

Tổng quan về ngôn ngữ Java

Nguyễn Thanh Quân – ntquan@fit.hcmuns.edu.vn

Nội dung

- Giới thiệu ngôn ngữ Java
- Đặc điểm về các kiểu dữ liệu
- Các khái niệm về hướng đối tượng trong ngôn ngữ Java
- Exception Handling
- Collection Framework
- Regular Expressions



Phần 1:

Giới thiệu về Ngôn Ngữ Java



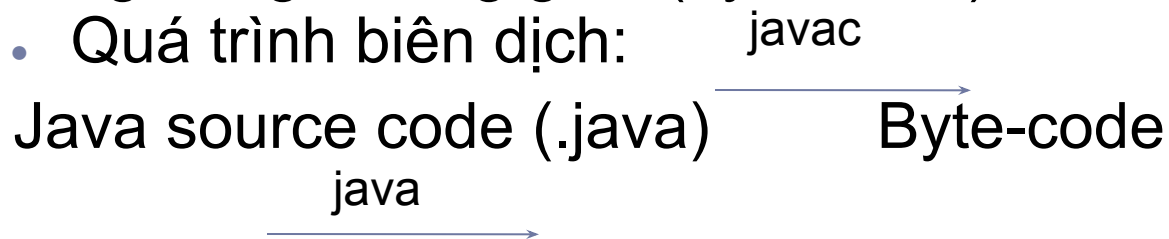
Lịch sử ra đời

- Java được Sun Microsystems (*Stanford University Network*) khai sinh năm 1991, hướng đến việc viết phần mềm cho các thiết bị điện tử (set-top box, microwave oven...)
- Java là một ngôn ngữ thuần hướng đối tượng, platform independent, hỗ trợ đầy đủ cho unicode và rất mạnh mẽ.
- Các dòng sản phẩm Java hiện có:
 - J2SE (Java 2 Standard Edition): nhắm đến việc phát triển các ứng dụng desktop thông thường
 - J2EE (Java 2 Enterprise Edition): nhắm đến các ứng dụng lớn: phân tán, web, web service...
 - J2ME (Java 2 Micro Edition): ra đời sau cùng, nhắm đến các sản phẩm nhỏ: mobile, palm, pda...



Platform Independent

- Một chương trình Java không được biên dịch trực tiếp thành mã máy (machine code) như các ngôn ngữ ra đời trước (C, C++) mà sẽ được biên dịch thành một dạng ngôn ngữ trung gian (byte-code)

Quá trình biên dịch: 

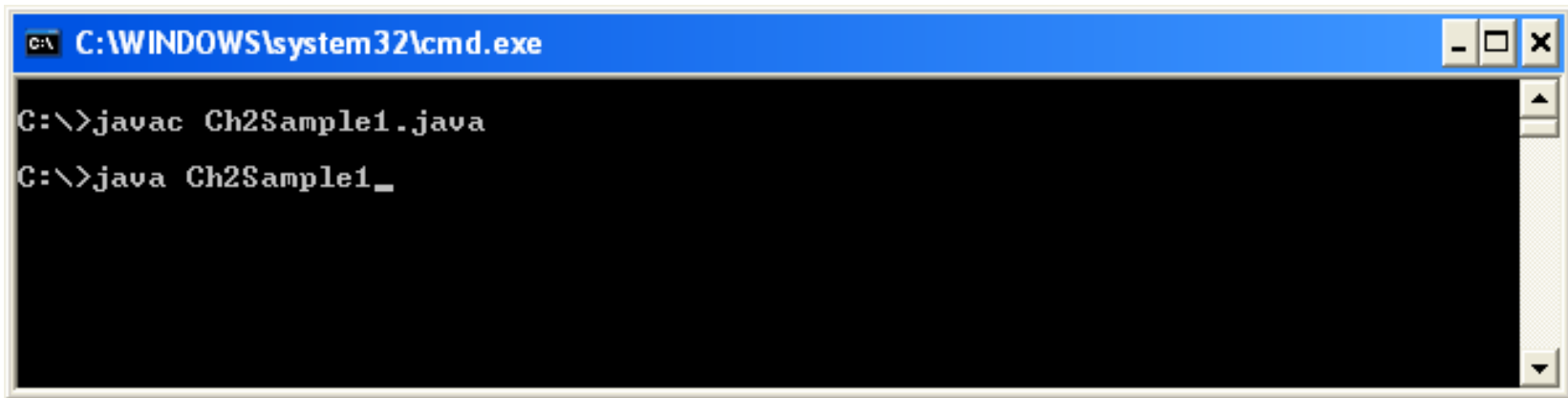
(.class) Machine code (phụ thuộc vào platform)

- Một ứng dụng hay thư viện của java có thể được đóng gói thành file .jar (tập hợp các file .class)
- Máy ảo Java (JVM) có chức năng biên dịch byte-code thành code thực thi trên một môi trường cụ thể. Mỗi HĐH sẽ được xây dựng một JVM riêng biệt



-
- Để biên dịch và thực thi ứng dụng Java, ta cần download bộ công cụ JDK của Sun, trong bộ công cụ này sẽ có các file `javac.exe`, `java.exe`, `jar.exe` được đặt tại thư mục `JDK/bin`
 - Set biến môi trường `JAVA_HOME` trỏ đến thư mục cài đặt JDK, và bổ sung đường dẫn `JAVA_HOME/bin` vào biến môi trường `PATH`
 - Các ứng dụng Java khi cần sử dụng các thư viện cộng thêm (thường dưới dạng `.jar`), đường dẫn đến các file `.jar` phải được set trong biến môi trường `CLASSPATH` hoặc chỉ ra từ câu lệnh biên dịch
-





```
C:\WINDOWS\system32\cmd.exe
C:\>javac Ch2Sample1.java
C:\>java Ch2Sample1_
```

- Biên dịch:

`javac [-cp path] Filename.java`

- Thực thi:

`java [-cp path] Filename` #Tên file không có .class

Đối với ứng dụng đã được đóng thành file .jar

`java -jar Filename.jar`

Trong classpath, ký tự “.” tượng trưng cho thư mục hiện hành.



