

# Bài 4: NGOẠI LỆ TRONG JAVA

# Nội dung

---



- Ngoại lệ - Exception
- Xử lý ngoại lệ - Exception Handling

# Exception



## ➤ Định nghĩa:

- Exception (ngoại lệ): là một sự kiện xảy ra trong tiến trình thực thi của một chương trình, nó làm ngưng tiến trình bình thường của chương trình.
- Khi xảy ngoại lệ, nếu không xử lý chương trình sẽ kết thúc ngay.
- Ví dụ: Lỗi chia cho 0, vượt kích thước của mảng, lỗi mở file

# Xử lý ngoại lệ - Exception Handling



- Mục đích: làm cho chương trình không bị ngắt đột ngột.
- Có 2 cách để xử lý ngoại lệ:
  - Sử dụng các mệnh đề điều kiện kết hợp với các giá trị cờ.
  - Sử dụng cơ chế bắt và xử lý ngoại lệ.

# Xử lý ngoại lệ (tt)



Sử dụng các mệnh đề điều kiện kết hợp với các giá trị cờ.

- **Mục đích:** thông qua tham số, giá trị trả lại hoặc giá trị cờ để viết mã xử lý tại nơi phát sinh lỗi.
- **Hạn chế:**
  - Làm chương trình thêm rối, gây khó hiểu.
  - Dễ nhầm lẫn

# Lớp Inventory



```
public class Inventory
{
    public final int MIN = 0;
    public final int MAX = 100;
    public final int CRITICAL = 10;
    public boolean addToInventory (int amount)
    {
        int temp;
        temp = stockLevel + amount;
        if (temp > MAX)
        {
            System.out.print("Adding " + amount + " item will cause stock ");
            System.out.println("to become greater than " + MAX + " units
(overstock)");
            return false;
        }
    }
}
```

# Lớp Inventory (tt)



---

```
else
{
    stockLevel = stockLevel + amount;
    return true;
}
} // End of method addToInventory
:
```

# Các vấn đề đối với cách tiếp cận điều kiện/cờ



```
reference1.method1 (){  
    if (reference2.method2() == false)  
        return false;  
}
```

```
reference2.method2 (){  
    if (store.addToInventory(amt) == false)  
        return false;  
}
```

```
store.addToInventory (int amt){  
    if (temp > MAX)  
        return false;  
}
```



# Các vấn đề đối với cách tiếp cận điều kiện/cờ



```
reference1.method1 (){  
    if (reference2.method2() == false)  
        return false;  
}
```

**Vấn đề 1:** Phương thức chủ có thể quên kiểm tra điều kiện trả về

```
reference2.method2 (){  
    if (store.addToInventory(amt) == false)  
        return false;  
}
```

```
store.addToInventory (int amt){  
    if (temp > MAX)  
        return false;  
}
```

# Các vấn đề đối với cách tiếp cận điều kiện/cờ



```
reference1.method1 ()  
    if (reference2.method2() == false)  
        return false;
```

```
reference2.method2 ()  
    if (store.addToInventory(amt) == false)  
        return false;
```

```
store.addToInventory (int amt)  
    if (temp > MAX)  
        return false;
```

**Vấn đề 2:** Phải sử dụng 1 loạt các phép kiểm tra giá trị cờ trả

# Các vấn đề đối với cách tiếp cận điều kiện/cờ



```
reference1.method1 ()  
    if (reference2.method2() == false)  
        return false;
```

```
reference.method2 ()  
    if (store.addToInventory(amt) == false)  
        return false;
```

??

??

**Vấn đề 3:** Phương thức  
chủ có thể không biết  
cách xử lý khi lỗi xảy ra

```
store.addToInventory (int amt)  
    if (temp > MAX)  
        return false;
```

# Các vấn đề đối với cách tiếp cận điều kiện/cờ



## ➤ **Nhược điểm:**

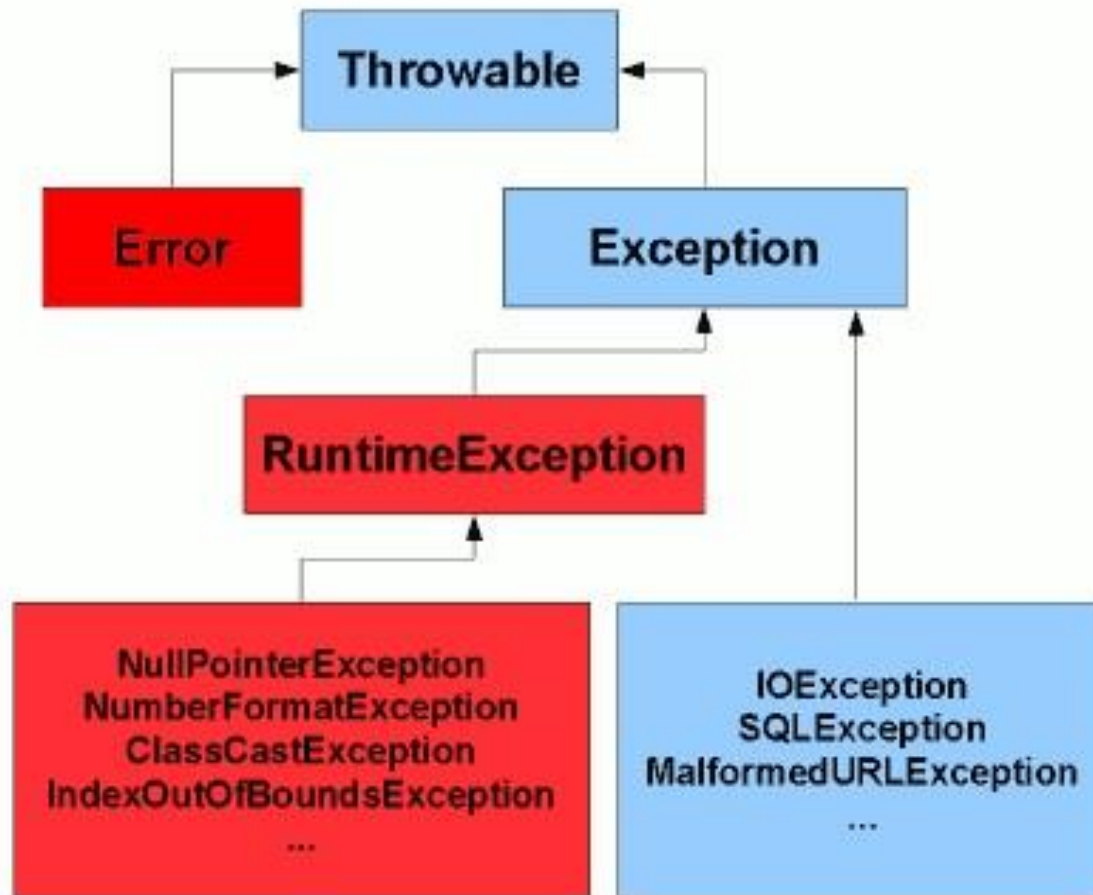
- Khó kiểm soát được hết các trường hợp
  - Lỗi số học, lỗi bộ nhớ,...
- Lập trình viên thường quên không xử lý lỗi

