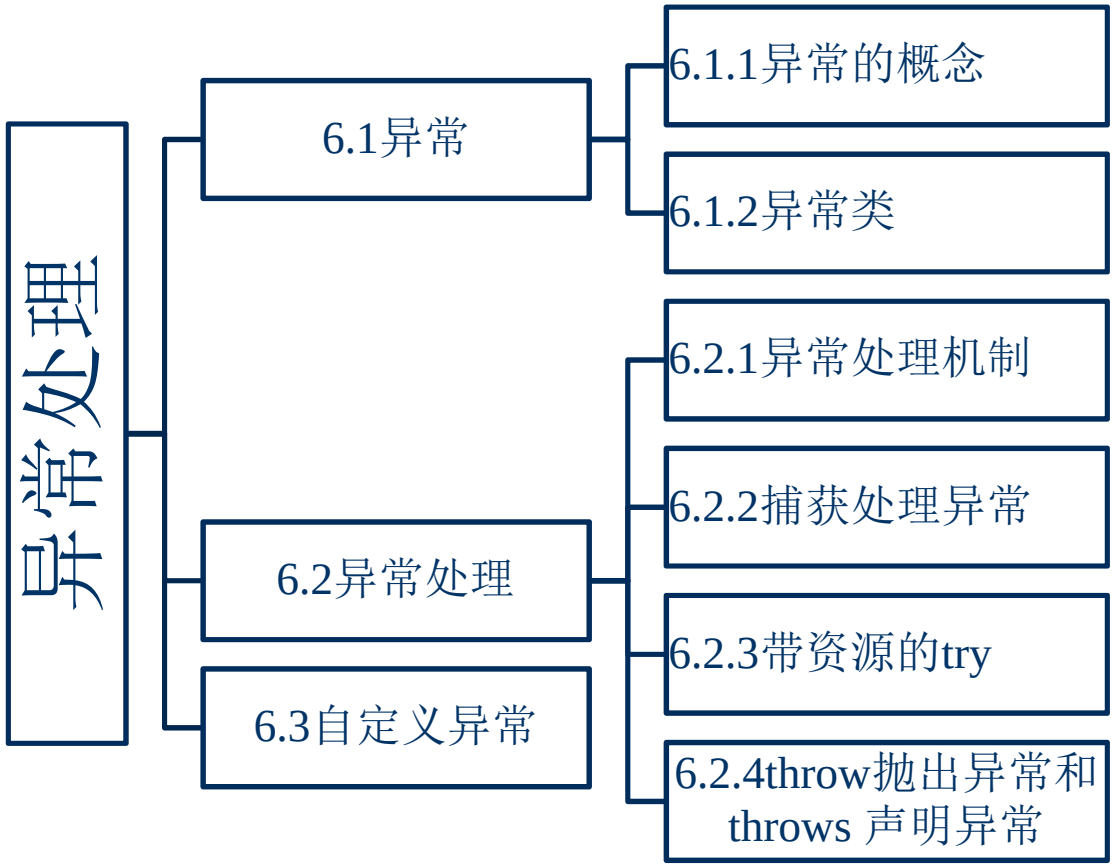
第六章 异常处理

第六章 异常处理


异常处理
6.1异常
6.1.1异常的概念
6.1.2异常类
6.2异常处理
6.2.1异常处理机制
6.2.2捕获处理异常
6.2.3带资源的try
6.2.4throw抛出异常和throws 声明异常
6.3自定义异常

第六章 异常处理

好的软件应该考虑到程序的健壮性，能够处理各种错误.

Java采用了面向对象的方法来表示程序中的各种运行错误，即异常，并提供了一套标准化的异常处理机制，实现了声明异常、抛出异常、捕获异常的操作。

6.1.1 异常的概念

异常（Exception），在Java中又称为例外，是程序在运行中由于一些特殊原因出现的错误，它会中断正在执行的程序。

6.1.1 异常的概念

程序出现的错误分为编译错误和运行错误两种。

1. 编译错误

因为所编写的程序存在语法问题，编译系统能直接检测出来。

2. 运行错误

程序在运行的时候才会出现的错误。除了因算法逻辑错误导致的，其他运行错误还分为两大类。

（1）**致命性的错误**，如Java 虚拟机产生错误、内存耗尽，系统硬件故障、动态链接失败等。这类错误应用程序无法处理。

（2）**一般性的（非致命性）错误**，是因编程错误或偶尔的外在因素导致的一般性问题，如：除数为零、数组越界、负数开平方，网络连接中断、读取不存在的文件等，通过某种处理后，程序还能继续运行。一般所说的异常（Exception）都是指这类错误。