Một số IDE để xây dựng ứng dụng

- Netbeans
- Jbuilder
- Eclipse
- Jcreator
- Java Sun Studio



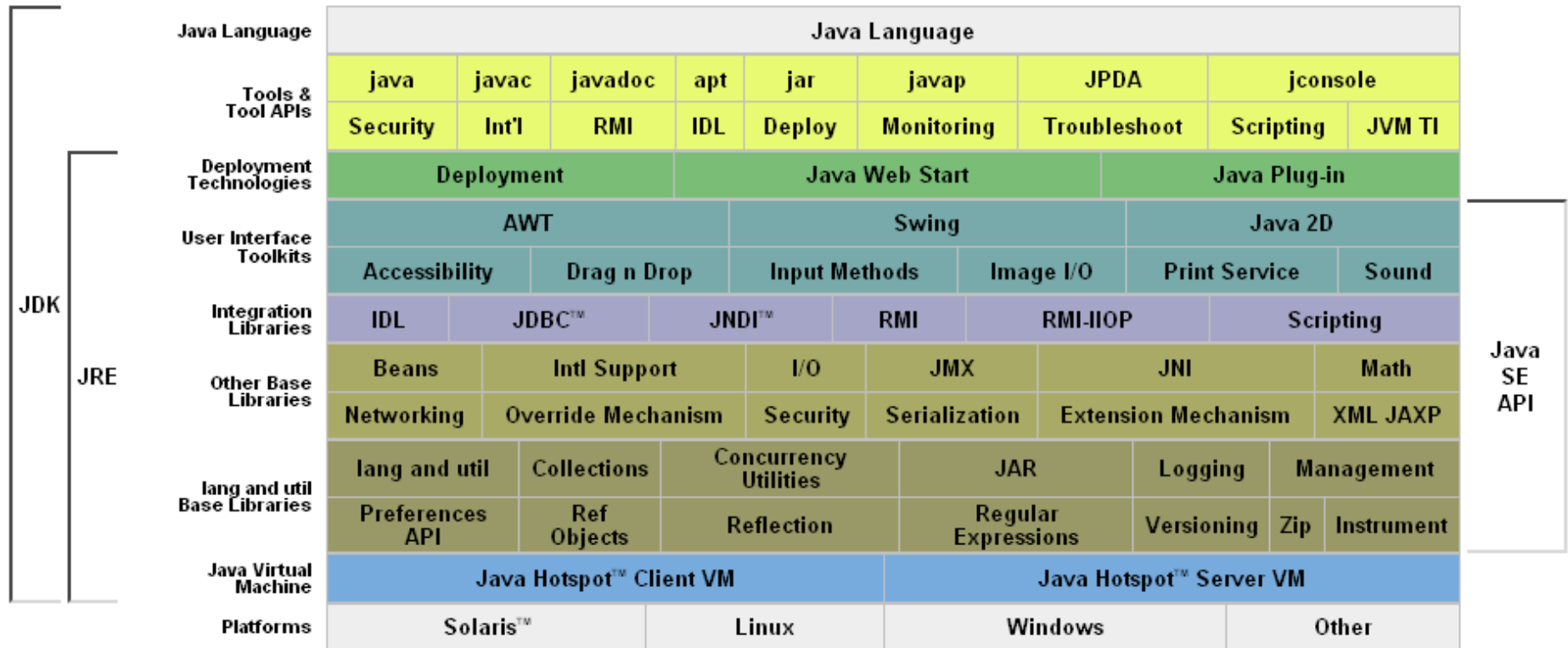
Một file source code cơ bản

- Một ứng dụng java cũng là 1 lớp, hàm main sẽ đặt trong lớp này (thuần hướng đối tượng).
- Không tồn tại khái niệm **biến toàn cục, hàm toàn cục**
- **Tên lớp và tên file phải giống nhau, ví dụ lớp test1 phải được nằm trong file có tên test1.java**

```
class vidu1
{
    public static void main (String[] args){
        System.out.println(" Hello world!");
    }
}
```



Cấu trúc gói phần mềm Java



Tài nguyên nghiên cứu:

- J2SE Doc (giống bộ tài liệu MSDN) : https://cds.sun.com/is-bin/INTERSHOP.enfinity/WFS/CDS-CDS_Developer-Site/en_US/-/USD/ViewProductDetail-Start?ProductRef=jdk-6-doc-oth-JPR@CDS-CDS_Developer
- J2EE Doc: https://cds.sun.com/is-bin/INTERSHOP.enfinity/WFS/CDS-CDS_Developer-Site/en_US/-/USD/ViewFilteredProducts-SimpleBundleDownload
- Java Tutorial: https://cds.sun.com/is-bin/INTERSHOP.enfinity/WFS/CDS-CDS_Developer-Site/en_US/-/USD/ViewProductDetail-Start?ProductRef=tutorial-2008_03_14-oth-JPR@CDS-CDS_Developer



