

# Nội dung môn học

---

Chương 1: Giới thiệu về NNLT Java

Chương 2: Hướng đối tượng trong Java

Chương 3: Các lớp tiện ích trong Java

Chương 4: Quản lý Exception

Chương 5: Nhập xuất

Chương 6: Xử lý đa luồng

Chương 7: Kết nối và thao tác CSDL với JDBC

Chương 8: Lập trình GUI với AWT & Swing

# QUẢN LÝ EXCEPTION

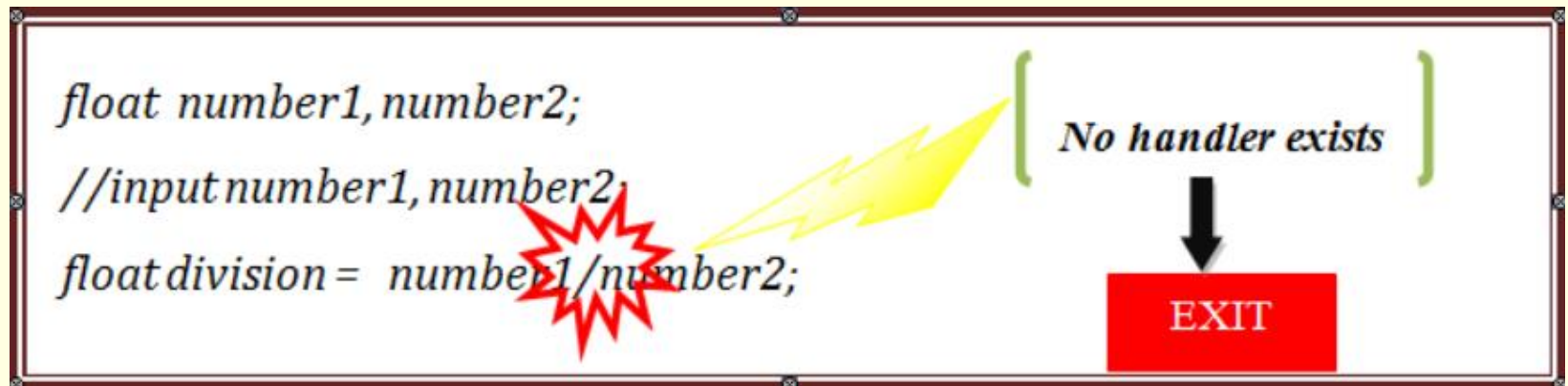
# Nội dung

---

1. Khái niệm ngoại lệ (Exception).
2. Xử lý Exception truyền thống. Hạn chế
3. Mô hình xử lý Exception.
4. Xử lý Exception trong java
  - Các loại Exception, Error
  - Cách quản lý (*try, catch, finally, throw, throws*)
  - Exception và Overriding

# Exception là gì?

- Exception là một sự kiện xuất hiện trong quá trình thực thi của chương trình và sẽ dừng luồng xử lý bình thường của chương trình. Nếu không xử lý thì CT kết thúc và trả quyền điều khiển cho HĐH [1].
- Exception, Error nên được report cho users và system administrators. Chương trình nên hồi phục và thực thi bình thường khi có Exception xảy ra.



# Tại sao cần quản lý Exception? (tt)

---

- ❑ Khi xảy ra ngoại lệ, nếu không có cơ chế xử lý thích hợp?
  - ❑ Chương trình bị ngắt khi ngoại lệ xảy ra
  - ❑ Các tài nguyên không được giải phóng → Lãng phí
- ❑ **Ví dụ: Vào/ra tệp tin**
  - ❑ Nếu ngoại lệ xảy ra (ví dụ như chuyển đổi kiểu không đúng) → Chương trình kết thúc mà không đóng tệp tin lại.
  - ❑ Tệp tin không thể truy cập/hởng
  - ❑ Tài nguyên cấp phát không được giải phóng

# Các xử lý truyền thông?

## ❑ Viết mã xử lý tại nơi phát sinh ra lỗi.

- Làm chương trình trở nên rối, khó hiểu

## ❑ Truyền trạng thái lên mức trên.

- Thông qua tham số, giá trị trả về hoặc biến flag.
- Dễ nhầm, khó hiểu

*Ví dụ:*

```
int devide(int num, int denom, int *error) {  
    if (denom != 0){  
        *error = 0;  
        return num/denom;  
    } else {  
        *error = 1;  
        return 0;  
    }  
}
```

