Java程序设计

计算机科学与技术

贡正仙

zhxgong@suda.edu.cn

第7章
JAVA中异常处理

主要内容

- 异常
- 异常捕获
- 异常抛出
- 编写自己的异常类

异常概述

- 程序除了按用户需求完成所规定的功能外，还应该对程序运行过程中可能出现的异常（例如：除0溢出、数组越界、文件找不到等）进行处理
  - 异常处理得越早，损失就越小
  - 首先应该预计所有可能出现的异常，然后考虑能否完全避免。如果不能完全避免，再考虑异常发生时的具体处理方法
- 在程序中处理异常主要考虑两个问题：
  - 如何表示异常
  - 如何控制处理流程

异常概述

- 传统的异常处理（以C为代表）：
  - 出现异常时得到一个“异常”的值或编码，程序通过if语句判断调用相应的异常处理代码
  - 缺点：
    - 表示异常情况的能力有限，仅靠值或编码，难以表达异常情况所包含的所有信息
    - 异常流程的代码和正常流程的代码混合在一起，会影响程序的可读性，容易增加程序结构的复杂性
    - 随着系统规模的不断扩大，这种处理方式已经成为创建大型、可维护应用程序的障碍

异常概述

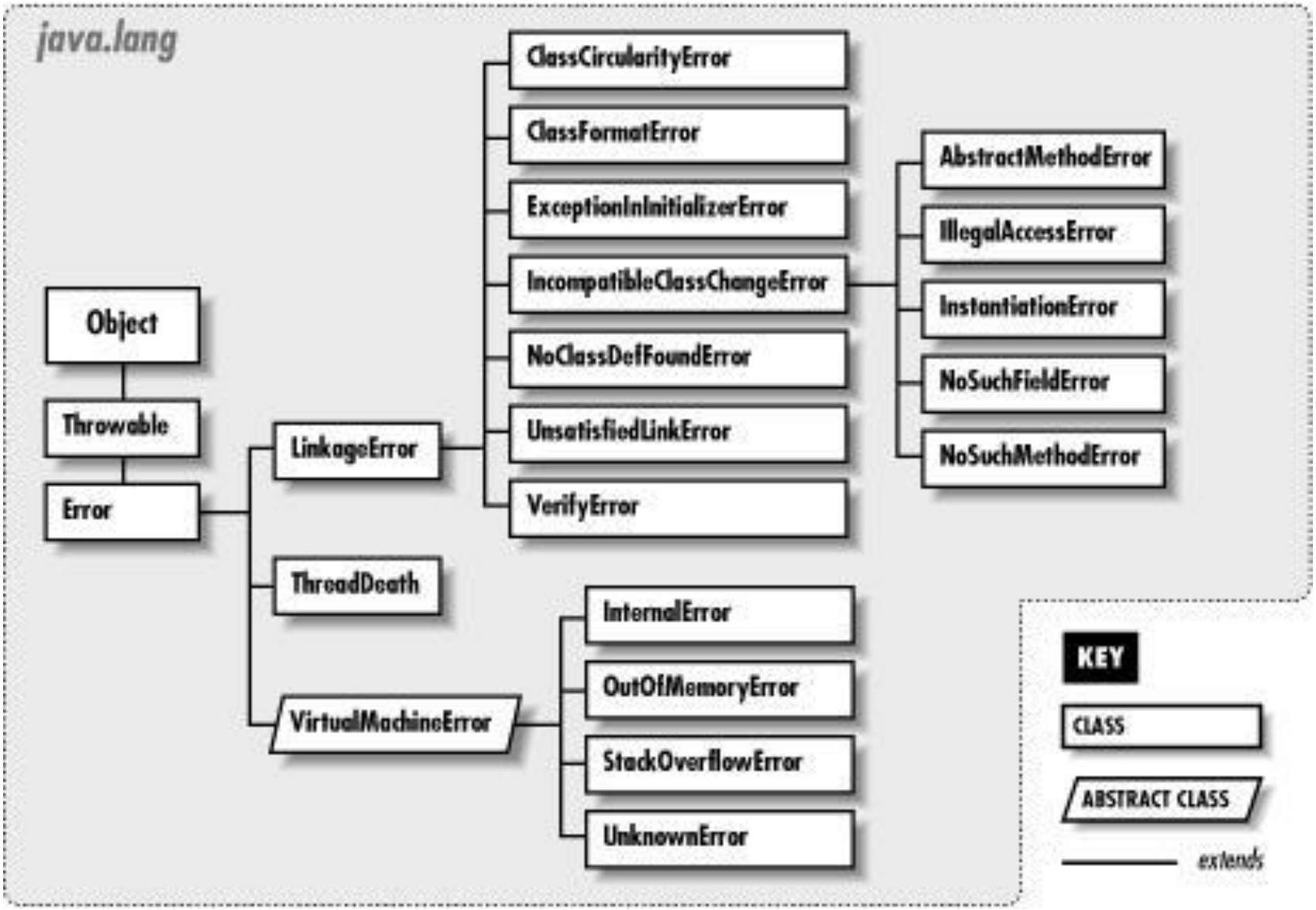
- 面向对象机制中的异常处理（以Java为代表）：
  - 发生的异常看成一个对象，封装了对应的错误类型以及程序运行的状态等信息
  - 优点：
    - 把各种不同类型的异常情况进行分类，形成不同的异常类，可以充分发挥类的可扩展和可重用的优势
    - 异常流程代码和正常流程代码分离，提高了程序的可读性，简化了程序的结构
    - 可以灵活地处理异常。如果当前方法有能力处理异常，就捕获并处理它，否则只需抛出异常，由调用者来处理