# Xử lý ngoại lệ (tt)



- Sử dụng cơ chế bắt và xử lý ngoại lệ.
- Mục đích:
  - Giúp chương trình đáng tin cậy hơn, tránh kết thúc bất thường
  - Tách biệt khối lệnh có thể gây ngoại lệ và khối lệnh xử lý ngoại lệ.
- Phân loại ngoại lệ:
  - Ngoại lệ không cần kiểm tra (unchecked)
  - Ngoại lệ phải kiểm tra (checked)

# Checked vs Unchecked



# Ngoại lệ không cần kiểm tra



- Trình biên dịch không yêu cầu phải bắt các ngoại lệ khi nó xảy ra.
  - *Không cần khối try-catch*
- Các ngoại lệ này có thể xảy ra bất cứ thời điểm nào khi thi hành chương trình.
- Thông thường là những lỗi nghiêm trọng mà chương trình không thể kiểm soát
  - Xử dụng các mệnh đề điều kiện để xử lý sẽ tốt hơn.
- Gồm các lớp **RuntimeException**, **Error** và các **lớp con** của chúng

# Ngoại lệ không cần kiểm tra (tt)



- **RuntimeException:** chỉ các ngoại lệ xảy ra khi JVM thực thi chương trình
  - `NullPointerException`: con trỏ null
  - `OutOfMemoryException`: hết bộ nhớ
  - `ArrayIndexOutOfBoundsException`: vượt quá chỉ số mảng
  - `ArithmeticException`: lỗi toán học
  - `ClassCastException`: lỗi ép kiểu
- **Lớp Error:**
  - Chỉ những lỗi nghiêm trọng và không dự đoán trước được: `ThreadDead`, `LinkageError`, `VirtualMachineError`...
  - Các ngoại lệ kiểu `Error` ít được xử lý.



# Ngoại lệ không cần kiểm tra: NullPointerException



```
int [] arr = null;
```

```
arr[0] = 1;
```

NullPointerException

```
arr = new int [4];
```

```
int i;
```

```
for (i = 0; i <= 4; i++)
```

```
    arr[i] = i;
```

```
arr[i-1] = arr[i-1] / 0;
```

# Ngoại lệ không cần kiểm tra: : ArrayIndexOutOfBoundsException



```
int [] arr = null;
```

```
arr[0] = 1;
```

```
arr = new int [4];
```

```
int i;
```

```
for (i = 0; i <= 4; i++)
```

```
    arr[i] = i;
```

**ArrayIndexOutOfBoundsException**  
(when i = 4)

```
arr[i-1] = arr[i-1] / 0;
```

# Ngoại lệ không cần kiểm tra: ArithmeticExceptions



```
int [] arr = null;
```

```
arr[0] = 1;
```

```
arr = new int [4];
```

```
int i;
```

```
for (i = 0; i <= 4; i++)
```

```
    arr[i] = i;
```

```
arr[i-1] = arr[i-1] / 0;
```

**ArithmeticException**  
(Division by zero)

# Ngoại lệ cần phải kiểm tra



- Là ngoại lệ bắt buộc kiểm tra.
- Phải xử lý khi ngoại lệ có khả năng xảy ra:
  - Sử dụng khối try-catch
  - Sử dụng throw, throws
- Ví dụ:
  - IOException, NumberFormatException

# Ngoại lệ cần phải kiểm tra try - catch



## ➤ Khối try...catch:

- try {...}: khối lệnh có khả năng gây ra ngoại lệ.
- catch {...}: nơi bắt và xử lý ngoại lệ.

## ➤ Cú pháp:

```
try
{
    // Code that may cause an error/exception to occur
}
catch (ExceptionType identifier)
{
    // Code to handle the exception
}
```

# Ngoại lệ cần phải kiểm tra: Đọc dữ liệu từ bàn phím

---



```
import java.io.*;
class Driver
{
    public static void main (String [] args)
    {
        BufferedReader stringInput;
        String s;
        int num;
        stringInput = new BufferedReader(new InputStreamReader(System.in));
```

# Ngoại lệ cần phải kiểm tra: Đọc dữ liệu từ bàn phím



```
try{
    System.out.print("Type an integer: ");
    s = stringInput.readLine();
    System.out.println("You typed in..." + s);
    num = Integer.parseInt (s);
    System.out.println("Converted to an integer..." + num);
}
catch (IOException e){
    System.out.println(e);
}
catch (NumberFormatException e{
    :      :      :
}
}
}
```

# Ngoại lệ cần phải kiểm tra: Ngoại lệ xảy ra khi nào

---



```
try
{
    System.out.print("Type an integer: ");
    s = stringInput.readLine();
    System.out.println("You typed in..." + s);
    num = Integer.parseInt (s);
    System.out.println("Converted to an integer..." + num);
}
```



# Kết quả của phương thức readLine()



```
try
{
    System.out.print("Type an integer: ");
    s = stringInput.readLine();
    System.out.println("You typed in..." + s);
    num = Integer.parseInt (s);
    System.out.println("Converted to an integer..." + num);
}
```