# Các xử lý truyền thống

❑ Ví dụ: đọc file truyền thống.

```
errorCodeType readFile {  
    initialize errorCode = 0;  
    open the file;  
    if (theFileIsOpen) {  
        determine the length of the file;  
        if (gotTheFileLength) {  
            allocate that much memory;  
            if (gotEnoughMemory) {  
                read the file into memory;  
                if (readFailed) {  
                    errorCode = -1;  
                }  
            } else {  
                errorCode = -2;  
            }  
        }  
    }  
}
```

# Các xử lý truyền thống

---

```
    } else {  
        errorCode = -3;  
    }  
    close the file;  
    if (theFileDintClose && errorCode == 0) {  
        errorCode = -4;  
    } else {  
        errorCode = errorCode and -4;  
    }  
} else {  
    errorCode = -5;  
}  
return errorCode;  
}
```



# Hạn chế

---

- ❑ Khó kiểm soát hết các trường hợp
  - ❑ Lỗi số học, lỗi xuất nhập, lỗi bộ nhớ, ...
- ❑ Lập trình viên thường quên không nhớ xử lý lỗi.
  - ❑ Bản chất con người
  - ❑ Thiếu kinh nghiệm, cố tình bỏ qua

# Tại sao cần cơ chế mới quản lý Exception?

---

- ❑ Làm thế nào giúp developer kiểm soát lỗi dễ dàng. Chương trình đáng tin cậy hơn, tránh kết thúc bất thường?
  - Cơ chế quản lý Exception mới → Tách biệt codes có thể gây ra lỗi (Exception) và khối lệnh xử lý lỗi.
  - Cho phép tập trung viết mã cho luồng chính và xử lý trường hợp bất thường ở nơi khác.
  - Tránh việc phải lan truyền mã lỗi trong dây chuyền các lời gọi hàm (cho đến khi gặp hàm quan tâm đến việc xử lý lỗi)

# Tại sao cần cơ chế mới quản lý Exception?

```
method1 {  
    errorCodeType error;  
    error = call method2;  
    if (error)  
        doErrorProcessing;  
    else  
        proceed;  
}
```

```
errorCodeType method2 {  
    errorCodeType error;  
    error = call method3;  
    if (error)  
        return error;  
    else  
        proceed;  
}
```

```
errorCodeType method3 {  
    errorCodeType error;  
    error = call readFile;  
    if (error)  
        return error;  
    else  
        proceed;  
}
```

- Bắt buộc method2 và method3 truyền mã lỗi trả về bởi readFile cho đến khi mã lỗi tới được method1 (phương thức duy nhất quan tâm đến việc xử lý lỗi).

# Tại sao cần cơ chế mới quản lý Exception?

```
readFile {  
    try {  
        open the file;  
        determine its size;  
        allocate that much memory;  
        read the file into memory;  
        close the file;  
    } catch (fileOpenFailed) {  
        doSomething;  
    } catch (sizeDeterminationFailed) {  
        doSomething;  
    }  
    catch (memoryAllocationFailed) {  
        doSomething;  
    }  
}
```

```
catch (readFailed) {  
    doSomething;  
} catch (fileCloseFailed) {  
    doSomething;  
}  
}
```

```
method1 {  
    call method2;  
}  
method2 {  
    call method3;  
}  
method3 {  
    call readFile;  
}
```

