.vn](mailto:thangld@uit.edu.vn)**

**Khoa Mạng máy tính và Truyền thông  
Đại học Công nghệ Thông tin**



# Nội dung

- Giới thiệu Java
- Ứng dụng Java
- OOP trong Java
- Wrapper Classes
- *String*
- Exception Handling
- Nhập/xuất dữ liệu



# Giới thiệu Java [1]

- Ngôn ngữ lập trình hướng đối tượng
- Ngôn ngữ thông dịch
- Độc lập hệ nền (Multi-platform / Platform-Independent)



# Giới thiệu Java [2]

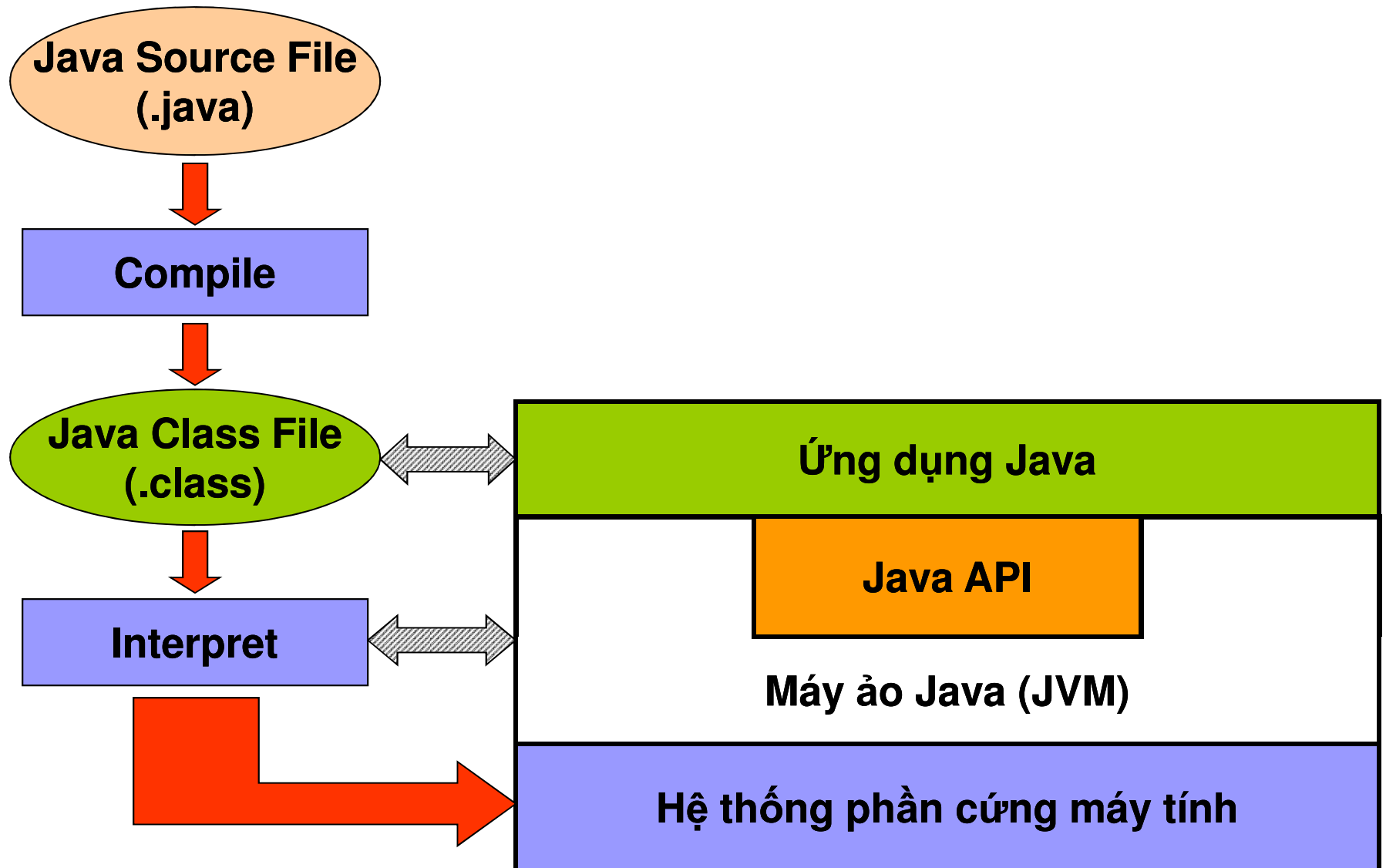
- Ngôn ngữ giống C/C++
- Không có khái niệm con trỏ
- Hủy đối tượng tự động
- Biến môi trường CLASSPATH: chỉ đến thư mục / zip file / jar file chứa các class thư viện