Ngoại lệ có thể xảy ra ở đây

# Lớp BufferedReader



<https://docs.oracle.com/javase/7/docs/api/java/io/BufferedReader.html>

```
public class BufferedReader
{
    public BufferedReader (Reader in);
    public BufferedReader (Reader in, int sz);
    public String readLine () throws IOException;
    :
}
```

# Kết quả của phương thức parseInt ()



```
try
{
    System.out.print("Type an integer: ");
    s = stringInput.readLine();
    System.out.println("You typed in..." + s);
    num = Integer.parseInt(s);
    System.out.println("Converted to an integer..." + num);
}
```

Ngoại lệ có thể xảy ra ở đây

# Lớp Integer



<http://docs.oracle.com/javase/7/docs/api/java/lang/Integer.html>

public class Integer

{

    public Integer (int value);

    public Integer (String s) *throws NumberFormatException*;

        :

        :

    public static int parseInt (String s) *throws NumberFormatException*;

        :

        :

}

# Cơ chế xử lý ngoại lệ



```
try
{
    System.out.print("Type an integer: ");
    s = stringInput.readLine();
    System.out.println("You typed in..." + s);
    num = Integer.parseInt (s);
    System.out.println("Converted to an integer..." + num);
}
catch (IOException e)
{
    System.out.println(e);
}
catch (NumberFormatException e)
{
    :      :      :
}
```

# Cơ chế xử lý ngoại lệ (tt)



```
Driver.main ()
try
{
    num = Integer.parseInt (s);
}
:
catch (NumberFormatException e)
{
    :
}
```

```
Integer.parseInt (String s)
{
    :
    :
}
```

# Cơ chế xử lý ngoại lệ (tt)



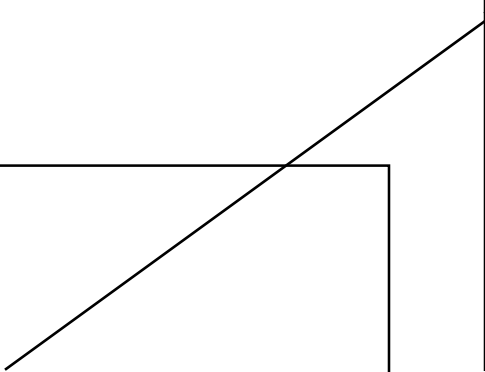
```
Driver.main ()
try
{
    num = Integer.parseInt (s);
}
:
catch (NumberFormatException e)
{
    :
}
```

```
Integer.parseInt (String s)
{
    Người sử dụng không  
nhập chuỗi số
}
```

# Cơ chế xử lý ngoại lệ (tt)



```
Driver.main ()
try
{
    num = Integer.parseInt (s);
}
:
catch (NumberFormatException e)
{
    :
}
```



```
Integer.parseInt (String s)
{
    NumberFormatException e =
        new
        NumberFormatException ();
}
```



# Cơ chế xử lý ngoại lệ (tt)



```
Driver.main ()
try
{
    num = Integer.parseInt (s);
}
:
catch (NumberFormatException e)
{
    :
}
```

```
Integer.parseInt (String s)
{
    NumberFormatException e =
        new NumberFormatException ();
}
```

# Cơ chế xử lý ngoại lệ (tt)



```
Driver.main ()
try
{
    num = Integer.parseInt (s);
}
:
catch (NumberFormatException e)
{
    Ngoại lệ sẽ được xử lý ở đây
}
```

```
Integer.parseInt (String s)
{
}
}
```

# Bắt ngoại lệ



```
catch (NumberFormatException e)
{
    :      :      :
}
```

# Bắt ngoại lệ (tt)



```
catch (NumberFormatException e)
{
    System.out.println(e.getMessage());
    System.out.println(e);
    e.printStackTrace();
}
```

# Bắt ngoại lệ (tt)



```
catch (NumberFormatException e)
```

```
{
```

```
    System.out.println(e.getMessage());
```

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

```
    e.printStackTrace();
```

```
}
```

For input string: "exception"

java.lang.NumberFormatException: For input string: "exception"

java.lang.NumberFormatException: For input string: "exception"  
at

java.lang.NumberFormatException.forInputString(NumberFormatException.java:48)  
at java.lang.Integer.parseInt(Integer.java:426)  
at java.lang.Integer.parseInt(Integer.java:476)  
at Driver.main(Driver.java:39)

# Tránh bỏ qua việc xử lý ngoại lệ



```
try
{
    s = stringInput.readLine();
    num = Integer.parseInt (s);
}
catch (IOException e)
{
    //System.out.println(e);
}
```

# Tránh bỏ qua việc xử lý ngoại lệ (tt)



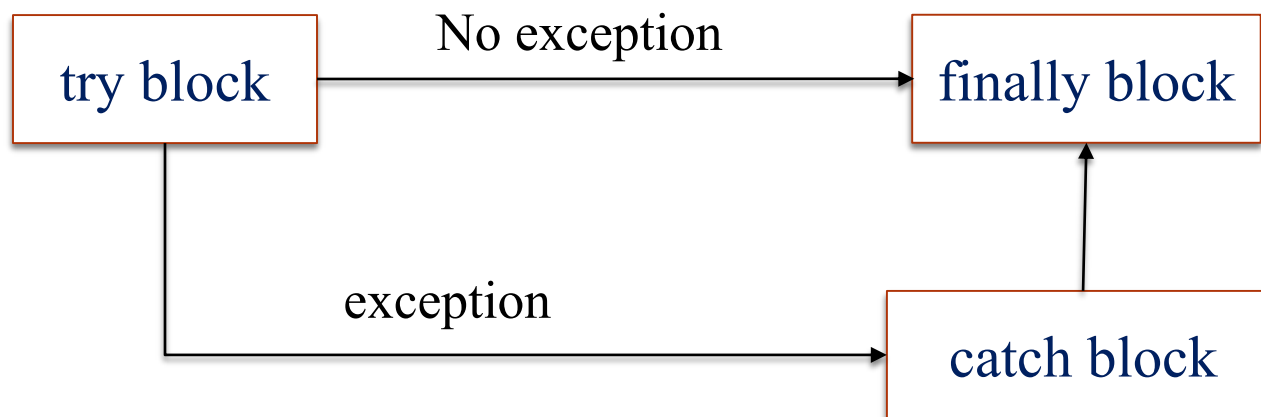
```
try
{
    s = StringInput.readLine();
    num = Integer.parseInt(s);
}
catch (IOException e)
{
    System.out.println(e);
}
catch (NumberFormatException e)
{
    // Do nothing here but set up the try-catch block to bypass the
    // annoying compiler error
}
```

NO!