# Mô hình xử lý ngoại lệ

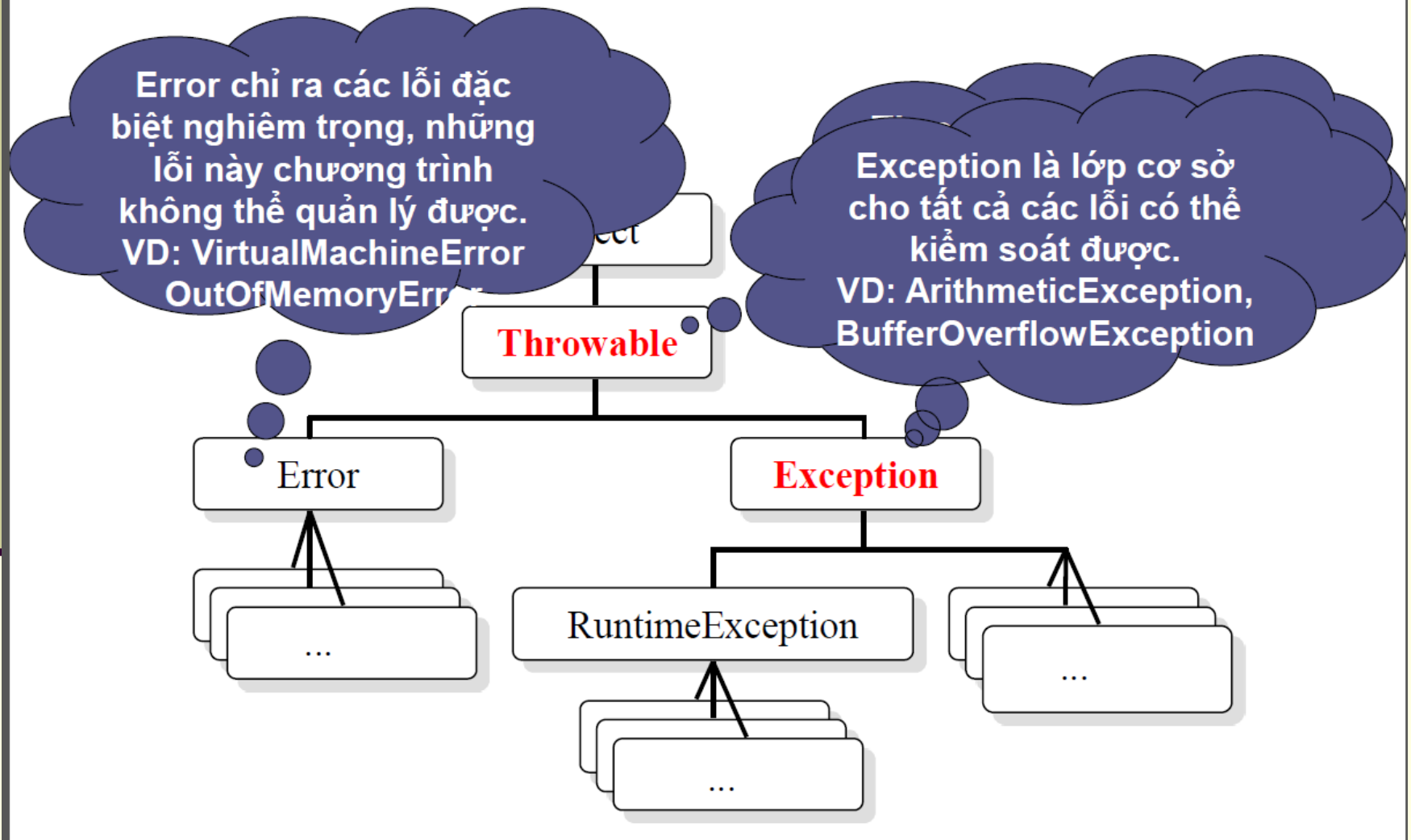
## ■ Hướng đối tượng

- Đóng gói các điều kiện không mong đợi trong một đối tượng.
- Khi xảy ra ngoại lệ, đối tượng tương ứng với ngoại lệ được tạo ra chứa thông tin chi tiết về ngoại lệ.
- Cung cấp cơ chế hiệu quả trong việc xử lý lỗi.
- Tách biệt luồng điều khiển bất thường với luồng bình thường.

```
Float totalpayment = getTotalPayment();  
Int numberOfEmployee = getNumberEmployee();  
Float averageSalary = totalpayment/numberOfEmployee;  
System.out.println("Average salary is " + averageSalary);
```

Handler

# Các loại Exception, Error trong Java



# Xử lý Exception trong java

- Một phương thức có thể ném ra các ngoại lệ trong thân của nó cho phương thức gọi nó “bắt”. Do vậy chỉ phương thức nào quan tâm đến lỗi mới phải phát hiện lỗi.

```
method1 {  
    try {  
        call method2;  
    } catch (exception e) {  
        doErrorProcessing;  
    }  
}  
  
method2 throws exception {  
    call method3;  
}  
  
method3 throws exception {  
    call readfile;  
}
```

