

**UIT**  
TRƯỜNG ĐẠI HỌC  
CÔNG NGHỆ THÔNG TIN

# **NGÔN NGỮ LẬP TRÌNH JAVA**

## **Chương 4** **NHẬP XUẤT VÀ QUẢN LÝ NGOẠI LỆ (P2)** **QUẢN LÝ NGOẠI LỆ**

**GVGD: ThS. Lê Thanh Trọng**

# NỘI DUNG

- ❖ Mục tiêu bài học
- ❖ Biệt lệ
- ❖ Bắt và xử lý biệt lệ
- ❖ Checked và unchecked exception
- ❖ Các từ khóa try-catch-finally
- ❖ Các từ khóa throw và throws
- ❖ Tạo biệt lệ mới và cách sử dụng
- ❖ Một số vấn đề liên quan
- ❖ Tóm tắt bài học

# MỤC TIÊU BÀI HỌC

- ❖ Cung cấp cho sinh viên các kiến thức về biệt lệ trong java.
- ❖ Cách bắt và xử lý biệt lệ cũng như phân biệt được 2 loại biệt lệ là unchecked và checked.
- ❖ Các khối try- catch và finally, throw và throws trong mô hình xử lý biệt lệ
- ❖ Có khả năng tạo và sử dụng một biệt lệ tự định nghĩa
- ❖ Vấn đề liên quan đến thừa kế trong việc bắt và xử lý biệt lệ

# Đặt vấn đề: Chương trình sau có an toàn?

```
class Example2 {  
    public static void main(String args[]) {  
        int a, b, c;  
        BufferedReader br = new BufferedReader(new  
FileReader("E:/example.txt"));  
        String str1=br.readLine();  
        a=Integer.parseInt(str1);  
        String str2=br.readLine();  
        b=Integer.parseInt(str2);  
        c=a+b;  
        System.out.println(c);  
    }  
}
```

# Khái niệm biệt lệ

- ❖ Sự kiện xảy ra trong quá trình thực thi
- ❖ Diễn tiến chương trình bị cắt ngang và chương trình dừng “bất bình thường” nếu không xử lý.

# Khi nào xảy ra biệt lệ

- ❖ Invalid data
- ❖ Không tìm thấy file cần truy xuất
- ❖ Ngắt kết nối mạng
- ❖ Hết bộ nhớ

# Các cách xử lý

- ❖ Sử dụng các mệnh đề điều kiện kết hợp với các giá trị cờ
- ❖ Sử dụng cơ chế xử lý biệt lệ

# Ví dụ: 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;
        }
        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

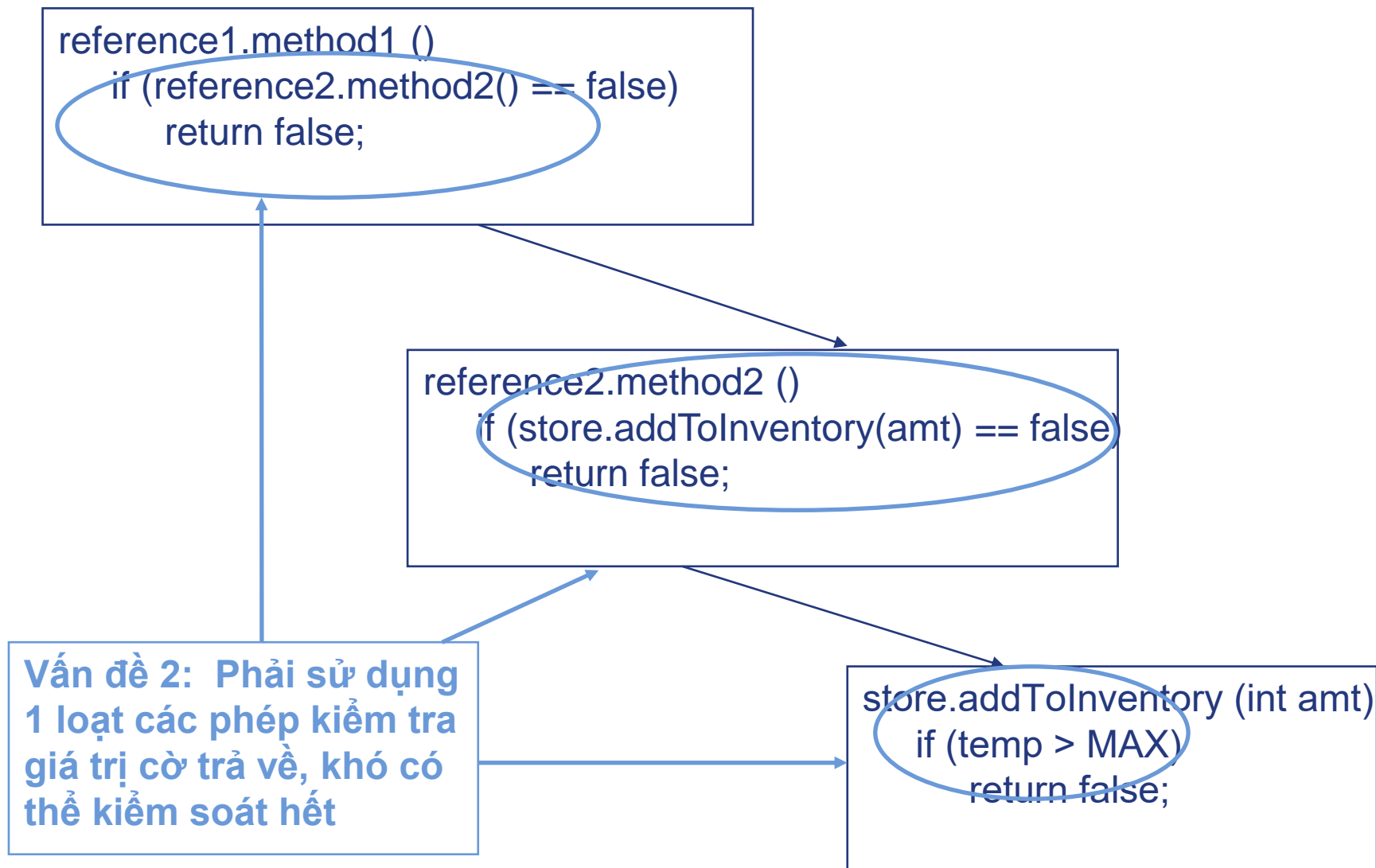
```
reference1.method1 ()  
    if (reference2.method2() == false)  
        return false;
```

