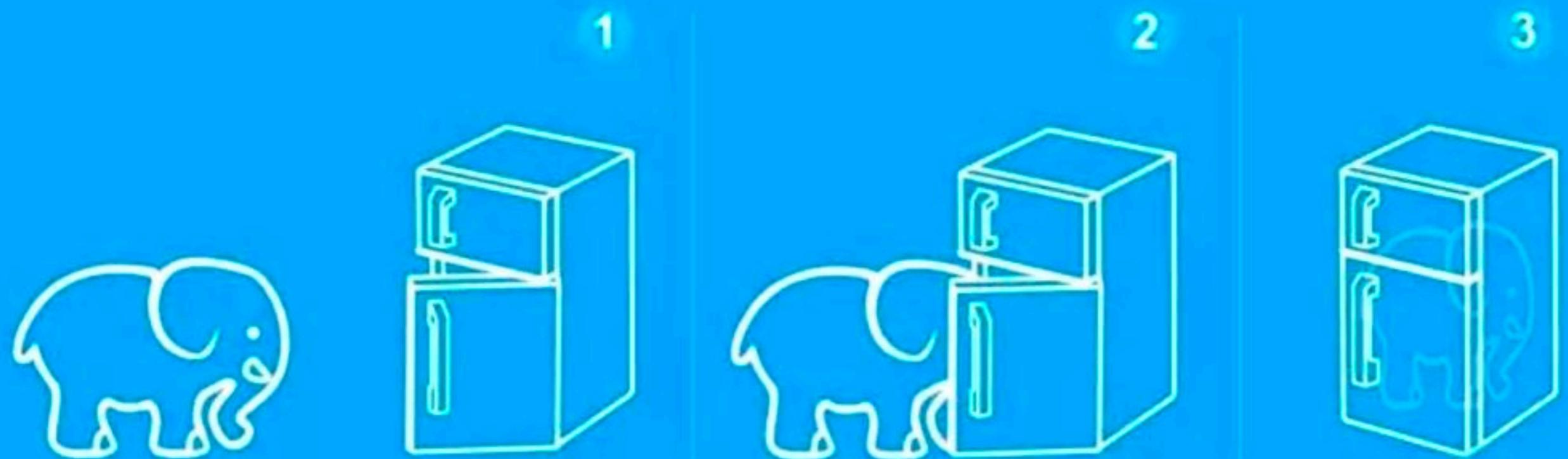


异常处理

杨亮



把大象放进冰箱需要几步



门打不开怎么办

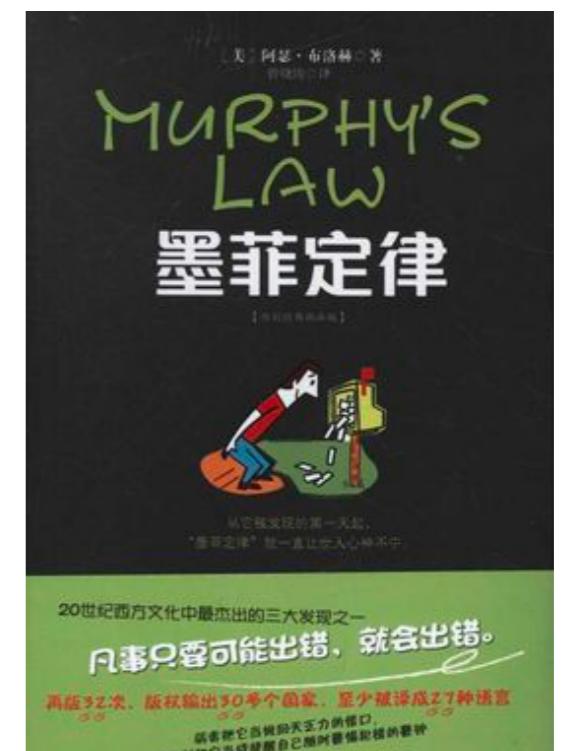
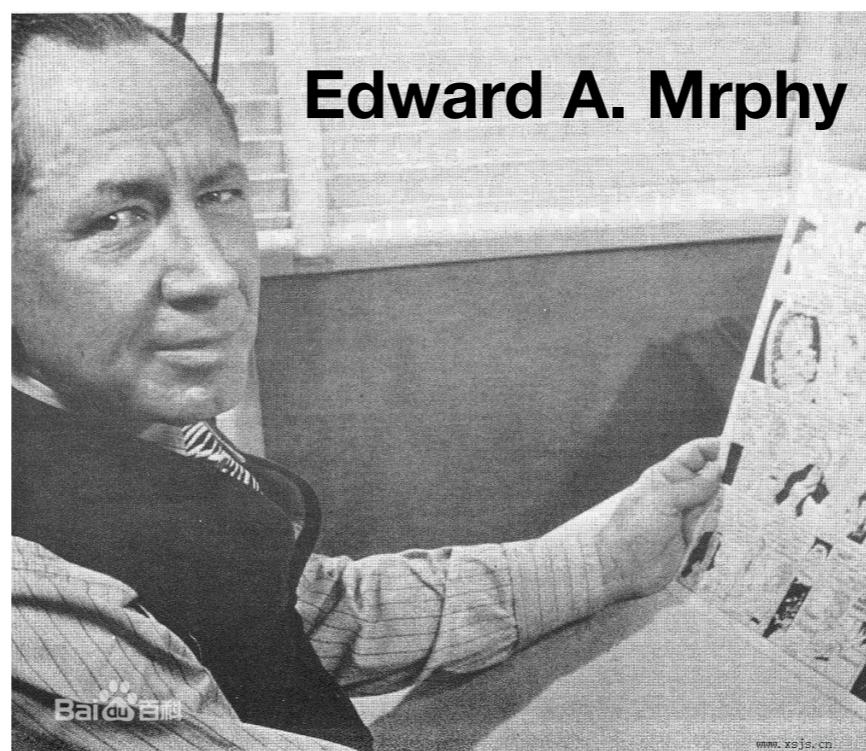
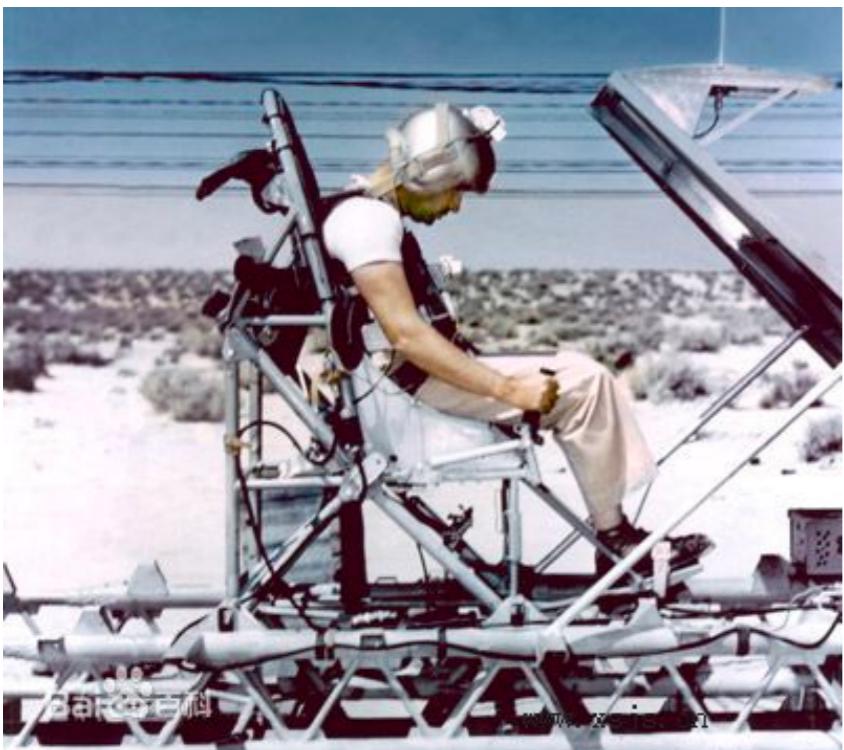
大象不进去怎么办