# Xử lý Exception trong java (tt)

---

- Xử lý ngoại lệ trong Java được thực hiện theo mô hình hướng đối tượng:
  - *Tất cả các ngoại lệ đều là thể hiện của một lớp kế thừa từ lớp **Throwable** hoặc các lớp con của nó*
  - *Các đối tượng này có nhiệm vụ chuyển thông tin về ngoại lệ (loại và trạng thái của chương trình) từ vị trí xảy ra ngoại lệ đến nơi quản lý/xử lý nó.*



# Từ khóa – Khối try/catch

---

- Từ khóa

- *try*

- *catch*

- *finally*

- *throw*

- *throws*

# Từ khóa – Khối try/catch (tt)

- ❑ **Khối try/catch:** Phân tách đoạn chương trình thông thường và phần xử lý ngoại lệ

- *try { ... }:* Khối lệnh có khả năng gây ra ngoại lệ


- *catch() { ... }:* Bắt và xử lý với ngoại lệ

```
try {  
    // Doan ma co the gay ngoai le  
}  
catch (ExceptionType e) {  
    // Xu ly ngoai le  
}
```

- ❑ **ExceptionType** là một đối tượng của lớp **Throwable**.

# Ví dụ không xử lý ngoại lệ

```
class NoException {  
    public static void main(String args[]) {  
        String text = args[0];  
        System.out.println(text);  
    }  
}
```



```
D:\FIT-HUT\Lectures\OOP\OOP-Java\Demo>java NoException  
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0  
    at NoException.main(NoException.java:3)  
D:\FIT-HUT\Lectures\OOP\OOP-Java\Demo>
```

# Ví dụ có xử lý ngoại lệ

```
class ArgExceptionDemo {  
    public static void main(String args[]) {  
        try {  
            String text = args[0];  
            System.out.println(text);  
        }  
        catch (Exception e) {  
            System.out.println("Hay nhap tham so khi chay!");  
        }  
    }  
}
```



```
D:\FIT-HUT\Lectures\OOP\OOP-Java\Demo>java ArgExceptionDemo  
Hay nhap tham so khi chay!  
D:\FIT-HUT\Lectures\OOP\OOP-Java\Demo>_
```

# Ví dụ có xử lý ngoại lệ

```
public class ChiaCho0Demo {  
    public static void main(String args[]) {  
        try {  
            int num = calculate(9,0);  
            System.out.println(num);  
        }  
        catch(Exception e) {  
            System.err.println("Co loi xay ra: " + e.toString());  
        }  
    }  
    static int calculate(int no, int nol){  
        int num = no / nol;  
        return num;  
    }  
}
```



```
Co loi xay ra: java.lang.ArithmeticException: / by zero  
Press any key to continue . . .
```

# Lớp Throwable

- Một biến kiểu `String` để lưu thông tin chi tiết về ngoại lệ đã xảy ra
- **Một số phương thức cơ bản**
  - **`new Throwable(String s)`**: Tạo một ngoại lệ với thông tin về ngoại lệ là `s`
  - **`String getMessage()`**: Lấy thông tin về ngoại lệ
  - **`String getString()`**: Mô tả ngắn gọn về ngoại lệ
  - **`void printStackTrace()`**: In ra tất cả các thông tin liên quan đến ngoại lệ (tên, loại, vị trí...)
  - ...

# Lớp Throwable

```
public class StckExceptionDemo {  
    public static void main(String args[]) {  
        try {  
            int num = calculate(9,0);  
            System.out.println(num);  
        }  
        catch(Exception e) {  
            System.err.println("Co loi xay ra :"  
                               + e.getMessage());  
            e.printStackTrace();  
        }  
    }  
    static int calculate(int no, int no1) {  
        int num = no / no1;  
        return num;  
    }  
}
```



```
Co loi xay ra :/ by zero  
java.lang.ArithmeticException: / by zero  
    at StckExceptionDemo.calculate(StckExceptionDemo.java:14)  
    at StckExceptionDemo.main(StckExceptionDemo.java:4)  
Press any key to continue . . . _
```