# Khối finally



- Là 1 khối không bắt buộc trong khối try-catch-*finally*.
- Dùng để đảm bảo khối lệnh sẽ được thi hành bất kể ngoại lệ có xảy ra hay không. VD:
  - Đóng file, đóng socket, connection
  - Giải phóng tài nguyên (nếu cần)...





# Khối finally (tt)



## ➤ Cú pháp:

```
try
{
    // Code that may cause an error/exception to occur
}
catch (ExceptionType identifier)
{
    // Code to handle the exception
}
finally
{
    // Implement code
}
```

# Khối finally: có ngoại lệ



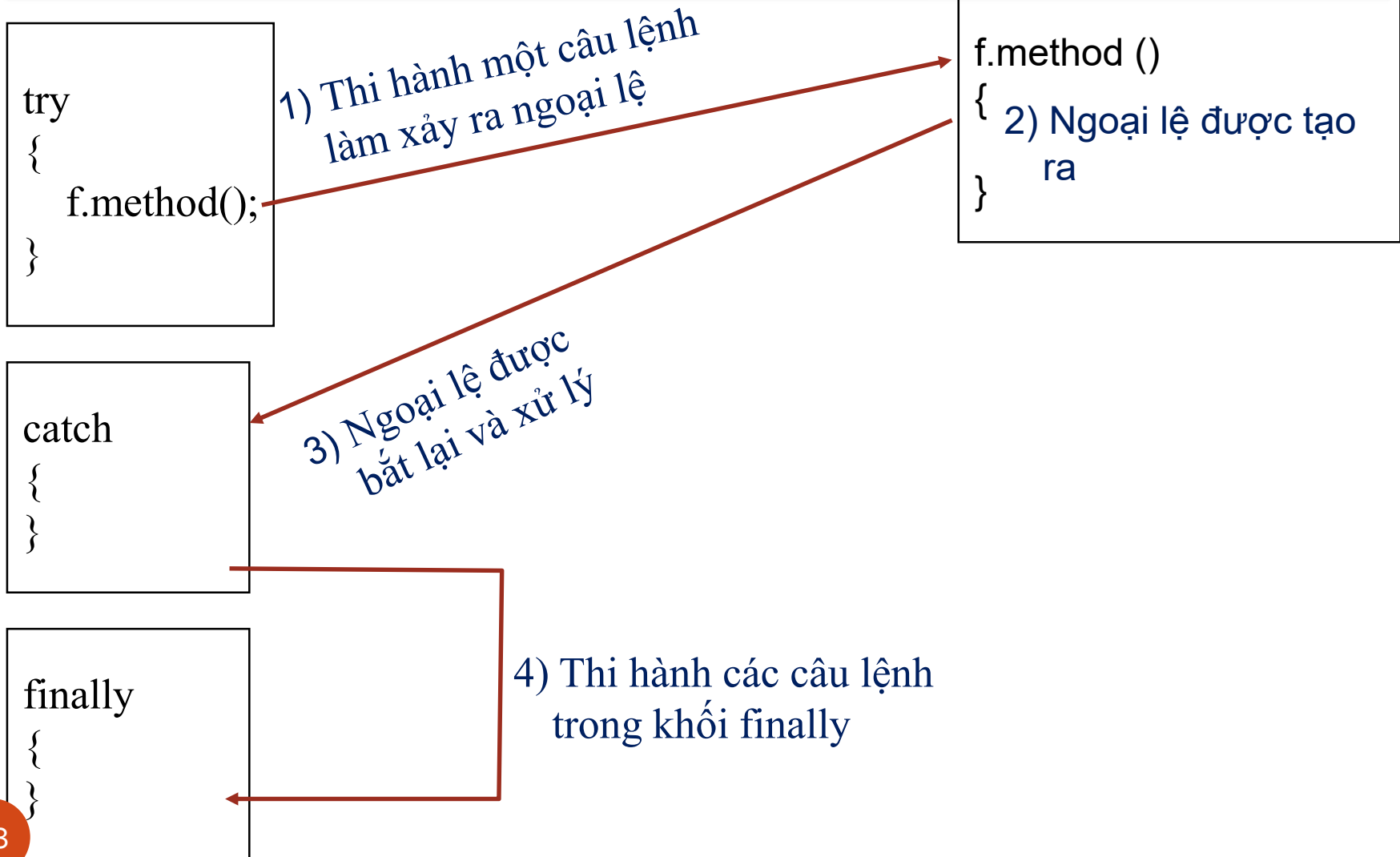
```
try
{
    f.method();
}
```

```
catch
{
}
```

```
finally
{
}
```

```
Foo.method ()
{
}
}
```

# Khối finally: có ngoại lệ (tt)



# Khối finally: không có ngoại lệ



```
try
{
    f.method();
}
```

1) Gọi thi hành 1 phương thức không làm phát sinh ngoại lệ

f.method ()

```
{
    2) Phương thức thi hành bình thường
}
```

```
catch
{
}
```

```
finally
{
}
```

3) Thi hành các câu lệnh trong khối finally

# Try-Catch-Finally: Ví dụ



```
class Driver
{
    public static void main (String [] args)
    {
        TCExample eg = new TCExample ();
        eg.method();
    }
}
```

# Try-Catch-Finally: Ví dụ



```
public class TCFExample
{
    public void method ()
    {
        BufferedReader br;
        String s;
        int num;
        try
        {
            System.out.print("Type in an integer: ");
            br = new BufferedReader(new InputStreamReader(System.in));
            s = br.readLine();
            num = Integer.parseInt(s);
            return;
        }
    }
}
```

