ĐẠI NAM
UNIVERSITY

CHƯƠNG 7
XỬ LÝ NGOẠI LỆ

Giảng viên: Phạm Văn Tiệp

ĐẠI NAM
UNIVERSITY

NGOẠI LỆ LÀ GÌ?

Ngoại lệ là gì?

- Là một sự kiện xảy ra trong quá trình thực thi chương trình, phá vỡ luồng bình thường của chương trình.
- **Không xử lý ngoại lệ:**
  - Lỗi lan truyền trong chương trình
  - Chương trình kết thúc bất thường
  - Tài nguyên không được giải phóng
  - Chương trình đưa ra kết quả không mong muốn → **bắt và xử lý ngoại lệ**

Ví dụ

```
public static void main(String[] args) {
   int x, fx;
   Scanner inputData = new Scanner(System.in);
   System.out.print("Enter x: ");
   x = inputData.nextInt();
   fx = 1/x;
   System.out.println("f(x) = " + fx);
}
```

```
Enter x: 0
Exception in thread "main" java.lang.ArithmeticException: / by zero at
java.oop.example.ExceptionExample.main (ExceptionExample.java:15)
```

Ví dụ

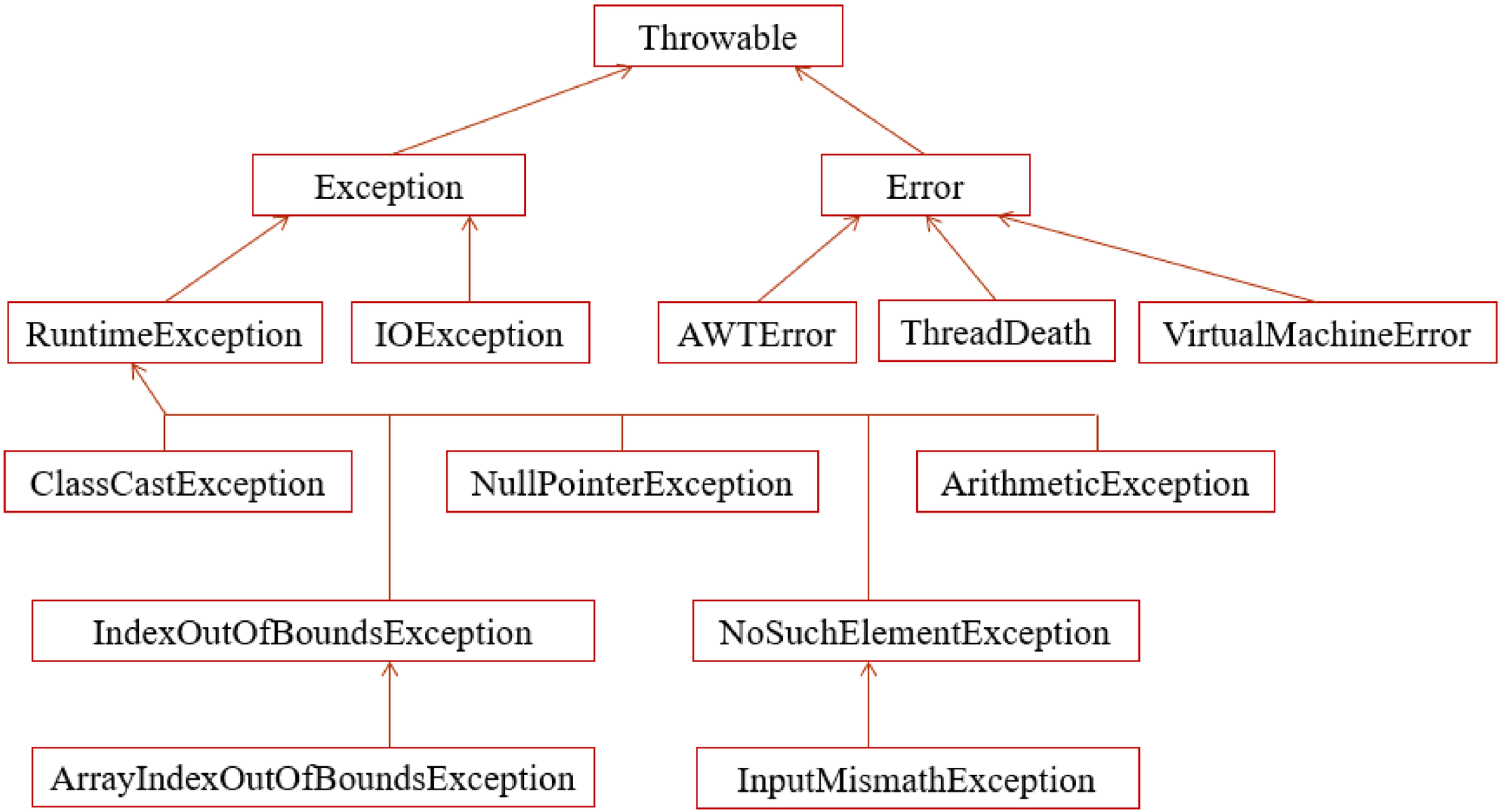
```
public static void main(String[] args) {
   int x, fx;
   Scanner inputData = new Scanner(System.in);
   System.out.print("Enter x: ");
   x = inputData.nextInt();
   fx = 1/x;
   System.out.println("f(x) = " + fx);
}
```

```
Enter x: a
Exception in thread "main"
java.util.InputMismatchException
    at java.util.Scanner.throwFor(Unknown Source)
    at java.util.Scanner.next(Unknown Source)...
```