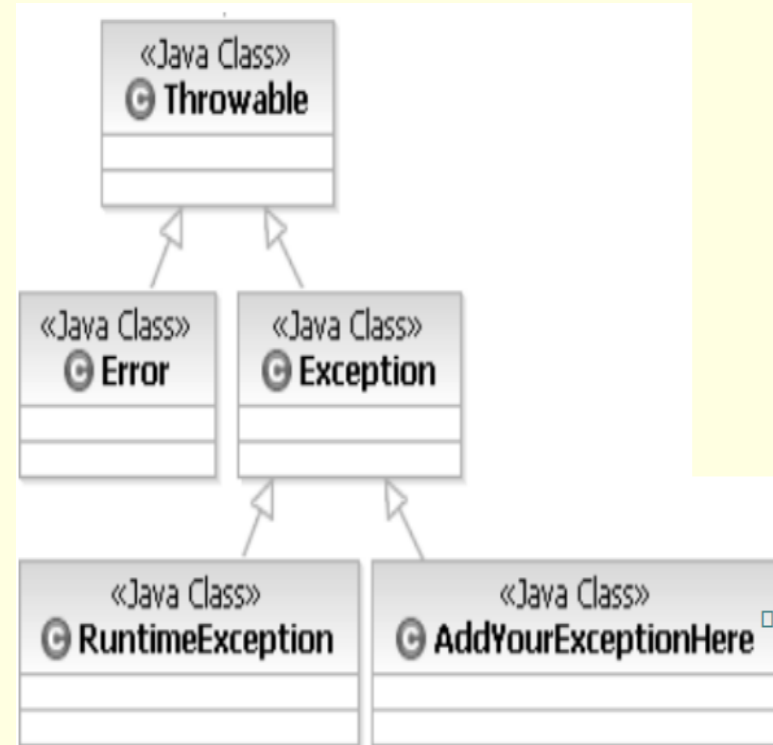
# Lớp Error

- Gồm các ngoại lệ nghiêm trọng không thể kiểm tra (unchecked exception) vì có thể xảy ra ở nhiều phần của chương trình.
- Còn gọi là ngoại lệ không thể phục hồi (un-recoverable exception)
- Không cần kiểm tra trong mã nguồn Java của bạn
- Các lớp con:
  - VirtualMachineError: InternalError, OutOfMemoryError, StackOverflowError, UnknownError
  - ThreadDeath
  - LinkageError:
    - IncompatibleClassChangeError
      - AbstractMethodError, InstantiationException, NoSuchFieldError, NoSuchMethodError...
    - ...
  - ...



# Lớp Exception

- Chứa các loại ngoại lệ nên/phải bắt và xử lý hoặc ủy nhiệm.
- Người dùng có thể tạo ra các ngoại lệ của riêng mình bằng cách kế thừa từ Exception
- RuntimeException có thể được “tung” ra trong quá trình JVM thực hiện
  - *Không bắt buộc phải bắt ngoại lệ dù có thể xảy ra lỗi*
  - *Không nên viết ngoại lệ của riêng mình kế thừa từ lớp này*



# Một số lớp con của Exception

---

- ClassNotFoundException, SQLException
- java.io.IOException:
  - FileNotFoundException, EOFException...
- RuntimeException:
  - NullPointerException, BufferOverflowException
  - ClassCastException, ArithmeticException
  - IndexOutOfBoundsException:
    - ArrayIndexOutOfBoundsException,
    - StringIndexOutOfBoundsException...
  - IllegalArgumentException:
    - NumberFormatException, InvalidParameterException...
  - ...

# Khối try/catch lồng nhau

- Những phần nhỏ trong khối mã sinh ra một lỗi, nhưng toàn bộ cả khối thì lại sinh ra một lỗi khác → Cần có các xử lý ngoại lệ lồng nhau.
- Khi các khối try lồng nhau, khối try bên trong sẽ được thực hiện trước.

```
try {  
    // Doan ma co the gay ra IOException  
    try {  
        // Doan ma co the gay ra NumberFormatException  
    } catch (NumberFormatException e1) {  
        // Xu ly loi sai dinh dang so  
    }  
} catch (IOException e2) {  
    // Xu ly loi vao ra  
}
```

# Nhiều khối catch

- Một đoạn mã có thể gây ra nhiều hơn một ngoại lệ → Sử dụng nhiều khối catch.

```
try {  
    // Đoạn mã có thể gây ra nhiều ngoại lệ  
} catch (ExceptionType1 e1) {  
    // Xử lý ngoại lệ 1  
} catch (ExceptionType2 e2) {  
    // Xử lý ngoại lệ 2  
} ...
```

- ExceptionType1 phải là lớp con hoặc ngang hàng với ExceptionType2 (trong cây phân cấp kế thừa)