代码divideException.java

```
public class divideException {
        public static void main(String args[]){
            int x=Integer.parseInt(args[0]);
            int y=Integer.parseInt(args[1]);
            int z=x/y;
            System.out.println(x+"/"+y+" = "+z);
    }}
```

例如

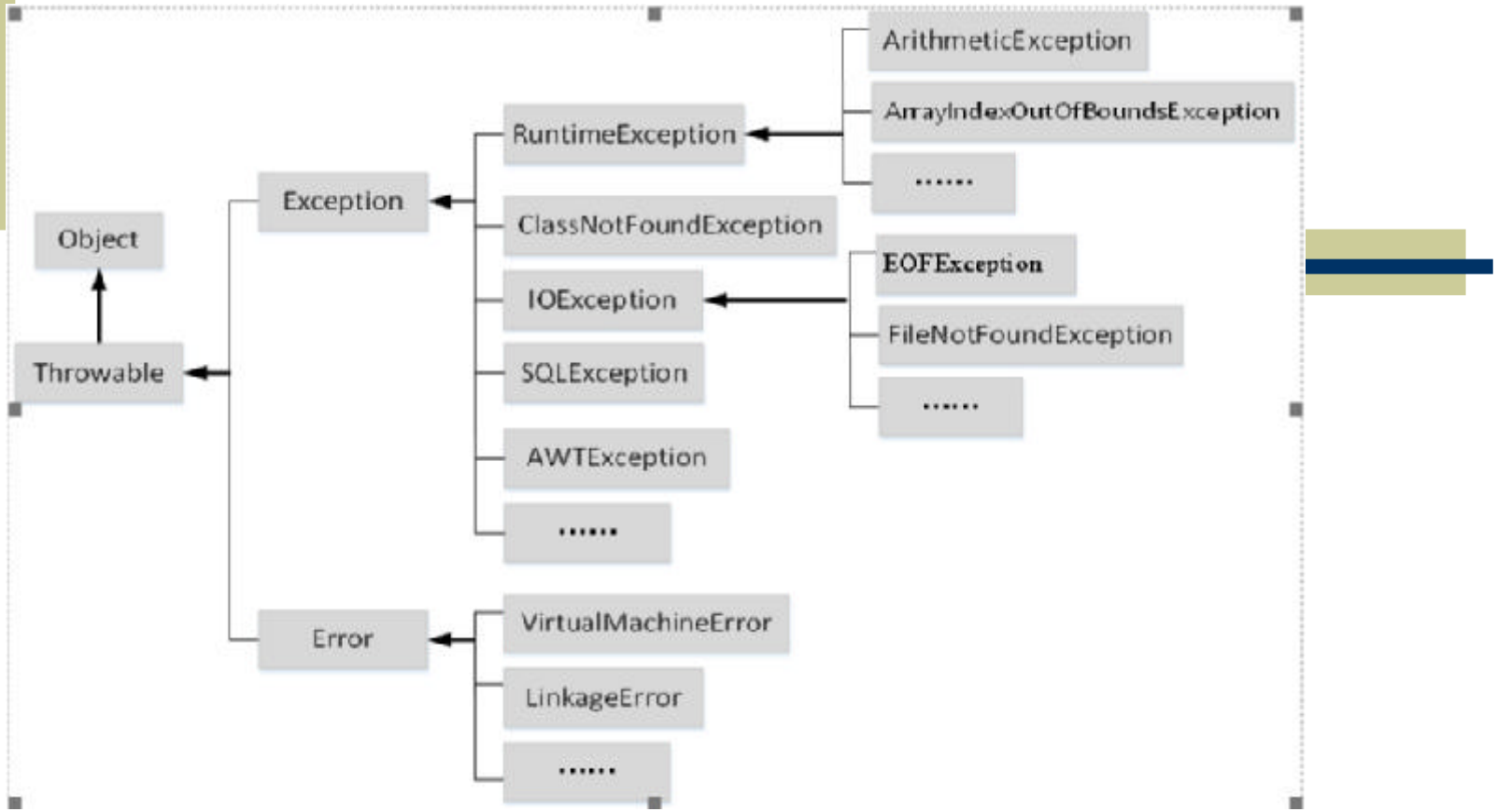
输入的第二个命令行参数是0：

```
java  divideException  6  0
```