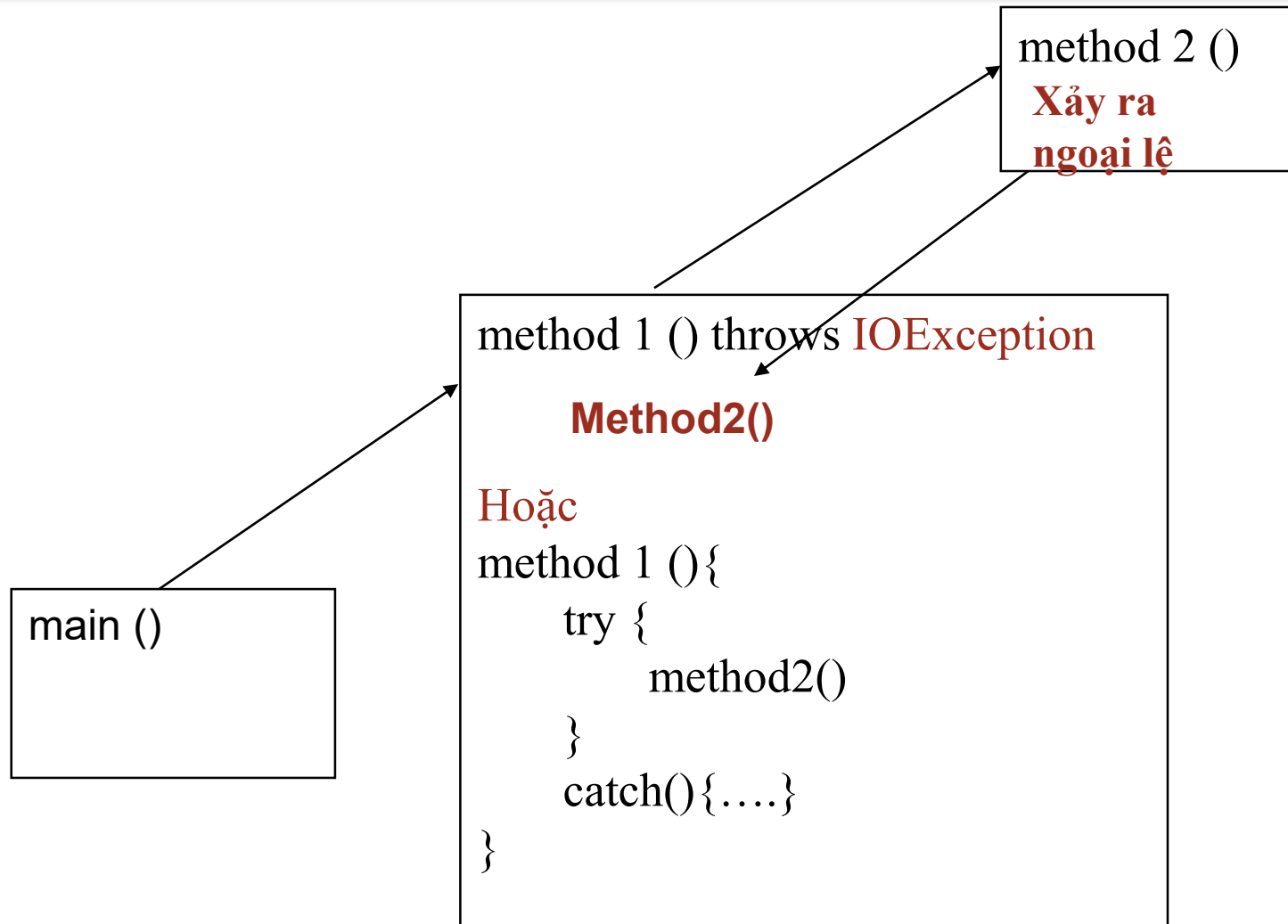
# Ngoại lệ cần phải kiểm tra: throws



Giả sử có `method1` và `method2`. `Method1` gọi `method2` và `method2` là phương thức có khả năng xảy ra ngoại lệ:

- Hoặc **`method2`** phải nằm trong khối **`try/catch`**.
- Hoặc phải khai báo **`method1`** có khả năng ném (**`throws`**) ngoại lệ.

# Ngoại lệ cần phải kiểm tra: throws (tt)





# Ngoại lệ cần phải kiểm tra: throws (tt)



```
import java.io.*;
public class TCExample
{
    public void method () throws IOException, NumberFormatException
    {
        BufferedReader br;
        String s;
        int num;

        System.out.print("Type in an integer: ");
        br = new BufferedReader(new InputStreamReader(System.in));
        s = br.readLine();
        num = Integer.parseInt(s);
    }
}
```

# Hàm main xử lý ngoại lệ



```
class Driver
```

```
{
```

```
    public static void main (String [] args) {
```

```
        TCExample eg = new TCExample ();
```

```
        boolean inputOkay = true;
```

```
        do{
```

```
            try
```

```
            {
```

```
                eg.method();
```

```
                inputOkay = true;
```

```
            }
```

```
            catch (IOException e){
```

```
                e.printStackTrace();
```

```
            }
```

```
            catch (NumberFormatException e){
```

```
                inputOkay = false;
```

```
                System.out.println("Please enter a whole number.");
```

```
            }
```

```
        } while (inputOkay == false);
```

```
    } // End of main
```

```
} // End of Driver class
```

Phải xử lý cả  
IOException và  
NumberFormatException



# Hàm main không xử lý ngoại lệ

---

```
class Driver
{
    public static void main (String [] args) throws IOException,
    NumberFormatException {
        TCExample eg = new TCExample ();
        eg.method();
    }
}
```

# Ngoại lệ cần phải kiểm tra: throw



➤ Sử dụng throw **anExceptionObject** trong thân phương thức để tung ra ngoại lệ khi cần

➤ Ví dụ:

```
public static void method (int so) throws Exception {  
    if (so < 5) {  
        throw new Exception("Qua nhỏ");  
    }  
}
```

➤ Nếu phương thức có chứa câu lệnh **throw** ngoại lệ thì phần khai báo phương thức phải khai báo **throws** ngoại lệ đó hoặc lớp cha của ngoại lệ đó.