Java中的异常处理

- Java中异常的产生：
  - 执行过程中如果发生异常，就会生成对应异常类的对象；
  - 异常对象可以由发生异常的方法生成，也可以由Java虚拟机生成
  - 抛出异常：生成异常对象，并把它提交给运行时系统的过程
  - Java中的异常是Throwable类及其子类生成的实例

Java中异常的层次

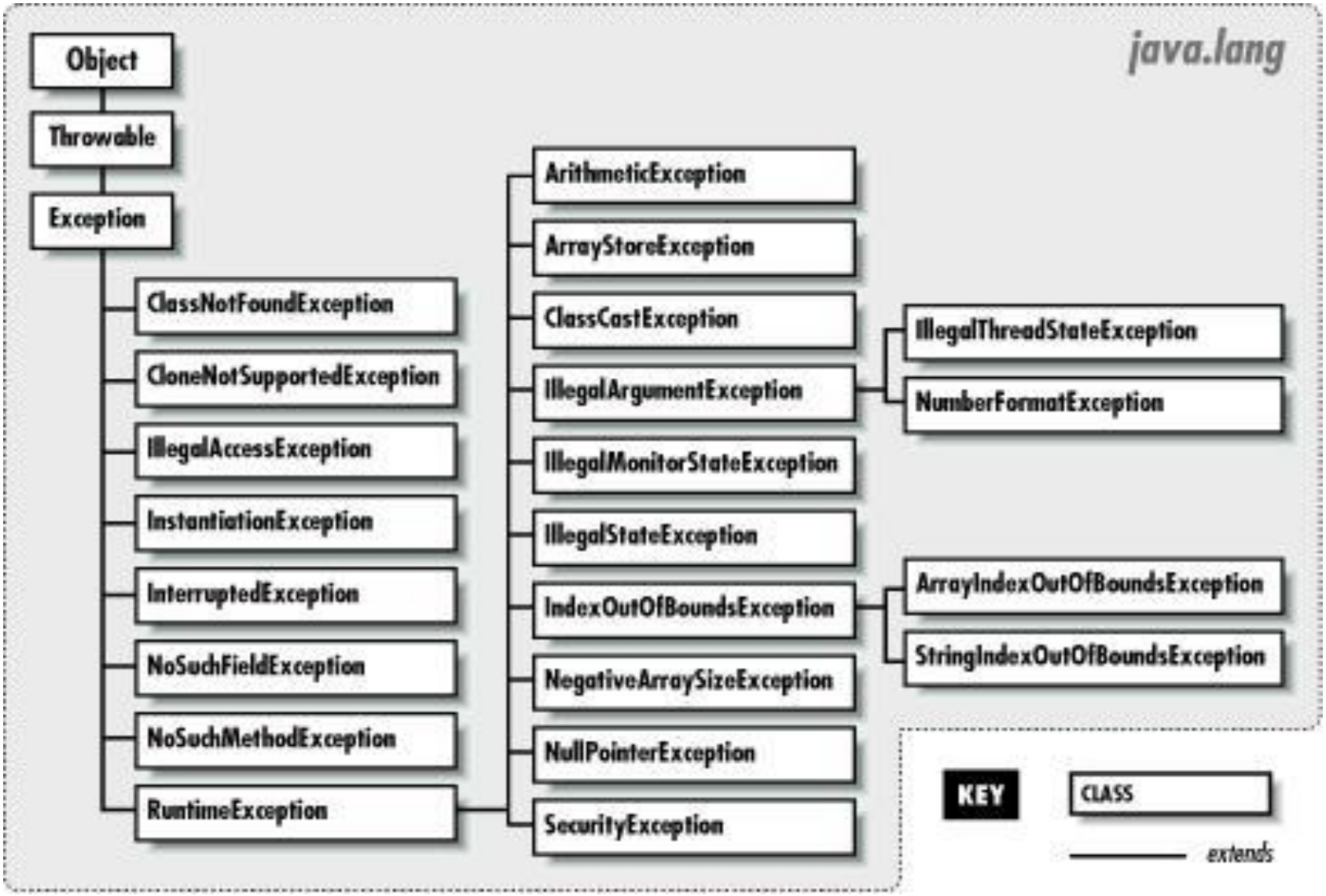
- java.lang.Throwable类：
  - 常用方法
    - getMessage()——返回String类型的异常信息
    - printStackTrace()——打印跟踪方法调用栈而获得的详细异常信息。在程序调试阶段，此方法可用于跟踪错误
- Throwable类有两个标准的子类：
  - java.lang.Error：
    - 错误，是指虚拟机或动态装载等相关的问题，如系统崩溃、动态链接失败、虚拟机错误等
    - 这类错误一般认为是无法恢复和不可捕获的，将导致应用程序中断。