Phần 2:

Đặc điểm về kiểu
dữ liệu



Các kiểu dữ liệu

- Trong Java có thể xem như tồn tại 2 kiểu dữ liệu chính: primitive type và Object (class) type

Primitive type	Object type
int long float char byte short double -> các kiểu dữ liệu cơ sở	String Wrapper (Integer, Long, Float...) Tất cả các kiểu dữ liệu dạng lớp (do Java cung cấp hoặc người dùng khai báo thêm)

Kiểu dữ liệu primitive: passing by value

Kiểu dữ liệu object: passing by reference

Các lớp wrapper cung cấp các hàm tiện ích cho các kiểu primitive:

Ví dụ `int i=Integer.parseInt("1234");`

Tất cả đều là pointer

- Có thể xem tất cả các đối tượng của các lớp đều là pointer trong ngôn ngữ C++.
- Luôn luôn phải new trước khi dùng (trừ lớp String)

Ví dụ :

C++	Java
<pre>CPhanSo a; a.TuSo=5; CPhanSo *b; b=new CPhanSo(); b-> TuSo=5; delete b;</pre>	<pre>CPhanSo a=new CPhanSo(); a.TuSo=5; a=null; //thu hồi bộ nhớ Không sử dụng từ khóa delete trong Java, bản thân Java có cơ chế Garbage collector, tự thu hồi vùng nhớ không dùng đến.</pre>



Mảng

- Mảng trong ngôn ngữ Java có cấu trúc tương tự C/C++:

```
int [] a = new int[5]; hoặc int a[] = new int [5];
```

- Lấy kích thước một mảng (C++ không hỗ trợ):

```
int size=a.size;
```

Hủy mảng:

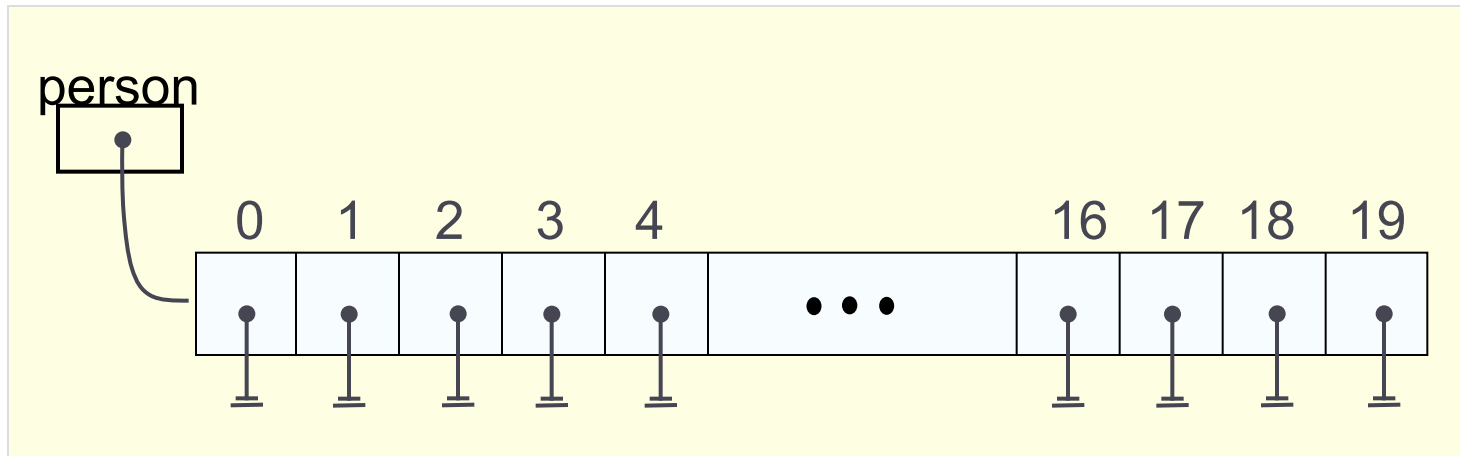
```
a=null;
```



Một lưu ý khi dùng mảng

```
Person [] person=new Person[20];  
person[0].setName("Nguyen Van A"); // báo lỗi
```

- Lý do: Con trỏ person[0] vẫn là NULL.



- Khắc phục:

```
person[0] = new Person();  
person[0].setName("Nguyen Van A");
```


Chuỗi:

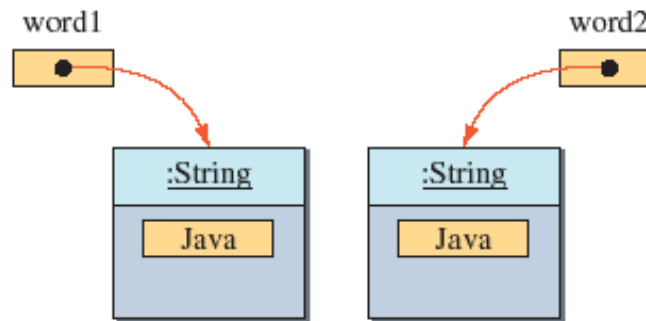
- Riêng với lớp String, có thể dùng từ khóa new hoặc không dùng. Để so sánh 2 chuỗi giống nhau, ta dùng hàm equals của lớp String

```
String word1, word2;
```

```
word1 = new String("Java");
```

```
word2 = new String("Java");
```

Whenever the **new** operator is used, there will be a new object.