# Java Development Kit

- Bộ công cụ phát triển Java (Windows)
  - Thư mục <j2sdk\_home>/bin
  - *javac.exe*: Java Compiler
    - *javac <java\_source\_file.java>*
  - *java.exe*: Java Interpreter
    - *java <java\_class\_file.class>*

# Thực thi ứng dụng Java





# Ứng dụng Java

HelloWorldApp.java

```
public class HelloWorldApp{  
    public static void main(String[] args) {  
        System.out.println("HelloWorld");  
    }  
}
```



# Kiểu dữ liệu

## ■ Primitive Types

- ☐ *byte*
- ☐ *char*
- ☐ *boolean*
- ☐ *short*
- ☐ *int*
- ☐ *long*
- ☐ *float*
- ☐ *double*

## ■ Reference Types

- ☐ *array*
- ☐ *class*
- ☐ *interface*



# Chuyển đổi kiểu dữ liệu [1]

- Một kiểu dữ liệu được chuyển đổi sang một kiểu dữ liệu khác
- Ví dụ

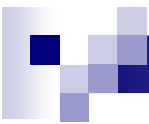
```
float c = 34.89675f;  
int b = (int)c + 10;  
c = b;
```

- Có hai cách chuyển đổi kiểu dữ liệu: tự động và ép kiểu



# Chuyển đổi kiểu dữ liệu [2]

- Khi dữ liệu , với một kiểu dữ liệu cho trước, được gán cho một biến có kiểu dữ liệu khác, quá trình chuyển đổi kiểu dữ liệu tự động thực hiện nếu thỏa các điều kiện sau:
  - Hai kiểu dữ liệu tương thích nhau
  - Kiểu dữ liệu đích lớn hơn kiểu dữ liệu nguồn
- Ép kiểu dữ liệu là sự chuyển đổi dữ liệu tường minh. Nó có thể làm mất thông tin



# Các luật mở rộng kiểu dữ liệu

- Tất cả các giá trị kiểu **byte** and **short** được mở rộng thành kiểu **int**
- Nếu một toán hạng có kiểu **long**, kiểu dữ liệu của toàn biểu thức sẽ được mở rộng thành kiểu **long**
- Nếu một toán hạng có kiểu **float**, kiểu dữ liệu của toàn biểu thức sẽ được mở rộng thành kiểu **float**
- Nếu một toán hạng có kiểu **double**, kiểu dữ liệu của toàn biểu thức sẽ được mở rộng thành kiểu **double**



# Biến

- Khai báo (giống C/C++)

*kiểu-dữ-liệu tên-biến [=giá-trị];*

- Ví dụ

*double d = 5.5;*

# Mảng

## ■ Khai báo

- *kiểu-dữ-liệu* *tên-biến*[];
- *kiểu-dữ-liệu* *tên-biến*[] =  
*new* *kiểu\_dữ\_liệu*[*số\_pt*];
- *kiểu-dữ-liệu* *tên-biến*[] =  
*{gtri1, gtri2, ...}*;

## ■ Ví dụ

```
int a[];
```

```
int a[] = new int [10];
```

```
float af[] = {5.3, 7.6, 8.9, 3.0};
```



# Cấu trúc điều khiển

- Cấu trúc rẽ nhánh

- *if-else*

- *switch-case*

- Cấu trúc lặp

- *while*

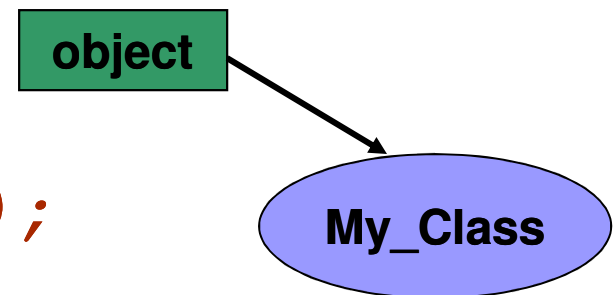
- *do-while*

- *for*

# Class và Object

- Lớp (class) định nghĩa một kiểu dữ liệu mới
- Đối tượng (object) thuộc một lớp trong Java luôn được cấp phát động, sử dụng từ khóa *new*
- Biến có kiểu dữ liệu là lớp có thể tham chiếu đến một đối tượng thuộc lớp
- Ví dụ

```
My_Class object;  
object = new My_Class();
```