输出结果：在控制台显示被0除的数学错误的异常信息。

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at divideException.main(divideException.java:5)
```

6.1.2 异常类



（1） **运行时异常（Runtime Exception）：**

由Java解释器抛出这些异常，并为他们提供默认的捕获处理代码。Java不要求捕获这类异常，当然应用程序也可以自己捕获处理运行时异常。也称为非受检查异常（unchecked Exception）。

（2） **受检查异常（Checked Exception）：**

程序必须强制对这类可能发生的异常进行处理，否则编译不能通过。

了解

表 6-1 常用的异常类

异常名称	类型	引起的原因
ArithmeticException	RuntimeException	数学错误，如被零除
ArrayIndexOutOfBounds	RuntimeException	数组下标越界
ArrayStoreException	RuntimeException	向数组中存放与声明类型不兼容对象异常
NullPointerException	RuntimeException	空对象引用异常
NumberFormatException	RuntimeException	字符串和数字间转换的故障
StringIndexOutOfBoundsException	RuntimeException	程序试图访问字符串中不存在的字符位置
ClassCastException	RuntimeException	类型强制转换异常
IllegalArgumentException	RuntimeException	传递非法参数异常
FileNotFoundException	Non_ RuntimeException	企图访问一个不存在的文件
IOException	Non_ RuntimeException	普通的 I/O 故障，例如不能从文件中读
ClassNotFoundException	Non_ RuntimeException	未找到相应类异常
EOFException	Non_ RuntimeException	文件已结束异常类
IllegalAccessException	Non_ RuntimeException	访问被拒绝时抛出的异常
SQLException	Non_ RuntimeException	操作数据库异常类
AWTException	Non_ RuntimeException	图形界面异常

【例6.2】抛出运行时异常的代码

```
class RuntimeEx{
        public static void main(String arg[]) {
                StringBuilder s[]=new StringBuilder[10];
                for(int i=0;i<=10;i++){
                        System.out.println(s[i].length());
                        }
        }
}
```

出现了两类运行时异常：**NullPointerException和ArrayIndexOutofBoundsException**

上述代码进行修改，消除上述异常。代码如下所示：

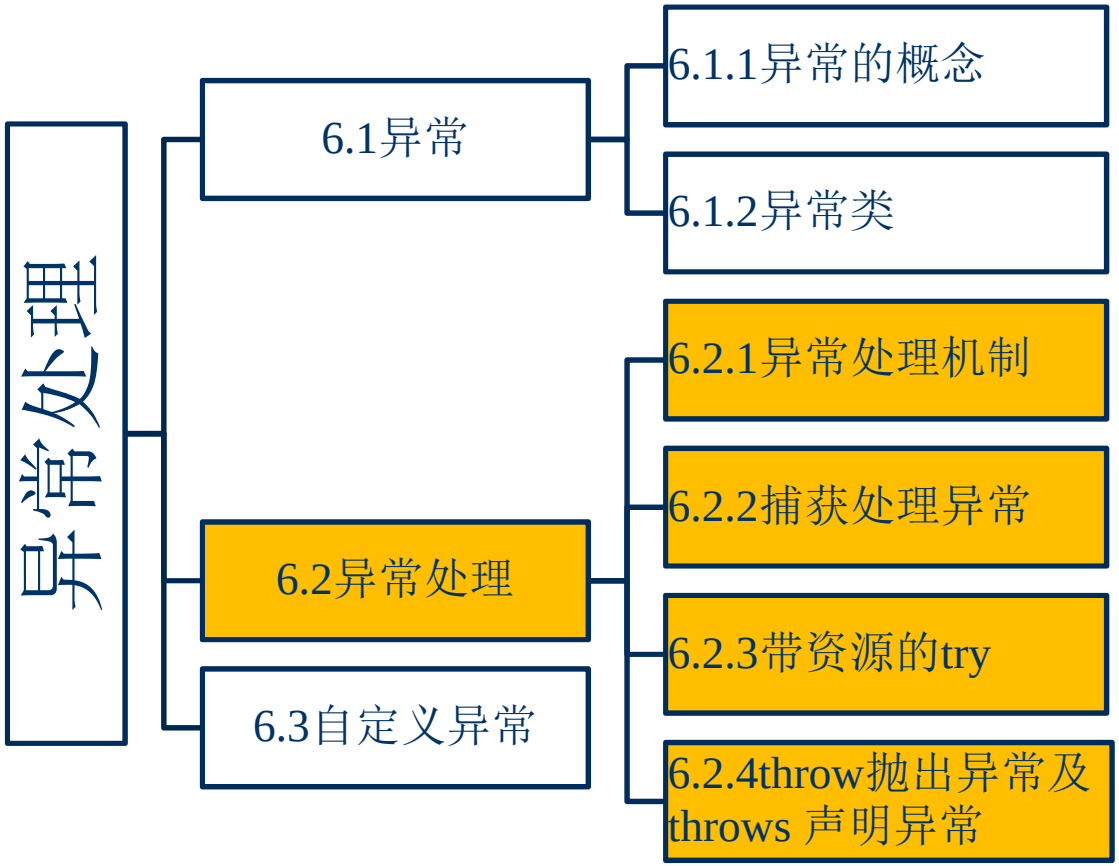
```
class RuntimeEx {
    public static void main(String arg[]) {
       StringBuilder s[]=new StringBuilder[10];
       for(int i=0;i<10;i++){
          s[i]=new StringBuilder("String"+Math.random()*10*i);
          System.out.println(s[i].length());
              }
       }
  }
```

【例6.3】受检查异常如果不处理，编译无法通过。