java.lang.Error子类

Java中异常的层次

- Throwable类有两个标准的子类：
  - java.lang.Exception
    - 异常，是可以捕获，可能恢复的异常情况
    - Exception类对象是Java程序处理或抛出的对象
    - Exception类及其子类（见下图）
    - 常见的一些异常（见下表）



java.lang.Exception子类

Java中的常见异常

常用的Java异常类	说明
Exception	异常层次结构的根类
RuntimeException	运行时异常。许多 java.lang 异常的基类
ArithmeticException	算术错误情形，如以零作除数
IllegalArgumentException	方法接收到非法参数
ArrayIndexOutOfBoundException	数组大小小于或大于实际的数组大小
NullPointerException	尝试访问 null 对象成员
ClassNotFoundException	不能加载所需的类
NumberFormatException	数字转化格式异常，比如字符串到 float
IOException	I/O 异常的根类
FileNotFoundException	找不到文件
EOFException	文件结束
InterruptedException	线程中断

异常处理

- Java异常处理包括
  - 声明异常
  - 抛出异常
  - 捕获异常等内容。
  - 回顾一下前面我们哪些地方需要异常处理却没有做？
  - 1. 接受main函数传入的两个参数，然后接受一个运算符进行计算，没有判断args的数量
  - 2.接受一个包含日期值的字符串要解析成日期型出错
  - 3.。。。。。。。。。。

异常处理

- 捕获和处理异常：

```
try{
    //执行的代码块
}catch(Exception1 e){
    //对异常类型1的处理
}catch(Exception2 e){
    //对异常类型2的处理
}finally{
    ……
}
```