# Class

## ■ Khai báo

*access-specifier modifier*

```
class <tên-class>  
    [extends <tên-super-class>]  
    [implements <d/s-interface>] {  
    /*  
    * class body  
    */  
}
```

□ *access-specifier:*

- *public,*
- *none (default)*

□ *modifier:*

- *abstract*
- *final*





# Thuộc tính

## ■ Khai báo

*access-specifier modifier*

*<khai-báo-biến>;*

□ *access-specifier:*

- *public*
- *protected*
- *none (default)*
- *private*

□ *modifier:*

- *static*
- *final*

# Phương thức [1]

## ■ Khai báo

```
access-specifier modifier  
kiểu-dữ-liệu tên-phương-thức (  
    danh-sách-tham-số) {  
  
}
```

□ *access-specifier:*

- *public*
- *protected*
- *none (default)*
- *private*

□ *modifier:*

- *static*
- *final*
- *abstract*



# Phương thức [2]

- Nguyên tắc truyền tham số:
  - Kiểu dữ liệu là kiểu cơ sở: truyền bằng tham số trị
  - Kiểu dữ liệu là kiểu tham chiếu: truyền bằng tham chiếu
- **Overloading**: các phương thức trong cùng một class có cùng tên nhưng khác danh sách tham số.
- **Overriding**: các phương thức giống nhau nhưng được khai báo trong các lớp khác nhau có quan hệ kế thừa.

# Phương thức [3]

## ■ Ví dụ

```
void f(int i) { ... }  
void g(My_Class o) { ... }
```

### □ Truyền tham trị

```
int k=5;  
f(5);  
f(k);
```

### □ Truyền tham chiếu

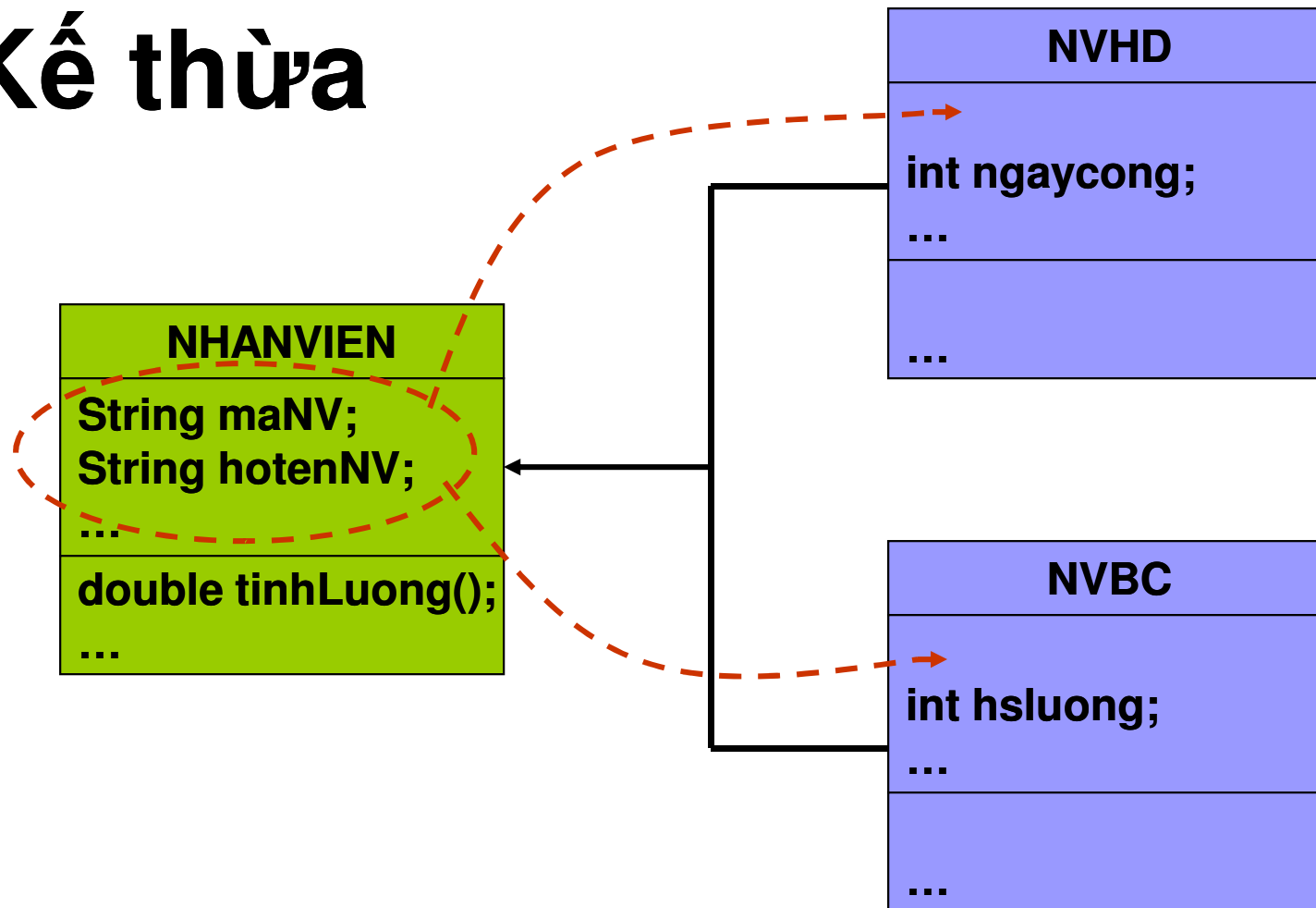
```
My_Class obj = new My_Class();  
g(obj);
```



