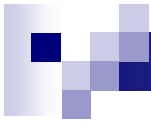




Xử lý ngoại lệ



Nội dung

- Khái niệm về xử lý ngoại lệ (exception handling)
- Ném và bắt ngoại lệ
- Khai báo ngoại lệ
- Ném lại ngoại lệ
- Định nghĩa ngoại lệ mới
- Xử lý ngoại lệ trong constructor




Tài liệu tham khảo

- *Thinking in Java*, chapter 9
- *Java how to program*, chapter 13




Lỗi và ngoại lệ

- Mọi đoạn chương trình đều tiềm ẩn khả năng sinh lỗi
 - lỗi chủ quan: do lập trình sai
 - lỗi khách quan: do dữ liệu, do trạng thái của hệ thống
- Ngoại lệ: các trường hợp hoạt động không bình thường
- Xử lý ngoại lệ như thế nào
 - làm thế nào để có thể tiếp tục (tái) thực hiện



Cách xử lý lỗi truyền thống

- Cài đặt mã xử lý tại nơi phát sinh ra lỗi
 - ☐ làm cho chương trình trở nên khó hiểu
 - ☐ 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
- Khó kiểm soát được hết các trường hợp
 - ☐ lỗi số học, lỗi bộ nhớ,...
- Người lập trình thường quên không xử lý lỗi
 - ☐ bản chất con người
 - ☐ thiếu kinh nghiệm, cố tình bỏ qua



Ví dụ (C++)

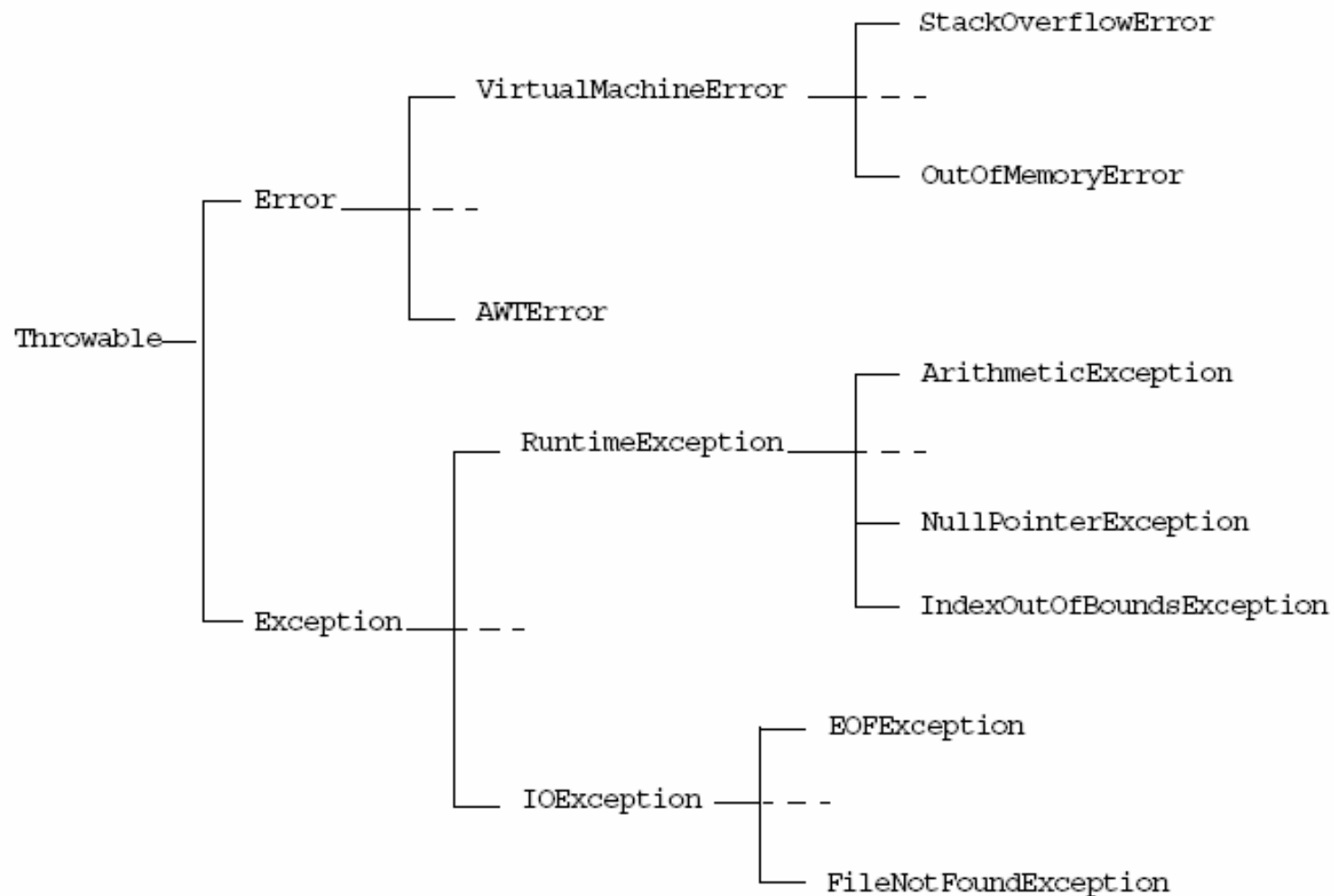
```
int devide(int num, int denom, int& error)
{
    if (0 != denom) {
        error = 0;
        return num/denom;
    } else {
        error = 1;
        return 0;
    }
}
```



