

XỬ LÝ NGOẠI LỆ (Exception Handling)

Sau khi kết thúc chương này, bạn có thể nắm được các nội dung sau:

- Định nghĩa một ngoại lệ (exception)
- Hiểu được mục đích của việc xử lý ngoại lệ
- Hiểu được các kiểu ngoại lệ khác nhau trong Java
- Mô tả mô hình xử lý ngoại lệ
- Hiểu được các khối lệnh chứa nhiều catch
- Mô tả cách sử dụng các khối 'try', 'catch' và 'finally'
- Giải thích cách sử dụng các từ khoá 'throw' và 'throws'
- Tự tạo ra các ngoại lệ

7.1 Giới thiệu

Exception là một lỗi đặc biệt. Lỗi này xuất hiện vào lúc thực thi chương trình. Các trạng thái không bình thường xảy ra trong khi thi hành chương trình tạo ra các exception. Những trạng thái này không được biết trước trong khi ta đang xây dựng chương trình. Nếu bạn không phân phối các trạng thái này thì exception có thể bị kết thúc đột ngột. Ví dụ, việc chia cho 0 sẽ tạo một lỗi trong chương trình. Ngôn ngữ Java cung cấp bộ máy dùng để xử lý ngoại lệ rất tuyệt vời. Việc xử lý này làm hạn chế tối đa trường hợp hệ thống bị phá vỡ (crash) hay hệ thống bị ngắt đột ngột. Tính năng này làm cho Java là một ngôn ngữ lập trình mạnh.

7.2 Mục đích của việc xử lý ngoại lệ

Một chương trình nên có cơ chế xử lý ngoại lệ thích hợp. Nếu không, chương trình sẽ bị ngắt khi một exception xảy ra. Trong trường hợp đó, tất cả các nguồn tài nguyên mà hệ thống trước kia phân phối sẽ được di dời trong cùng trạng thái. Điều này gây lãng phí tài nguyên. Để tránh trường hợp này, tất cả các nguồn tài nguyên mà hệ thống phân phối nên được thu hồi lại. Tiến trình này đòi hỏi cơ chế xử lý ngoại lệ thích hợp.

Cho ví dụ, xét thao tác nhập xuất (I/O) trong một tập tin. Nếu việc chuyển đổi kiểu dữ liệu không thực hiện đúng, một ngoại lệ sẽ xảy ra và chương trình bị hủy mà không đóng lại tập tin. Lúc đó tập tin dễ bị hư hại và các nguồn tài nguyên được cấp phát cho tập tin không được thu hồi lại cho hệ thống.

7.3 Xử lý ngoại lệ

Khi một ngoại lệ xảy ra, đối tượng tương ứng với ngoại lệ đó được tạo ra. Đối tượng này sau đó được truyền cho phương thức là nơi mà ngoại lệ xảy ra. Đối tượng này chứa thông tin chi tiết về ngoại lệ. Thông tin này có thể được nhận về và được xử lý. Các môi trường runtime như 'IllegalAccessException', 'EmptyStackException' v.v... có thể chặn được các ngoại lệ. Đoạn mã trong chương trình đôi khi có thể tạo ra các ngoại lệ. Lớp 'Throwable' được Java cung cấp là lớp trên nhất của lớp Exception, lớp này là lớp cha của các ngoại lệ

khác nhau.

7.4 Mô hình xử lý ngoại lệ

Trong Java, mô hình xử lý ngoại lệ kiểm tra việc xử lý những hiệu ứng lề (lỗi), được biết đến là mô hình 'catch và throw'. Trong mô hình này, khi một lỗi xảy ra, một ngoại lệ sẽ bị chặn và được đưa vào trong một khối. Người lập trình viên nên xét các trạng thái ngoại lệ độc lập nhau từ việc điều khiển thông thường trong chương trình. Các ngoại lệ phải được bắt giữ nếu không chương trình sẽ bị ngắt.