# Constructor

- Có cùng tên với tên lớp và không có kiểu trả về
- Được tự động gọi thực hiện ngay khi đối tượng thuộc lớp được tạo ra
- Phương thức khởi tạo
  - Không tham số: phương thức khởi tạo mặc định
  - Có tham số
- Tùy thuộc vào cách khởi tạo đối tượng mà phương thức khởi tạo tương ứng được gọi thực hiện

# Kế thừa



- Lớp dẫn xuất kế thừa tập các thuộc tính, phương thức được khai báo trong lớp cơ sở



# Kế thừa & Constructor

- Lệnh gọi thực thi phương thức khởi tạo của lớp cơ sở phải là câu lệnh đầu tiên trong hàm khởi tạo của lớp dẫn xuất
- Nếu trong phương thức khởi tạo của lớp dẫn xuất không gọi (tường minh) phương thức khởi tạo của lớp cơ sở thì phương thức khởi tạo mặc định của lớp cơ sở luôn được tự động gọi thực hiện



# Ví dụ

```
class A {  
    A () {  
    }  
    A (int i) {  
    }  
} //end class A
```

```
class B extends A {  
    B () {  
        [super();]  
        ...  
    }  
    B (int i, int j) {  
        super(i);  
        ...  
    }  
    B (int k) {  
        ...  
    }  
} //end class B
```

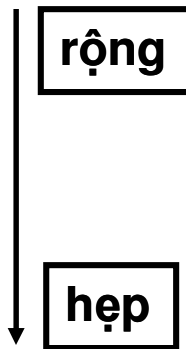


# Overriding & Phạm vi truy cập

- Phạm vi truy cập của các phương thức overriding trong lớp dẫn xuất phải bằng hoặc rộng hơn trong lớp cơ sở

- Thứ tự truy xuất

- ☐ *public*
- ☐ *protected*
- ☐ *none (default)*
- ☐ *private*





# Ví dụ

```
class A {  
    void f () { ... }  
    public void g (int i) { ... }  
} //end class A
```

```
class B extends A {  
    //[none], protected, public  
    void f () { ... }  
  
    //public  
    ____ void g (int i) { ... }  
} //end class B
```

# Từ khóa **super**

- Từ khóa **super** được sử dụng để gọi thực hiện phương thức khởi tạo của lớp cơ sở

**super () ;** //gọi constructor của lớp cơ sở

- Từ khóa **super** có thể được sử dụng để tham chiếu đến các thuộc tính hoặc gọi thực hiện các phương thức của lớp cơ sở