Xử lý ngoại lệ (Exception handling) trong Java

- Xử lý ngoại lệ trong Java được kế thừa từ C++
- Dựa trên cơ chế ném và bắt ngoại lệ
 - ném ngoại lệ: dừng chương trình và chuyển điều khiển lên mức trên (nơi bắt ngoại lệ)
 - bắt ngoại lệ: xử lý với ngoại lệ
- Ngoại lệ: là đối tượng mang thông tin về lỗi đã xảy ra
 - ngoại lệ được ném tự động
 - ngoại lệ được ném tường minh


Phản hệ ngoại lệ trong Java





Ưu điểm của cơ chế ném bắt ngoại lệ

- Dễ sử dụng
 - ☐ 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 thuật toán
 - ☐ tách mã xử lý
 - ☐ sử dụng cú pháp khác
- Không bỏ sót ngoại lệ (ném tự động)
- Làm chương trình dễ đọc hơn, an toàn hơn



Ném ngoại lệ (tường minh)

■ Ném ngoại lệ bằng câu lệnh **throw**

```
if (0==denominator) {  
    throw new Exception();  
} else res = nominator / denominator;
```

Cú pháp `try - catch`

- Việc phân tách đoạn chương trình thông thường và phần xử lý ngoại lệ được thể hiện thông qua cú pháp **`try - catch`**
 - Khối lệnh `try { ... }`: khối lệnh có khả năng ném ngoại lệ
 - Khối lệnh `catch() { ... }`: bắt và xử lý với ngoại lệ

```
try {  
    // throw an exception  
}  
catch (Throwable e) {  
    // exception-handling statements  
}
```



Ví dụ

```
try {  
    if (0 == denom) {  
        throw new Exception("denom = 0");  
    } else res = num/denom;  
} catch(Exception e) {  
    System.out.println(e.getMessage());  
}
```



Cú pháp `try catch finally`

- Có thể bắt nhiều loại ngoại lệ khác nhau bằng cách sử dụng nhiều khối lệnh `catch` đặt kế tiếp
 - khối lệnh `catch` sau không thể bắt ngoại lệ là lớp dẫn xuất của ngoại lệ được bắt trong khối lệnh `catch` trước
- Khối lệnh `finally` có thể được đặt cuối cùng để thực hiện các công việc “dọn dẹp” cần thiết
 - `finally` luôn được thực hiện dù ngoại lệ có được bắt hay không
 - `finally` được thực hiện cả khi không có ngoại lệ được ném ra