# Ngoại lệ cần phải kiểm tra: throw (tt)



➤ Đối với **RuntimeException** phương thức không cần phải khai báo throws RuntimeException vì ngoại lệ này mặc định được ủy nhiệm cho JVM

➤ Ví dụ

```
public static void method (int so) {  
    if (so < 5) {  
        throw new RuntimeException("Qua nhỏ");  
    }  
}
```

# Ngoại lệ cần phải kiểm tra: throw (tt)



## ➤ Ví dụ 1:

```
static int cal(int no, int no1){  
    if (no1 == 0){  
        throw new ArithmeticException ("Khong the chia cho 0");  
    }  
    int num = no/no1;  
    return num;  
}  
  
public static void main(String[] args) {  
    int num = cal(6,0);  
}
```

**Lỗi ngoại lệ:**

Exception in thread "main" java.lang.ArithmeticException: Khong the chia cho 0

# Ngoại lệ cần phải kiểm tra: throw (tt)



## ➤ Ví dụ 2:

```
static int cal(int no, int no1) throws Exception{  
    if (no1 == 0){  
        throw new ArithmeticException ("Khong the chia cho 0");  
    }  
    int num = no/no1;  
    return num;  
}  
public static void main(String[] args) {  
    int num = cal(6,0);  
}
```

### Lỗi biên dịch:

Exception in thread "main" java.lang.RuntimeException:  
Uncompilable source code - unreported exception java.lang.Exception;  
must be caught or declared to be thrown  
at exceptionex.ExceptionEx.main(ExceptionEx.java:58)

# Ngoại lệ cần phải kiểm tra: throw (tt)



## ➤ Ví dụ 3:

```
static int cal(int no, int no1){  
    if (no1 == 0){  
        throw new ArithmeticException ("Khong the chia cho 0");  
    }  
    int num = no/no1;  
    return num;  
}  
public static void main(String[] args) {  
    try{  
        int num = cal(6,0);  
    }  
    catch(Exception e) {  
        System.out.println(e.getMessage());  
    }  
}
```



# Ngoại lệ cần phải kiểm tra (tt)



- Một phương thức có thể throw nhiều hơn 1 ngoại lệ:

```
public void method(int tuoi, String ten) throws ArithmeticException,  
    NullPointerException {  
    if (tuoi < 18) {  
        throw new ArithmeticException("Chua du tuoi!");  
    }  
    if (ten == null) {  
        throw new NullPointerException("Thieu ten!");  
    }  
}
```

- Lan truyền ngoại lệ:

- Trong main() gọi phương thức A(), trong A() gọi B(), trong B() gọi C().
- Giả sử trong C() xảy ra ngoại lệ.

# Ngoại lệ cần phải kiểm tra (tt)



- Nếu C() gặp lỗi và throw ra ngoại lệ nhưng trong C() lại không xử lý ngoại lệ này, thì nơi gọi C() là phương thức B() là nơi có thể xử lý ngoại lệ.
- Nếu trong B() cũng không xử lý thì phải xử lý ngoại lệ này trong A()... Quá trình này gọi là lan truyền ngoại lệ.
- Nếu đến main() cũng không xử lý ngoại lệ được throw từ C() thì chương trình sẽ phải dừng lại.

## ➤ **Kế thừa ngoại lệ:**

- Khi override một phương thức của lớp cha, phương thức ở lớp con không được phép tung ra các ngoại lệ mới.
- Phương thức ghi đè trong lớp con chỉ được phép tung ra các ngoại lệ **giống** hoặc là **lớp con** hoặc là **tập con** của các ngoại lệ được **tung ra ở lớp cha**.

# Ví dụ kế thừa ngoại lệ



```
class Disk {  
    public void readFile() throws EOFException {}  
}  
class FloppyDisk extends Disk {  
    public void readFile() throws IOException {} // ERROR!  
}
```



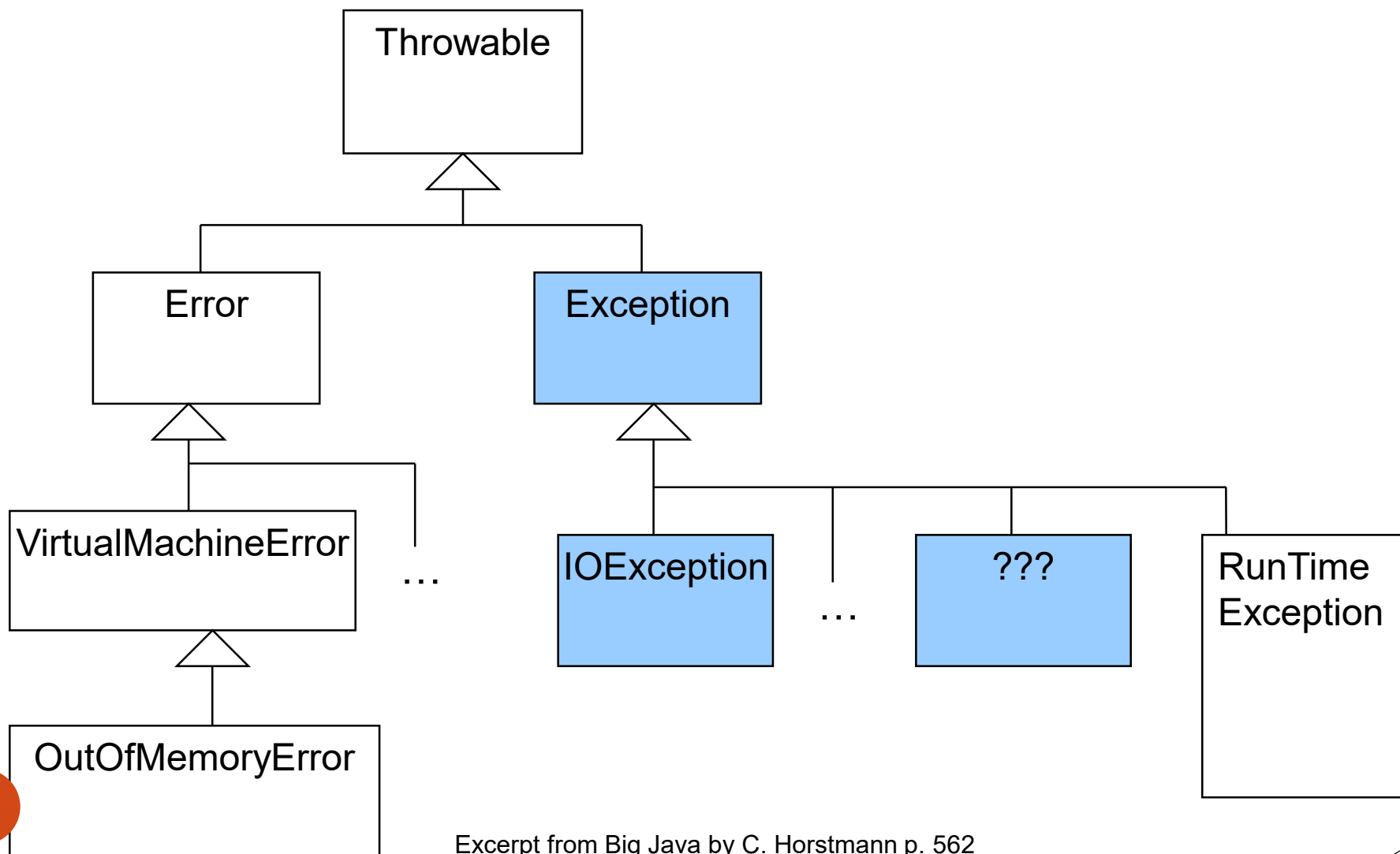
```
class Disk {  
    public void readFile() throws IOException {}  
}  
class FloppyDisk extends Disk {  
    public void readFile() throws EOFException {} //OK  
}
```

