ĐẠI NAM
UNIVERSITY

# XỬ LÝ NGOẠI LỆ

**Thời gian: 4 tiết**

# Nội dung

- ✓ Ngoại lệ
- ✓ Bắt và xử lý ngoại lệ
- ✓ Ủy nhiệm ngoại lệ
- ✓ Tự định nghĩa ngoại lệ

# Ngoại lệ

- Ngoại lệ (exception) = Exceptional event
- Định nghĩa: Ngoại lệ là một sự kiện xảy ra trong quá trình thực thi chương trình, phá vỡ luồng bình thường của chương trình
- Ví dụ

```
int i = 4/0;
```

ERROR!

# Ngoại lệ

- Ngoại lệ là một lỗi đặc biệt
- Xảy ra tại thời điểm chạy chương trình (runtime)
- Khi xảy ra một ngoại lệ, nếu không xử lý thì chương trình kết thúc ngay và trả lại quyền điều khiển cho hệ điều hành.
- Kết thúc bất thường chương trình
- Kết quả thực thi không mong muốn

# Cách xử lý lỗi truyền thống

- Viết mã xử lý tại nơi phát sinh ra lỗi
  - Làm cho chương trình trở nên rối
  - Không phải lúc nào cũng đầy đủ thông tin để xử lý
  - Không nhất thiết phải xử lý
- Truyền trạng thái lên mức trên
  - Thông qua tham số, giá trị trả lại hoặc biến tổng thể (flag)
  - Dễ nhầm
  - Vẫn còn khó hiểu

# 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
  - Bản chất con người
  - Thiếu kinh nghiệm, cố tình bỏ qua

# Mục đích của xử lý ngoại lệ

- 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

```
.....
IF B IS ZERO GO TO ERROR
C = A/B
PRINT C
GO TO EXIT

ERROR:
    DISPLAY “DIVISION BY ZERO”

EXIT:
    END
```

Khối xử lý lỗi

# Phân tách code

- Lập trình truyền thống: hàm readFile cần nhiều mã nguồn để phát hiện, thông báo và xử lý lỗi.

```
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;
                }
            }
            …
```

# Xử lý ngoại lệ (tiếp)

```
                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;
}
```

# Cơ chế bắt ngoại lệ



```
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;
    }
```

*Cơ chế ngoại lệ cho phép tập trung viết mã cho luồng chính và xử lý trường hợp bất thường ở nơi khác*

# Mục đích của xử lý ngoại lệ

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):

```
method1 {
    call method2;
}

method2 {
    call method3;
}

method3 {
    call readFile;
}
```

# Phương pháp lan truyền lỗi



```
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.*

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;
}
```