Cú pháp try catch finally

```
try {  
    ...  
}  
catch(Exception1 e1) {  
    ...  
}  
catch(Exception2 e2) {  
    ...  
}  
finally {  
    ...  
}
```



Ví dụ

```
InputStreamReader reader = new InputStreamReader(System.in);
BufferedReader buf = new BufferedReader(reader);
try {
    String str = buf.readLine();
    num = Integer.valueOf(str).intValue();
}
catch (IOException e) {
    System.err.println("IO Exception");
}
catch (NumberFormatException e) {
    System.err.println("NumberFormatException");
}
catch (Exception e) {
    System.err.println(e.getMessage());
}
finally {
    buf.close();
}
```



Ném ngoại lệ khỏi phương thức

- Không nhất thiết phải xử lý ngoại lệ trong phương thức
 - không đủ thông tin để xử lý
 - không đủ thẩm quyền
- Một phương thức muốn ném ngoại lệ ra ngoài phải khai báo việc ném ngoại lệ bằng từ khóa **throws**
 - có thể ném ngoại lệ thuộc lớp dẫn xuất của ngoại lệ được khai báo



Ví dụ

```
int readInt() throws IOException, NumberFormatException
{
    InputStreamReader reader;
    reader = new InputStreamReader(System.in);
    BufferedReader buf = new BufferedReader(reader);
    String str = buf.readLine();
    return Integer.valueOf(str).intValue();
}
```



Ví dụ

```
try {  
    int n = readInt();  
}  
catch (IOException e) {  
    System.err.println("IO Exception");  
}  
catch (NumberFormatException e) {  
    System.err.println("NumberFormatException");  
}
```



Ngoại lệ và phương thức được định nghĩa lại

- Phương thức được định nghĩa lại tại lớp dẫn xuất có thể không ném ngoại lệ
- Nếu ném ngoại lệ, chỉ có thể ném ngoại lệ giống như tại phương thức của lớp cơ sở hoặc ngoại lệ là lớp dẫn xuất của ngoại lệ được ném tại phương thức của lớp cơ sở
 - đảm bảo bắt được ngoại lệ khi sử dụng cơ chế đa hình



Ví dụ

```
class A {  
    public void methodA() throws RuntimeException {  
    }  
}
```

```
class B extends A {  
    public void methodA() throws ArithmeticException {  
    }  
}
```

```
class C extends A {  
    public void methodA() throws Exception {  
    }  
}
```

```
class D extends A {  
    public void methodA() {  
    }  
}
```



Ví dụ:


```
A a = new B();  
try {  
    a.methodA();  
}  
catch (RuntimeException e) {  
    ...  
}
```



Ném lại ngoại lệ


- Sau khi bắt ngoại lệ, nếu thấy cần thiết chúng ta có thể ném lại chính ngoại lệ vừa bắt được để cho chương trình mức trên tiếp tục xử lý

```
try {...  
}  
catch (Exception e) {  
    System.out.println(e.getMessage());  
    throw e;  
}
```



Lần vết ngoại lệ StackTrace

- Có thể sử dụng phương thức `printStackTrace()` để lần vết vị trí phát sinh ngoại lệ
 - debug chương trình



```
public class Test4 {  
    void methodA() throws Exception {  
        methodB();  
        throw new Exception();  
    }  
    void methodB() throws Exception {  
        methodC();  
        throw new Exception();  
    }  
    void methodC() throws Exception {  
        throw new Exception();  
    }  
    public static void main(String[] args) {  
        Test4 t = new Test4();  
        try {  
            t.methodA();  
        }  
        catch(Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```




Ném ngoại lệ từ `main()`

- Nếu không có phương thức nào bắt ngoại lệ, ngoại lệ sẽ được truyền lên phương thức `main()` và được cần được xử lý tại đây.
- Nếu vẫn không muốn xử lý ngoại lệ, chúng ta có thể để ngoại lệ truyền lên mức điều khiển của máy ảo bằng cách khai báo `main()` ném ngoại lệ
 - chương trình sẽ bị dừng và hệ thống sẽ in thông tin về ngoại lệ trên Console (`printStackTrace()`)



Ví dụ