**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



# 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;
```

# So sánh 2 phương pháp xử lý

```
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; //**
}
} else {
```

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

# So sánh 2 phương pháp xử lý



UIT  
TRƯỜNG ĐẠI HỌC  
CÔNG NGHỆ THÔNG TIN

```
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;  
}  
}
```

## ❖ Mục đích

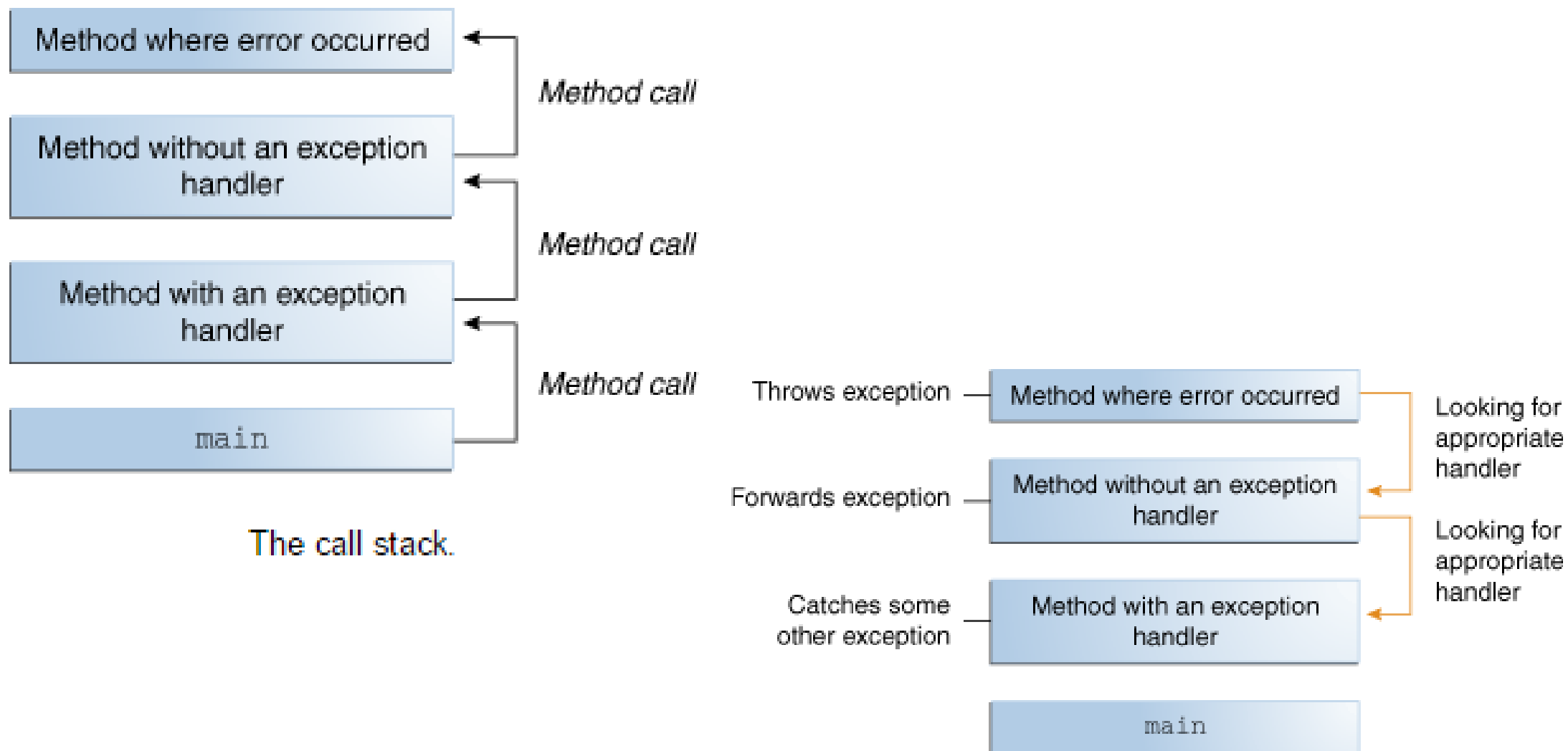
- An toàn hơn
- Tránh kết thúc bất thường
- Tách biệt các lệnh có thể xảy ra ngoại lệ và các lệnh xử lý ngoại lệ
- Tránh việc lan truyền mã lỗi trong dãy các lời gọi hàm tuần tự
- Giải phóng tài nguyên

# Mô hình xử lý biệt lệ

- ❖ Đóng gói nơi có thể 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 xử lý lỗi
- ❖ Tách biệt luồng điều khiển bất thường với luồng bình thường
- ❖ Phải được xử lý ở tại phương thức sinh ra ngoại lệ hoặc ủy nhiệm cho phương thức gọi đến



# Mô hình xử lý biệt lệ



# Mô hình xử lý biệt lệ

- ❖ 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 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.

# Các từ khóa

- ❖ try
- ❖ catch
- ❖ finally
- ❖ throws
- ❖ throw

# Xử lý biệt lệ

❖ Cú pháp:

**try**

{

// Code that may cause an error/exception to occur

}

**catch** (ExceptionType identifier)

{

// Code to handle the exception

}

# Xử lý biệt lệ: đọc dữ liệu từ bàn phím

```
import java.io.*;

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

# Xử lý biệt lệ: 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)
{
    //
}
}
```

# Xử lý biệt lệ: Biệt 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);
```

```
}
```