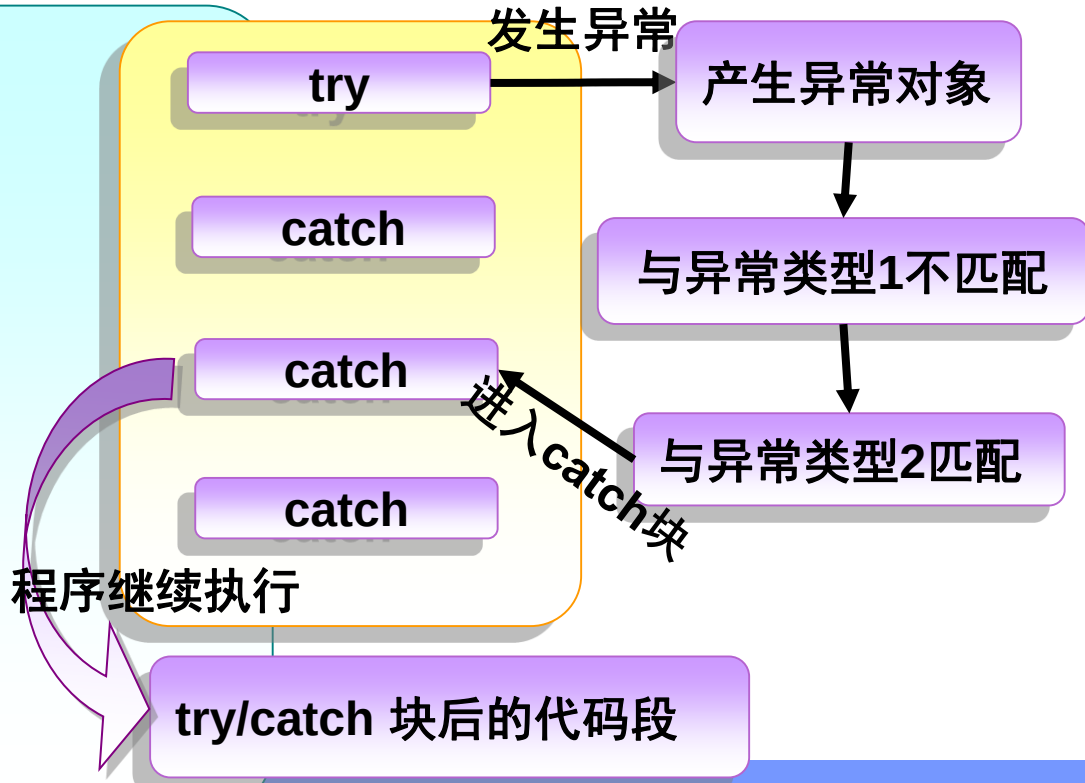
异常处理

- 捕获和处理异常
  - try语句：try{.....}选定捕获异常的范围，在该范围中的语句会生成异常对象并抛出
  - catch语句：一个try后面可以跟多个包含异常引用的catch块来处理指定的异常
  - catch子句按顺序计算，执行第一个可能匹配
  - 最多只能有一个catch子句执行

多重catch块 2-1

- 一段代码可能会引发多种类型的异常
- 当引发异常时，会按顺序来查看每个 **catch** 语句，并执行第一个与异常类型匹配的**catch**语句
- 执行其中的一条 **catch** 语句之后，其后的 **catch** 语句将被忽略

```
public void method(){
    try {
        // 代码段
        // 产生异常(异常类型2)
    } catch (异常类型1 ex) {
        // 对异常进行处理的代码段
    } catch (异常类型2 ex) {
        // 对异常进行处理的代码段
    } catch (异常类型3 ex) {
        // 对异常进行处理的代码段
    }
    // 代码段
}
```



异常处理

- 说明：
  - 异常应该按从最特殊到最一般的顺序排列

```
void calculate(){
    try{   }
    catch(Exception ex){
        ex.printStackTrace();
    }
    catch(ArithmeticException ex){
        System.out.println("such exception can not play
        effectiveness");
    }
}
```

异常处理

- 捕获和处理异常
  - 由于异常会强制中断正常流程，这会使得某些不管在任何情况下都必须执行的步骤被忽略，从而影响程序的健壮性。
  - finally语句：这块中的内容不论异常是否发生，其内容都必须被执行，提供了统一的出口

- 捕获和处理异常：

```
try{
    //执行的代码块
}catch(Exception1 e){
    //对异常类型1的处理
}catch(Exception2 e){
    //对异常类型2的处理
}finally{
    ......
}
```

- 异常结构中各成员之间的关系

TRY

CATCH

FINALLY

我，做、做、做！

看看、有麻烦没，该我管吗？

放心干吧，出了事我兜着！

异常处理

- 例如，我们在读取文件正常的流程为：打开文件，读取文件，关闭文件。异常流程为：打开文件之后没有读取。以下ReadFile()方法表示如下：

```
public void ReadFile(){
        try{
            打开文件；
            读取文件；//可能会抛出异常
            关闭文件；
         }catch(FileException fe){
                    ……
        }
}
```

