



Chapter 6

Exception

异常处理

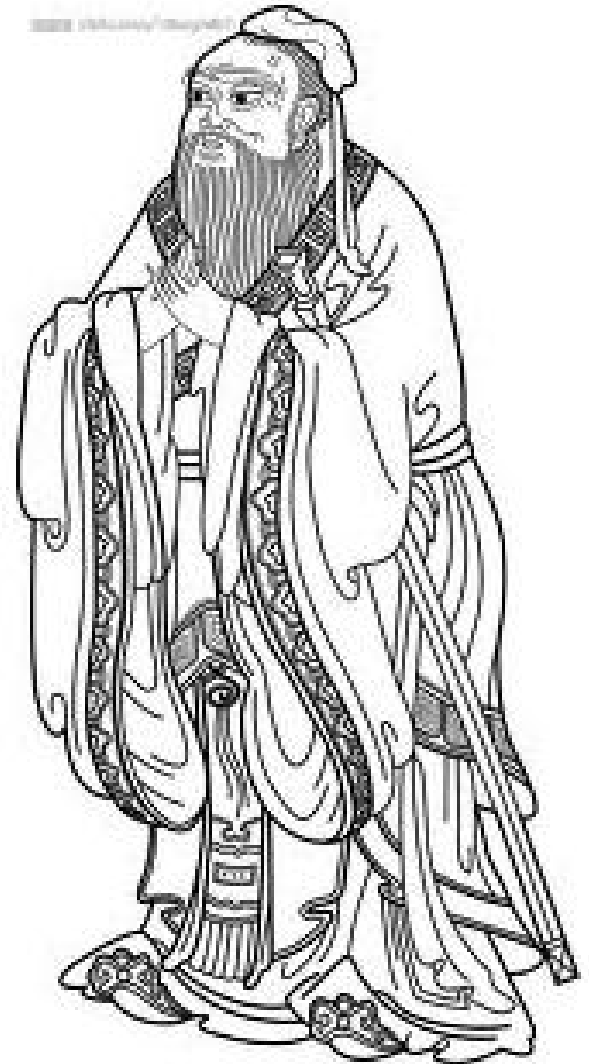


6.1 异常概述

非正常事件，指程序运行过程中出现的影响正常程序流程的事件。

- 异常：Exception（异常，例外，除外）

**人非圣贤，孰能无过。
过而能改，善莫大焉。**





目录

6.1 异常概述

6.1.1 异常与错误

6.1.2 标准异常类

6.2 异常处理方法

6.2.1 捕获和处理异常

6.2.2 抛出异常

6.2.3 异常传递链

6.3 自定义异常

6.3.1 自定义异常的创建

6.3.2 自定义异常抛出与捕获



示例：读取文件的程序

初始程序：

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



```
int 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 (theFileDintClose && errorCode == 0) {  
            errorCode = -4;  
        } else { errorCode = -5; }  
    } else { errorCode = -6; }  
    return errorCode;  
}
```

传统的异常处理方式：

- errorCode信息表达能力弱；
- 异常处理代码和正常流程代码混在一起；
- 异常识别处理能力弱。



```
void 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;  
    }  
}
```

Java的异常处理方式:

- 异常事件信息表达能力强;
- 异常处理代码和正常流程代码分开编写;
- 异常识别处理能力强。



6.1 异常概述

1. 异常和错误

Java语言中的非正常事件分为两种：

- **Exception**是指那些程序中可能发生的、经过处理有可能恢复正常的非正常事件。经过处理后，可以不中断程序的执行。**非致命的**

如：输入输出异常、运行时异常.....

- **Error**是指那些程序中可能发生的、非常严重且无法恢复的非正常事件。将使程序中断执行而退出系统。**致命的**

如：虚拟机错误、内存溢出错误.....

- **Java语言异常处理机制体现了Java语言鲁棒性的特点。**



6.1 异常概述

```
package testabstractclass;

import java.io.IOException;

public class Test1 {

    public static void main(String[] args) throws IOException{
        int i = 1;
        i = i/0;
        System.out.println(i);
    }
}
```

Exception in thread "main" java.lang.ArithmeticException / by zero
at testabstractclass.Test1.main(Test1.java:8)