**super.f () ;**

# Từ khóa `static`

- Khai báo thuộc tính/phương thức

*`static int i;`*

*`public static void f ( ) {...}`*

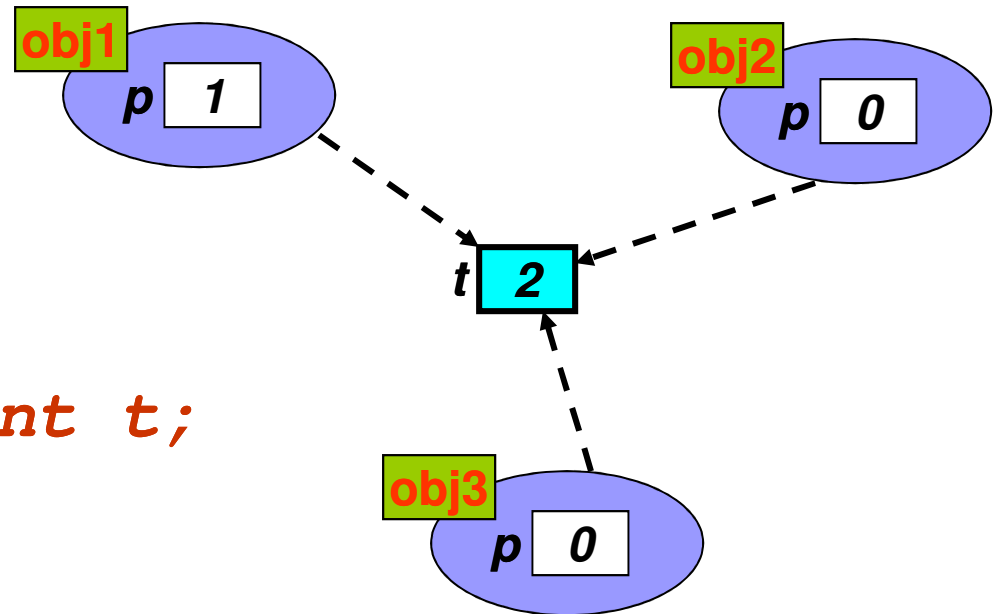
- Thuộc tính *`static`* là duy nhất, được chia sẻ bởi tất cả các đối tượng thuộc lớp.
- Phương thức *`static`* chỉ truy xuất được các thuộc tính *`static`*.
- Các thành viên *`static`* có thể được truy xuất thông qua tên lớp

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

# Ví dụ

```
class A {  
    public int p;  
    public static int t;  
}
```

```
class Main {  
    public static void main(String[] args) {  
        A obj1= new A();  
        A obj2= new A();  
        A obj3= new A();  
        A.p= 10; //Error  
        A.t= 5;  
    }  
}
```



```
obj1.p= 1;  
obj1.t= 2;
```

# Interface [1]

- Là một kiểu dữ liệu tham chiếu -- có thể được xem như một class hoàn toàn trừu tượng
- Được sử dụng với mục đích
  - Hỗ trợ đa kế thừa
  - Là giao tiếp đại diện cho một hoặc nhiều kiểu dữ liệu class cụ thể
- Quy ước
  - Tất cả các thành viên: ***public***
  - Tất cả các thuộc tính: ***static final***
  - Tất cả các phương thức: ***abstract***

# Interface [2]

## ■ Khai báo

```
access-specifier interface  
                                <tên-interface>  
                                [extends <d/s-interface>] {  
    /*  
    * interface body  
    */  
}
```

□ *access-specifier*:

- *public*,
- *none (default)*



# Tham chiếu

- Tham chiếu, hình thành khi đối tượng được tạo ra, được sử dụng để truy xuất các thuộc tính của đối tượng
- Khi gán một đối tượng vào một biến nhớ, hoặc truyền đối tượng vào phương thức, chỉ có tham chiếu của đối tượng được truyền vào