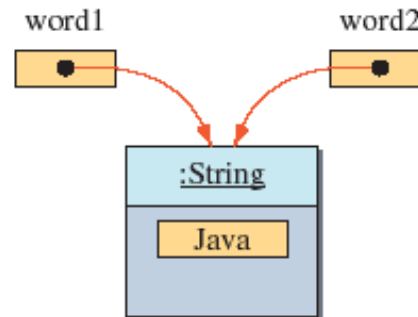
word1 == word2 → false

```
String word1, word2;
```

```
word1 = "Java";
```

```
word2 = "Java";
```

Literal string constant such as "Java" will always refer to one object.



word1 == word2 → true

-
- Xem thêm các hàm của các lớp Java hỗ trợ sẵn tại package `java.lang`



Phần 3:

Hướng đối tượng với Java



Lớp

- Một ứng dụng Java là tập hợp các lớp (class), bản thân hàm main cũng nằm trong một lớp cụ thể nào đó.
- Một ứng dụng chỉ được có một lớp chứa hàm main
- Một lớp nếu khai báo public (hoặc không chứa bất kỳ từ khóa private/public nào) phải được đặt trong file có **cùng tên** với tên lớp.



Khai báo lớp

- Lớp test phải được đặt trong file **test.java**

```
//các câu lệnh import  
import java.io.*;
```

```
.....
```

```
class test{  
    //các hàm  
    public static void main(String[] args){  
    }  
}
```



Package

- Các lớp có chức năng gần giống nhau thường được gom nhóm vào thành các gói (package).
- Ngoài ra, việc đặt tên gói còn giúp phân biệt các lớp cùng tên với nhau. Ví dụ user X và Y đều xây dựng lớp Math, ta có thể đặt tên gói là packx, packy và tên lớp tương ứng sẽ là packx.Math, packy.Math.
- Quy tắc đặt tên gói được SUN gợi ý là đặt theo tên miền, ví dụ các gói của công ty Microsoft thường được đặt là com.microsoft và thường được viết thường.
- Để sử dụng thuận tiện các gói và tên lớp quá dài, có thể sử dụng từ khóa import



Từ khóa import

- Giả sử trong source code ta sử dụng lớp `javax.swing.JOptionPane` (tương tự như `MessageDialog` của C++), ta có thể khai báo

`javax.swing.JOptionPane` pane;

- Ta có thể dùng từ khóa `import` (ở phía trước khai báo lớp)

`import javax.swing.JOptionPane;`

`class`

sau đó để khai báo biến ta có thể viết ngắn gọn

`JOptionPane` pane;

- Để import tất cả các lớp trong 1 package, ta dùng dấu `*`:

`import javax.swing.*;`

- Có thể xem từ khóa `import` gần giống từ khóa `#include` của ngôn ngữ C++, tuy nhiên trong C++ bắt buộc phải có, trong Java có thể sử dụng `import` hoặc không (optional).



Khai báo package

- Để khai báo một lớp thuộc một package nào đó, ta dùng từ khóa package. Ví dụ:

```
package vietnam;
```

```
class A{ ....}
```

- Như vậy ta đã khai báo class A thuộc package vietnam, tên đầy đủ của class A là vietnam.A
- Cấu trúc lưu trữ: lớp được đặt trong file .java có tên cùng với tên lớp. Package thể hiện bởi một thư mục cùng tên. Giả sử source code được lưu trong thư mục src, lớp A tương ứng bên trên sẽ có đường dẫn: **src/vietnam/A.java**



inner class

- Ngôn ngữ Java cho phép khai báo 1 lớp bên trong phạm vi một lớp khác, gọi là inner class. Inner class này có thể dạng static hoặc dạng thông thường.
- Ví dụ:

```
class A{  
    class B{...}  
    static class C{...}  
}
```

- Các lớp nội (inner class) có thể sử dụng lại các biến của lớp cha (kể cả private), riêng lớp static inner class chỉ có thể sử dụng lại thành phần static của lớp cha.



Kế thừa

- Để xây dựng một lớp kế thừa từ lớp khác, ta dùng từ khóa `extends`
- Ví dụ: `class A extends B{....}`
- Khái niệm kế thừa của Java tương tự khái niệm **kế thừa public** của ngôn ngữ C++ (`class A : public B`)
- Các từ khóa phạm vi truy cập (access modifier):
 - `public`: bất cứ nơi đâu
 - `private`: chỉ trong lớp hiện tại
 - `protected`: trong lớp hiện tại và các lớp con
 - `default` (không có bất kỳ từ khóa nào): được phép truy cập từ những lớp trong cùng package.



Lớp abstract

- Để ngăn không cho tạo 1 đối tượng của 1 lớp, ta có thể dùng từ khóa `abstract`.
- Giả sử ta đang xây dựng 1 bài toán quản lý nhân viên. Nhân viên của 1 công ty có thể gồm 2 dạng là `NVKyThuat` và `NVVanPhong`. Theo thiết kết HĐT, ta có thể xây dựng lớp `NhanVien` là lớp cha:

```
class NhanVien{...}
class NVKyThuat extends NhanVien{...}
class NVVanPhong extends NhanVien{...}
```
- Lớp `NhanVien` dùng để cài đặt các phần chung của 2 loại nhân viên, ta không muốn cho phép người dùng tạo ra object của lớp này, ta khai báo:
`abstract class NhanVien{ ... }`
- Khi đó câu lệnh `NhanVien n = new NhanVien()` sẽ bị báo lỗi.