Ngoại lệ trong Java

- Đóng gói các thông tin và phương thức của ngoại lệ trong các lớp
- Tất cả các ngoại lệ xuất hiện được coi như là một đối tượng của các lớp kế thừa từ lớp Throwable hoặc các lớp con của nó
- Throwable có 2 lớp con trực tiếp:
  - Error: Các ngoại lệ nghiêm trọng, chương trình không thể quản lý
  - Exception: Các ngoại lệ có thể kiểm soát
- **Ngoại lệ che được và ngoại lệ không che được:**
  - **Ngoại lệ che được**: Không bắt buộc phải xử lý trong chương trình. Gồm mọi ngoại lệ thuộc RuntimeException và các lớp con
  - **Ngoại lệ không che được**: Bắt buộc phải xử lý trong chương trình.

Ngoại lệ trong Java



ĐẠI NAM
UNIVERSITY

BẮT VÀ XỬ LÝ NGOẠI LỆ

Cách thức xử lý ngoại lệ

- Khi ngoại lệ xảy ra, có 3 cách thức xử lý:
  - Bỏ qua (chỉ áp dụng với ngoại lệ che được)
  - Xử lý tại nơi xảy ra ngoại lệ
  - Xử lý ở vị trí khác, thời điểm khác trong chương trình → ủy nhiệm ngoại lệ

Xử lý tại chỗ

- Sử dụng cấu trúc try ... catch ... finally
- Cú pháp chung

```
try{
    //Khối lệnh có thể xảy ra ngoại lệ
}
catch(Exception1 e1){//Exception1 phải là lớp con hoặc
                     //ngang hàng với Exception2
    //Xử lý ngoại lệ e1
}
catch(Exception e2){
    //Xử lý ngoại lệ e2
}
...
finally{
    // Khối lệnh luôn được thực hiện cho dù ngoại lệ có
    // xảy ra hay không
}
```

Ví dụ

```
public static void main(String[] args) {
    int x, fx;
    Scanner inputData = new Scanner(System.in);
    System.out.print("Enter x: ");
    try{
       x = inputData.nextInt();
       fx = 1/x;
    } catch (InputMismatchException inputMisException){
       System.out.println(“You entered not-integer value!”);
    } catch (ArithmeticException arithException){
       System.out.println(“Error ” + arithException);
    }
    System.out.println("f(x) = " + fx);
}
```

Báo lỗi vì giá trị fx có thể không được khởi tạo

Ví dụ - Sửa lại

```
public static void main(String[] args) {
   int x, fx;
   Scanner inputData = new Scanner(System.in);
   System.out.print("Enter x: ");
   try{
      x = inputData.nextInt();
      fx = 1/x;
      System.out.println("f(x) = " + fx);
   } catch (InputMismatchException inputMisException){
      System.out.println(“You entered not-integer value!”);
   } catch (ArithmeticException arithException){
      System.out.println(“Error ” + arithException);
   }
}
```