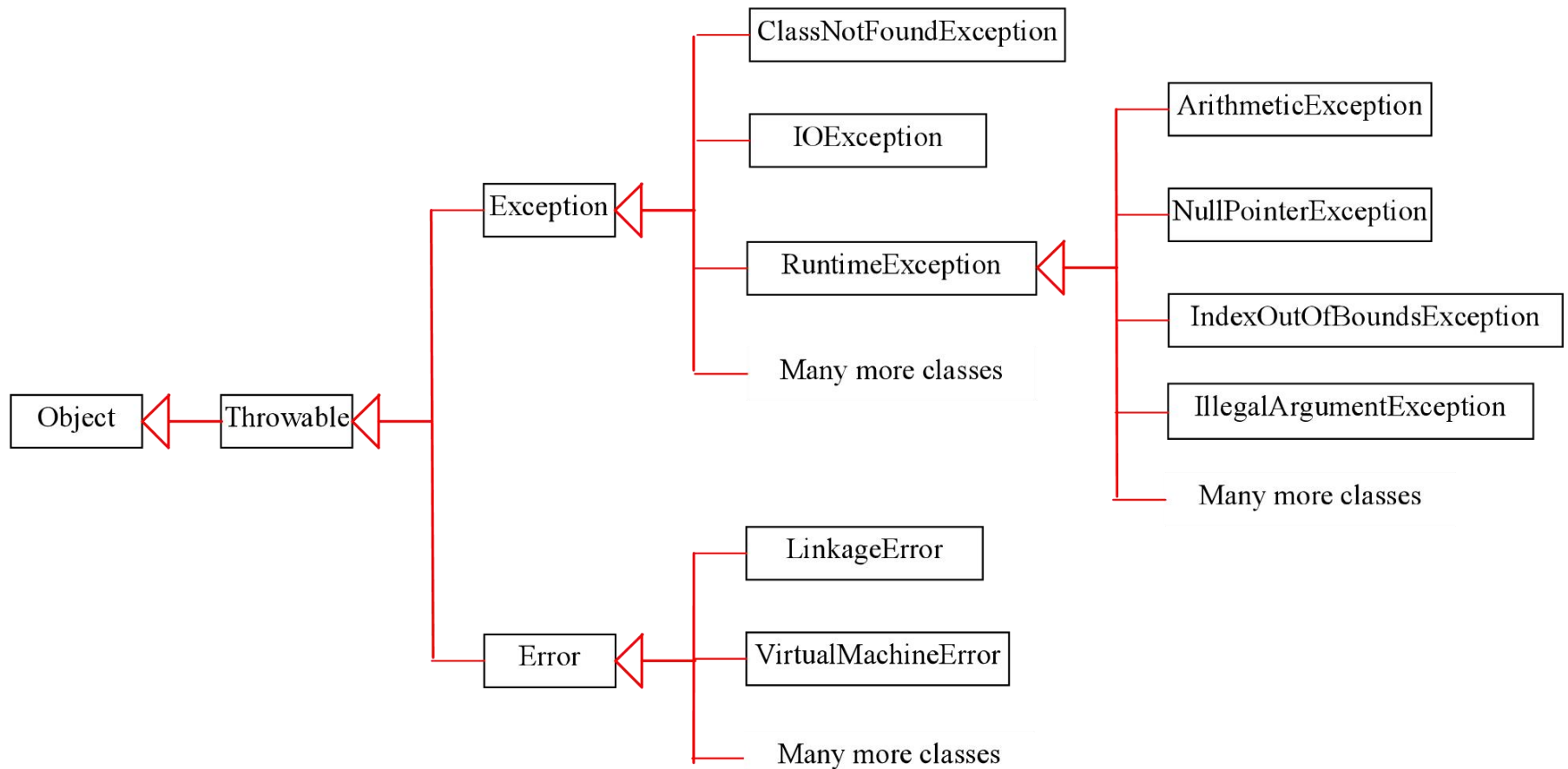
类型名称

行号

提示信息



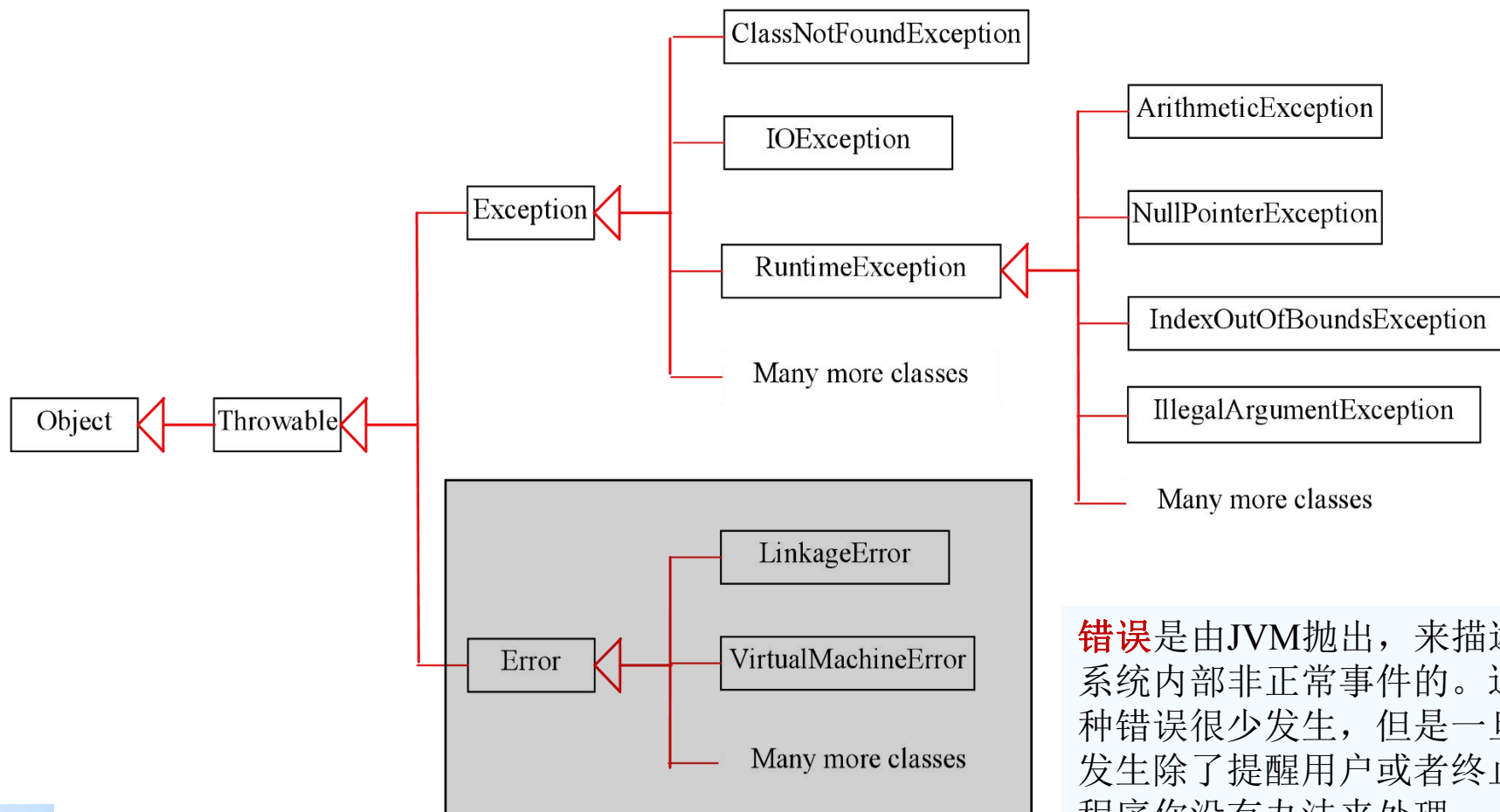
6.1 异常概述



Java中的异常处理是**基于面向对象**的一种运行态非正常事件（异常）处理机制，通过对异常信息的封装实现对用户非法操作、参数设置异常、硬件系统异常，运行时网络状态更换等在运行态中可能出现的异常信息的**处理机制**。