# Mục đích của xử lý ngoại lệ

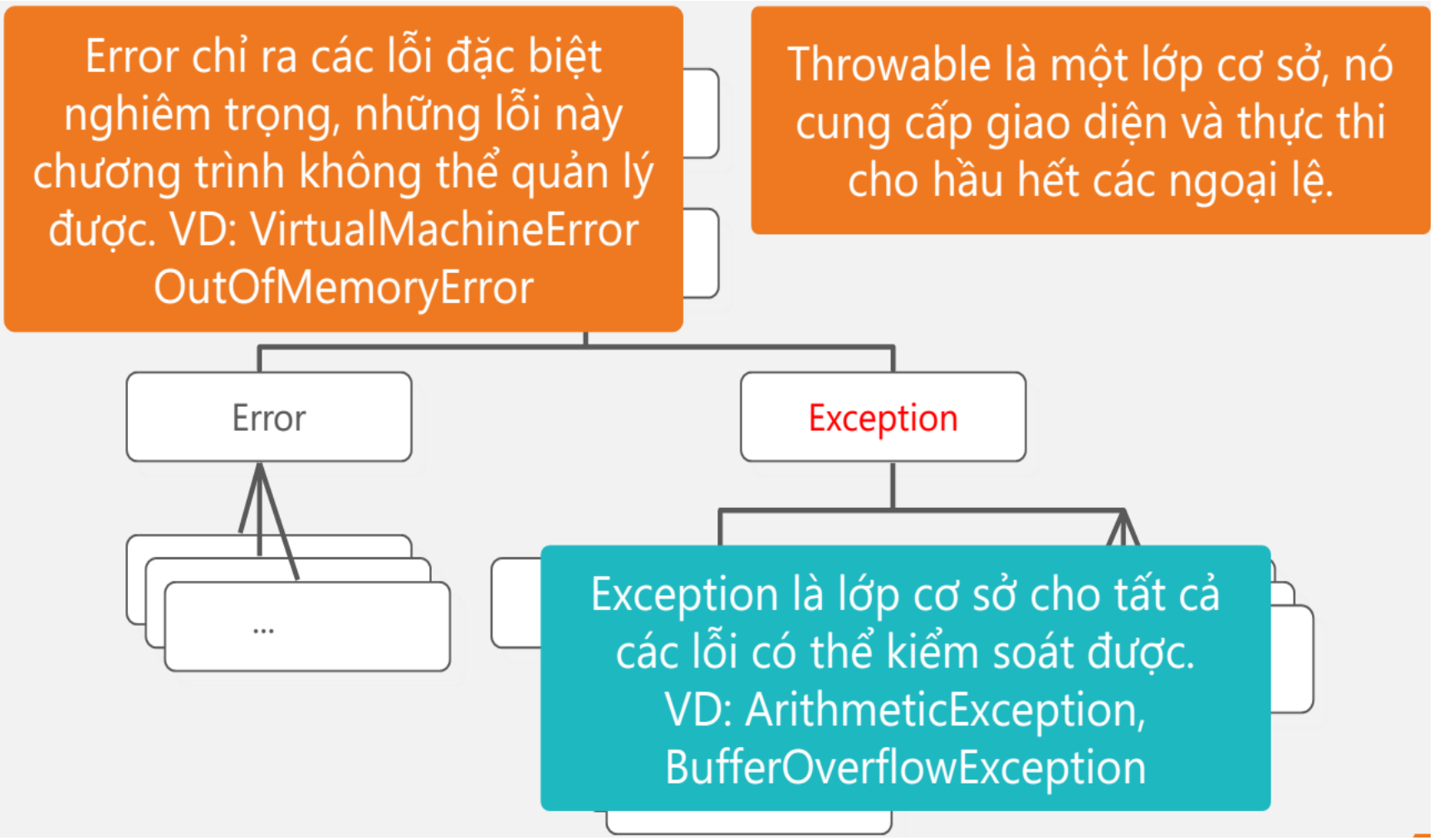
- Khi xảy ra ngoại lệ, nếu không có cơ chế xử lý thích hợp:
  - Chương trình bị ngắt khi ngoại lệ xảy ra
  - Các tài nguyên không được giải phóng → Lãng phí
- Ví dụ: Vào/ra tệp tin
  - Nếu ngoại lệ xảy ra (ví dụ như chuyển đổi kiểu không đúng) → Chương trình kết thúc mà không đóng tệp tin lại
    - Tệp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