Ví dụ - Cách viết khác

```
public static void main(String[] args) {
   int x, fx;
   Scanner inputData = new Scanner(System.in);
   System.out.print("Enter x: ");
   try{
      x = inputData.nextInt();
   } catch (InputMismatchException inputMisException){
   System.out.println(“You entered not-integer value!”);
   try{
      fx = 1/x;
      System.out.println("f(x) = " + fx);
   } catch (ArithmeticException arithException){
      System.out.println(“Error ” + arithException);
   }
}
```

Xử lý tại chỗ

- Cú pháp try...catch lồng nhau khi ngoại lệ có thể xảy ra trong khi xử lý một ngoại lệ khác.

```
try{
  //Khối lệnh có thể xảy ra ngoại lệ
}
catch (Exception1 e1){
  //Xử lý ngoại lệ e1
  try{
     //Ngoại lệ mới xảy ra
  } catch (Exception e2){
     //Xử lý ngoại lệ e2 trước e1
  }
}
...
finally{
}
```

Xử lý tại chỗ

- Khi không thể kiểm soát quá nhiều ngoại lệ, hoặc không chắc chắn về các ngoại lệ có khả năng xảy ra, có thể bắt ngoại lệ chung Exception
- Ví dụ

```
try{
   x = inputData.nextInt();
   fx = 1/x;
   System.out.println("f(x) = " + fx);
}catch (ArithmeticException arithException){
   System.out.println(“Error ” + arithException);
}catch (Exception anyException){
   System.out.println(“Something is wrong!”);
}
```

Xử lý tại chỗ

- Luôn phải có khối xử lý catch hoặc finally khi chờ bắt ngoại lệ.

```
try{
    x = inputData.nextInt();
    fx = 1/x;
    System.out.println("f(x) = " + fx);
} catch (InputMismatchException inputMisException){
    System.out.println(“You entered not-integer value!”);
  } catch (ArithmeticException arithException){
    System.out.println(“Error ” + arithException);
  } finally{
    System.out.println(“Finally this must have happened!”)
  }
}
```

Bắt và xử lý nhiều ngoại lệ

- Từ Java 7, cho phép sử dụng 1 khối catch để bắt và xử lý nhiều ngoại lệ.

```
try{
  //Khối lệnh có thể xảy ra ngoại lệ
}
catch(Exception1 | Exception2 e){
  //Xử lý khi ngoại lệ Exception1 hoặc Exception2 xảy ra
}
...
finally{
  // Khối lệnh luôn được thực hiện cho dù ngoại lệ có
  // xảy ra hay không
}
```