```
class NonRuntimeExceptionDemo{
    public static void main(String args[]){
            FileInputStream in=new FileInputStream("text.txt");
            int s;
            while((s=in.read())!=-1)
                    System.out.print(s);
            in.close();
    }
}
```

必须对异常进行用户定义的处理，否则编译无法通过。

第六章 异常处理


异常处理
6.1异常
6.1.1异常的概念
6.1.2异常类
6.2异常处理
6.2.1异常处理机制
6.2.2捕获处理异常
6.2.3带资源的try
6.2.4throw抛出异常及throws 声明异常
6.3自定义异常

6.2.1 异常处理机制

- 1、传统语言处理异常（比如C语言）

```
openFiles;
if (theFilesOpen) {
  determine the lenth of the file;
  if (gotTheFileLength){
    allocate that much memory;
    if (gotEnoughMemory) {
       read the file into memory;
       if (readFailed) errorCode=-1;
       else errorCode=1;
    }else  errorCode=-3;
  }else errorCode=-5 ;
 }else errorCode=-5;}
```

6.2.1 异常处理机制

1、传统语言处理异常（比如C语言）

- **处理方法：**

一般是用if语句进行测试，如果有错误，则返回一个特定的错误码，然后据此来处理各种错误情况。

- **存在的问题：**

1. 业务代码与异常处理代码混合在一起，其可读性低。
2. 返回的出错信息往往是一些整数编码，携带的信息量有限且标准化困难。
3. 每次调用一个方法时都进行全面细致的检查，使得程序结构极其复杂，可维护性降低，而且，当程序存在多个分支时，往往会有遗漏某些错误的情况出现。
4. 由谁来处理错误的职责不清晰。

6.2.1 异常处理机制

2、Java异常处理方法

Java提供了专门的异常处理机制。一个方法如果可能抛出（throw）异常，可以有两种处理方法：

1）声明异常

不捕获异常，而只是声明方法有可能抛出的异常，从而让该方法的上层调用方法捕获异常。

2）捕获处理异常

Java运行时系统捕获异常并找到相应异常的捕获处理代码并运行，这一过程称为捕获处理异常。如果系统找不到捕获异常代码，程序将终止。

6.2.2 捕获处理异常

下面是try-catch-finally结构的形式：

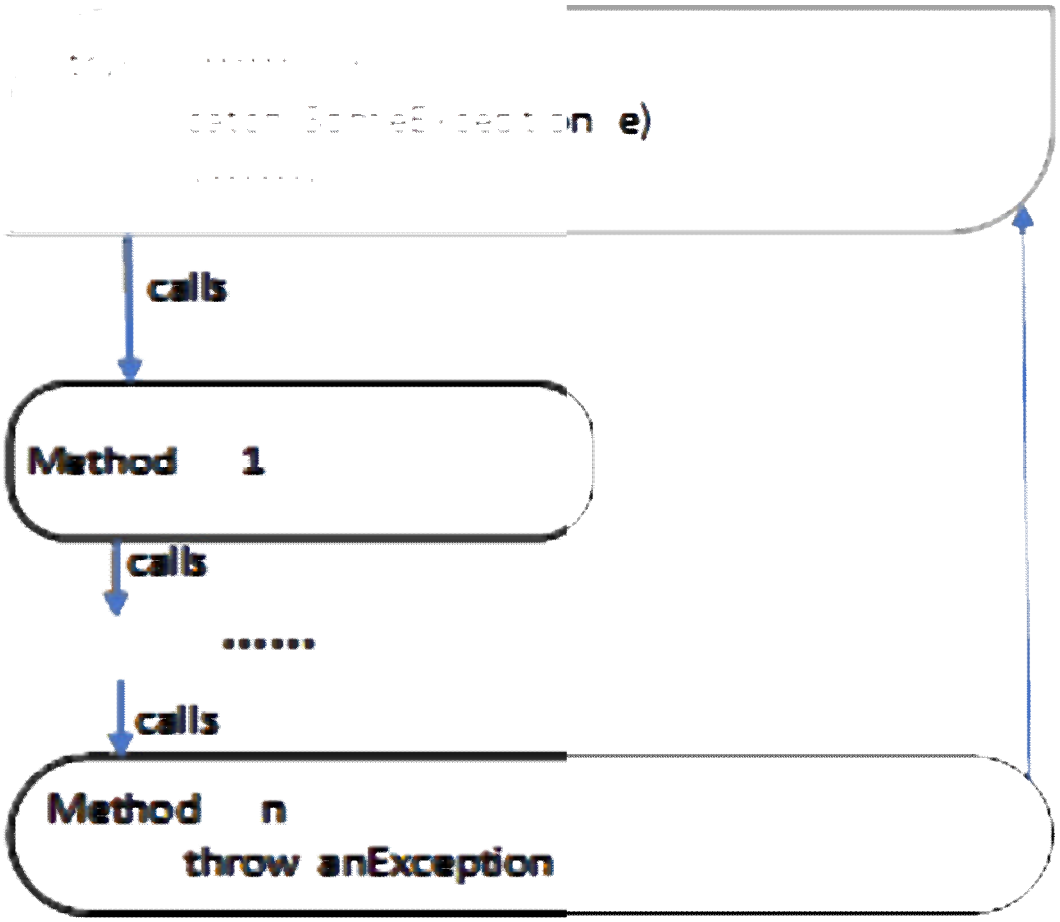
```
try{