6.1 异常概述

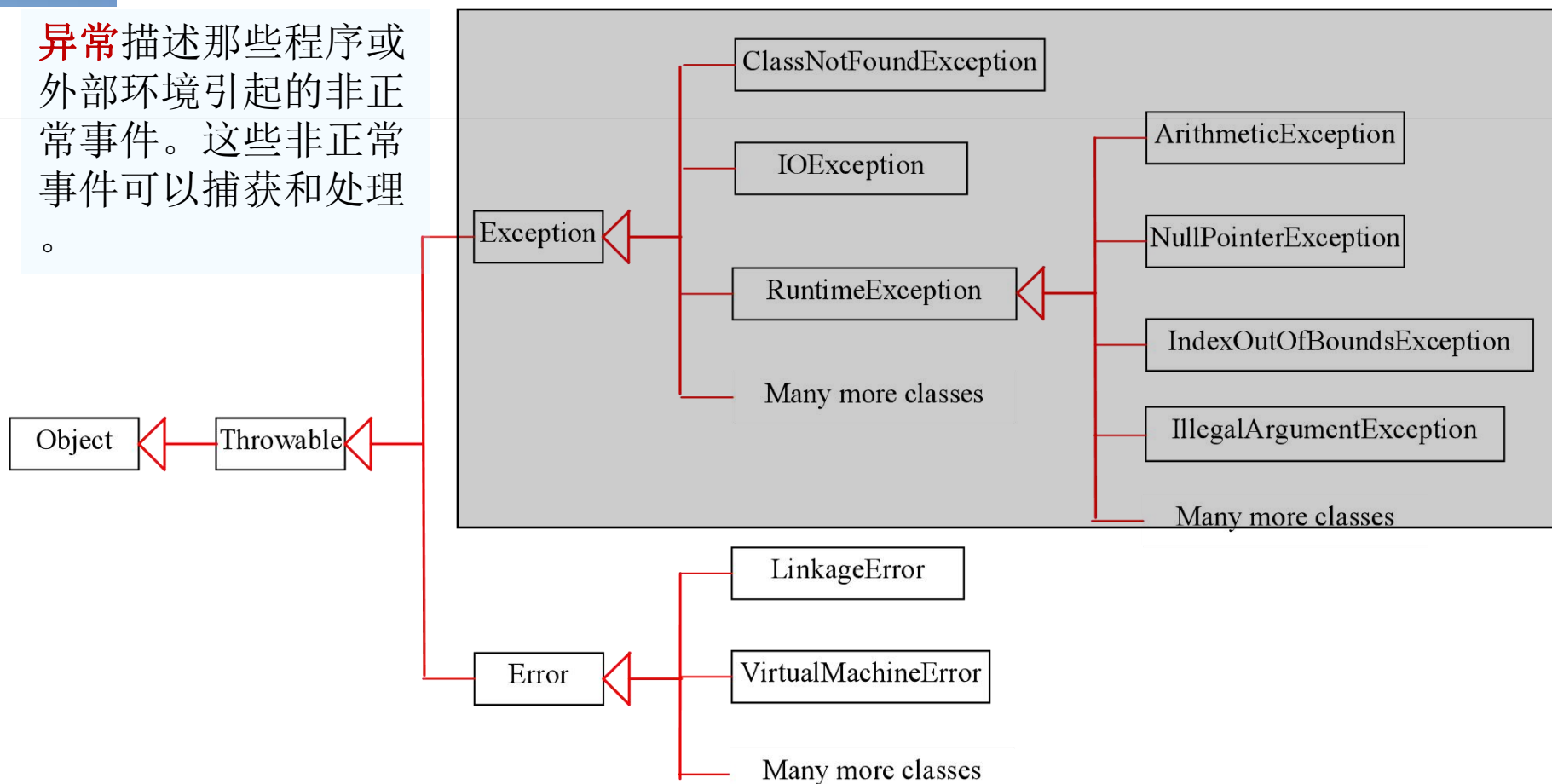


错误是由JVM抛出，来描述系统内部非正常事件的。这种错误很少发生，但是一旦发生除了提醒用户或者终止程序你没有办法来处理。



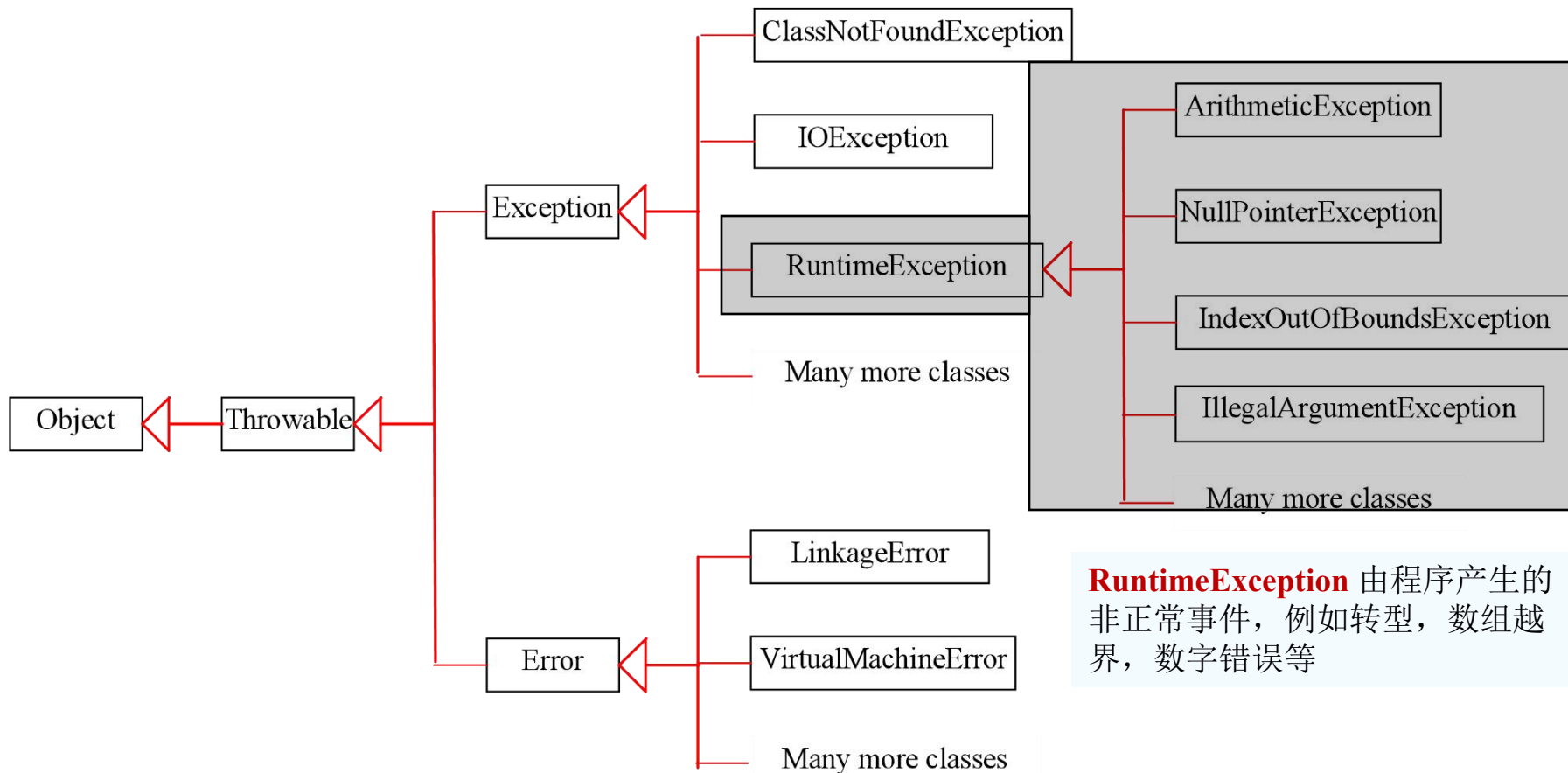
6.1 异常概述

异常描述那些程序或外部环境引起的非正常事件。这些非正常事件可以捕获和处理。