Ngôn ngữ Java cung cấp 5 từ khoá sau để xử lý các ngoại lệ:

- try
- catch
- throw
- throws
- finally

Dưới đây là cấu trúc của mô hình xử lý ngoại lệ:

```
try
{
    // place code that is expected to throw an exception
}
catch(Exception e1)
{
    // If an exception is thrown in 'try', which is of type e1, then perform
    // necessary actions here, else go to the next catch block
}
catch(Exception e2)
{
    // If an exception is thrown in, try which is of type e2, then perform
    // necessary actions here, else go to the next catch block
}
catch(Exception eN)
{
    // If an exception is thrown in, try which is of type eN, then perform
    // necessary actions here, else go to the next catch block
}
finally
{
    // this block is executed, whether or not the exception is throw.
}
```

7.4.1 Các ưu điểm của mô hình 'catch và throw'

Mô hình 'catch và throw' có hai ưu điểm:

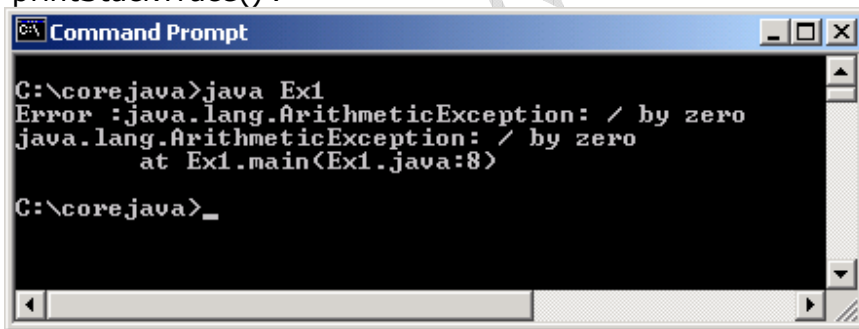
- Người lập trình viên phải phân phối trạng thái lỗi chỉ vào những nơi cần thiết. Không cần phải thực hiện tại mọi mức.
- Một thông báo lỗi có thể được in ra khi tiến hành xử lý ngoại lệ.

7.4.2 Các khối 'try' và 'catch'

Khối 'try-catch' được sử dụng để thi hành mô hình 'catch và throw' của việc xử lý ngoại lệ. Khối 'try' chứa một bộ các lệnh có thể thi hành được. Các ngoại lệ có thể bị chặn khi thi hành những câu lệnh này. Phương thức dùng để chặn ngoại lệ có thể được khai báo trong khối 'try'. Một hay nhiều khối 'catch' có thể theo sau khối 'try'. Các khối 'catch' này bắt các ngoại lệ bị chặn trong khối 'try'. Hãy nhìn khối 'try' dưới đây:

```
try
{
    doFileProcessing(); // user-defined method
    displayResults();
}
catch (Exception e) // exception object
{
    System.err.println("Error :" + e.toString());
    e.printStackTrace();
}
```

Ở đây, 'e' là đối tượng của lớp 'Exception'. Chúng ta có thể sử dụng đối tượng này để in các chi tiết về ngoại lệ. Các phương thức 'toString' và 'printStackTrace' được sử dụng để mô tả các exception phát sinh ra. Hình sau chỉ ra kết xuất của phương thức 'printStackTrace()'.



```
Command Prompt
C:\corejava>java Ex1
Error :java.lang.ArithmeticException: / by zero
java.lang.ArithmeticException: / by zero
    at Ex1.main(Ex1.java:8)
C:\corejava>_
```

Hình 7.1 Khối Try và Catch

Để bắt giữ bất cứ ngoại lệ nào, ta phải chỉ ra kiểu ngoại lệ là 'Exception'.

catch(Exception e)

Khi ngoại lệ bị bắt giữ không biết thuộc kiểu nào, chúng ta có thể sử dụng lớp 'Exception' để bắt ngoại lệ đó.

Khối `catch()` bắt giữ bất cứ các lỗi xảy ra trong khi thi hành phương thức `doFileProcessing` hay `display`. Nếu một lỗi xảy ra trong khi thi hành phương thức `doFileProcessing()`, lúc đó phương thức `displayResults()` sẽ không bao giờ được gọi. Sự thi hành sẽ tiếp tục thực hiện khối `catch`. Để có nhiều lớp xử lý lỗi hơn, như là `LookupException` thay vì một đối tượng ngoại lệ chung (Exception e), lỗi thật sự sẽ là một instance của `LookupException` hay một trong số những lớp con của nó. Lỗi sẽ được truyền qua khối `try catch` cho tới khi chúng bắt gặp một `catch` tham chiếu tới nó hay toàn bộ chương trình phải bị huỷ bỏ.