门关不上怎么办

墨菲定律

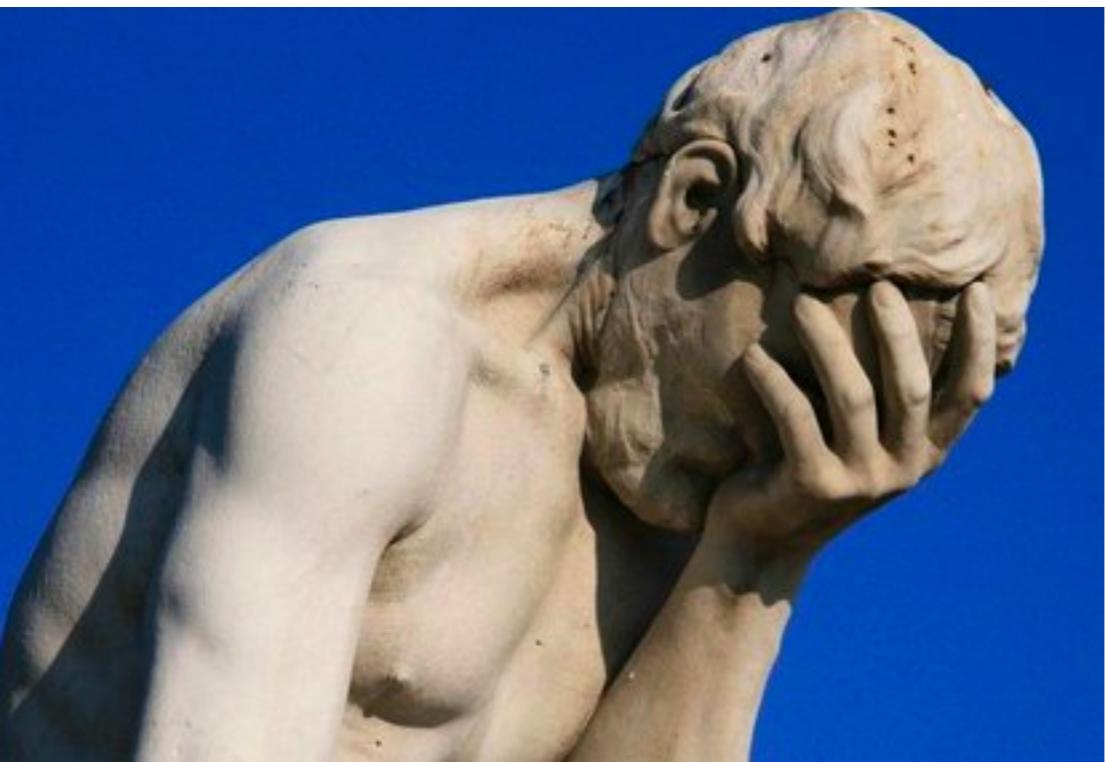
凡是可能出错的事有很大几率会出错

- 一、任何事都没有表面看起来那么简单；
- 二、所有的事都会比你预计的时间长；
- 三、会出错的事总会出错；
- 四、如果你担心某种情况发生，那么它就更有可能发生。



一个简单的程序例子

```
readFile {  
    open the file;  
    determine its size;  
    allocate that much memory;  
    read the file into memory;  
    close the file;  
}
```



- 如果无法打开文件会发生什么?
- 如果无法确定文件的长度，会发生什么?
- 如果不能分配足够的内存，会发生什么?
- 如果读取失败会发生什么?
- 如果文件无法关闭会怎么样?

面向过程的代码

```
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;
                }
            } 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;
}
```

```
readFile {
    open the file;
    determine its size;
    allocate that much memory;
    read the file into memory;
    close the file;
}
```

- 如果无法打开文件会发生什么?
- 如果无法确定文件的长度, 会发生什么?
- 如果不能分配足够的内存, 会发生什么?
- 如果读取失败会发生什么?
- 如果文件无法关闭会怎么样?



将错误处理代码与“常规”代码分离

```
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;  
    }  
}
```