异常处理

- 从上面的代码我们可以发现，如果在读取文件时抛出异常，那么关闭文件这个操作装被跳过而不会被执行，系统占用的资源不会被释放。这样的流程显然是不安全的。finally代码块能保证特定的操作总是会被执行。它的形式如下：

```
public void ReadFile(){
        try{
            打开文件；
            读取文件；//可能会抛出异常
          }catch(FileNotFoundException e){
                  ……
          }finally{
            关闭文件；
          }
}
```

```
public class TryCatchFinallySample{
    static void Proc(int sel){
        System.out.println("-----In Situation "+sel+"-----");
        try{
            if (sel==0){
                System.out.println("No Exception caught");
            }
            else if (sel==1){
                int i=0;
                int j=4/i;
            }
            else if (sel==2){
                int iArray[]=new int[4];
                iArray[10]=3;
            }
        }catch(ArithmeticException e){
            System.out.println("Catch "+e);
        }catch(ArrayIndexOutOfBoundsException e){
            System.out.println("Will not be executed");
        }finally{
            System.out.println("in Proc finally ");
        }
    }
    public static void main(String args[])
    {
        Proc(0);Proc(1);Proc(2);
    }
```

完整trycatchfinally例子1

Try、catch及finally语句块使用注意的问题

- 无**catch**时**finally**必须紧跟**try**
- **Catch**与**finally**不能同时省略
- **Try**、**catch**以及**finally**块之间不能插入任何其他代码

```
//使用finally语句
  try
  {
       if(a==10) return;
   }
   finally
   {
        ….;
   }
```

有了try语句，其后必须跟catch或finally或其中之一。Finally无论代码是否产生异常都要执行，finally的优先级比return要高，

- Try-catch-finally嵌套

```
try{
    int[] s=new int[5];
    try{
        System.out.println(s[-1]);
    }
    catch(NegativeArraySizeException ex){
        System.out.println("inner catch");
        ex.printStackTrace();
    }
}
catch(ArrayIndexOutOfBoundsException
   ex){
    System.out.println("outer catch");
    ex.printStackTrace();
}
```

```
finally{
    System.out.println("end");
}
```

异常处理-throws抛出异常

某个方法可能会发生异常，但不想在当前方法中处理异常，可以将异常抛出，在调用该方法的代码中捕获异常进行处理

异常处理-throws抛出异常

用throws抛出异常的格式：

returnType **method() throws Exception1,Exception2,……**

关键字throws表示，method()方法可能抛出Exception1, Exception2,等多个异常。如果方法可能抛出多个异常可以在throws后面添加异常列表，用逗号隔开。

```
class throwsExceptionSample{
    static void Proc(int sel) throws
            ArithmeticException,ArrayIndexOutOfBoundsException{
        System.out.println("-----In Situation "+sel+" -----");
        if (sel==0){
                System.out.println("No Exception caught");
        }
        else if (sel==1){
                int iArray[]=new int[4];
                iArray[10]=3;
        }
    }                                   public static void main(String args[]){
}                                               try{
                                                        Proc(0);
                                                        Proc(1);
                                                }catch(ArrayIndexOutOfBoundsException e){
                                                        System.out.println("Catch "+e);
                                                }finally{
                                                        System.out.println("in Proc finally ");
                                                }
                                        }
```

异常处理-throws抛出异常

- Java不需要在方法中显式声明Error和RuntimeException,但非运行时异常需要处理

```
class throwsExceptionSample{
    static void Proc(int sel) throws
    //ArithmeticException,ArrayIndexOutOfBoundsException
      {
            System.out.println("-----In Situation "+sel+" -----");
            if (sel==0){
                  System.out.println("No Exception caught");
            }
            else if (sel==1){
                  int iArray[]=new int[4];
                  iArray[10]=3;
            }
      }
}
```

可以省略

例子：TestThrows.java(运行时，可省略)　TestThrows2.java（非运行时，不可省略）

异常处理-throws抛出异常

- Java不需要在方法中显式声明Error和RuntimeException,但非运行时异常需要处理

```
public void throwsException()
    throws ClassNotFoundException {
      // 非运行时异常，不可以省略
      Class.forName("com.mysql.jdbc.Driver");
    }
```

```
try {
    throwsException();
}
catch (ClassNotFoundException e) {
    System.out.println("the odbc is not set!");
 }
finally {
 System.out.println("finally part");
 }
```

例子：TestThrows2.java（非运行时，不可省略）

异常处理-throws抛出异常

有些情况下，一个方法并不需要处理它所生成的异常，或者不知道该如何处理这一异常，这时就向上传递，由调用它的方法来处理这些异常，这就需要throw子句

异常处理-throw抛出异常

用throw语句抛出异常：

- throw语句可以明显地引发一个异常
- 基本格式：

throw throwable**Instance**;

```
class ThrowSample{
    static void doExample(){
        try{
                NullPointerException t=new NullPointerException("demo");
                throw t;
        }catch(NullPointerException e){
                System.out.println("First catch inside Exception");
                throw e;
         }
    }
}
                public static void main(String args[]){
                        try{
                                doExample();
                        }catch(NullPointerException e){
                                System.out.println("recatch Exception:"+e);
                        }
                }
```