Biệt lệ có thể xảy ra ở  
đây





# Lớp BufferedReader

<http://java.sun.com/j2se/1.4.1/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);
}
```

Biệt lệ có thể xảy ra ở đây

# Lớp Integer

❖ <http://java.sun.com/j2se/1.4.1/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 biệt lệ

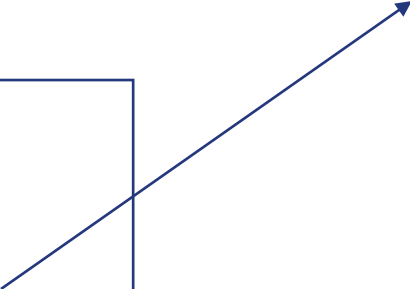


**UIT**  
TRƯỜNG ĐẠI HỌC  
CÔNG NGHỆ THÔNG TIN

```
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 biệt lệ

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



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

# Cơ chế xử lý biệt lệ

```
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 biệt lệ

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

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

# Cơ chế xử lý biệt lệ

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

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



# Cơ chế xử lý biệt lệ

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

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

# Bắt biệt lệ

```
catch (NumberFormatException e)
```

```
{
```

```
    :    :    :
```

```
}
```

```
}
```

```
}
```

# Bắt biệt lệ

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

# Bắt biệt lệ

Nhập vào: "exception"

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

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)
```

# Các loại biệt lệ

❖ Unchecked

❖ Checked

# Đặc điểm của biệt lệ unchecked

- ❖ Trình biên dịch không yêu cầu phải bắt các biệt lệ khi nó xảy ra.
  - *Không cần khối try-catch*
- ❖ Các biệt 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.
- ❖ Ví dụ:
  - `NullPointerException`, `IndexOutOfBoundsException`, `ArithmeticException`...

# Biệt lệ unchecked: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;
```

# 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;
```



# 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)

# Biệt lệ checked

- ❖ Phải xử lý khi biệt lệ có khả năng xảy ra
  - Phải sử dụng khối try-catch
- ❖ Liên quan đến 1 vấn đề cụ thể
  - Khi một phương thức được gọi thi hành
- ❖ Ví dụ:
  - IOException

# Tránh bỏ qua việc xử lý biệt lệ

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

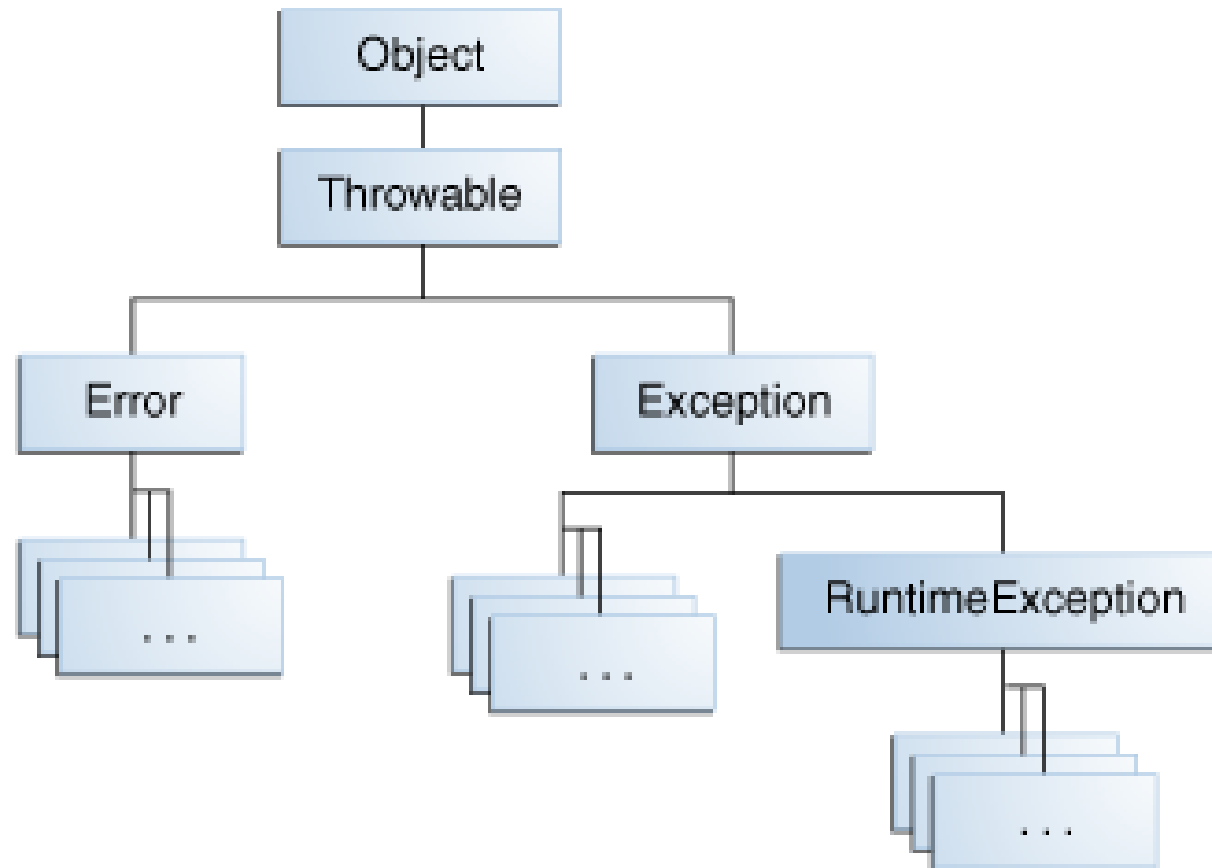
# Tránh bỏ qua việc xử lý biệt lệ

**NO!**

```
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
    // block to bypass the
    // annoying compiler error
}
```

**try-catch**

# Checked vs Unchecked



# Lớp Throwable

- ❖ `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...)