7.5 Các khối chứa nhiều Catch

Các khối chứa nhiều `catch` xử lý các kiểu ngoại lệ khác nhau một cách độc lập. Chúng được liệt kê trong đoạn mã sau:

```
try
{
    doFileProcessing(); // user defined method
    displayResults(); // user defined method
}
catch(LookupException e) // e - Lookupexception object
{
    handleLookupException(e); // user defined handler
}
catch(Exception e)
{
    System.err.println("Error:" + e.printStackTrace());
}
}
```

Trong trường hợp này, khối `catch` đầu tiên sẽ bắt giữ một `LockupException`. Khối `catch` thứ hai sẽ xử lý kiểu ngoại lệ khác với khối `catch` thứ nhất.

Một chương trình cũng có thể chứa các khối `try` lồng nhau. Ví dụ đoạn mã dưới đây:

```
try
{
    statement 1;
    statement 2;
    try
    {
        statement1;
        statement2;
    }
    catch(Exception e) // of the inner try block
    {
    }
}
}
```

```
catch(Exception e) // of the outer try block
{
}
...
```

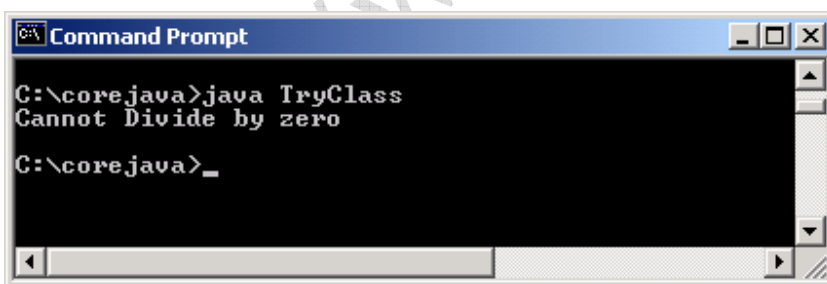
Khi sử dụng các 'try' lồng nhau, khối 'try' bên trong được thi hành đầu tiên. Bất kỳ ngoại lệ nào bị chặn trong khối 'try' sẽ bị bắt giữ trong các khối 'catch' theo sau. Nếu khối 'catch' thích hợp không được tìm thấy thì các khối 'catch' của các khối 'try' bên ngoài sẽ được xem xét. Nếu không, Java Runtime Environment xử lý các ngoại lệ.

chương trình 7.1 minh họa cách sử dụng các khối 'try' và 'catch'.

Chương trình 7.1

```
class TryClass
{
    public static void main(String args[])
    {
        int demo=0;
        try
        {
            System.out.println(20/demo);
        }
        catch(ArithmeticException a)
        {
            System.out.println("Cannot Divide by zero");
        }
    }
}
```

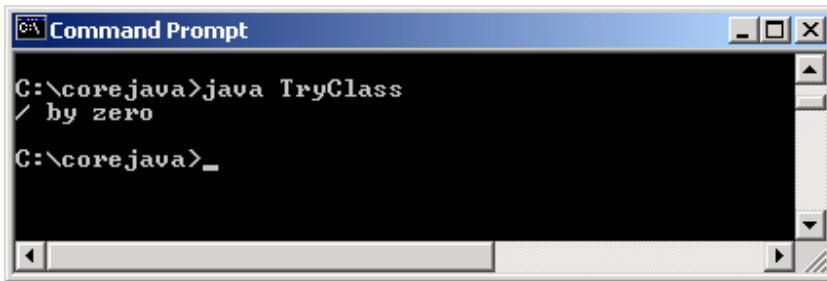
Kết xuất của chương trình:



Hình 7.2 ArithmeticException

Trong chương trình này, một số được chia cho 0. Đây không là toán tử số học hợp lệ. Do đó một ngoại lệ bị chặn và được bắt giữ trong khối catch. Khi người lập trình viên nhận biết được loại ngoại lệ nào có thể xảy ra, anh ta hay cô ta viết một câu lệnh trong khối 'catch'. Ở đây, 'a' được sử dụng như một đối tượng của ArithmeticException để in các chi tiết về các toán tử ngoại lệ mà hệ thống cung cấp. Nếu bạn thay thế lệnh

'System.out.println' của khối 'catch' bằng lệnh '**System.out.println(a.getMessage())**' thì kết xuất của chương trình như sau:



Hình 7.3 Câu thông báo lỗi

Khi các khối 'try' được sử dụng mà không có các khối 'catch' nào, chương trình sẽ biên dịch mà không gặp sự cố nào nhưng sẽ bị ngắt khi thực thi. Bởi vì ngoại lệ đã xảy ra khi thực thi chương trình.