Hàm abstract

- Trong lớp NhanVien ta sẽ cài đặt một số hàm virtual, ví dụ như hàm Work()

```
abstract class NhanVien{  
    .....  
    public void Work() { } //hàm rỗng  
}
```

- Các lớp con NVKyThuat, NVVanPhong sẽ cài đặt lại hàm Work() tùy loại NhanVien. Tuy nhiên, lập trình viên có thể không xây dựng hàm này ở lớp con (cố tình/ vô ý). Để bắt buộc lập trình viên phải xây dựng lại hàm này, ta có thể dùng từ khóa abstract

```
abstract class NhanVien{  
    .....  
    public abstract void Work(); //chú ý dấu ;  
}
```



Hàm abstract

- Lúc này các lớp con bắt buộc phải cài đặt lại hàm `Work()`, nếu không các lớp con cũng sẽ là lớp abstract và không được phép khởi tạo object (các trình biên dịch sẽ cảnh báo khi biên dịch).
- Một lớp abstract có thể có hoặc không có hàm abstract.



Interface

- Trong Java không hỗ trợ khái niệm đa kế thừa, ví dụ:
class A **extends B,C** <- báo lỗi biên dịch
- Khái niệm đa kế thừa trong Java được hỗ trợ thông qua interface. Một interface có thể xem như một lớp, trong lớp đó chỉ được khai báo các hằng số (từ khóa final) và các hàm ảo.

```
interface IMyinterface{  
    final float PI=3.1415926;  
    void Work();  
    void Swim();  
    void Sing();  
}
```

- Các hàm trong interface đều là hàm ảo (không cần từ khóa abstract) và luôn là hàm public (không cần từ khóa public).
-



Interface

- Java cho phép một lớp implements nhiều interface cùng lúc. Ví dụ:

```
class A extends B implements C,D {  
    //các hàm của C  
    .....  
    //các hàm của D  
    ....  
}
```

- Khi implements một interface, bắt buộc phải cài đặt lại tất cả các hàm của interface đó qui định.
- Câu hỏi: Interface khác gì với một lớp chỉ toàn hàm abstract? Trả lời: Interface hỗ trợ “đa kế thừa”, các class thì không.



Từ khóa final

- Nếu sử dụng với biến, có tác dụng định nghĩa các hằng số (constant).

`final float PI=3.14;`

- Nếu sử dụng trước tên lớp, có ý nghĩa ngăn không cho lớp khác kế thừa lại lớp này:

Ví dụ:

`final class A{...}`

~~`class B extends A {...}`~~



Phần 4:

Exception Handling



Cơ chế xử lý lỗi thông qua Exception

- Java xử lý lỗi thông qua Exception. Khi một lỗi xảy ra, ta gọi là phát sinh một Exception, ta có thể “catch” exception này để xử lý.

- Ví dụ:

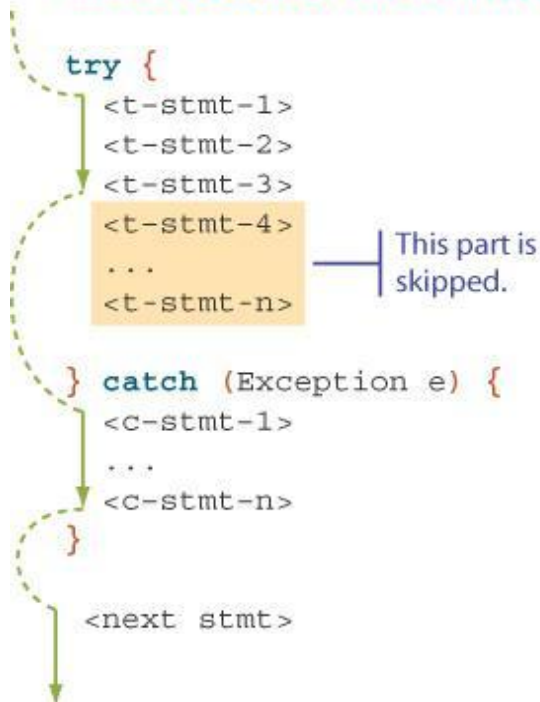
```
try{
    String number="nine";
    int i=Integer.parseInt(number);
}
catch (Exception e)
{
    System.out.println("Co loi xay ra, message:"+e.getMessage() );
    e.printStackTrace(); //xuất ra thứ tự các hàm gây lỗi
}
```



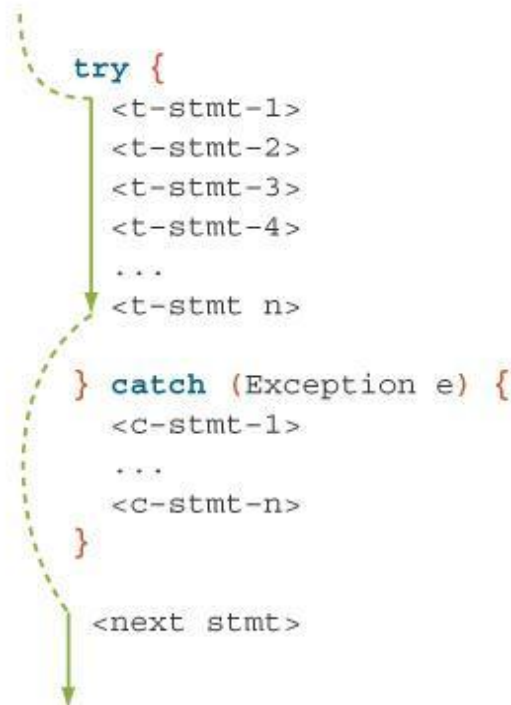
- Khi không có lỗi, đoạn code nằm giữa catch sẽ không được xử lý

Exception

Assume `<t-stmt-3>` throws an exception.