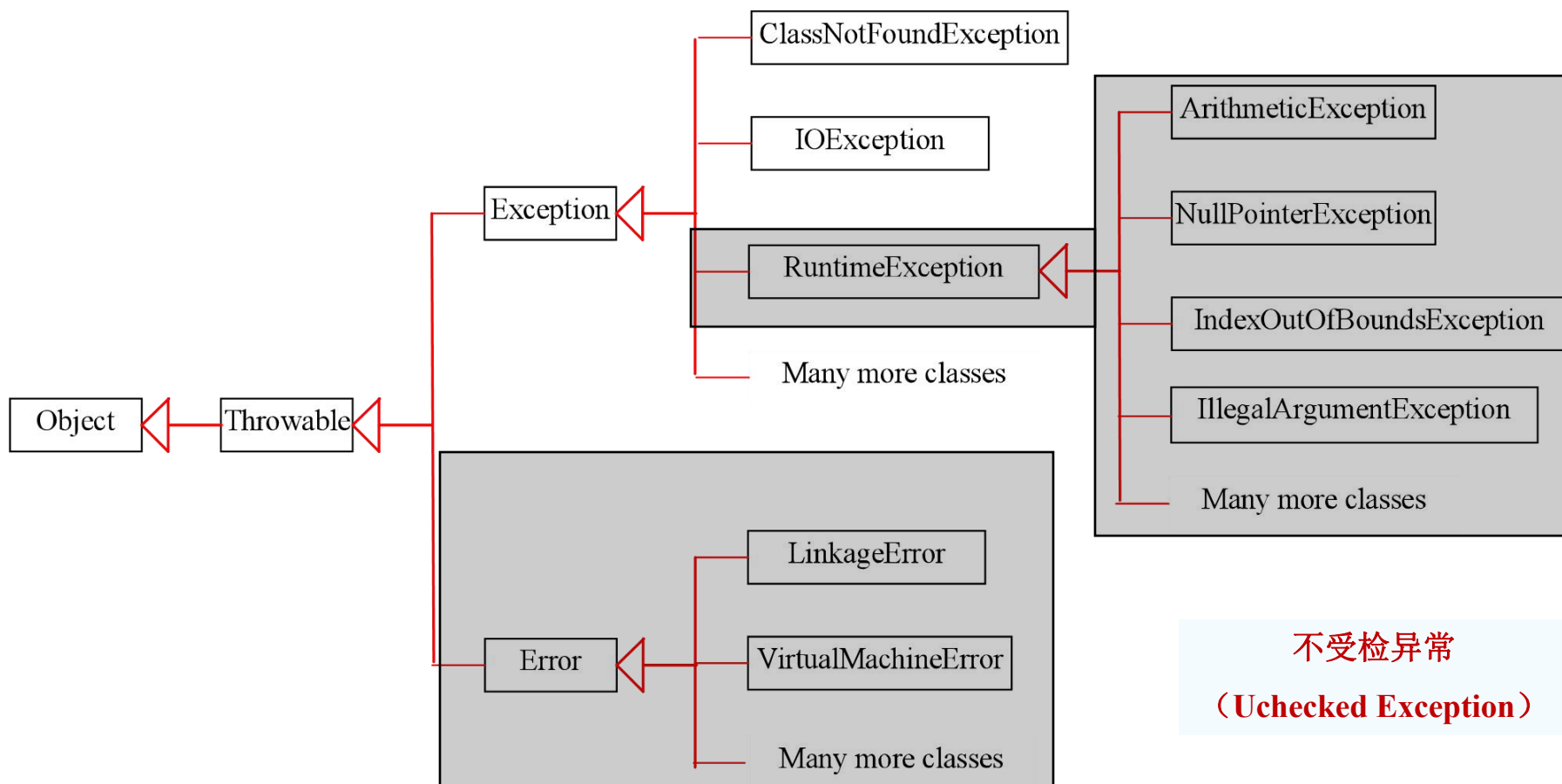
6.1 异常概述



RuntimeException 由程序产生的非正常事件，例如转型，数组越界，数字错误等



6.1 异常概述



不受检异常
(Unchecked Exception)



6.1 异常概述

不受检异常和受检异常：

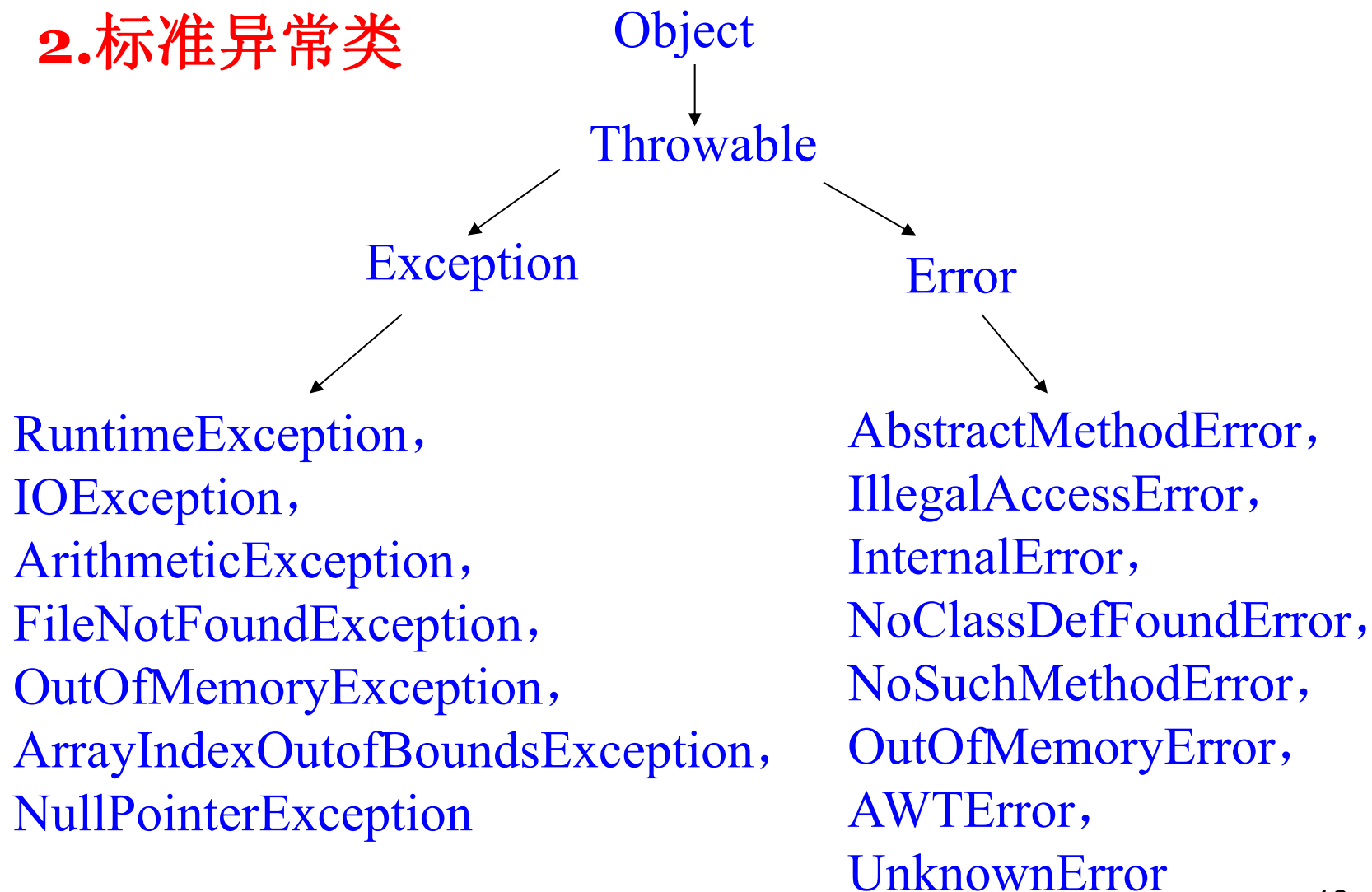
◆**不受检异常（Unchecked Exception）**：不要求程序员捕获和处理的异常，最终可从`main()`抛出并调用`printStackTrace()`。包括`RuntimeException`、`Error` 及其子类。多数情况下，unchecked异常反应程序的逻辑错误，如`NullPointerException` 异常，`IndexOutOfBoundsException` 异常。