Uỷ nhiệm ngoại lệ

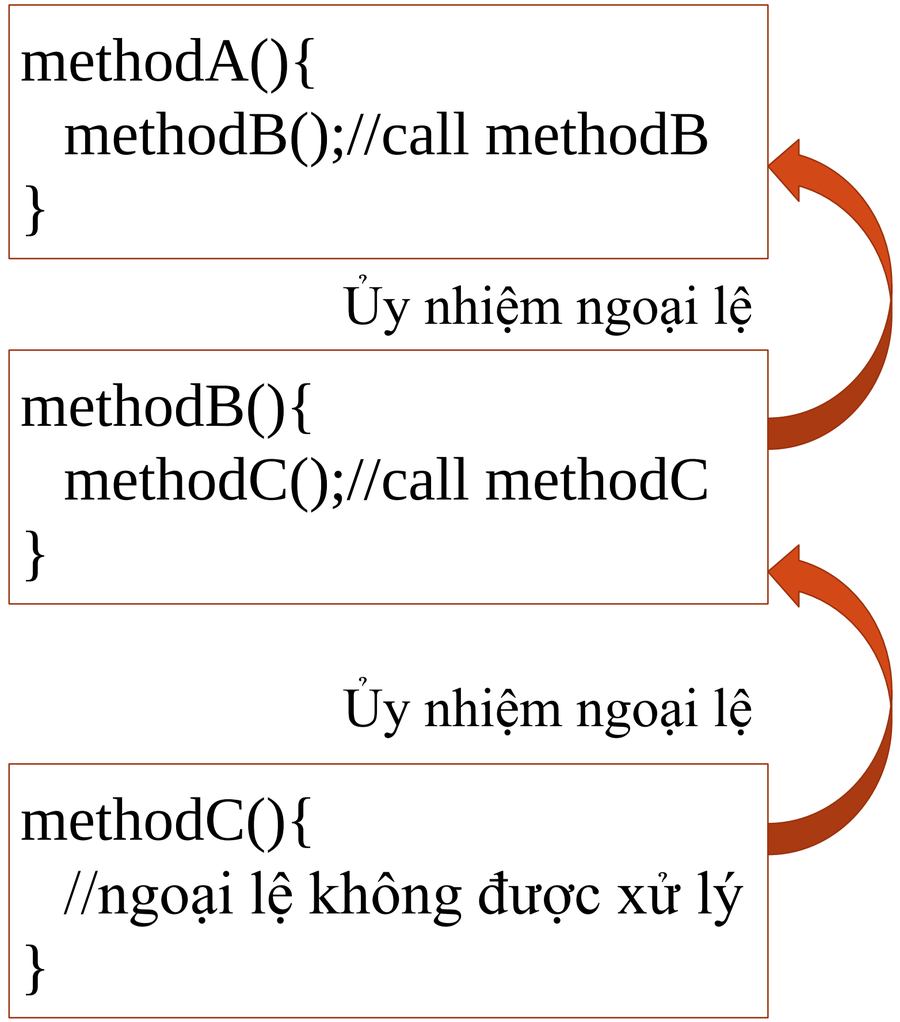
- Ngoại lệ có thể không cần xử lý tại chỗ mà ủy nhiệm cho mức cao hơn xử lý.
- Ở mức nào có phần xử lý ngoại lệ thì ngoại lệ được xử lý tại mức đó.
- Mức cao nhất có thể được ủy nhiệm là máy ảo JVM.
- **Cú pháp:**

```
Modifier DataType methodName(parameters) throws ExceptionType {
   //method's body
}
```

- Các ngoại lệ RuntimeException mặc định được ủy nhiệm cho các mức cao hơn.

Uỷ nhiệm ngoại lệ

- Giả sử ngoại lệ xảy ra ở methodC() nhưng methodC() không xử lý mà ủy nhiệm cho phương thức gọi nó là methodB()
- Nếu ở methodB() không xử lý, có thể ủy nhiệm cho mức cao hơn là methodA()
- Nếu methodA() có khối lệnh để bắt và xử lý, ngoại lệ sinh ra do methodC() được xử lý tại đây



Lợi ích khi uỷ nhiệm ngoại lệ

- Dễ sử dụng:
  - Chương trình dễ đọc và an toàn
  - Chuyển ngoại lệ đến vị trí có khả năng xử lý
- Tách xử lý ngoại lệ khỏi những đoạn mã thông thường
- Không bỏ sót ngoại lệ
- Gom nhóm và phân loại ngoại lệ

Lợi ích – Ví dụ

```
private int getInt(){
   int data = 0;
   BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
   String str;
   System.out.println("Enter an integer: ");
   try {
       str = br.readLine();
       data = Integer.parseInt(str);
   } catch (IOException e) {
       System.out.println("Could not enter data!");
   }
   return data;
}
```

Lợi ích – Ví dụ

```
private double getDouble(){
   double data = 0;
   BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
   String str;
   System.out.println("Enter a double: ");
   try {
      str = br.readLine();
      data = Double.parseDouble(str);
   } catch (IOException e) {
      System.out.println("Could not enter data!");
   }
   return data;
}
```

Lợi ích – Ví dụ

```
private String getString(){
   String data = null;
   BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
   String str;
   System.out.println("Enter a string: ");
   try {
      data = br.readLine();
   } catch (IOException e) {
      System.out.println("Could not enter data!");
   }
   return data;
}
```

Lợi ích – Ví dụ

```
public void getData(){
   int intData = 0;
   double doubleData = 0;
   String strData = null;
   intData = getInt();
   doubleData = getDouble();
   strData = getString();
}
```