异常处理

在重新抛出异常中，要注意如下几点：

（1）利用catch子句的参数或使用new运算获取一个Throwable实例的句柄；

(2)程序执行到throw语句时立即中止，不再执行它后面的语句；

(3)在包含throw语句的**try块后面**寻找与其相匹配的catch子句来捕获抛出的异常；

(4)找到，则执行；如果找不到则向上一层程序抛出直到由JVM来处理。

- 如果想在当前的方法中捕获并处理throw抛出的异常，则必须使用try…catch语句;
- 通过throw抛出异常后，如果想由上一级代码来捕获并处理异常，则同样需要在抛异常的方法中使用throws关键字在方法的声明中指明要抛出的异常；
- 上述两种情况，若throw抛出的异常是Error、RuntimeException或它们的子类，则无须使用throws关键字或try…catch语句。

例子：

TestThrow3.java TestThrowNotRuntimeException.java

- Throws与Throw的区别和联系

  1、throws出现在方法函数头；而throw出现在函数体。

  2、throws表示出现异常的一种可能性，并不一定会发生这些异常；throw则是抛出了异常，执行throw则一定抛出了某种异常。

  3、两者都是消极处理异常的方式（这里的消极并不是说这种方式不好），只是抛出或者可能抛出异常，但是不会由函数去处理异常，真正的处理异常由函数的上层调用处理。

编写自己的异常类

- Java提供了多个异常类。但是，如果Java所提供的异常类无法适当描述我们遇到的问题，因此Java允许编程人员创建异常类，以处理Java的异常类未包含的异常或处理他们自己的异常。
- 可以通过扩展Exception类或者Exception类的子类来自定义异常。

编写自己的异常类

- 在自定义异常时，应该注意以下几点：

(1)自定义异常需要继承Exception 及其子类；

(2)若要抛出自定义的异常对象，使用throw关键字；

(3)要想抛出自定义异常，一定要将所调用的方法定义为可抛出异常的方法（即使用throws）。

```
class MyException extends Exception {
    //创建自定义异常类
    String message;
    public MyException(String ErrorMessagr)
    {
        message = ErrorMessagr;
    }
     public String getMessage(){
            return message;
     }
}
```

```
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");
        }
        return x/y;//返回值
    }
```

另一种方式

```
class MyException extends Exception {
    //创建自定义异常类
    String message;
   public MyException(String ErrorMessagr)
   {
      message = ErrorMessagr;
   }
    public String getMessage(){
          return message;
    }
}
```

```
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
```

```
class MyException extends Exception {}
    //创建自定义异常类
    public MyException(String ErrorMessagr)
    {
        Super(ErrorMessagr);
    }
}
```

ExceptionEx1.java

- 查错

```
public int mtd(int a,int b)
{
    try{
        if(b==0)
            throw Exception1(“”);
        a=a/b;
    }
    catch(Exception e1){
        ;
    }
}
```

```
class Exception1 extends
            Exception{
    Exception1(String str)
    {
        System.out.println(str);
    }
}
```

Throw后少new

课内练习

- 二、查错

```
public int mtd(int a,int
  b)
{
    try{
        int r=a/b;
    }
    catch(Exception e1)
    {          ;      }
    catch(Exception e2)
    {
        ;
    }
     catch(Exception e3)
    {
        ;
    }
     return r;
}
```

异常类重复

课内练习

- 三、查错

```
public int mtd(int a,int b){
    try{
       int r=a/b;
    }
    catch(异常类1 e1)
    {
          ;
    }
    finally{
    }
    catch(异常类2 e2)
    {
         ;
    }
    catch(异常类3 e3)
    {
          ;
    }
    return r;
```

Finally语句必须放在所有catch语句之后

课内练习

- 五、判断。给出结果，或指出错误

```
public int mtd(int a,int b)
 {
      try{
         a=a/(b-b);
      }
      catch(Exception e1)
      {
          a=10;
      }
      catch(
       ArithmetricException e)
       {
           a=100;
       }
       return a;
    }
    }
    //mtd()的返回值？
```