## Ví dụ

```
public class
PrinStackTrace{
public static void
main(String args[]){
try {
    int a= Div(9,0);
    System.out.println(num);
}
catch(Exception e) {
    System.err.println("Lỗi:"
```

```
+ e.getMessage());
    e.printStackTrace();
}
}
static int Div(int a, int b)
{
    int re= a/ b;
    return re;
}
}
```

# Checked vs Unchecked

## ❖ Checked exceptions

- Xảy ra lúc biên dịch
- Không thể bỏ qua lúc biên dịch
- Ex: FileReader đọc dữ liệu file không tồn tại

## ❖ Unchecked exceptions

- Xảy ra lúc thực thi
- Có thể bỏ qua lúc biên dịch
- Ex: truy cập phần tử với chỉ số vượt quá kích thước mảng



# Unchecked exceptions

Exception	Description
ArithmeticException	Arithmetic error, such as divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
ArrayStoreException	Assignment to an array element of an incompatible type.
ClassCastException	Invalid cast.
IllegalArgumentException	Illegal argument used to invoke a method.
IllegalMonitorStateException	Illegal monitor operation, such as waiting on an unlocked thread.
IllegalStateException	Environment or application is in incorrect state.
IllegalThreadStateException	Requested operation not compatible with current thread state.

# Checked exceptions

## Exception

ClassNotFoundException

CloneNotSupportedException

IllegalAccessException

InstantiationException

InterruptedException

NoSuchFieldException

NoSuchMethodException

## Description

Class not found.

Attempt to clone an object that does not implement the Cloneable interface.

Access to a class is denied.

Attempt to create an object of an abstract class or interface.

One thread has been interrupted by another thread.

A requested field does not exist.

A requested method does not exist.

# Mệnh đề *finally*

- ❖ Là 1 mệnh đề không bắt buộc trong khối try-catch-*finally*
- ❖ Dùng để đặt khối lệnh sẽ được thi hành bất kể biệt lệ có xảy ra hay không
- ❖ Khi có try, bắt buộc phải có catch hoặc finally hoặc cả 2

# Mệnh đề finally: có biệt lệ

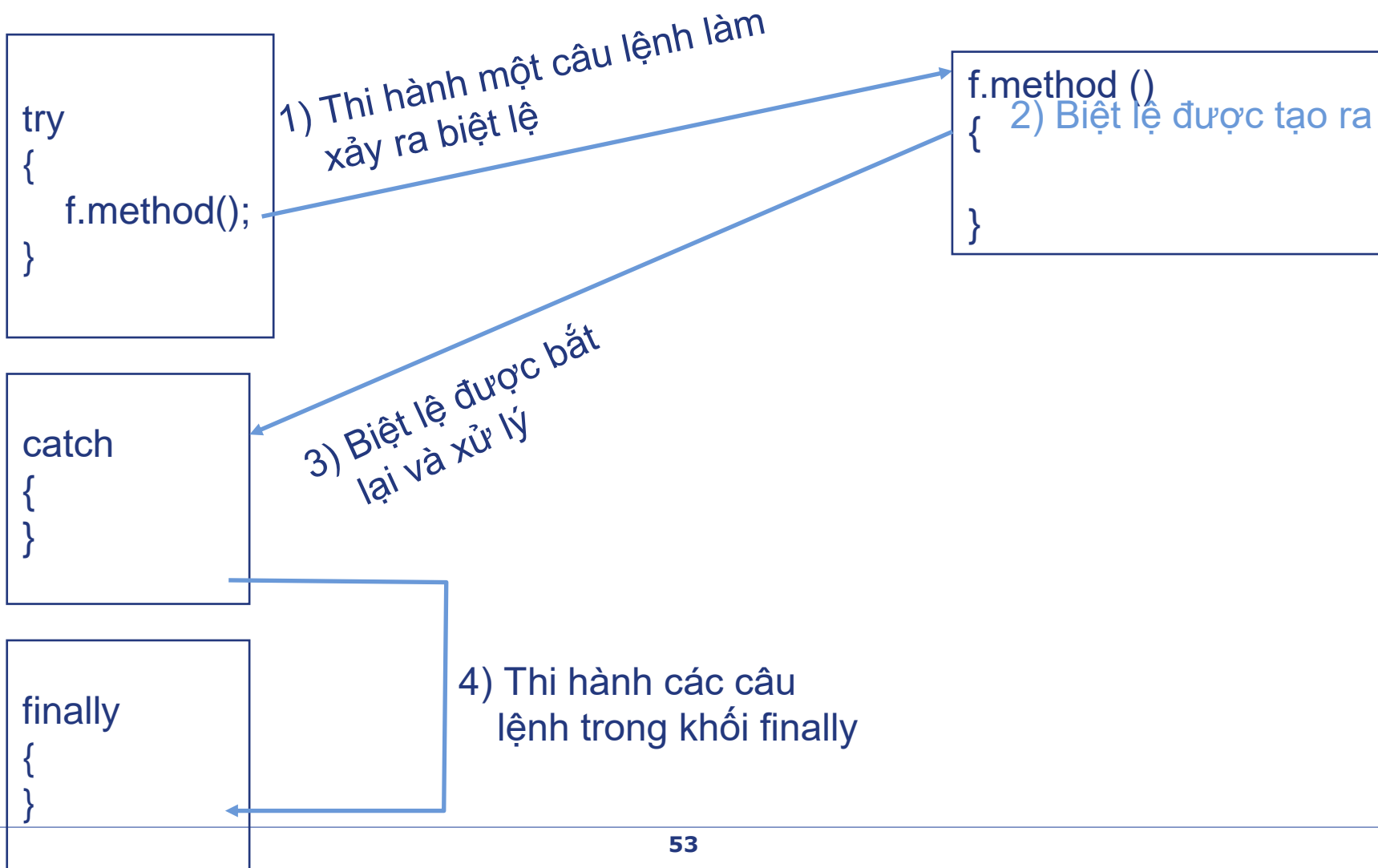
```
try  
{  
    f.method();  
}
```

```
catch  
{  
}  
}
```