◆**受检异常（Checked Exception）**：指编译器强制程序员检测和处理的异常。指`RuntimeException`、`Error` 及其子类之外的其余类异常。



6.1 异常概述

2. 标准异常类





6.2 异常处理方法

两种处理异常的方法：

- 使用try-catch-finally语句块结构在程序代码中**捕获和处理异常**；
- 把异常对象通过层层向上抛出直至转交给JVM处理。Java语言称产生异常和转交异常的过程为**抛出异常**。



6.2 异常处理方法

1. 捕获和处理异常

- 捕获和处理异常是通过**try-catch-finally**语句块实现的。语句块实际上就是在正常的程序中采用的几个标记，并不是新的程序语句。
- 程序在运行过程中对try语句块中的语句进行监测，**根据其中出现的异常的种类**决定是否采用catch语句块中的语句以及采用哪个catch语句块中的语句处理异常，**最后**，再运行finally语句块中的语句来结束捕获和处理异常的过程。
- 多个catch块时候，**只会匹配其中一个异常类**并执行catch块代码，而不会再执行别的catch块，并且匹配catch语句的**顺序是由上到下**。



6.2 异常处理方法

程序员认为可能出现异常的语句，在语句序列中划定捕获异常的范围。

try{ statements }

不同的异常参数

处理在try语句块中捕获的异常，每个catch块负责处理一种类型的异常。

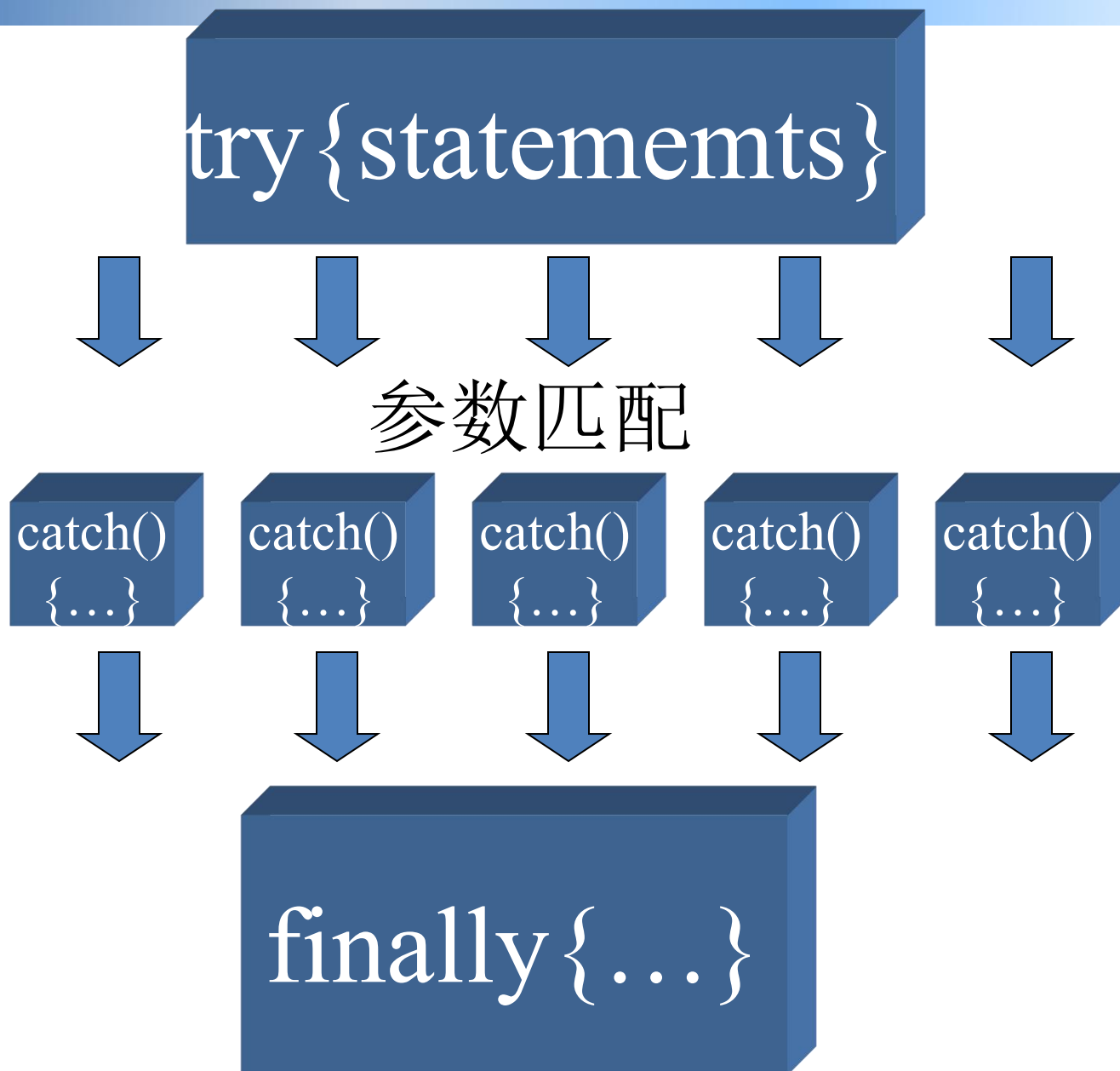
catch(ExceptionClassName obj){ statements }