# Nhiều khối catch

```
class MultipleCatch1 {  
    public static void main(String args[])  
    {  
        try {  
            String num = args[0];  
            int numValue = Integer.parseInt(num);  
            System.out.println("Dien tich hv la: "  
                               + numValue * numValue);  
        } catch (Exception e1) {  
            System.out.println("Hay nhap canh cua  
hv!");  
        } catch (NumberFormatException e2) {  
            System.out.println("Not a number!");  
        }  
    }  
}
```

Lỗi

D:\exception java.lang.NumberFormatException  
has already been caught

# Nhiều khối catch

```
class MultipleCatch1 {  
    public static void main(String args[])  
    {  
        try {  
            String num = args[0];  
            int numValue = Integer.parseInt(num);  
            System.out.println("Dien tich hv la: "  
                               + numValue * numValue);  
        } catch (ArrayIndexOutOfBoundsException e1) {  
            System.out.println("Hay nhap canh cua hv!");  
        } catch (NumberFormatException e2) {  
            System.out.println("Hay nhap 1 so!");  
        }  
    }  
}
```

# Nhiều khối catch

```
...
public void openFile() {
    try {
        // constructor may throw FileNotFoundException
        FileReader reader = new FileReader("someFile");
        int i=0;
        while(i != -1) {
            //reader.read() may throw IOException
            i = reader.read();
            System.out.println((char) i );
        }
        reader.close();
        System.out.println("--- File End ---");
    } catch (FileNotFoundException e) {
        //do something clever with the exception
    } catch (IOException e) {
        //do something clever with the exception
    }
}
...
```

# try ... catch... finally

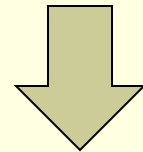
```
try {  
    // Khoi lenh co the sinh ngoai le  
}  
catch(ExceptionType e) {  
    // Bat va xu ly ngoai le  
}  
finally {  
    /* Thuc hien cac cong viec can thiet du  
    ngoai le co xay ra hay khong */  
}
```

- Nếu đã có khối try thì bắt buộc phải có khối catch hoặc khối finally hoặc cả hai



# try ... catch... finally

```
class StrExceptionDemo {  
    static String str;  
    public static void main(String s[]) {  
        try {  
            System.out.println("Truoc ngoai le");  
            staticLengthmethod();  
            System.out.println("Sau ngoai le");  
        }  
        catch (NullPointerException ne) {  
            System.out.println("Da xay ra loi");  
            System.out.println(ne.toString());  
        }  
        finally {  
            System.out.println("Trong finally");  
        }  
    }  
    static void staticLengthmethod() {  
        System.out.println(str.length());  
    }  
}
```



Truoc ngoai le

Da xay ra loi

java.lang.NullPointerException

Trong finally

# Quản lý Exception với các phương thức Overriding method

- Overriding method (trong lớp con) *có thể throw/throws* unchecked exception (RuntimeException hoặc Error), bất kể overridden method (trong lớp cha) có mô tả Exception hay không.
- Overriding method *không thể throw/throws* những checked exception “mới” hay “rộng hơn” các Exception mô tả trong overridden method.

```
class Base {  
    public void method1() {  
        System.out.println("Overriden method");  
    }  
}  
  
class Sub extends Base {  
    @Override  
    public void method1() throws OutOfMemoryError,  
        ArrayIndexOutOfBoundsException {  
        System.out.println("Overriding method");  
        throw new OutOfMemoryError();  
    }  
}
```

```
class Base {  
    public void method1() throws FileNotFoundException {  
        System.out.println("Overriden method");  
    }  
}  
  
class Sub extends Base {  
    public void method1() throws OutOfMemoryError,  
        ArrayIndexOutOfBoundsException, IOException {  
        System.out.println("Overriding method");  
        throw new OutOfMemoryError();  
    }  
}
```

# Tài liệu tham khảo

---

1. Sun tutorial group, The Java Tutorial, *A practical guide for programmers*, java.sun.com
2. Eckel B., Thinking in Java, Prentice Hall PTR, 1998, ISBN 0-13-659723-8
3. Oser P. (POS), Exception Handling Guidelines, Internal Leaf project, 2000.
4. <https://docs.oracle.com/javase/tutorial/essential/exceptions/>
5. <https://docs.oracle.com/javase/tutorial/essential/exceptions/QandE/questions.html>