7.6 Khối 'finally'

Khi một ngoại lệ xuất hiện, phương thức đang được thực thi có thể bị dừng mà không được thi hành toàn vẹn. Nếu điều này xảy ra, thì các đoạn mã (ví dụ như đoạn mã với chức năng thu hồi tài nguyên có các lệnh đóng lại tập tin khai báo cuối phương thức) sẽ không bao giờ được gọi. Java cung cấp khối 'finally' để giải quyết việc này. Khối 'finally' thực hiện tất cả các việc thu dọn khi một ngoại lệ xảy ra. Khối này có thể được sử dụng kết hợp với khối 'try'. Khối 'finally' chứa các câu lệnh thu hồi tài nguyên về cho hệ thống hay lệnh in ra các câu thông báo. Các lệnh này bao gồm:

- Đóng tập tin.
- Đóng lại bộ kết quả (được sử dụng trong chương trình cơ sở dữ liệu).
- Đóng lại các kết nối được tạo trong cơ sở dữ liệu.

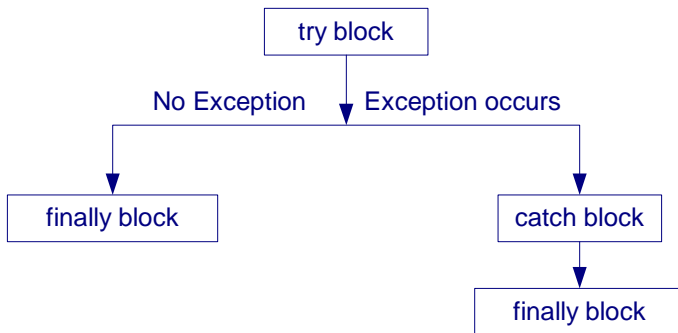
```
try
{
    doSomethingThatMightThrowAnException();
}
finally
{
    cleanup();
}
```

Phương thức 'cleanup()' được gọi nếu phương thức 'doSomethingThatMightThrowAnException()' chặn một ngoại lệ. Mặt khác 'cleanup()' cũng được gọi ngay khi không có ngoại lệ nào bị chặn và thi hành tiếp tục sau khối lệnh 'finally'.

Khối 'finally' là tùy ý, không bắt buộc. Khối này được đặt sau khối 'catch'. Hệ thống sẽ duyệt từ câu lệnh đầu tiên của khối 'finally' sau khi gặp câu lệnh 'return' hay lệnh 'break' được dùng trong khối 'try'.

Khối 'finally' bảo đảm lúc nào cũng được thực thi, bất chấp có ngoại lệ xảy ra hay không.

Hình 7.4 minh họa sự thi hành của các khối 'try', 'catch' và 'finally'.



Hình 7.4 Khối lệnh 'try', 'catch' và 'finally'

Hình 7.2 sử dụng khối 'finally'. Ở đây, khối 'finally' được thi hành bất chấp 'ArithmeticException' có xảy ra hay không. Khối này khai báo các hoạt động thu dọn.

Chương trình 7.2

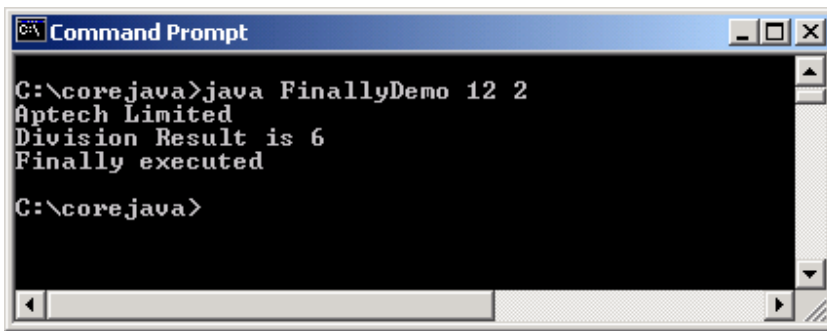
```
class FinallyDemo
{
    String name;
    int no1,no2;
    FinallyDemo(String args[])
    {
        try
        {
            name=new String("Aptech Limited");
            no1=Integer.parseInt(args[0]);
            no2=Integer.parseInt(args[1]);
            System.out.println(name);
            System.out.println("Division Result is" + no1/no2);
        }
        catch(ArithmeticException i)
        {
            System.out.println("Cannot Divide by zero");
        }
        finally
        {
            name=null; // clean up code
            System.out.println("Finally executed");
        }
    }
    public static void main(String args[])
```

```

    {
        new FinallyDemo(args);
    }
}

```

Kết xuất của chương trình:



```

C:\corejava>java FinallyDemo 12 2
Aptech Limited
Division Result is 6
Finally executed

C:\corejava>

```

Hình 7.5 Khối Finally

Trong ví dụ này, các câu lệnh trong khối 'Finally' luôn luôn thi hành, bất chấp ngoại lệ có xảy ra hay không. Trong kết xuất bên trên, khối 'finally' được thi hành mặc dù không có ngoại lệ xảy ra.