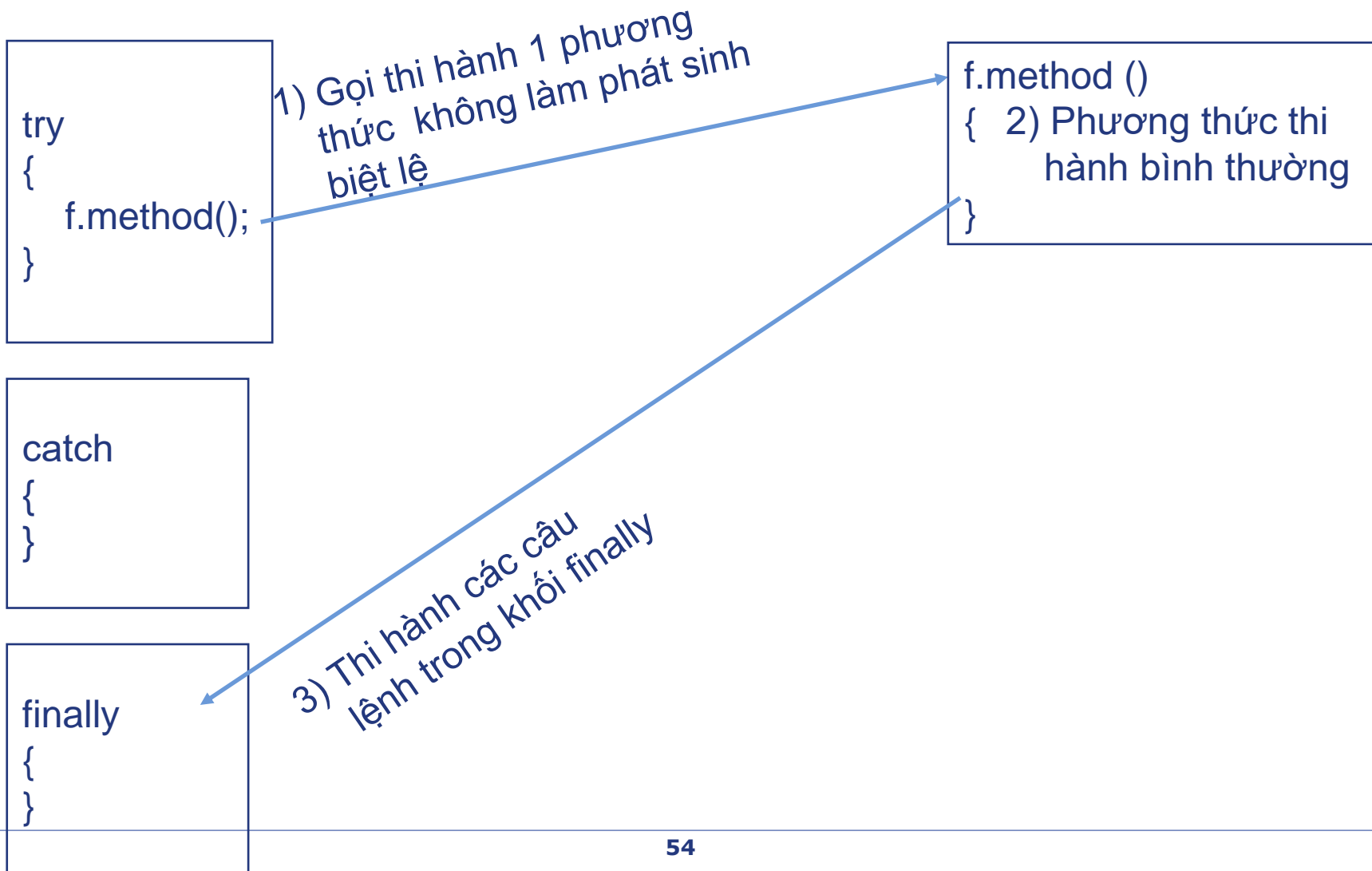
```
finally  
{  
}  
}
```

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

# Mệnh đề finally: có biệt lệ



# Mệnh đề finally: không có biệt lệ



# Try-Catch-Finally: Ví dụ

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



# Try-Catch-Finally: Ví dụ

```
    catch (IOException e)
    {
        e.printStackTrace();
        return;
    }
    catch (NumberFormatException e)
    {
        e.printStackTrace ();
        return;
    }
    finally
    {
        System.out.println("<<<This code will always execute>>>");
        return;
    }
}
```

# Từ khóa throw

- ❖ Dùng để chủ động phát sinh một biệt lệ một cách tường minh.
- ❖ Thường dùng chủ yếu trong việc phát sinh các biệt lệ do người dùng tạo ra.

# Từ khóa throw

```
// Java program that demonstrates the use of throw
class ThrowExcep
{
    static void fun()
    {
        try
        {
            throw new NullPointerException("demo");
        }
        catch(NullPointerException e)
        {
            System.out.println("Caught inside fun().");
            throw e; // rethrowing the exception
        }
    }
}
```