“常规”代码

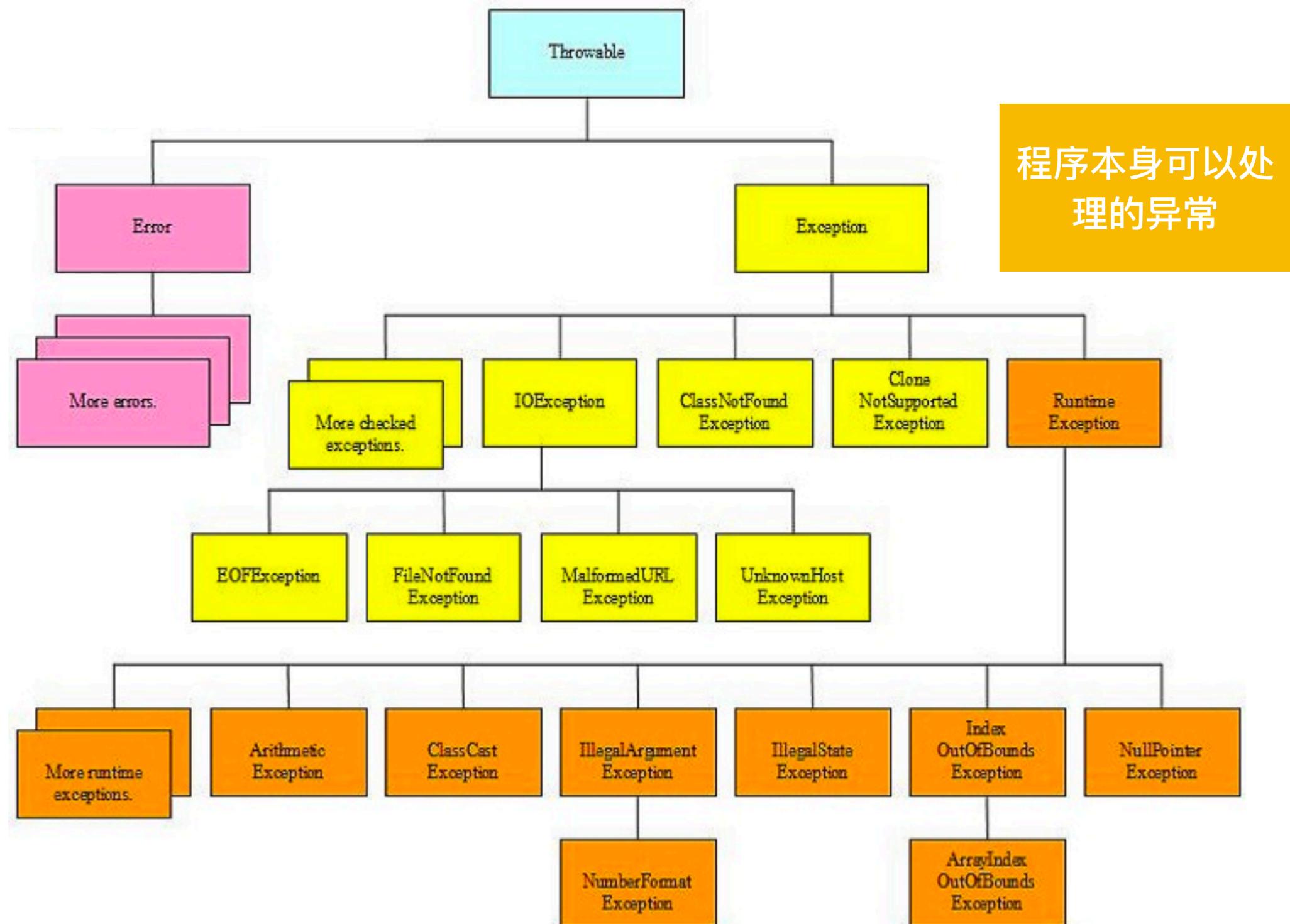
错误处理代码

异常处理的优势

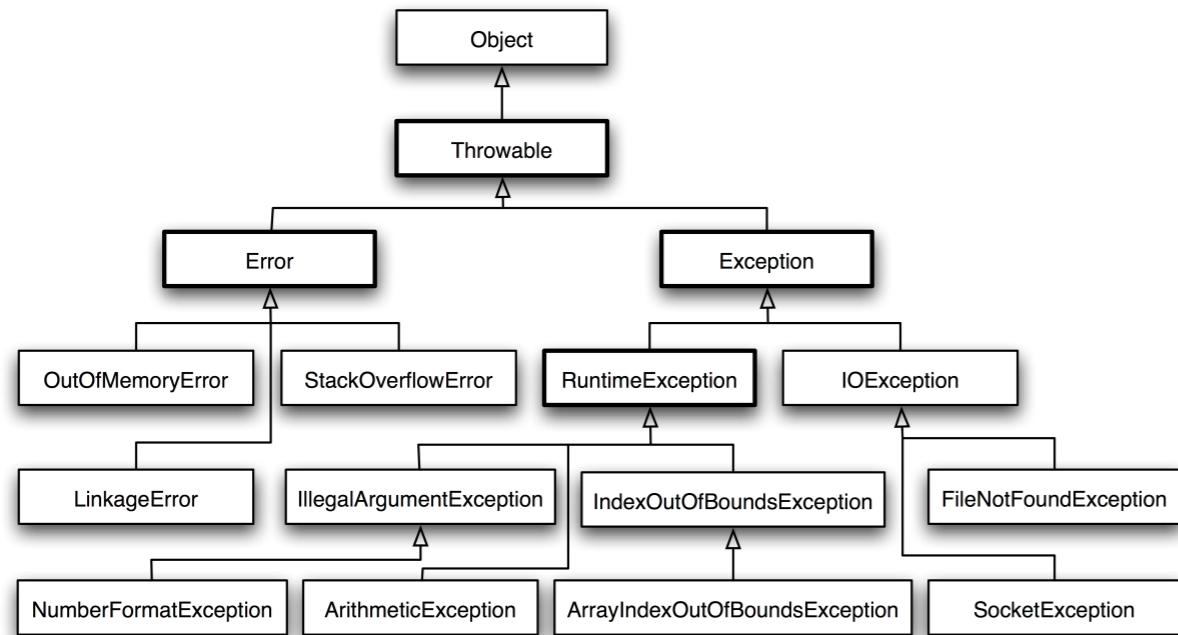
异常也是对象

JVM的异常
与程序无关

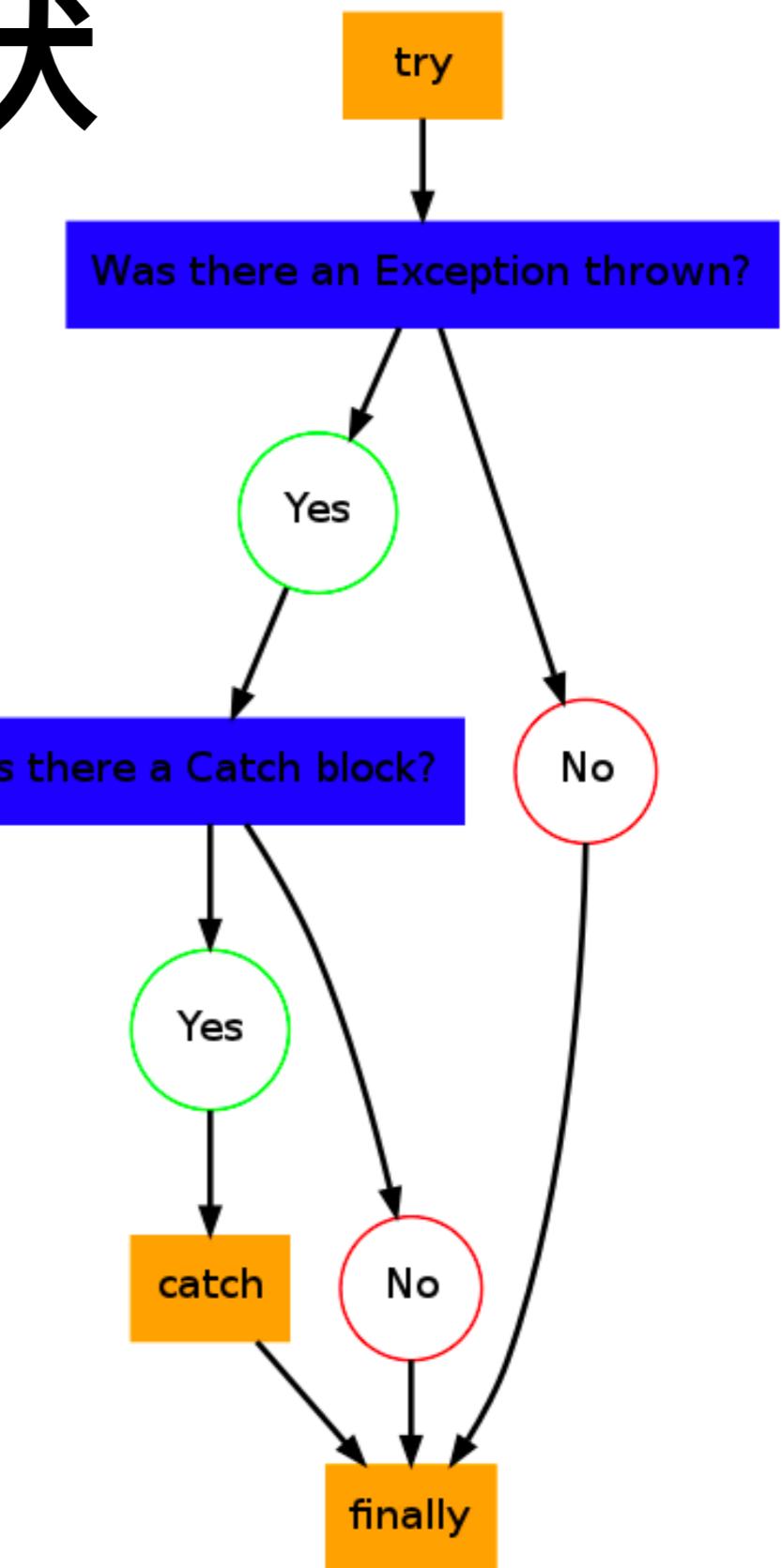
程序本身可以处
理的异常



异常的捕获



```
try {  
    // 可能会发生异常的程序代码  
} catch (Type1 id1) {  
    // 捕获并处理try抛出的异常类型Type1  
} catch (Type2 id2) {  
    // 捕获并处理try抛出的异常类型Type2  
} finally {  
    // 无论是否发生异常，都将执行的语句块  
}
```



系统定义的异常会自动抛出，程序只需要负责**捕获**和**处理**

看个例子

```
1 public class TestException {
2     public static void main(String args[]) {
3         int i = 0;
4         String greetings[] = { " Hello world !", " Hello World !! ",
5                               " HELLO WORLD !!!" };
6         while (i < 4) {
7             try {
8                 // 特别注意循环控制变量i的设计，避免造成无限循环
9                 System.out.println(greetings[i++]);
10            } catch (ArrayIndexOutOfBoundsException e) {
11                System.out.println("数组下标越界异常");
12            } finally {
13                System.out.println("-----");
14            }
15        }
16    }
17 }
```