No Exception



Exception

- Một đoạn lệnh try có thể tương ứng nhiều câu lệnh catch:

```
int a[] = {1,2,3,4,5};
```

```
try{
```

```
    String s = ..... ; //Lấy dữ liệu từ người dùng nhập.
```

```
    int i=Integer.parseInt(s);
```

```
    System.out.println("Gia tri a[" + i + "]="+a[i]);
```

```
}
```

```
catch (NumberFormatException e)
```

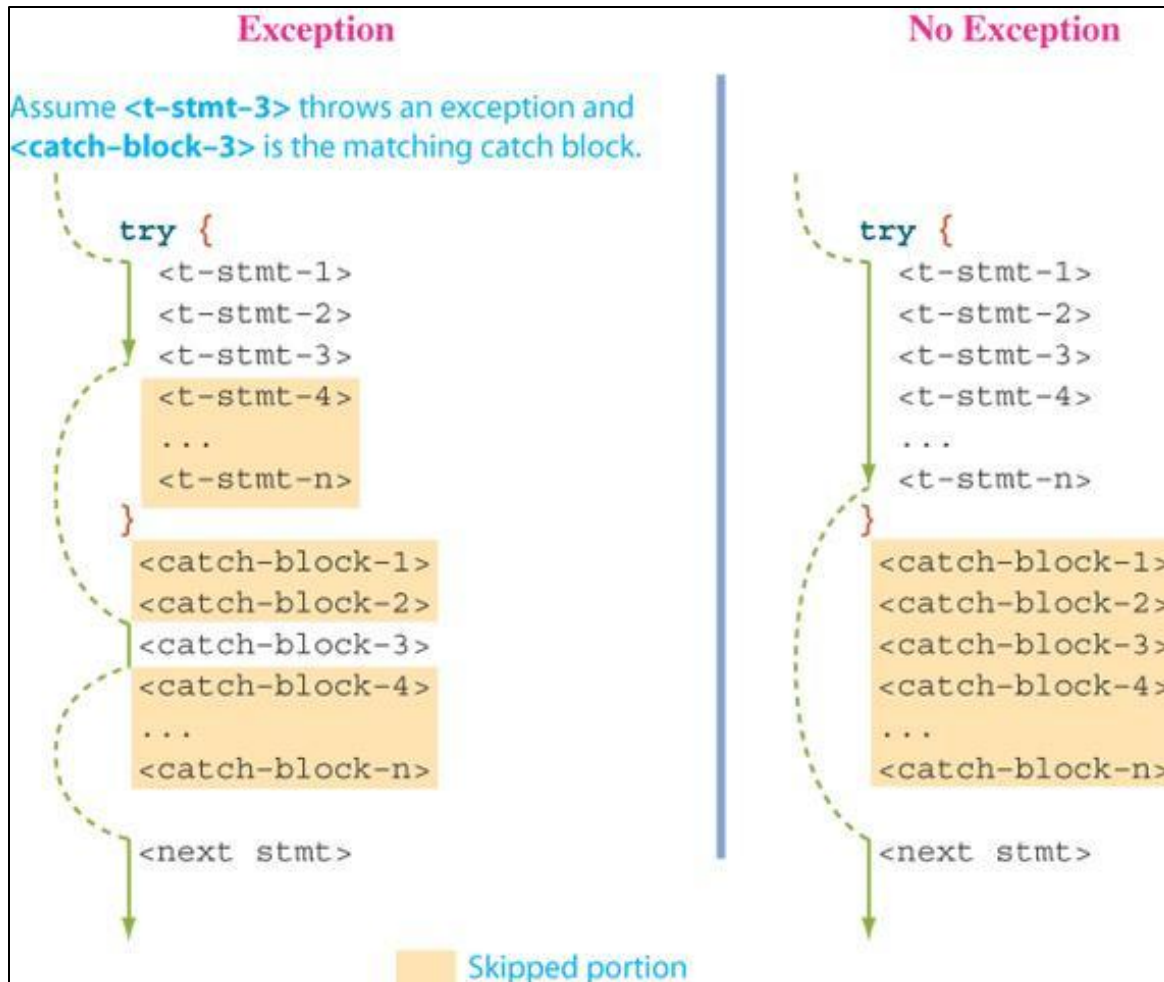
```
{.....}
```

```
catch (ArrayIndexOutOfBoundsException e)
```

```
{...}
```

- Nếu người dùng nhập s là một chuỗi không thể biến thành số, đoạn `NumberFormatException` được thực thi, nếu i là một số ≥ 5 , đoạn code `ArrayIndexOutOfBoundsException` sẽ thực thi.