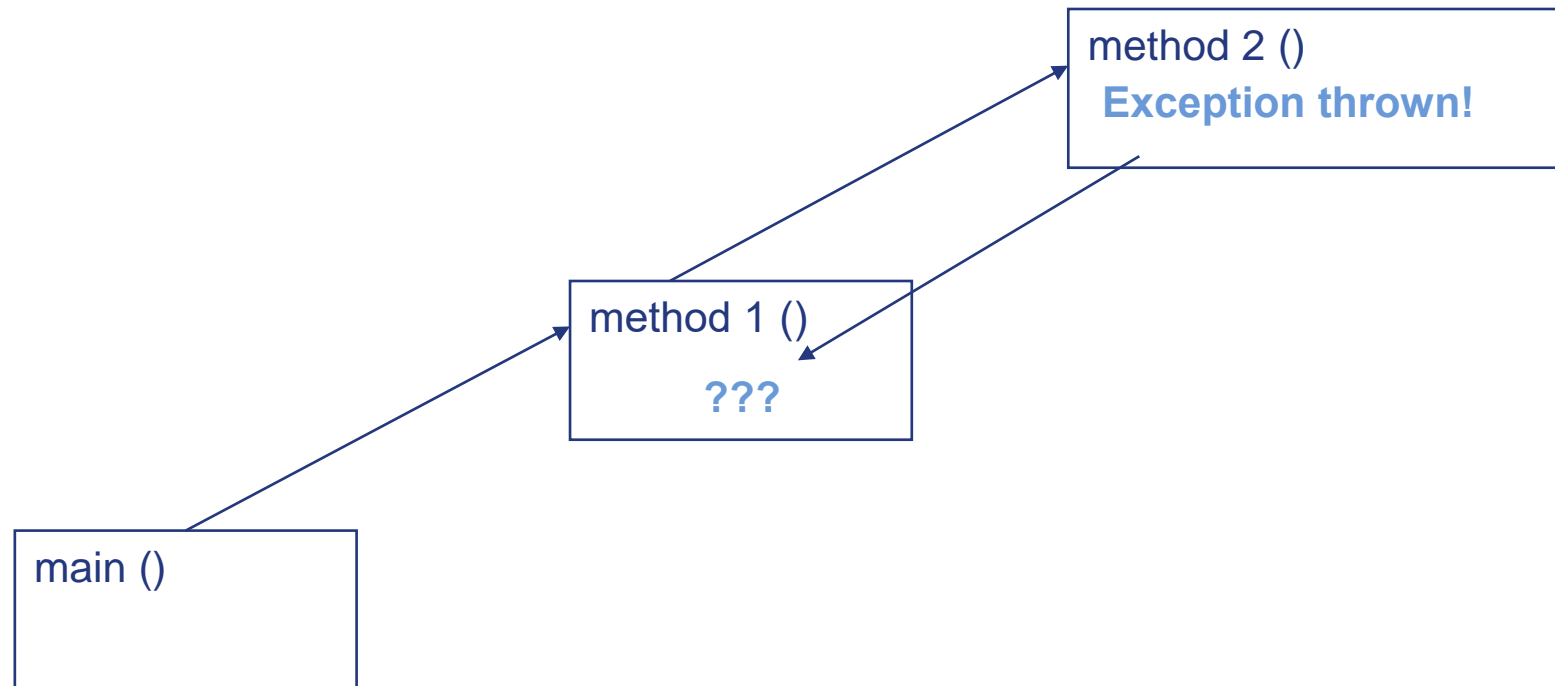
## Từ khóa throws

- ❖ Throws là keyword của Java được sử dụng để làm kí hiệu cho biết 1 phương thức nào đó sẽ có thể phát sinh một trong cách ngoại lệ được chỉ định.
- ❖ cú pháp:
- ❖ Returntype `method_name(parameters) throws EXP1, EXP2`

# Từ khóa throws

❖ Ví dụ:

```
class ThrowsExcep
{
    static void fun() throws IllegalAccessException
    {
        System.out.println("Inside fun(). ");
        throw new IllegalAccessException("demo");
    }
}
```



# Hàm được gọi không thể xử lý biệt lệ

```
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 được gọi không thể xử lý biệt lệ

```
class Driver
{
    public static void main (String [] args)
    {
        TCExample eg = new TCExample ();
        boolean inputOkay = true;
```



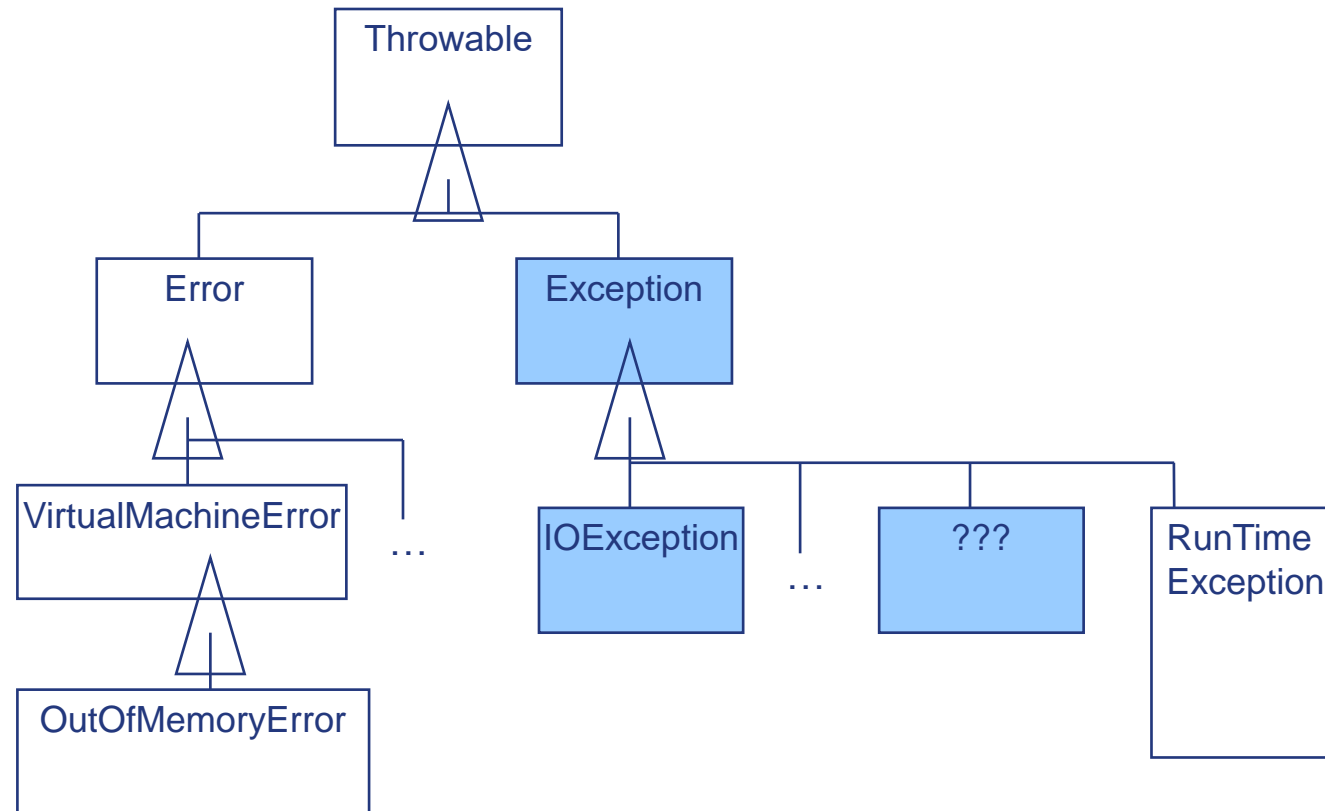
# Hàm được gọi không thể xử lý biệt lệ