异常消息的设置与获取

```
try {
    int test = 10 / 0;
    Log.d(TAG, String.valueOf(test));
} catch (Exception e) {
    Log.d(TAG, "=====Exception Start=====");
    e.printStackTrace();           Name of the Exception: Message of the Exception
    Log.d(TAG, e.getMessage());   Stack trace
    Log.d(TAG, e.toString());    =====Exception Start=====
    Log.d(TAG, "=====Exception End=====");   Message of the Exception
}
```

```
D/TAG_Test: =====Exception Start=====
W/System.err: java.lang.ArithmetricException: divide by zero
W/System.err:     at com.toby.personal.testlistview.MainActivity.onCreate(MainActivity.java:17)
W/System.err:     at android.app.Activity.performCreate(Activity.java:6555)
W/System.err:     at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1108)
W/System.err:     at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2385)
W/System.err:     at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2492)
W/System.err:     at android.app.ActivityThread.access$900(ActivityThread.java:156)
W/System.err:     at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1352)
W/System.err:     at android.os.Handler.dispatchMessage(Handler.java:102)
W/System.err:     at android.os.Looper.loop(Looper.java:148)
W/System.err:     at android.app.ActivityThread.main(ActivityThread.java:5469) <1 internal calls>
W/System.err:     at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:726)
W/System.err:     at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:616)
D/TAG_Test: divide by zero
D/TAG_Test: java.lang.ArithmetricException: divide by zero
D/TAG_Test: =====Exception End=====
```

异常的抛出与声明

对于会**抛出但不处理异常的函数**
应该声明抛出异常的类型

```
[修饰符] 返回值类型 方法名([参数列表]) [throws 异常名] {  
语句序列;  
throw new 异常名(message);  
语句序列;  
}
```

程序也可以在函数中**手动地**抛出异常
(Throwable子类)