catch(ExceptionClassName obj){ statements }

finally{ statements }

无论在try语句块中是否发生异常，finally语句块都是要执行的。

6.2 异常处理方法





示例：带有异常处理功能的数组下标越界的例子。

```
import javax.swing.JOptionPane;
public class ExcepDemo3
{
    public static void main(String[] args)
    {
        String string[] = {"Easter Sunday", "Thanksgiving", "Christmas"};
        String output = "";
        int k = 0, m = 0;
        try{
            for(int i=0;i<5;i++)
            {
                k = i + 1;
                output += string[i];
                output += "\n";
                m = i + 1;
            }
        }
        catch(Exception e)
        {
            output += e.toString();
        }
        finally
        {
            output += "\nround " + k + " started";
            output += "\nIt is terminated at round " + m;
        }
        JOptionPane.showMessageDialog( null, output );
        System.exit( 0 );
    }
}
```





6.2 异常处理方法

获取异常有关信息的三个方法：

- **getMessage()** : 获取异常性质。
- **toString()** : 给出异常的类型与性质。
- **printStackTrace()** : 指出异常的类型、性质、栈层次及出现在程序中的位置。



6.2 异常处理方法

两种处理异常的方法：

- 使用try-catch-finally语句块结构在程序代码中**捕获和处理异常**；
- 把异常对象通过层层向上抛出直至转交给JVM处理。Java语言称产生异常和转交异常的过程为**抛出异常**。



6.2 异常处理方法

2. 抛出异常

➤Java语言也允许指明出现的异常不在当前方法内处理，而是将其抛出，送交到调用它的方法来处理，在调用序列中逐级向上传递，乃至传递到Java运行时系统，直至找到一个运行层次可以处理它为止。

➤声明抛出异常是在一个方法声明中的**throws**子句中给出的。其语法格式为

returnType methodName([paramList])

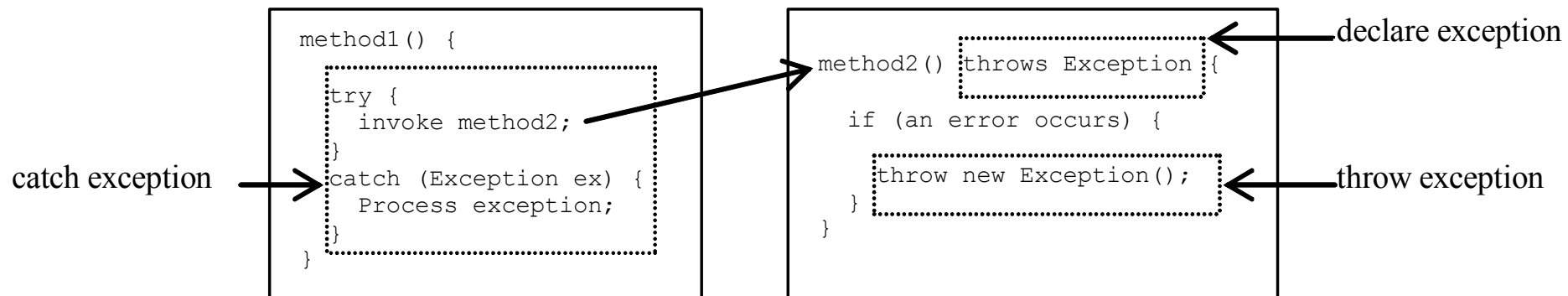
throws ExceptionList { ... }

➤抛出异常用**throw** 子局实现，语法格式为

throw ExceptionObject;



6.2 异常处理方法



声明，抛出，捕获异常



示例：抛出异常程序

```
public class ThrowsException {  
    public void catchThowException(int str)  
        throws ArrayIndexOutOfBoundsException, ArithmeticException,  NullPointerException {  
        System.out.print(str + " ");  
        if (str == 1) {  
            int a = 0;  
            int b = 6 / a;  
        } else if (str == 2) {  
            String s[] = new String[5];  
            s[0].toCharArray();  
        } else if (str == 3) {  
            int[] a = new int[5];  
            a[10] = 10;  
        } else {  
            System.out.println("没有发现异常,系统正常执行");  
        }  
    }  
}
```

```

public static void main(String args[]) {
    ThrowsException te02 = new ThrowsException();
    try {
        te02.catchThrowException(0);
    } catch (Exception e) {
        System.out.println("异常:" + e.getMessage());
    }
    try {
        te02.catchThrowException(1);
    } catch (Exception e) {
        System.out.println("异常:" + e);
    }
    try {
        te02.catchThrowException(2);
    } catch (Exception e) {
        System.out.println("异常:" + e);
    }
    try {
        te02.catchThrowException(3);
    } catch (Exception e) {
        System.out.println("异常:" + e);
    }
}

```

- 0 没有发现异常,系统正常执行
- 1 异常:[java.lang.ArithmeticException](#): / by zero
- 2 异常:[java.lang.NullPointerException](#)
- 3 异常:[java.lang.ArrayIndexOutOfBoundsException](#): 10