Từ khóa throws và throw

- Để báo hiệu một hàm có thể phát sinh exception, ta dùng từ khóa throws. Ví dụ:

```
public int getUserData() throws NumberFormatException {  
    String s= JOptionPane.showInputDialog(null,"Nhập  
    vào một số:");  
    //câu lệnh có khả năng phát sinh exception  
    int i=Integer.parseInt(s);  
    return i;  
}
```

Khi đó, lời gọi hàm getUserData() phải được đặt vào try, catch...



-
- Để phát sinh một exception, ta dùng từ khóa throw
 - Ví dụ:

```
public void rutTien(float soTien) {  
    if (soDu < soTien){  
        throw new Exception("Rut qua so du");  
    }  
    else  
        soDu-=soTien;  
}
```



Phần 5:

Collection Framework



Collection Framework

- Mảng trong Java, cũng như trong ngôn ngữ C++ cần biết trước số lượng phần tử khi cấp phát.
- Ví dụ: `int[] a=new int[5];`
- Để giải quyết vấn đề lưu trữ “động”, có thể dùng các lớp trong gói `java.util`;
- Collection là interface “cha” của các interface khác trong Collection Framework



Collection interface

boolean	<u>add</u> (<u>E</u> e) Ensures that this collection contains the
boolean	<u>addAll</u> (<u>Collection</u> <? extends <u>E</u> > c) Adds all of the elements in the specified
void	<u>clear</u> () Removes all of the elements from this co
boolean	<u>contains</u> (<u>Object</u> o) Returns true if this collection contains t
boolean	<u>containsAll</u> (<u>Collection</u> <?> c) Returns true if this collection contains a
boolean	<u>equals</u> (<u>Object</u> o) Compares the specified object with this
int	<u>hashCode</u> () Returns the hash code value for this coll
boolean	<u>isEmpty</u> () Returns true if this collection contains r
<u>Iterator</u> < <u>E</u> >	<u>iterator</u> () Returns an iterator over the elements in
boolean	<u>remove</u> (<u>Object</u> o) Removes a single instance of the specifi
boolean	<u>removeAll</u> (<u>Collection</u> <?> c) Removes all of this collection's elements
boolean	<u>retainAll</u> (<u>Collection</u> <?> c)

List

- Là một **Interface** hỗ trợ cho việc lưu trữ theo dạng tập hợp: $L = (l_0, l_1, l_2, \dots, l_N)$

<code>boolean add (Object o)</code>
Adds an object o to the list
<code>void clear ()</code>
Clears this list, i.e., make the list empty
<code>Object get (int idx)</code>
Returns the element at position idx
<code>boolean remove (int idx)</code>
Removes the element at position idx
<code>int size ()</code>
Returns the number of elements in the list
<code>Iterator iterator() // Browse by iterator</code>

-
- Hai lớp (class) implements interface List là
 - ArrayList
 - LinkedList



Map

- Map là một Interface phục vụ cho việc lưu trữ theo dạng (key, value) (gần giống khái niệm session trong lập trình web).

key	value
k_0	v_0
k_1	v_1
.	.
.	.
.	.
k_n	v_n



- Các hàm Map interface hỗ trợ:

<code>void clear ()</code>
<code>Clears this list, i.e., make the map empty</code>
<code>boolean containsKey (Object key)</code>
<code>Returns true if the map contains an entry with a given key</code>
<code>Object put (Object key, Object value)</code>
<code>Adds the given (key, value) entry to the map</code>
<code>Object get (Object key); //return value</code>
<code>boolean remove (Object key)</code>
<code>Removes the entry with the given key from the map</code>
<code>int size ()</code>
<code>Returns the number of elements in the map</code>



• Các lớp cài đặt Map: HashMap, TreeMap

```
import java.util.*;

Map        catalog;
catalog = new TreeMap( );

catalog.put("CS101", "Intro Java Programming");
catalog.put("CS301", "Database Design");
catalog.put("CS413", "Software Design for Mobile Devices");

if (catalog.containsKey("CS101")) {
    System.out.println("We teach Java this semester");
} else {
    System.out.println("No Java courses this semester");
}
```



Set & SortedSet

- Set: tương tự như List, tuy nhiên không cho phép lưu các phần tử giống nhau trong cùng 1 set.
- SortedSet: Hỗ trợ cơ chế sắp xếp tự động, người dùng định nghĩa 1 tiêu chí, khi add thêm phần tử vào set này phần tử sẽ tự động được sắp xếp vào vị trí phù hợp (các tiêu chí có thể như: sắp xếp theo MaSV, Điểm TB, tuổi...)



Iterator

- Duyệt Collection theo Iterator: hầu như mọi lớp trong collection framework đều hỗ trợ hàm `Iterator iterator()`; để duyệt danh sách.
- Các hàm Iterator hỗ trợ:

Method Summary	
boolean	<u>hasNext</u> () Returns true if the iteration has more elements.
<u>E</u>	<u>next</u> () Returns the next element in the iteration.
void	<u>remove</u> () Removes from the underlying collection the last element returned by the iterator (optional operation).