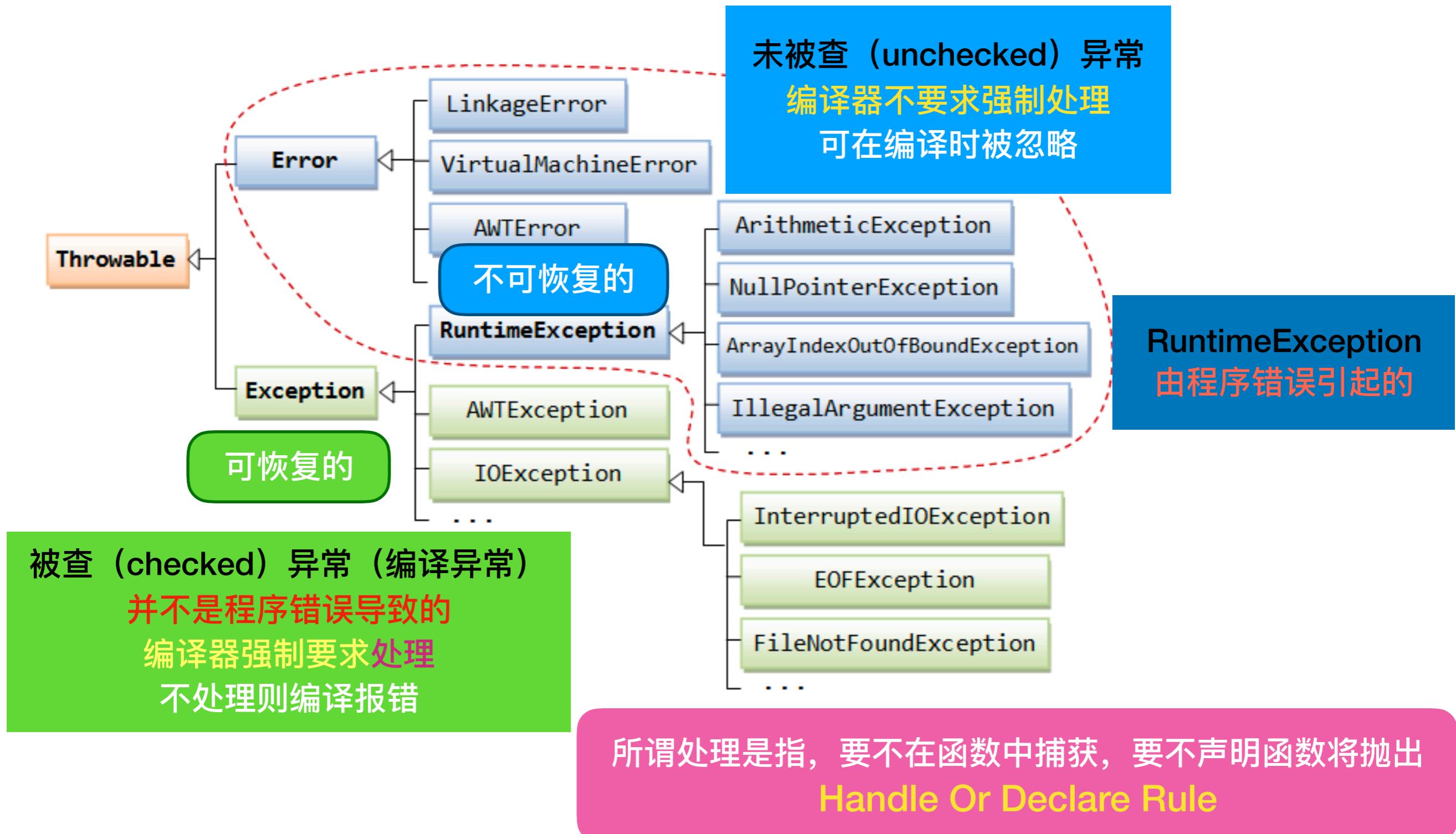


手动抛出异常例子

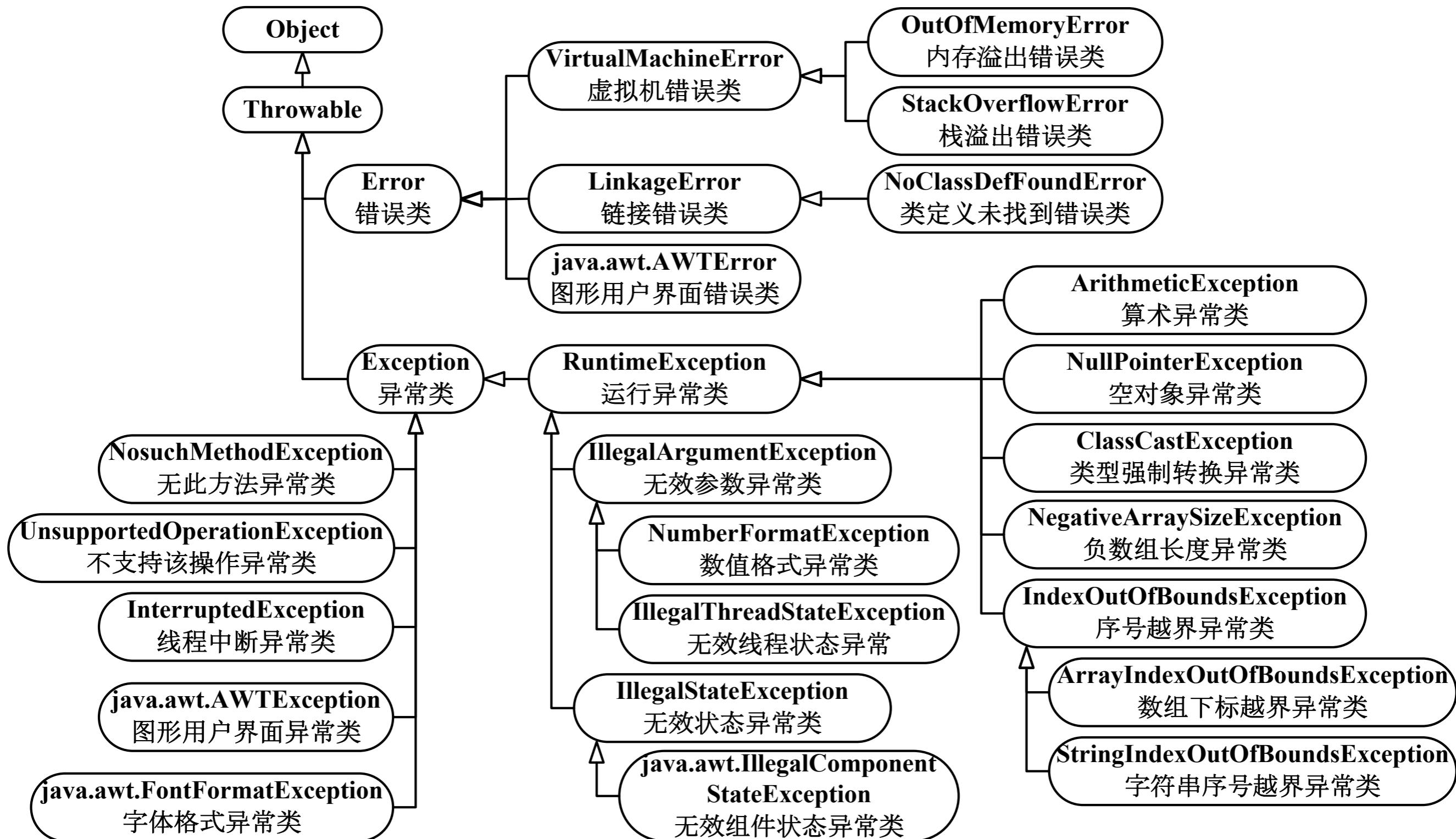
```
public class TestException {  
    static int quotient(int x, int y) throws MyException { // 定义方法抛出异常  
        if (y < 0) { // 判断参数是否小于0  
            throw new MyException("除数不能是负数"); // 异常信息  
        }  
        return x/y; // 返回值  
    }  
    public static void main(String args[]) { //  
        int a =3;  
        int b =0;  
        try { // try语句包含可能发生异常的语句  
            int result = quotient(a, b); // 调用方法quotient()  
        } catch (MyException e) { // 处理自定义异常  
            System.out.println(e.getMessage()); // 输出异常信息  
        } catch (ArithmetricException e) { // 处理ArithmetricException异常  
            System.out.println("除数不能为0"); // 输出提示信息  
        } catch (Exception e) { // 处理其他异常  
            System.out.println("程序发生了其他的异常"); // 输出提示信息  
        }  
    }  
}
```

```
class MyException extends Exception { // 创建自定义异常类  
    String message; // 定义String类型变量  
    public MyException(String ErrorMessagr) { // 父类方法  
        message = ErrorMessagr;  
    }  
    public String getMessage() { // 覆盖getMessage()方法  
        return message;  
    }  
}
```