Uỷ nhiệm ngoại lệ – Ví dụ

```
private int getInt() throws IOException{
  int data = 0;
  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
  String str;
  System.out.println("Enter an integer: ");
  str = br.readLine();
  data = Integer.parseInt(str);
  return data;
}
```

Uỷ nhiệm ngoại lệ – Ví dụ

```
private int getDouble() throws IOException{}
```

```
private int getString() throws IOException{}
```

```
public void getData(){
   int intData = 0;
   double doubleData = 0;
   String strData = null;
   try{
      intData = getInt();
      doubleData = getDouble();
      strData = getString();
   }catch(IOException e){
      System.out.println("Could not enter data!");
   }
}
```

Tung ngoại lệ cưỡng bức

- Phương thức được gọi (called method) có thể tung ngoại lệ cưỡng bức và ủy nhiệm xử lý ngoại lệ đó cho phương thức gọi nó (calling method)
- Thường sử dụng khi kiểm soát dữ liệu vào-ra hoặc một bước nào đó trong quá trình xử lý dữ liệu
- Cách thức: Sử dụng từ khóa throw

```
Modifier DataType methodName(parameters) throws ExceptionType {
    //method's body
    throw new exceptionTypeConstruct();
}
```

Nếu ExceptionType thuộc RuntimeException thì không cần sử dụng throws để ủy nhiệm

Tung ngoại lệ cưỡng bức – Ví dụ

```
public void restrictedContent throws IOException{
   int age;
   System.out.println(“This page contains 18+ contents. How old are you?”);
   Scanner inputData = new Scanner(System.in);
   age = inputData.nextInt();
   if(age < 18)
      throw new IOException(“You’re restricted!”);
}
```

```
public static void main(String arg[]) {
    try{
       restrictedContent();
    }catch (Exception e){
       System.out.print(e.getMessage());
    }
}
```

Ngoại lệ và kế thừa

- Khi kế thừa, lớp con có thể ghi đè (overriding) phương thức của lớp cha
- Khi ghi đè phương thức, lớp con chỉ có thể tung ra và ủy nhiệm các ngoại lệ giống hoặc là ngoại lệ con của các ngoại lệ trong phương thức của lớp cha

```
public class Disk {
   public void readData() throws IOException{}
}
```

```
public class FloppyDisk extends Disk{
   public void readData() throws Exception{} //wrong
}
```

```
public class FloppyDisk {
   public void readData() throws EOFException{} //OK
}
```

ĐẠI NAM
UNIVERSITY

TỰ ĐỊNH NGHĨA NGOẠI LỆ

Tự định nghĩa ngoại lệ

- Khi tung ngoại lệ cưỡng bức thuộc vào các ngoại lệ do Java định nghĩa, trong nhiều trường hợp có thể gây ra nhập nhằng.

```
public void restrictedContent throws IOException{
  int age;
  System.out.println(“This page contains 18+ contents. How old are you?”);
  Scanner inputData = new Scanner(System.in);
  age = inputData.nextInt();
  if(age < 18)
    throw new IOException(“You’re restricted!”);
}
```

- **Khắc phục**: Tự định nghĩa ngoại lệ.

Tự định nghĩa ngoại lệ

- **Định nghĩa lớp ngoại lệ mới:**
  - Kế thừa từ lớp Exception nếu muốn ngoại lệ mới thuộc dạng không che được
  - Kế thừa từ RuntimeException nếu muốn ngoại lệ mới che được

```
public NewException extends Exception{
   public NewException(String msg){
      super(msg);
   }
}
```

- Ví dụ:

```
public AgeException extends Exception{
   public NewException(String msg){
      super(msg);
   }
}
```

Tự định nghĩa ngoại lệ - Ví dụ

```
public void restrictedContent throws AgeException{
  int age;
  System.out.println("This page contains 18+ contents. How old are you?");
  Scanner inputData = new Scanner(System.in);
  age = inputData.nextInt();
  if (age < 0)
     throw new AgeException("Are you kidding me?");
  else if(age < 18)
     throw new AgeException("You're restricted!");
  //show 18+ contents
  //...
}
```