# 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

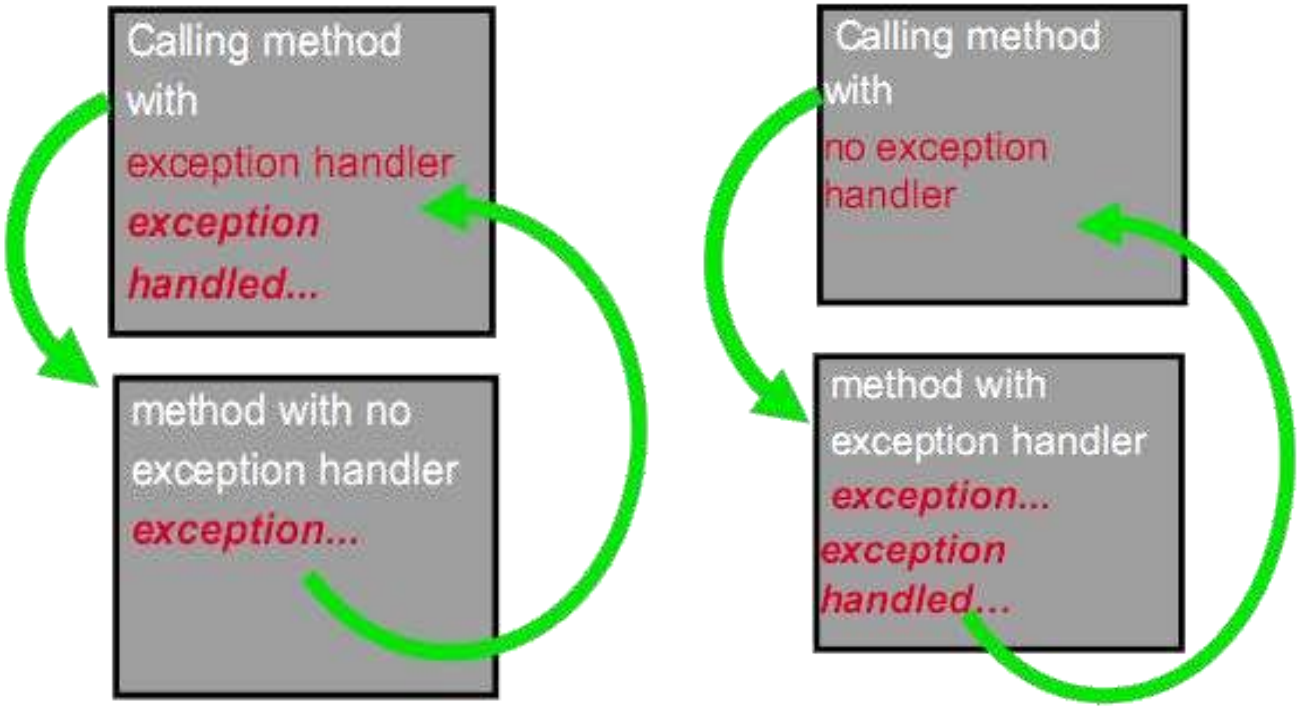
    được tạo ra chứa thông tin chi tiết về ngoại lệ
  - Cung cấp cơ chế hiệu quả trong việc xử lý lỗi
  - Tách biệt luồng điều khiển bất thường với luồng bình thường

# Cây phân cấp ngoại lệ trong Java



# Mô hình xử lý ngoại lệ

- Ngoại lệ cần phải được xử lý ở tại phương thức sinh ra ngoại lệ hoặc ủy nhiệm cho phương thức gọi đến



# Xử lý ngoại lệ trong Java

- 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.

# Xử lý ngoại lệ trong Java

- Keyword
  - `try`
  - `catch`
  - `finally`
  - `throw`
  - `throws`

# Khối try/catch

- 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:

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

# Ví dụ

```
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!");
    }
  }
}
```

# Ví dụ

```
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 no1){
        int num = no / no1;
        return num;
    }
}
```