- Đối với List còn hỗ trợ duyệt theo ListIterator

Method Summary

void	<u>add</u> (<u>E</u> e) Inserts the specified element into the list (optional operation).
boolean	<u>hasNext</u> () Returns true if this list iterator has more elements when traversing the list in the forward direction.
boolean	<u>hasPrevious</u> () Returns true if this list iterator has more elements when traversing the list in the reverse direction.
<u>E</u>	<u>next</u> () Returns the next element in the list.
int	<u>nextIndex</u> () Returns the index of the element that would be returned by a subsequent call to next.
<u>E</u>	<u>previous</u> () Returns the previous element in the list.
int	<u>previousIndex</u> () Returns the index of the element that would be returned by a subsequent call to previous.
void	<u>remove</u> () Removes from the list the last element that was returned by next or previous (optional operation).
void	<u>set</u> (<u>E</u> e) Replaces the last element returned by next or previous with the specified element (optional operation).

Enumeration

- Một số lớp hỗ trợ duyệt theo cơ chế Enumeration thông qua hàm `elements()`

Method Summary	
boolean	<u>hasMoreElements</u> () Tests if this enumeration contains more elements.
<u>E</u>	<u>nextElement</u> () Returns the next element of this enumeration if this enumeration object has at least one more element to provide.



Vector

- Có thể dùng lớp Vector, có vai trò gần tương tự như List.

boolean	<u>add</u> (<u>E</u> e) Appends the specified element.
void	<u>add</u> (int index, <u>E</u> element) Inserts the specified element at the specified position.
boolean	<u>addAll</u> (<u>Collection</u> <? extends <u>E</u> > c) Appends all of the elements from the specified Collection to the end of this Vector, using the Collection's Iterator.
boolean	<u>addAll</u> (int index, <u>Collection</u> <? extends <u>E</u> > c) Inserts all of the elements from the specified Collection into this Vector at the position indicated by the index.
void	<u>addElement</u> (<u>E</u> obj) Adds the specified component to the end of this Vector.
int	<u>capacity</u> () Returns the current capacity of this Vector.
void	<u>clear</u> () Removes all of the elements from this Vector.
<u>Object</u>	<u>clone</u> () Returns a clone of this Vector.
boolean	<u>contains</u> (<u>Object</u> o) Returns true if this Vector contains the specified element.
boolean	<u>containsAll</u> (<u>Collection</u> <? <u>Object</u> > c) Returns true if this Vector contains all of the elements in the specified Collection.
void	<u>copyInto</u> (<u>Object</u> [] anArray) Copies the components of this Vector into the array.
<u>E</u>	<u>elementAt</u> (int index) Returns the element at the specified position in this Vector.

Đổi mới từ phiên bản Java 1.5

- Cách sử dụng chung:

```
ArrayList list = new ArrayList();
```

```
list.add (new String("hello"));
```

```
String s=(String)list.get(0);
```

- Từ phiên bản Java 1.5 có thể sử dụng:

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

```
list.add(new String("hello"));
```

```
String s=list.get(0); //Không cần ép kiểu
```



Phần 6

Regular Expression



Regular Expression

- Cơ chế so khớp chuỗi của Java
- Ví dụ muốn kiểm tra dữ liệu người dùng nhập vào có phải là số điện thoại hay không (6 chữ số), ta xây dựng một bộ mẫu

```
String pattern = "[0-9][0-9][0-9] [0-9][0-9][0-9]";
```

```
String data=...; //lấy dữ liệu input từ người dùng
```

```
boolean b=data.matches(pattern);
```



Xây dựng pattern:

- Rules

- The brackets **[]** represent choices
- The asterisk symbol ***** means zero or more occurrences.
- The plus symbol **+** means one or more occurrences.
- The hat symbol **^** means negation.
- The hyphen **–** means ranges.
- The parentheses **()** and the vertical bar **|** mean a range of choices for multiple characters.



Sample rules

Expression	Description
[013]	A single digit 0, 1, or 3.
[0-9][0-9]	Any two-digit number from 00 to 99.
[0-9&&[^4567]]	A single digit that is 0, 1, 2, 3, 8, or 9.
[a-z0-9]	A single character that is either a lowercase letter or a digit.
[a-zA-Z][a-zA-Z0-9_]*	A valid Java identifier consisting of alphanumeric characters, underscores, and dollar signs, with the first character being an alphabet.
[wb](ad eed)	Matches wad, weed, bad, and bead.
(AZ CA CO)[0-9][0-9]	Matches AZxx, CAxx, and COxx, where x is a single digit.



Một số luật bổ sung

- \wedge :The beginning of a line
- $\$$:The end of a line
- $X?$: X , once or not at all
- X^* : X , zero or more times
- X^+ : X , one or more times
- $X\{n\}$: X , exactly n times
- $X\{n,\}$: X , at least n times
- $X\{n,m\}$: X , at least n but not more than m times
- $.$: any character



Ví dụ:

- Kiểm tra chuỗi có chứa ký tự n ở đầu chuỗi hay không? `data.matches("^n.*");`
- Kiểm tra chuỗi có ký tự kế cuối là a hay không? `data.matches(".*a.$");`
- Kiểm tra chuỗi có chứa ít nhất 2 lần cụm từ xy liên nhau hay không? `data.matches(".*(xy){2,}.*");`
- Các mẫu so khớp khác: Tìm hiểu lớp **java.util.regex.Pattern**



Các vấn đề chưa đề cập:

- Java IO: Đọc ghi dữ liệu dạng binary/text/object với Java (xem slide đi kèm)
- Multi-threaded Programming và synchronization:
Xem thêm trong Java Tutorial
- Lập trình giao diện với thư viện Swing: Java Tutorial
- Kết nối CSDL : buổi tiếp theo.



Q & A