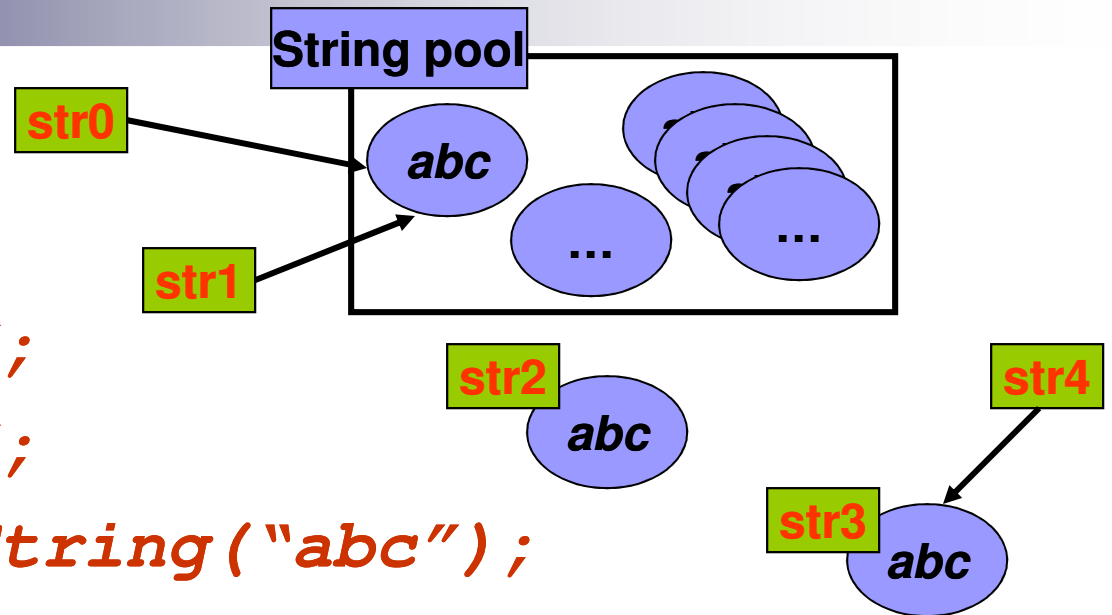




# So sánh trên tham chiếu

- Toán tử so sánh `==` và `!=` được sử dụng để xác định xem hai biến có cùng tham chiếu đến một đối tượng (một vùng nhớ) hay không
- Class *java.lang.Object* cung cấp phương thức *equals()* để so sánh giá trị các đối tượng
- Các class cần override phương thức *equals()* để thực hiện so sánh bằng giá trị của các đối tượng thuộc lớp

# Ví dụ



```
String str0= "abc";  
String str1= "abc";  
String str2= new String("abc");  
String str3= new String("abc");  
String str4= str3;
```

<code>str0==str1 //true</code>	<code>str0.equals(str1) //true</code>
<code>str1==str2 //false</code>	<code>str1.equals(str2) //false</code>
<code>str2==str3 //false</code>	<code>str2.equals(str3) //false</code>
<code>str3==str4 //true</code>	<code>str3.equals(str4) //true</code>



# Wrapper Classes [1]

- Nằm trong *package* ***java.lang***
- Đóng gói các kiểu dữ liệu cơ sở dưới dạng các lớp
- Được sử dụng khi cần dùng một đối tượng biểu diễn một kiểu cơ sở
- Cung cấp các phương thức ***static*** tiện ích chuyển đổi kiểu dữ liệu:

***int Integer.parseInt(String)***

***double Double.parseDouble(String)***

***...***



# Wrapper Classes [2]

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




# Lớp `String` [1]

- Trong Java, một chuỗi ký tự là một đối tượng thuộc lớp *`String`*.
- Mỗi khi thực hiện thay đổi trên một *`String`*, một đối tượng *`String`* mới sẽ được tạo nên với những thay đổi thể hiện trong đó. Chuỗi ký tự ban đầu không thay đổi

# Lớp String [2]

- Các phương thức thao tác trên chuỗi ký tự
  - *int length()*: xác định chiều dài của một *String*
  - *int indexOf(String)*: tìm một chuỗi con trong một chuỗi
  - *static String valueOf(...)*: chuyển đổi một kiểu dữ liệu cơ sở bất kỳ sang chuỗi ký tự
  - *String toLowerCase()*: chuyển đổi thành chuỗi thường
  - *String toUpperCase()*: chuyển đổi thành chuỗi hoa
  - ...



# Lớp String [3]

```
String s1= "abc";
```

```
int len= s1.length();           //len= 3
```

```
int pos= s1.indexOf("a");       //pos= 0
```

```
pos= s1.indexOf("g");           //pos= -1
```

```
String s2= s1.toUpperCase();    //s2= "ABC"
```

```
String s3= s2.toLowerCase();    //s3= "abc"
```

```
String s4= String.valueOf(2.5);
```

```
           //s4= "2.5"
```