7.7 Các ngoại lệ được định nghĩa với lệnh 'throw' và 'throws'

Các ngoại lệ bị chặn với sự trợ giúp của từ khoá 'throw'. Từ khoá 'throw' chỉ ra một ngoại lệ vừa xảy ra. Toán tử của throw là một đối tượng của lớp, lớp này được dẫn xuất từ 'Throwable'.

Đoạn lệnh sau chỉ ra cách sử dụng của lệnh 'throw':

```

try
{
    if (flag<0)
    {
        throw new MyException(); // user-defined
    }
}

```

Một phương thức đơn có thể chặn nhiều ngoại lệ. Để xử lý những ngoại lệ này, ta cần cung cấp một danh sách các ngoại lệ mà phương thức chặn trong phần định nghĩa của phương thức. Giả sử rằng phương thức 'x()' gọi phương thức 'y()'. Phương thức 'y()' chặn một ngoại lệ không được xử lý. Trong trường hợp này, phương thức gọi 'x()' nên khai báo việc chặn cùng một ngoại lệ với phương thức được gọi 'y()'. Ta nên khai báo khối 'try catch' trong phương thức x() để đảm bảo rằng ngoại lệ không được truyền cho các phương thức mà gọi phương thức này.

Đoạn mã sau minh họa cách sử dụng của từ khoá 'throws' để xử lý nhiều ngoại lệ:

```
public class Example
{
    // multiple exceptions separated by a comma
    public void exceptionExample() throws ExException, LookupException
    {
        try
        {
            // statements
        }
        catch(ExException exmp)
        {
        }
        catch(LookupException lkpex)
        {
        }
    }
}
```

Trong ví dụ trên, phương thức 'exceptionExample' khai báo từ khoá 'throws'. Từ khoá này được theo sau bởi danh sách các ngoại lệ mà phương thức này có thể chặn – Trong trường hợp này là 'ExException' và 'LookupException'. Hàm xử lý ngoại lệ cho các phương thức này nên khai báo các khối 'catch' để có thể xử lý tất cả các ngoại lệ mà các phương thức chặn.

Lớp 'Exception' thực thi giao diện 'Throwable' và cung cấp các tính năng hữu dụng để phân phối các ngoại lệ. Ưu điểm của nó là tạo các lớp ngoại lệ được định nghĩa bởi người dùng. Để làm điều này, một lớp con của lớp Exception được tạo ra. Ưu điểm của lớp con là một kiểu ngoại lệ mới có thể bị bắt giữ độc lập từ các loại Throwable khác.

Chương trình 7.3 minh họa ngoại lệ được định nghĩa bởi người dùng 'ArraySizeException':

Chương trình 7.3

```
class ArraySizeException extends NegativeArraySizeException
{
    ArraySizeException() // constructor
    {
        super("You have passed an illegal array size");
    }
}
class ThrowDemo
{
    int size, array[];
    ThrowDemo(int s)
    {
        size=s;
    }
}
```

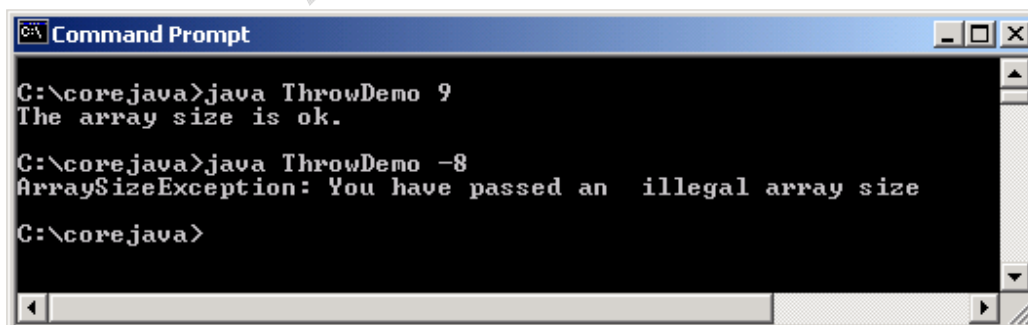
```

    try
    {
        checkSize();
    }
    catch(ArraySizeException e)
    {
        System.out.println(e);
    }
}
void checkSize() throws ArraySizeException
{
    if (size < 0)
        throw new ArraySizeException();
    else
        System.out.println("The array size is ok.");
    array = new int[3];
    for (int i=0; i<3; i++)
        array[i] = i+1;
}
public static void main(String arg[])
{
    new ThrowDemo(Integer.parseInt(arg[0]));
}
}