异常的分类

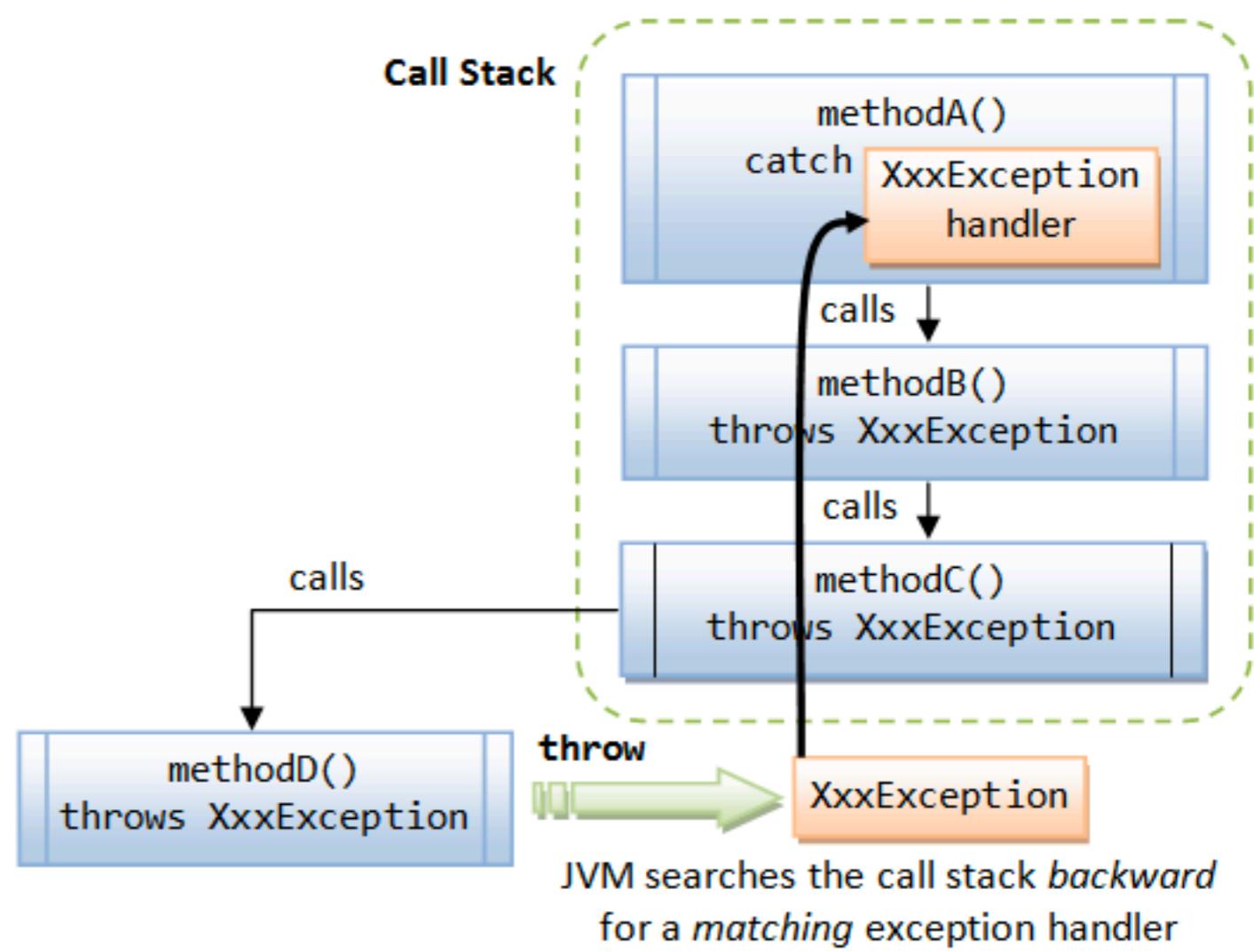


常见的异常



该什么时候处理问题

```
methodD {  
    open the file;  
    determine its size;  
    allocate that much memory;  
    read the file into memory;  
    close the file;  
}  
  
methodA {  
    try {  
        call methodB;  
    } catch (exception e) {  
        doErrorProcessing;  
    }  
}  
  
methodB throws exception {  
    call methodC;  
}  
  
methodC throws exception {  
    call methodD;  
}
```



异常的优势

将错误处理代码与“常规”代码分离

将错误沿调用推栈向上传递