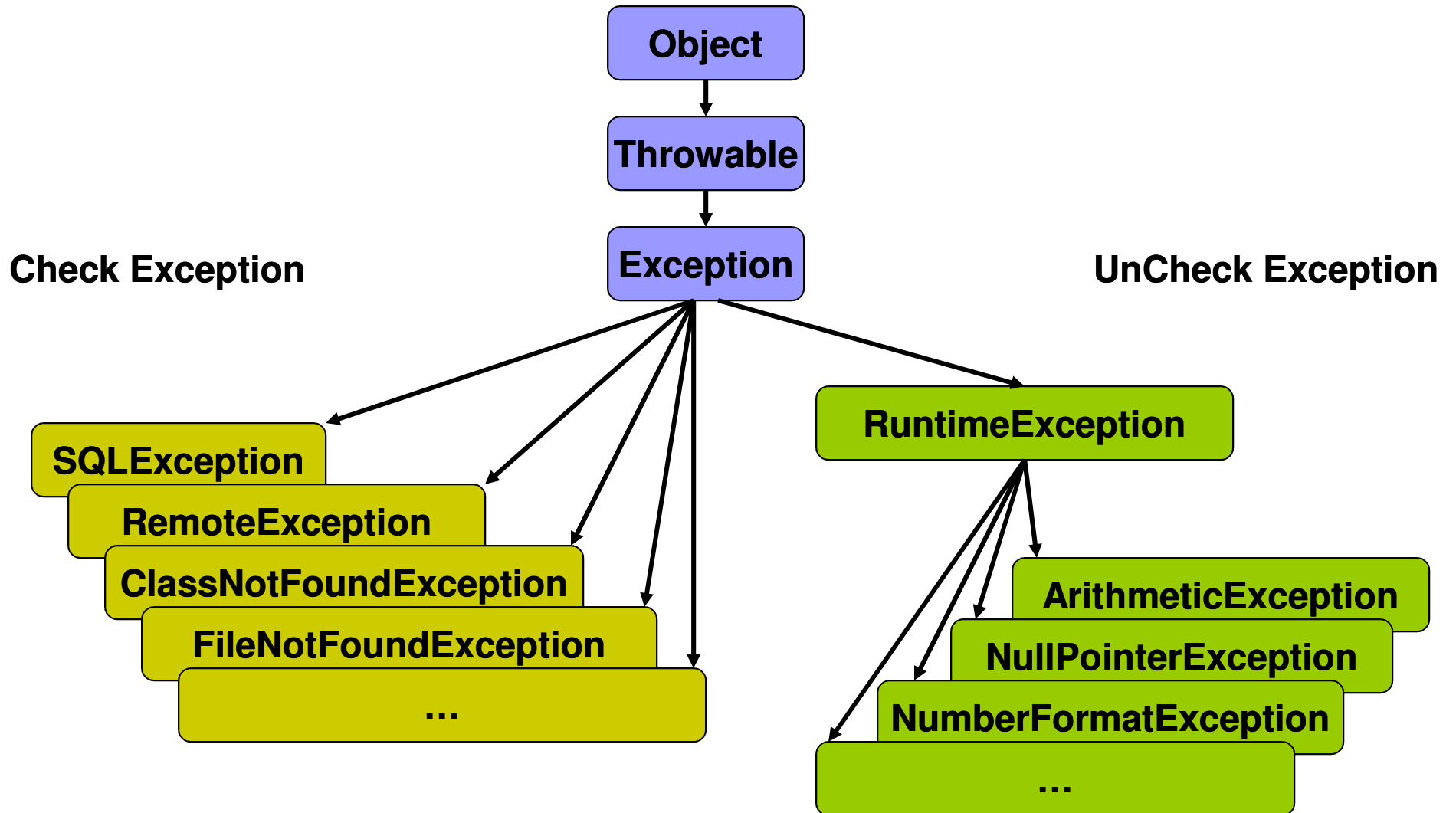


# Exception

- Được sử dụng để xử lý các tình huống bất thường xảy ra trong chương trình
- Các đối tượng exception được các phương thức ném ra để thông báo về một trường hợp bất thường xảy ra trong quá trình thực thi
- Có nhiều loại exception trong Java
- Java cung cấp class *java.lang.Exception* là lớp cơ sở cho tất cả các exception khác



# Các lớp exception





# Khai báo sử dụng exception [1]

## ■ Khai báo class exception

```
class MyException extends Exception {  
    ...  
    public MyException() {  
        super();  
    }  
  
    public MyException(String msg) {  
        super(msg);  
    }  
}
```

# Khai báo sử dụng exception [2]

## ■ Khai báo phương thức phát sinh exception

```
public int method1 throws MyException {  
    ...  
    if (...) {  
        throw new MyException();  
    }  
    ...  
    return 1;  
}
```

# Xử lý exception [1]

- Phải xử lý exception khi gọi thực thi phương thức có khả năng phát sinh exception
- Có hai cách xử lý exception trong Java:
  - Khai báo phương thức ném ra exception tương ứng -- không xử lý exception