# 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...)

# Ví dụ:

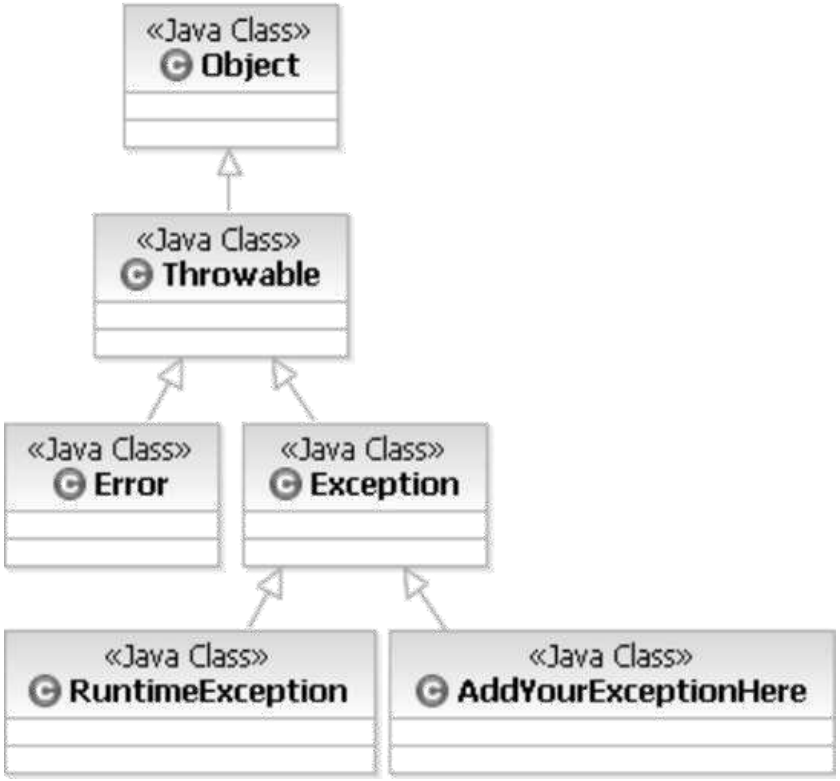
```
public class StckExceptionDemo {
    public static void main(String args[]){
        try {
            int num = calculate(9,0);
            System.out.println(num);
        }
        catch(Exception e) {
            System.err.println("Co loi xay ra :" +
                                e.getMessage());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

# 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 (unrecoverable exception)
- Không cần kiểm tra trong mã nguồn Java của bạn
- Các lớp con:
  - VirtualMachineError: InternalError, OutOfMemoryError, StackOverflowError, UnknownError
  - ThreadDeath
  - LinkageError

# 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 thực hiện 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

# Ví dụ IOException

```
import java.io.InputStreamReader;
import java.io.IOException;
public class HelloWorld {
    public static void main(String[] args) {
      InputStreamReader isr = new
          InputStreamReader(System.in);
      try {
          System.out.print("Nhap vao 1 ky tu: ");
          char c = (char) isr.read();
          System.out.println("Ky tu vua nhap: " + c);
       } catch(IOException ioe) {
           ioe.printStackTrace();
       }
    }
}
```

```
Nhap vao 1 ky tu: b
Ky tu vua nhap: b
Press any key to continue . . .
```

# 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 {
 // Doan ma co the gay ra nhieu ngoai le
} catch (ExceptionType1 e1) {
    // Xu ly ngoai le 1
} catch (ExceptionType2 e2) {
    // Xu ly ngoai le 2
} ...
```

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

# Chú ý

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

```
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 hv!");
     }
     catch(NumberFormatException e2) {
        System.out.println("Not a number!");
     }
   }
 }
```

# Ví dụ

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

```
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 hv!");
     }
     catch(NumberFormatException e2){
        System.out.println(“Hay nhap 1 so!");
     }
  } }
```

# Ví dụ:

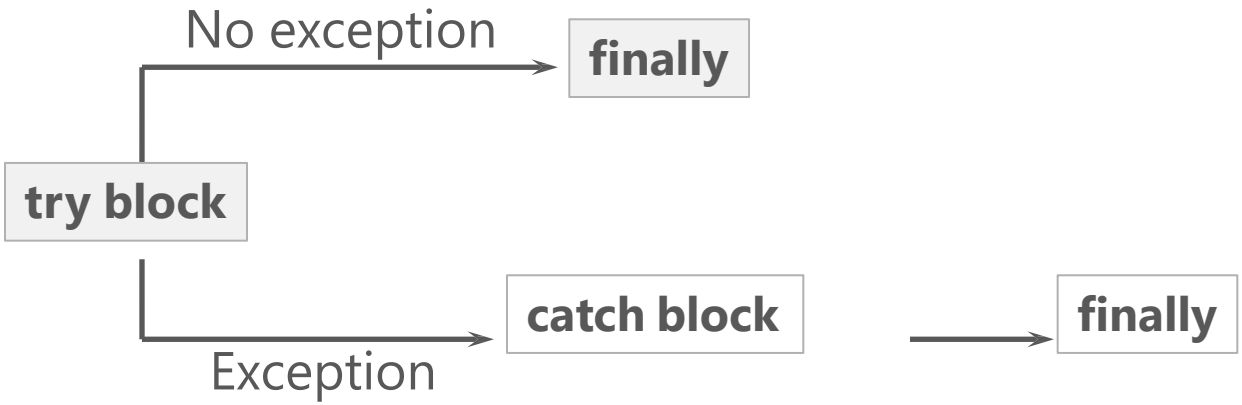
```
class MultiCatch2 {
  public static void main( String args[]) {
  try {
    // format a number
    // read a file
    // something else...
   }
   catch(IOException e) {
    System.out.println("I/O error "+e.getMessage();
   }
   catch(NumberFormatException e) {
    System.out.println("Bad data "+e.getMessage();
   }
   catch(Throwable e) { // catch all
    System.out.println("error: " + e.getMessage();}
   }
  }
}
```