调用可能产生异常的方法及其它语句;

}catch(异常类1 e){
      异常处理语句块;
}catch(异常类2 e){
      异常处理语句块;
}
  ...
[finally{

     最终处理 }
]
```

6.2.2 捕获处理异常



6.2.2 捕获处理异常

说明：

（1）try语句：发生异常时，将跳过try块中异常点后面的语句，且异常处理需要更多的时间和资源。因此，应当仔细分析代码，尽量缩减try代码块。

（2）catch语句可以有零个或多个，finally语句可以有零个或一个。

（3）catch语句：设计捕获异常代码要注意其顺序，按照从“特殊到一般”的顺序来设计。将子类的catch块放在前面，父类的catch块放在后面。

（4）从java7开始，多个异常可以写在一个catch中，它们之间用竖线隔开，例如：

```
try{
    ……
} catch( 异常类A |异常类B |异常类 C  ex) {
… }
```

但需要注意，用“|”操作符合并的异常不要出现互为父子的关系。

（5）finally语句是可选项。如果有该语句，无论是否捕获或处理异常，即使try或者catch块中包含break或return语句，finally块里的语句也会被执行。

finally语句一般用来在最后做一些资源回收工作，比如在try语句中打开了文件流，可以在finally中确保文件被有效关闭。

6.2.2 捕获处理异常

应用实例：需要实现如下功能：

从键盘上读取一个整数，如果读取的不是整数,请提示重新输入，直到获取正确的数字为止，最后进行除法运算。

下面给出几个例子，各实例通过调用不同的方法实现上述功能，因为不同的方法抛出的异常不同，因此每个实例处理的异常也不尽相同。

1、用System.in.read(byte[] b)输入数据

```
import java.io.IOException;
public class ReadBySystem {
    public static void main(String arg[]){
        byte[] b;
        int i=0,result=0;
        System.out.println("please input an int digital");
        while(true)
         { b=new byte[6];
           try{
1.              System.in.read(b);    //可能抛出IOException异常
              //可能抛出NumberFormatException异常
2、             i=Integer.parseInt((new String(b).trim()));
3、             result=25/i;//可能抛出ArithmeticException
                System.out.println("25/i is:"+result);
                break;
             }catch(IOException e){
                 System.out.println("io error");
                 break;
               }
           catch(NumberFormatException e1){
                   System.out.println("input an exact int digital..");
           }
           catch(ArithmeticException  e1){
                   System.out.println("a Non_zero digital");
                   }
 }  }}
```

【运行】