# Ưu điểm của việc throws/throw ngoại lệ

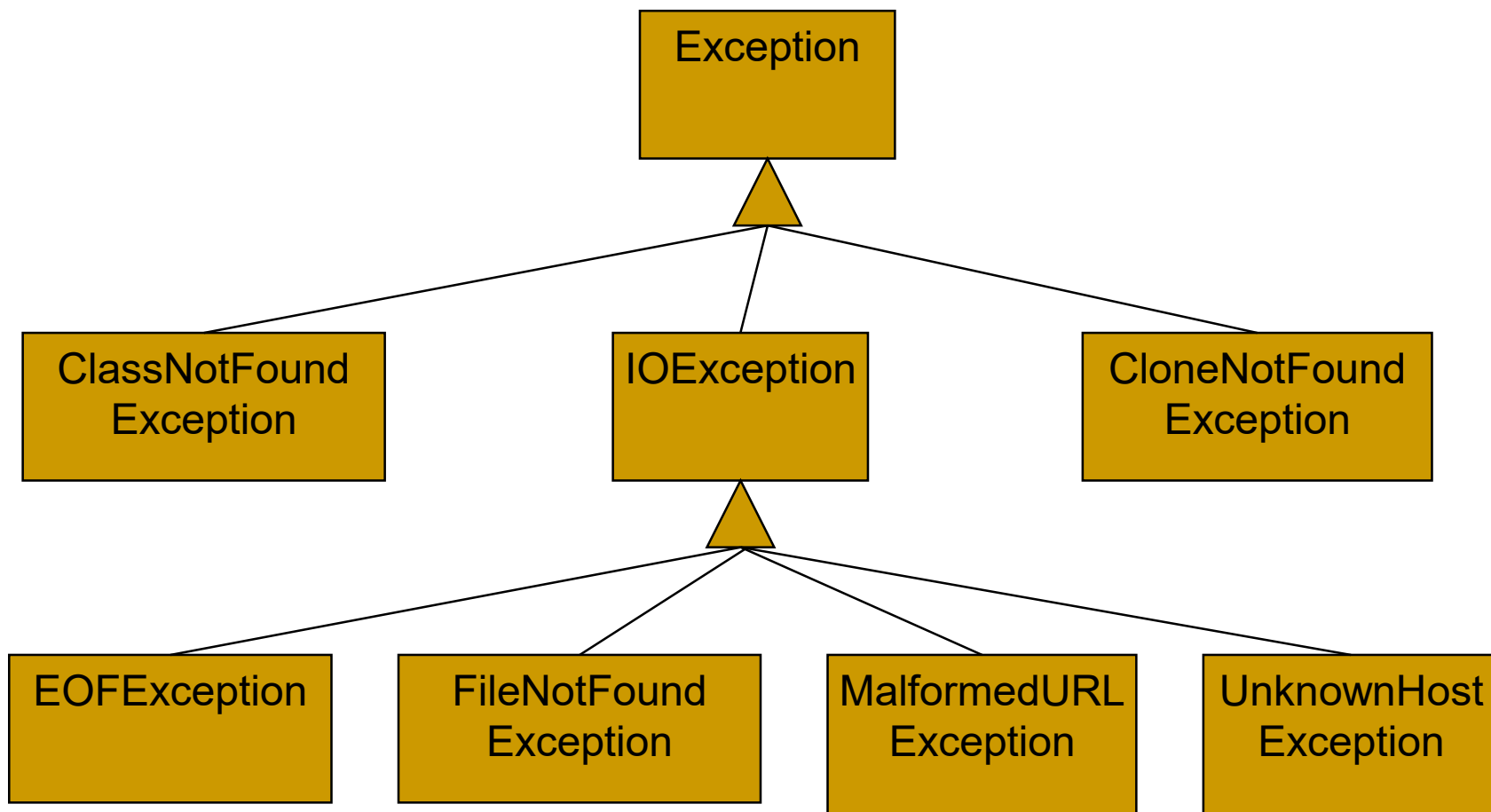


- Dễ sử dụng
  - Dễ dàng chuyển điều khiển đến nơi có khả năng xử lý ngoại lệ
  - Có thể throw nhiều loại ngoại lệ
- Tách xử lý ngoại lệ khỏi đoạn mã thông thường
- Không bỏ sót ngoại lệ (throws)
- Gom nhóm và phân loại các ngoại lệ