# Ví dụ:

```
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
    }
}
...
```

# Khối finally

- Đảm bảo thực hiện tất cả các công việc cần thiết khi có ngoại lệ xảy ra
  - Đóng file, đóng socket, connection
  - Giải phóng tài nguyên (nếu cần)...
- Chắc chắn sẽ thực hiện dù ngoại lệ có xảy ra hay không.



# Cú pháp 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

# Ví dụ

```
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");
         }
         finally {
             System.out.println(“Khoi finally");
         }
    }
    static void staticLengthmethod() {
        System.out.println(str.length());
    }
}
```

```
Truoc ngoai le
Da xay ra loi
Khoi finally
```

# Ví dụ:

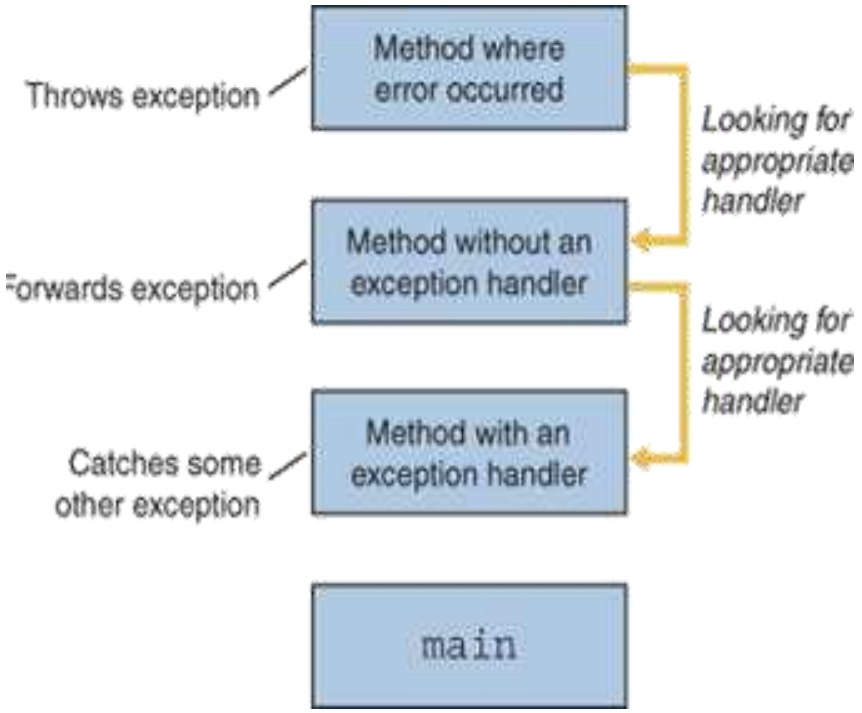
```
public void openFile(){
    try {
        // constructor may throw
        // FileNotFoundException
        FileReader reader = new FileReader("foo");
        int i=0;
        while(i != -1) {
            //reader.read() may throw IOException  i =
            reader.read(); System.out.println((char)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

# Ủy nhiệm ngoại lệ

Throw một ngoại lệ

# Hai cách làm việc với ngoại lệ

- Xử lý ngay
  - Sử dụng khối try ... catch (finally nếu cần).
- Ủy nhiệm cho vị trí gọi nó
  - Xử lý ngay, hoặc
  - throw ra ngoài phạm vi
    - Sử dụng từ khóa throw



# Ủy nhiệm ngoại lệ

- Phương thức có thể ủy nhiệm ngoại lệ cho vị trí gọi nó bằng cách:
  - Sử dụng throws ở phần định nghĩa phương thức để báo hiệu cho vị trí ExceptionType gọi nó biết là nó có thể phát sinh ngoại lệ ExceptionType
    Sử dụng throw anExceptionObject trong thân phương thức để tung ra ngoại lệ khi cần.
  - Ví dụ