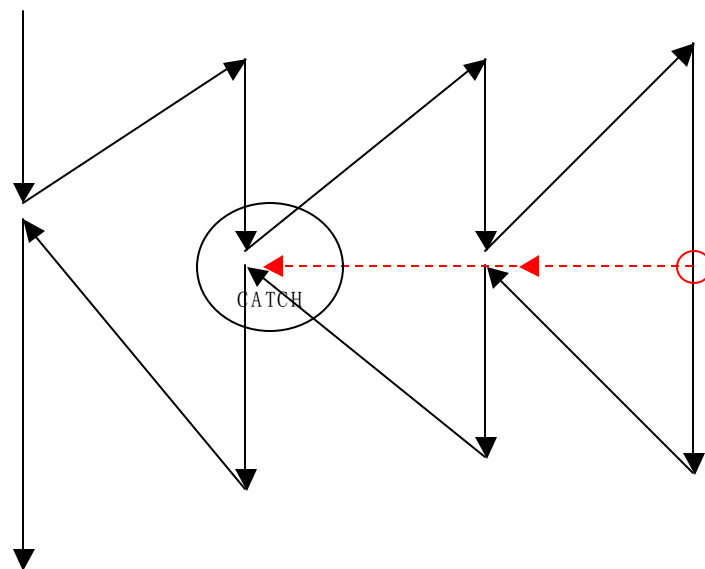
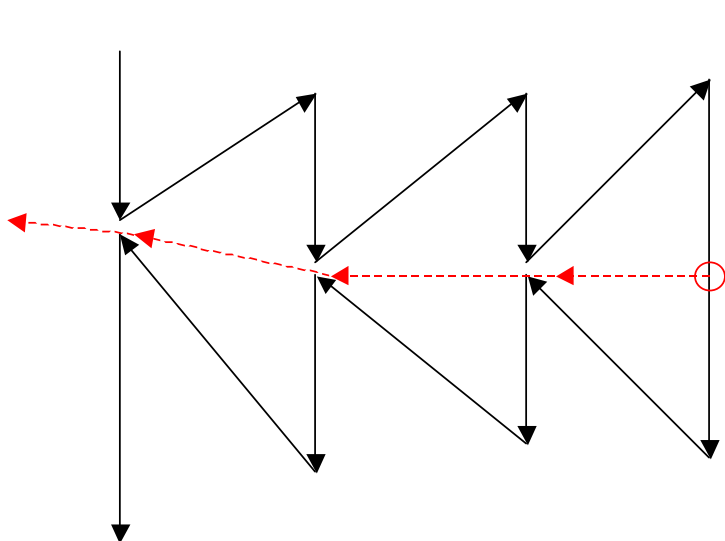
6.2 异常处理方法

➤ throws 和 throw 的区别:

- **throws** 是在方法声明时放在方法头中的，作用是声明一个方法可能抛出的所有异常；
- **throw** 则出现在方法体的内部，是一个具体的执行动作，作用是抛出一个具体异常对象。

6.2 异常处理方法

3. 异常传递链



如果在方法中产生了异常，方法会在抛出异常的地方退出；
如果不想终止方法，那就需要在特定的区域用**try**来捕获异常。



示例：异常处理综合示例

```
import java.io.*;
public class ThrowingExceptions
{ public static void main( String args[] )
  {   try {   throw1();           1   8
      }
      catch ( EOFException eofe )
      {       System.err.println( eofe.getMessage() + "\n" );
      }
      catch ( IOException ioe )
      {       System.err.println( ioe.getMessage() + "\n" );
      }
      catch ( Exception e )      9
      {       System.err.println( e.getMessage() + "\n" ); 10
              e.printStackTrace(); 11
      }
  }
}
```

Exception e 捕获提前
会发生什么现象？



```
public static void throw1() throws Exception 7
{
    throw2(); 2
}

public static void throw2() throws Exception 6
{
    try{
        System.out.println("方法二") 3
        throw new Exception("在方法二中抛出"); 4
    }
    catch(RuntimeException re)
    {
        System.err.println("在方法二的捕获中抛出");
    }
    finally{
        System.err.println("finally总是执行的") 5;
    }
}
```

```
命令提示符
方法二
finally总是执行的
在方法二中抛出

java.lang.Exception: 在方法二中抛出
    at ThrowingExceptions.throw2(ThrowingExceptions.java:35)
    at ThrowingExceptions.throw1(ThrowingExceptions.java:28)
    at ThrowingExceptions.main(ThrowingExceptions.java:9)
```



6.2 异常处理方法





6.3 自定义异常

1. 自定义异常的创建

可以在Java类库中现有异常类的基础上由用户创建新的异常类，新的异常类必须用**extends**子句声明是Exception类的子类。

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



6.3 自定义异常

2.自定义异常的抛出与捕获

➤自定义异常的抛出:

定义异常类的代码写在程序中，与其他类定义并列成为程序的一部分，在使用时与已有的异常类基本相同，只是在throw子句中使用下面的语法:

```
throw new MyException( );
```



示例：自定义异常的抛出与捕获

```
class SimpleException extends Exception {}

public class SimpleExceptionDemo {
    public void f() throws SimpleException {
        System.out.println( "Throwing SimpleException from f()" );
        throw new SimpleException ();
    }
    public static void main(String[] args) {
        SimpleExceptionDemo sed = new SimpleExceptionDemo();
        try {
            sed.f();
        } catch (SimpleException e) {
            System.err.println("Caught it!");
        }
    }
}
```

Throw SimpleException from
f()

Caught it!



JAVA异常处理机制的优点

- 把各种不同的异常事件进行分类，体现了良好的层次性。
- 处理异常的代码和“常规”代码分开，减少代码，增强可读性。
- 使异常事件可以沿调用栈自动向上传播，而非通过函数返回值传播。
- 可以统一或分别处理具有相同父类的异常。
- 为具有动态运行特性的复杂程序提供了强有力的控制方式。



总 结

- 异常和错误
- 不可检异常与可检异常
- 处理异常的两种方法：
 - 捕获和处理异常 try-catch-finally
 - 抛出异常 throws, throw
- 异常传递链
- 自定义异常与抛出
- throw 与 throws 的区别