# Tạo ra kiểu ngoại lệ mới



# Lớp Exception



# Tạo ngoại lệ mới



- Mục đích: tạo ra ngoại lệ do người dùng định nghĩa để kiểm soát các lỗi
  - Kế thừa lớp **Exception** hoặc lớp con của nó
  - Có tất cả phương thức của lớp **Throwable**

# Tạo ngoại lệ tự định nghĩa



```
public class MyException extends Exception {  
    public MyException(String msg) {  
        super(msg);  
    }  
    public MyException(String msg, Throwable cause){  
        super(msg, cause);  
    }  
}
```



# Sử dụng ngoại lệ tự định nghĩa (tt)



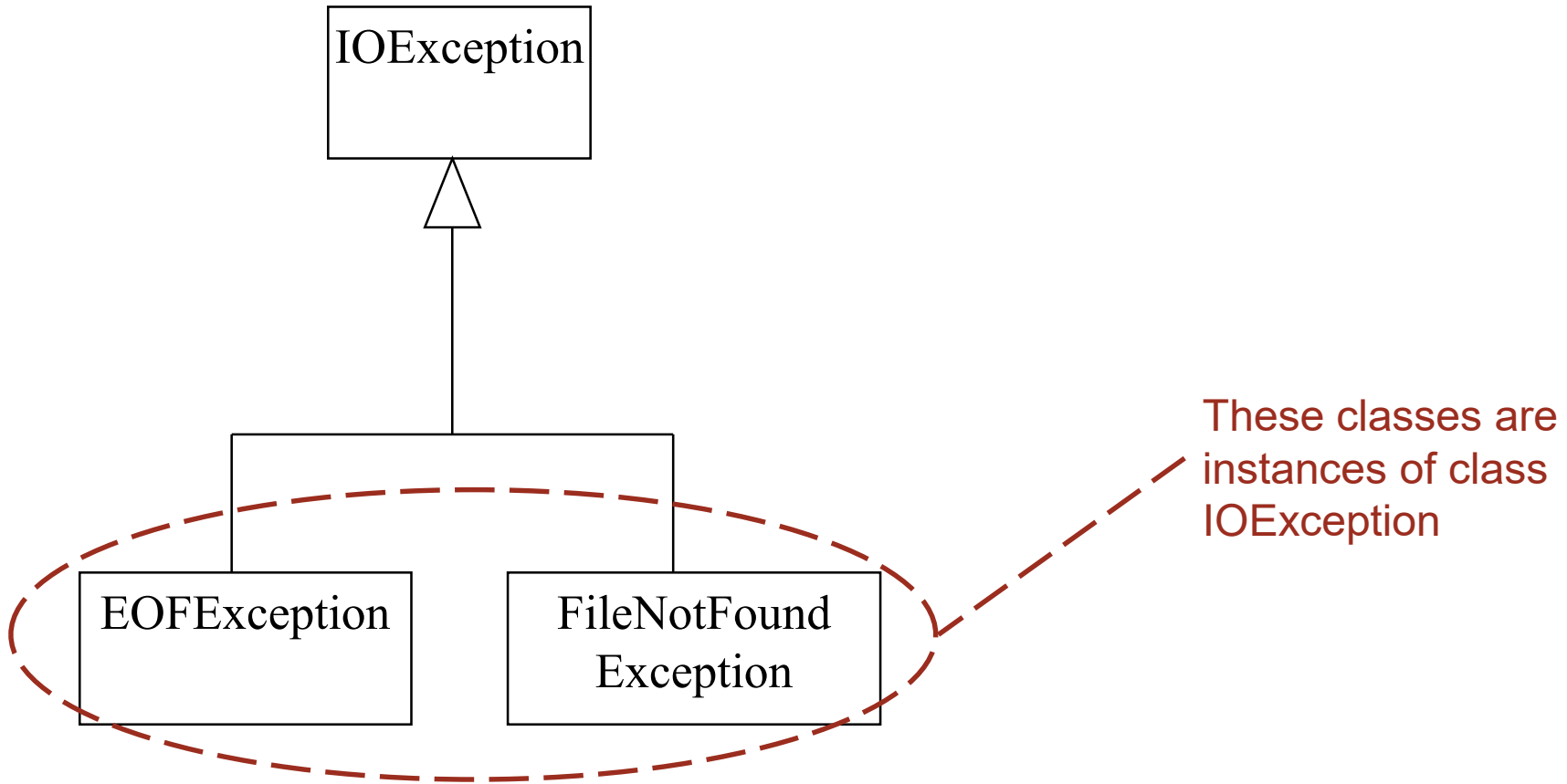
```
public class Example {  
    public void kiemTra(String fName1,String fName2) throws  
        MyException {  
        if (fName1.equals(fName2))  
            throw new MyException("Trung nhau");  
        System.out.println("Khong trung");  
    }  
}
```

# Sử dụng ngoại lệ tự định nghĩa (tt)



```
public class Test {  
    public static void main(String[] args) {  
        Example ex= new Example();  
        try {  
            String a = "Test";  
            String b = "Test";  
            ex.kiemTra(a,b);  
        } catch (MyException e) {  
            System.out.println(e.getMessage());  
        }  
    }  
}
```

# Cây thừa kế của lớp IOException



# Vấn đề bắt ngoại lệ



- Khi xử lý một chuỗi các ngoại lệ cần phải đảm bảo rằng các ngoại lệ lớp con được xử lý trước các ngoại lệ của lớp cha.
- Xử lý các trường hợp cụ thể trước khi xử lý các trường hợp tổng quát

# Vấn đề bắt ngoại lệ (tt)



## Đúng

```
try
{

}
catch (EOFException e)
{

}
catch (IOException e)
{

}
```

## Sai

```
try
{

}
catch (IOException e)
{

}
catch (EOFException e)
{

}
```

# Q & A

---