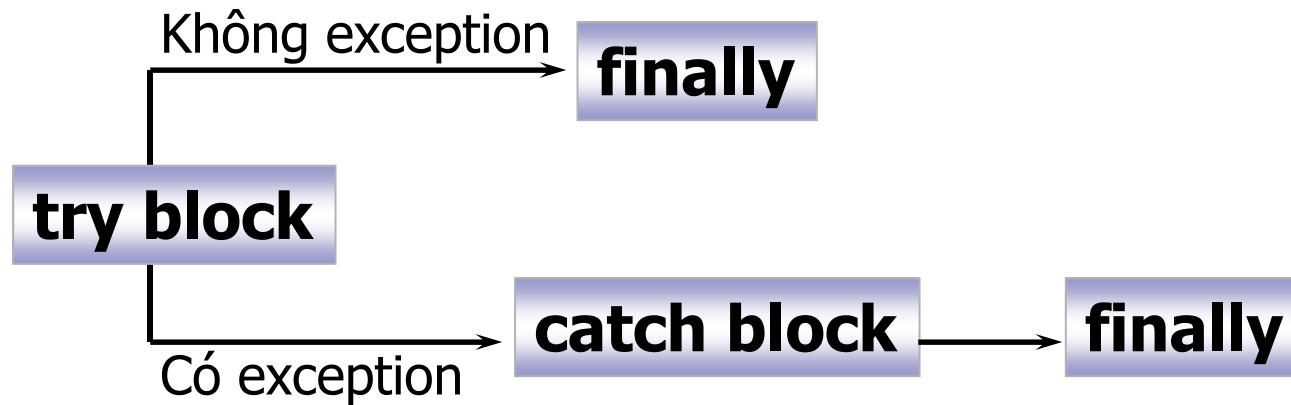
```
void fMethod() throws MyException {  
    ...  
    method1 ();  
    ...  
}
```

# Xử lý exception [2]

- Xử lý exception trong khối lệnh *try...catch*

```
try {  
    Khối_lệnh_cần_thực_hiện;  
} catch ( ) {  
    Khối_lệnh_xử_lý_exception;  
} catch ( ) {  
    ...  
} finally {  
    Khối_lệnh_kết_thúc;  
}
```

# Xử lý exception [3]





# Ví dụ

```
try {  
    byte[] buffer= new byte[128];  
    int len= System.in.read(buffer);  
    System.out.println(  
        new String(buffer, 0, len) );  
} catch ( IOException e ) {  
    e.printStackTrace();  
    [throw e;]  
} finally {  
    System.out.println("Finally.");  
}
```

# Nhập / xuất dữ liệu

- Sử dụng các luồng nhập xuất trong gói *java.io*
- Có hai loại luồng nhập/xuất trong Java:
  - Các luồng dữ liệu kiểu byte:
    - Xử lý dữ liệu nhập/xuất theo từng byte.
    - Hai lớp cơ sở là: *InputStream* và *OutputStream*
  - Các luồng dữ liệu kiểu ký tự:
    - Xử lý dữ liệu theo từng ký tự
    - Hai lớp cơ sở là: *Reader* và *Writer*



# Đọc dữ liệu từ Console[1]

- Sử dụng đối tượng *System.in*

```
try {  
    byte data[] = new byte[128];  
    System.out.print("Enter a string: ");  
    int len = System.in.read(data);  
    String str = new String(data, 0, len);  
    System.out.println(str);  
} catch (IOException ex) {  
    ex.printStackTrace();  
}
```

# Đọc dữ liệu từ Console[2]

- Sử dụng lớp đối tượng *BufferedReader*

```
try {  
    BufferedReader br=  
        new BufferedReader(  
            new InputStreamReader(  
                System.in));  
    System.out.print("Enter a string: ");  
    String str= br.readLine();  
    System.out.println(str);  
} catch (IOException ex) {  
    ex.printStackTrace();  
}
```

# Đọc dữ liệu từ Console[3]

- Sử dụng lớp đối tượng *java.util.Scanner*

```
Scanner scanner=  
        new Scanner(System.in);  
System.out.print("Enter a string: ");  
String str1= scanner.nextLine();  
System.out.println("and a number: ");  
int v= scanner.nextInt();  
scanner.nextLine();  
System.out.println("and a string: ");  
String str2= scanner.nextLine();  
System.out.println("Result: "  
        + str1 + " " + str2 + " " + v);
```

# Đọc dữ liệu từ tập tin

- Đọc dữ liệu, sử dụng lớp đối tượng *FileInputStream*

```
FileInputStream fis=  
    new FileInputStream(...);  
byte buffer[]= new byte[128];  
int len=0;  
while (len != -1) {  
    len= fis.read(buffer);  
    ...  
}  
fis.close();
```



# Ghi dữ liệu ra tập tin

- Xuất dữ liệu, sử dụng lớp đối tượng *FileOutputStream*

```
FileOutputStream fos=  
                new FileOutputStream(...);  
String str= "hello";  
fos.write(str.getBytes());  
fos.close();
```

A decorative graphic on the left side of the slide consists of a large, light blue square with a smaller, slightly offset light blue square inside it. Overlapping these are several smaller squares in various shades of blue, creating a layered effect. A solid dark blue horizontal bar extends from the right side of this graphic across the middle of the slide.

**Q&A**