```
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
```

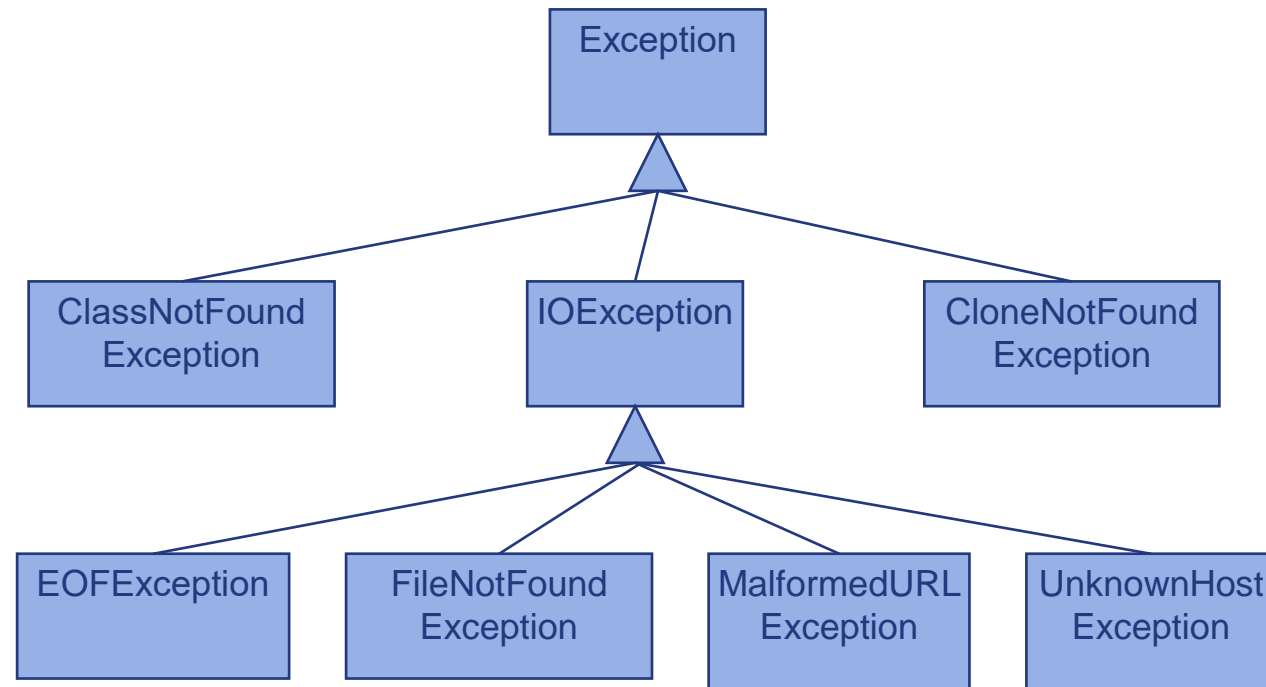
# Hàm main() không xử lý biệt lệ

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

# Sơ đồ phân cấp biệt lệ



# Lớp Exception



# Một số lớp con của Exception

❖ ClassNotFoundException, SQLException

❖ java.io.IOException:

- FileNotFoundException, EOFException...

❖ RuntimeException:

- NullPointerException, BufferOverflowException
- ClassCastException, ArithmeticException
- IndexOutOfBoundsException:
  - ArrayIndexOutOfBoundsException
  - StringIndexOutOfBoundsException
- IllegalArgumentException:
  - NumberFormatException,
  - InvalidParameterException...

## Tạo biệt lệ mới

- ❖ Các ngoại lệ do hệ thống xây dựng không đáp ứng chưa đủ yêu cầu kiểm soát lỗi → người dùng xây dựng các lớp ngoại lệ mới.
  - Kế thừa từ một lớp Exception hoặc lớp con của nó
  - Có tất cả các phương thức của lớp Throwable

## Tạo biệt lệ mới

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

# Tạo biệt lệ mới