```
public void myMethod(int param) throws Exception{
    if (param < 10) {
            throw new Exception("Too low!");
}
    //...
}
```

# Ủy nhiệm ngoại lệ

- Nếu phương thức có chứa câu lệnh tung ngoại lệ (throw) thì phần khai báo phương thức phai khai báo là có tung ngoại lệ đó

```
public void myMethod(int param) {
    if (param < 10) {
            throw new Exception("Too low!");
}
    //Blah, Blah, Blah...
}
```

→ unreported exception java.lang.Exception; must be caught or declared to be thrown

# Ủy nhiệm ngoại lệ

- Phương thức không cần phải khai báo sẽ tung ra RuntimeException vì ngoại lệ này mặc định được ủy nhiệm cho JVM
- Ví dụ

```
class Test {
 public void myMethod(int param) {
     if (param < 10) {
       throw new RuntimeException("Too low!");
     }
     // ...
 }
} //OK
```

# Ủy nhiệm ngoại lệ

- Tại vị trí gọi phương thức có ủy nhiệm ngoại lệ (trừ RuntimeException):
  - Hoặc là phương thức chứa vị trí đó phải ủy nhiệm tiếp cho vị trí gọi mình
  - Hoặc là tại vị trí gọi phải bắt ngoại lệ ủy nhiệm (hoặc lớp cha) và xử lý ngay bằng try...catch (finally nếu cần)

# Ví dụ

```
public class DelegateExceptionDemo {
  public static void main(String args[]){
       int num = calculate(9,3);
       System.out.println(“Lan 1: ” + num);
       num = calculate(9,0);
       System.out.println(“Lan 2: ” + num);
  }
  static int calculate(int no, int no1)
                          throws ArithmeticException {
              if (no1 == 0)
                   throw new
          ArithmeticException("Khong the chia cho 0!");
    int num = no / no1;
    return num;
   }
}
```

```
Lan 1: 3
Exception in thread "main" java.lang.ArithmeticException: Khong the chia cho 0!
        at DelegateExceptionDemo.calculate(DelegateExceptionDemo.java:11)
        at DelegateExceptionDemo.main(DelegateExceptionDemo.java:5)
Press any key to continue . . .
```

# Ví dụ

```
public class DelegateExceptionDemo {
        public static void main(String args[]) {
                int num = calculate(9,3);
                System.out.println("Lan 1: " + num);
                num = calculate(9,0);
                System.out.println("Lan 2: " + num);
        }
        static int calculate(int no, int no1) {
                  if (no1 == 0) throw new
                                ArithmeticException("Khong the chia cho 0!");
                  throws Exception {
                                int num = no / no1;
                                return num;
                  }
         }
}
```

```
G:\Java Example\DelegateExceptionDemo.java:3: unreported exception
java.lang.Exception; must be caught or declared to be thrown
                  int num = calculate(9,3);
                                      ^
G:\Java Example\DelegateExceptionDemo.java:5: unreported exception
java.lang.Exception; must be caught or declared to be thrown
                  num = calculate(9,0);
```

# Ví dụ

```
public class DelegateExceptionDemo {
    public static void main(String args[]) {
        try {
                int num = calculate(9,3);
                System.out.println("Lan 1: " + num);
                num = calculate(9,0);
                System.out.println("Lan 2: " + num);
        } catch(Exception e) {
                System.out.println(e.getMessage());
        }
    }
    static int calculate(int no, int no1)
        throws ArithmeticException {
                if (no1 == 0) throw new
                        ArithmeticException("Khong the chia cho 0!");
         int num = no / no1;
         return num;
    }
}
```