异常类位置不对，应从特殊到一般

课内练习

- 六、判断。给出结果，或指出错误

```
public int mtd(int a,int b)
 {
      try{
         a=a/(b-b);
        }
    catch(IOException
  e1)
     {
         a=10;
     }
}
```

```
  catch(Exception e)
  {
       a=100;
  }
  return a;
}
```

结果

Try语句块中不可能抛出IO异常，而catch里捕获该异常，不能通过编译。

课内练习

- 七、判断结果

```
try{
   int a[ ]=new int[5];
   a[4]=8;
   int k=a[5];
}
catch(
ArithmeticException e)
{
    k=50;
}
catch(
  NumberFormatException e)
{
     k=2;
}
catch(
 IndexOutOfBoundsException e)
{
     k=80;
}
//k=?
```

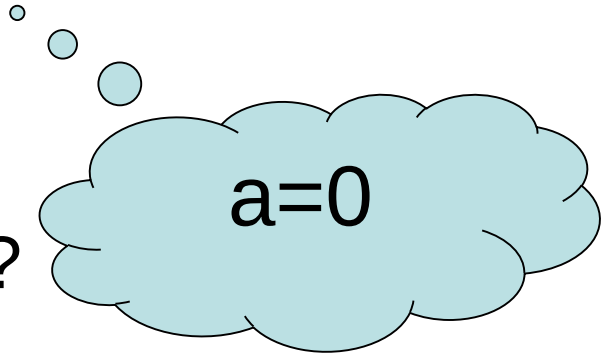
80

课内练习

- 八、判断结果

```
try{
    int a=10,int b=5;
    int c=5;
    a=a/(b-c);
}
catch(Exception e)
{
     a=100;
}
finally
{
      a=0;
 }
```

//a==?



课内练习

- 九、判断结果

```
int mtd(){
  try{
     int a=10,int b=5;
     int c=5;
     a=a/(b-c);
  }
   catch(Exception e)
  {
      return 100;
  }
  finally
  {
      System.out.println
       (“ error occur!”);
   }
}

//函数的返
```

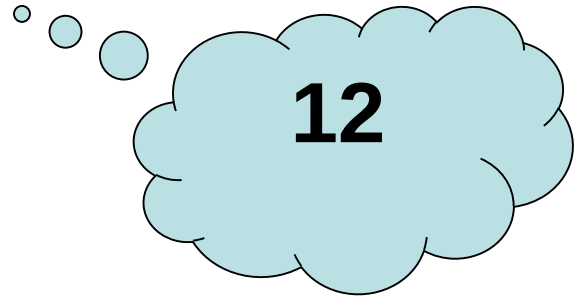
100
输出
“error occur！”

课内练习

- 十、判断结果

```
int mtd(){
   try{
      int a=10,int b=5;
      int c=4;
      a=a/(b-c);
   }
    catch(Exception e)
   {
       return 100;
   }
    finally{
        return 12;
    }
}
```

//函数的返回结果是？



实例

- 定义包Bank
- 定义银行帐户
  - 存储用户信息、帐户信息
  - 定义deposit和withdraw方法
  - 编写AccountOverdrawnException，当要取出比帐户上更多钱时触发
  - 编写InvalidDepositException，当无效钱数（小于0）存入时触发

小结

Java的异常处理涉及到五个关键字：try、catch、throw 、throws和finally。一般的异常处理流程由try、catch、finally三个代码块组成。try代码块包含了可能发生异常的程序；catch代码块用来捕获并处理异常。finally代码块主要用于释放被占用的相关资源。

try、catch、finally这三个关键字都不能单独使用。try语句可以和catch、finally组成 try...catch...finally或者try...catch或者try...finally三种结构。catch语句可以有一个或多个，finally语句最多只能有一个。当有多个catch块时候，Java虚拟机会匹配其中一个异常类或其子类，就执行这个catch块，而不会再执行其它的catch块。try、catch、finally三个代码块中变量的作用域相互独立，不能相互访问。如果非要在这些块中都可以相互访问，则需要将变量定义到这些块的外面。throw用于抛出异常或重新抛出异常。因此，throw语句后不允许有紧跟其他语句，即使有，这些语句没有执行的机会。

throws用于声明异常。如果方法可能抛出多个异常可以在throws后面添加异常列表，用逗号隔开。

Java允许编程人员创建异常类，以处理Java的异常类未包含的异常或处理它们自己的异常。自定义异常类必须是通过扩展Exception类,或者Exception类的子类来定义的。