```

Lớp được định nghĩa bởi người dùng 'ArraySizeException' là lớp con của lớp 'NegativeArraySizeException'. Khi một đối tượng được tạo từ lớp này, thông báo về ngoại lệ được in ra. Phương thức 'checkSize()' được gọi để chặn ngoại lệ 'ArraySizeException' mà được chỉ ra bởi mệnh đề 'throws'. Kích thước của mảng được kiểm tra trong cấu trúc 'if'. Nếu kích thước là số âm thì đối tượng của lớp 'ArraySizeException' được tạo. Phương thức 'call()' được bao quanh trong khối 'try-catch', là nơi mà giá trị của đối tượng được in ra. Phương thức 'call()' cần được bao trong khối 'try', để cho khối 'catch' tương ứng có thể in ra giá trị.

Kết xuất của chương trình được chỉ ra ở hình 7.6.



```

Command Prompt
C:\corejava>java ThrowDemo 9
The array size is ok.

C:\corejava>java ThrowDemo -8
ArraySizeException: You have passed an illegal array size

C:\corejava>

```

Hình 7.6 Ngoại lệ tự định nghĩa

7.8 Danh sách các ngoại lệ

Bảng sau đây liệt kê một số ngoại lệ:

Ngoại lệ	Lớp cha của thứ tự phân cấp ngoại lệ
RuntimeException	Lớp cơ sở cho nhiều ngoại lệ java.lang
ArithmeticException	Trạng thái lỗi về số, ví dụ như 'chia cho 0'
IllegalAccessException	Lớp không thể truy cập
IllegalArgumentException	Phương thức nhận một đối số không hợp lệ
ArrayIndexOutOfBoundsException	Kích thước của mảng lớn hơn 0 hay lớn hơn kích thước thật sự của mảng
NullPointerException	Khi muốn truy cập đối tượng null
SecurityException	Việc thiết lập cơ chế bảo mật không được hoạt động
ClassNotFoundException	Không thể nạp lớp yêu cầu
NumberFormatException	Việc chuyển đổi không thành công từ chuỗi sang số thực
AWTException	Ngoại lệ về AWT
IOException	Lớp cha của các ngoại lệ I/O
FileNotFoundException	Không thể định vị tập tin
EOFException	Kết thúc một tập tin
NoSuchMethodException	Phương thức yêu cầu không tồn tại
InterruptedException	Khi một luồng bị ngắt

Bảng 7.1 Danh sách một số ngoại lệ

Tóm tắt

- Bất cứ khi nào một lỗi xuất hiện trong khi thi hành chương trình, nghĩa là một ngoại lệ đã xuất hiện.
- Ngoại lệ phát sinh vào lúc thực thi chương trình theo trình tự mã.
- Mỗi ngoại lệ phát sinh ra phải bị bắt giữ, nếu không ứng dụng sẽ bị ngắt.
- Việc xử lý ngoại lệ cho phép bạn kết hợp tất cả tiến trình xử lý lỗi trong một nơi. Lúc đó đoạn mã của bạn sẽ rõ ràng hơn.
- Java sử dụng các khối 'try' và 'catch' để xử lý các ngoại lệ. Các câu lệnh trong khối 'try' chặn ngoại lệ còn khối 'catch' xử lý ngoại lệ.
- Các khối chứa nhiều catch có thể được sử dụng để xử lý các kiểu ngoại lệ khác nhau theo cách khác nhau.
- Từ khoá 'throw' liệt kê các ngoại lệ mà phương thức chặn.
- Từ khoá 'throw' chỉ ra một ngoại lệ vừa xuất hiện.
- Khối 'finally' khai báo các câu lệnh trả về nguồn tài nguyên cho hệ thống và in những câu thông báo.

!!! Check your progress & Exercise !!!

<http://www.ngohaianh.info>