```
public class FileExample
{
    public void copyFile(String fName1, String fName2)
throws MyException
    {
        if (fName1.equals(fName2))
            throw new MyException("File trùng tên");
        // Copy file
        System.out.println("Copy completed");
    }
}
```

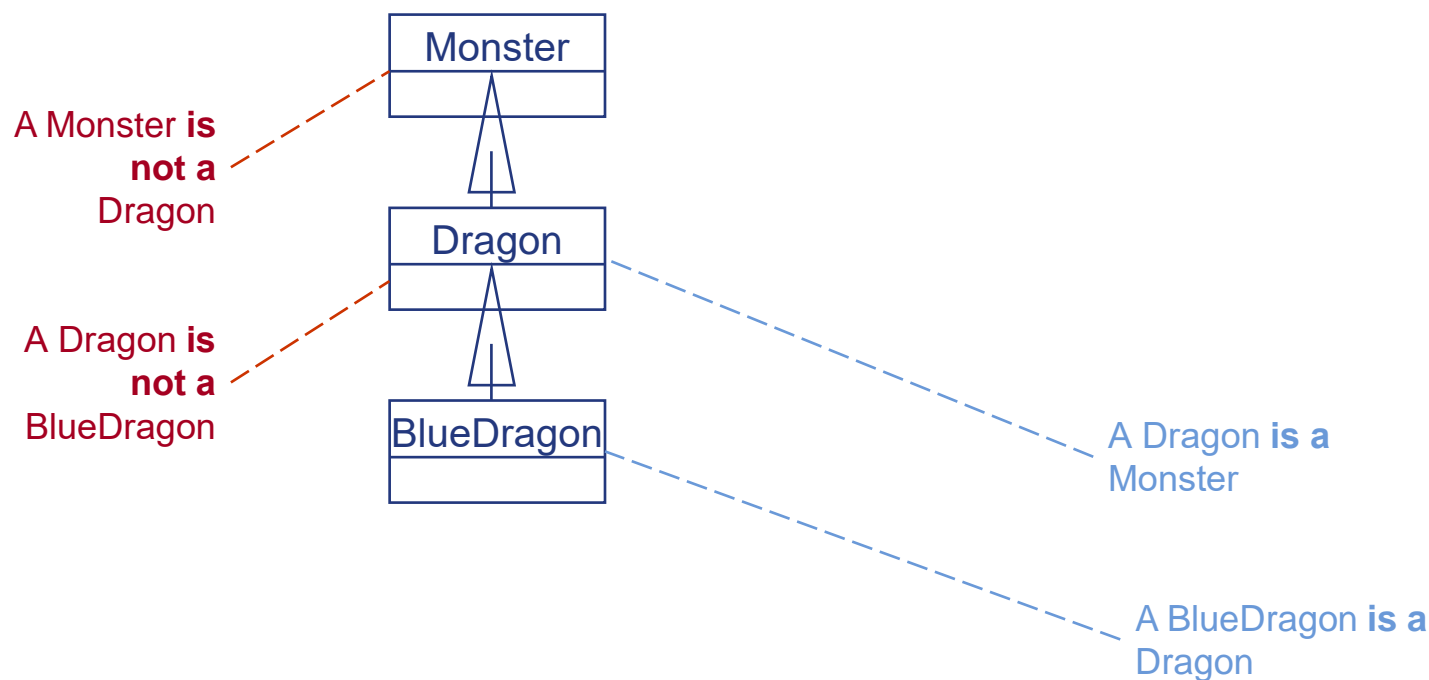


## Tạo biệt lệ mới

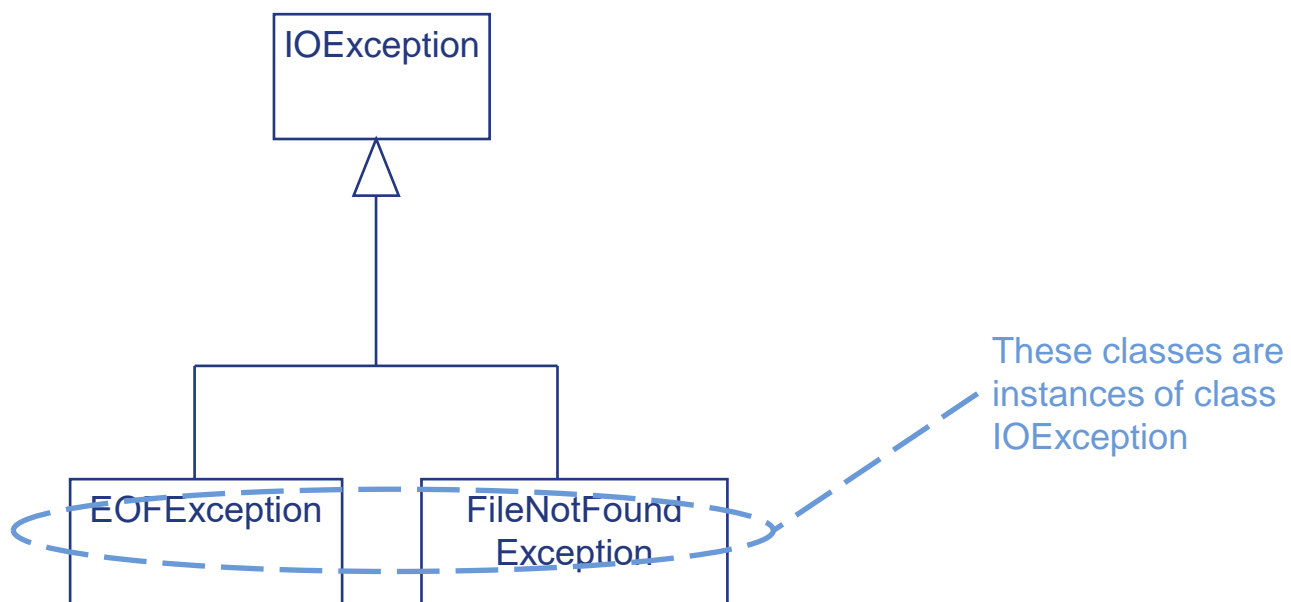
```
public class Test {  
    public static void main(String[] args) {  
        FileExample obj = new FileExample();  
        try {  
            String a = args[0];  
            String b = args[1];  
            obj.copyFile(a,b);  
        } catch (MyException e1) {  
            System.out.println(e1.getMessage());  
        }  
        catch(Exception e2) {  
            System.out.println(e2.toString());  
        }  
    }  
}
```

## Nhắc lại thừa kế

- ❖ Có thể thay thế một đối tượng của lớp con cho 1 đối tượng của lớp cha (ngược lại không đúng).



# Cây thừa kế của lớp IOException



## Thừa kế và vấn đề bắt biệt lệ

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

# Thừa kế và vấn đề bắt biệt lệ

## Đúng

```
try
{

}
catch (EOFException e)
{

}
catch (IOException e)
{

}
}
```

## Sai

```
try
{

}
catch (IOException e)
{

}
catch (EOFException e)
{

}
}
```

# TÓM TẮT BÀI HỌC

- ❖ Trong quá trình lập trình, ta nên tiến hành đánh bắt và xử lý các ngoại lệ có thể phát sinh bằng các từ khóa try, catch, finally
- ❖ Các ngoại lệ dạng checked là bắt buộc bắt để đảm bảo hoạt động của chương trình
- ❖ Code có thể phát sinh ngoại lệ sẽ được đặt trong khối try, catch sẽ xử lý khi xuất hiện ngoại lệ và khối finally (nếu có) sẽ luôn được thi hành

# TÓM TẮT BÀI HỌC

- ❖ Từ khóa throws để đánh dấu một phương thức có thể xảy ra biệt lệ.
- ❖ Từ khóa throw chủ động phát sinh biệt lệ trong code.
- ❖ Tạo một biệt lệ mới bằng cách xây dựng một lớp kế thừa từ lớp Exception và cài đặt các phương thức phù hợp.
- ❖ Biệt lệ lớp con cần được xử lý trước các biệt lệ của lớp cha.