```
Lan 1: 3
Khong the chia cho 0!
Press any key to continue
```

# Ủy nhiệm ngoại lệ

- Một phương thức có thể ủy nhiệm nhiều hơn 1 ngoại lệ

```
public void myMethod(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

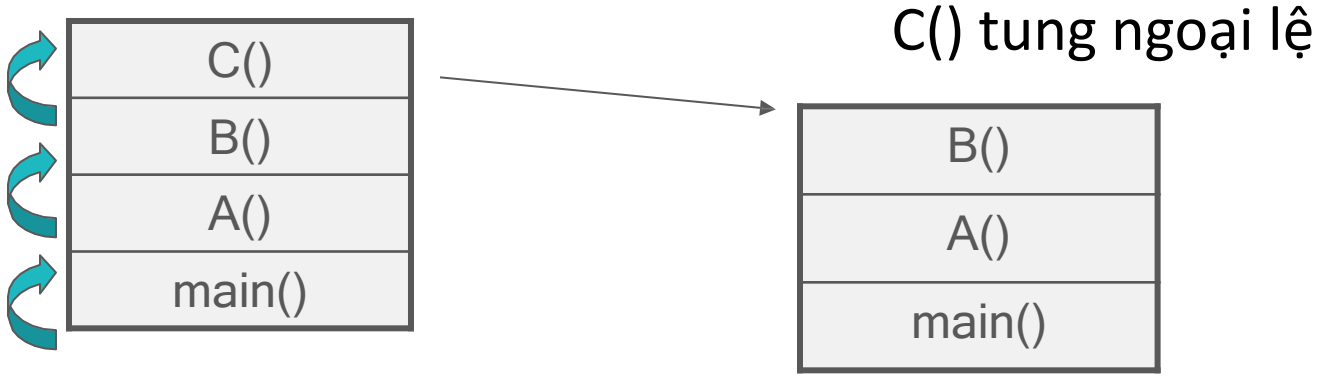
- Tình huống:
  - Giả sử trong main() gọi phương thức A(), trong A() gọi B(), trong B() gọi C(). Khi đó một ngăn xếp các phương thức được tạo ra.
  - Giả sử trong C() xảy ra ngoại lệ.

# Lan truyền ngoại lệ

Nếu C() gặp lỗi và tung ra ngoại lệ nhưng trong C() lại không xử lý ngoại lệ này, thì chỉ còn một nơi có thể xử lý chính là nơi mà C() được gọi, đó là trong phương thức B().

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 tung từ C() thì chương trình sẽ phải dừng lại.



# Kế thừa và ủy nhiệm 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.

# Kế thừa và ủy nhiệm ngoại lệ

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

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

# Ưu điểm của ủy nhiệm ngoại lệ

- Dễ sử dụng
  - Làm chương trình dễ đọc và an toàn hơn
  - Dễ dàng chuyển điều khiển đến nơi có khả năng xử lý ngoại lệ
  - Có thể ném nhiều loại ngoại lệ
- Tách xử lý ngoại lệ khỏi đoạn mã thông thường
- Không bỏ sót ngoại lệ (ném tự động)
- Gom nhóm và phân loại các ngoại lệ
- Làm chương trình dễ đọc và an toàn hơn

# Ngoại lệ tự định nghĩa

## Lớp Exception tự viết

# Tạo ngoại lệ tự định nghĩa

- Các ngoại lệ do hệ thống xây dựng không đủ để kiểm soát tất cả các lỗi. Cần phải có các lớp ngoại lệ do người dùng định nghĩa.
- Kế thừa từ một lớp Exception hoặc lớp con của nó
- Có tất cả các phương thức của lớp Throwable

```
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

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

# Sử dụng ngoại lệ tự định nghĩa

- Bắt và xử lý ngoại lệ

```
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());
        }
   }
}
```

```
C:\>java Test a1.txt a1.txt
File trung ten

C:\>java Test
java.lang.ArrayIndexOutOfBoundsException: 0
```

# XIN CẢM ƠN!