```
import java.io.*;

public class Echo {
    public static void main(String[] args)
        throws IOException {
        InputStreamReader reader;
        BufferedReader bufReader;
        reader = new InputStreamReader(System.in);
        bufReader = new BufferedReader(reader);
        String s;
        while( null != (s = bufReader.readLine())
            System.out.println(s);
        }
    }
}
```



Hai loại ngoại lệ

- Java phân biệt hai loại ngoại lệ là ngoại lệ cần kiểm tra và ngoại lệ không cần kiểm tra
- Ngoại lệ cần kiểm tra: chương trình dịch luôn kiểm tra xem chúng ta có viết code xử lý với các ngoại lệ này không (`try catch/ throws`)
 - `IOException`
- Ngoại lệ không cần kiểm tra: các ngoại lệ có thể loại trừ nếu viết chương trình tốt hơn
 - `RuntimeException`



Ví dụ: Checked Exception


```
InputStreamReader reader;  
BufferedReader bufReader;  
reader = new InputStreamReader(System.in);  
bufReader = new BufferedReader(reader);  
  
try {  
    String s = bufReader.readLine();  
}  
catch (IOException e) {  
    ...  
}
```



Ví dụ: Unchecked Exception

```
int num1 = Integer.valueOf(str1).intValue();  
int num2 = Integer.valueOf(str2).intValue();  
int num3 = num1 / num2;
```


- Các ngoại lệ thuộc lớp RuntimeException được hệ thống ném tự động
 - ☐ lỗi số học
 - ☐ lỗi chỉ số



Hoán đổi ngoại lệ

- Có thể đổi ngoại lệ cần kiểm tra thành ngoại lệ không cần kiểm tra
 - *chưa biết nên làm gì*

```
void wrapException() {  
    try {  
        throw new IOException();  
    }  
    catch (IOException e) {  
        throw new RuntimeException(e);  
    }  
}
```



```
try {  
    wrapException();  
} catch (RuntimeException e) {  
    try {  
        throw e.getCause();  
    }  
    catch (IOException e1) {  
        ...  
    }  
}
```



Tự định nghĩa ngoại lệ

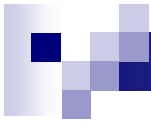
- Chúng ta có thể tạo lớp ngoại lệ để phục vụ các mục đích riêng
- Lớp ngoại lệ mới phải kế thừa từ lớp Exception hoặc lớp dẫn xuất của lớp này
- Có thể cung cấp hai constructor
 - constructor mặc định (không tham số)
 - constructor nhận một tham số String và truyền tham số này cho phương thức khởi tạo của lớp cơ sở



Ví dụ


```
class SimpleException extends Exception {  
}
```

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



Khởi tạo đối tượng và xử lý ngoại lệ

- Làm thế nào để thông báo khi hàm khởi tạo đối tượng gặp lỗi
 - không có giá trị trả lại
- Một cách là khởi tạo với một trạng thái đặc biệt và hi vọng sẽ có mã chương trình kiểm tra trạng thái này
- Cách hợp lý hơn là ném ngoại lệ



```
class InputFile {  
    public InputFile(String fname) throws IOException {  
        ...  
    }  
    ...  
}  
  
---  
try {  
    InputFile fin = new InputFile("data.txt");  
}  
catch (IOException e) {  
    System.err.println(e.getMessage);  
}
```



Bài tập và thực hành

- Tìm hiểu về phả hệ ngoại lệ của Java
- Thực hành
 - ☐ ném và bắt ngoại lệ
 - ☐ khai báo phương thức ném ngoại lệ
 - ☐ constructor ném ngoại lệ
 - ☐ tự định nghĩa ngoại lệ