```
please input an int digital
3e
input an exact int digital..
35.6
input an exact int digital..
5
25/i is:5
```

2、用finally回收资源. 用Scanner（System.in）输入数据

```
public class ReadByScanner {
    public static void main(String arg[]){
        Scanner scanner=null;
        boolean flag=false;
        System.out.println("1、请输入一个数值型数据");
        while(true) {
          try{
            scanner=new Scanner(System.in);
            double tempV=0;
            tempV=scanner.nextDouble();
            System.out.println("result is:"+2.5/tempV);
            flag=true;
            break;
              }
```

```
catch(InputMismatchException e1){
                System.out.println("2、请输入数值类型");
                                                }
 catch(NoSuchElementException e2){
                System.out.println("3、请输入数据");
                                        }
  catch(IllegalStateException e3){//如果scanner被关闭了，抛出异常
                e3.printStackTrace();
                 break;
                 }
finally{
        try{
              if(flag&&scanner!=null) {
                        scanner.close();
                        System.out.print("scanner.close() in finally block");
              }
         }catch(Exception e1){
              e1.printStackTrace();
              }
          }
}// end of while
}}
```

3、多异常的处理

```
import java.util.NoSuchElementException;
import java.util.Scanner;
public class MultiException {
     public static void main(String arg[]){
        Scanner in=null;
        int tempV=0;
        try{
       in=new Scanner(System.in);
       tempV=in.nextInt();
       System.out.println(2.5/tempV);
       }
        catch(NoSuchElementException |ArithmeticException  e){
       System.out.println("请输入非0数值");}
        catch(IllegalStateException e) {
               e.printStackTrace();}
        finally{
            try{
               if(in!=null)
                   in.close();
       }catch(Exception e){}
  }}}
```

4、从异常中获取信息

方法	描述
getMessage( ):String	返回描述该异常对象的信息
toString( ):String	返回“异常类全名 +异常对象的描述信息”
printStackTrace( ):void	在控制台打印异常对象和它的调用堆栈信息
getStackTrace( ):StackTraceElement[ ]	返回和异常对象相关的堆栈跟踪元素的数组

4、从异常中获取信息

```
public class MessageException {
    public static void main(String[] args){
       try{
          double a=area(-1);
          System.out.println("圆的面积是: "+a);
       }catch(IllegalArgumentException e){
          System.out.println("1、use printStackTrace");
          e.printStackTrace();
          System.out.println("2、"+e.toString());
          System.out.println("3、"+e.getMessage());
          System.out.println("4、Trace Info ");
          StackTraceElement[] tElem=e.getStackTrace();
          for(StackTraceElement st:tElem) {
           System.out.printf("%s (%s:%d)\n", st.getMethodName(),
           st.getClassName(),
           st.getLineNumber());
                       }
       }}
     public static double area(int r){
       if(r<0)
             throw new IllegalArgumentException("radius<0!");
       else return 3.14*r*r;
       }}
```

```
1、use printStackTrace
java.lang.IllegalArgumentException: radius<0!
        at chapter6/chapter6.MessageException.area(MessageException.java:22)
        at chapter6/chapter6.MessageException.main(MessageException.java:6)
2、java.lang.IllegalArgumentException: radius<0!
3、radius<0!
4、Trace Info
area (chapter6.MessageException:22)
main (chapter6.MessageException:6)
```

6.2.3 带资源的try

Java7之后，为了简化资源清理工作，允许在try关键字后声明并初始化资源，当try语句执行结束，系统会自动调用资源的close()方法关闭这些资源，不需要再显式地用finally关闭。其基本用法如下所示：

```
try（类型名1 资源变量1= 表达式1; 类型名2  资源变量2=表达式2; …）{

 使用资源，不需要考虑关闭资源res
}
```

注意：

可以被自动关闭的资源有一个前提，这个资源的类已经实现了java.lang.AutoCloseable接口，这个接口有一个方法：void close()。

```
import java.util.*;
import java.io.*;
public class resourceTry1 {
   public static void main(String arg[]){
       int d;
       Scanner sc=new Scanner(System.in);
       try{d=sc.nextInt();
           System.out.println(25/d);

       }catch(InputMismatchException e1){
              e1.printStackTrace();
       }catch(NoSuchElementException e2){
              e2.printStackTrace();

       }catch(ArithmeticException e3){
              e3.printStackTrace();
       }finally{
         if(sc!=null)
             sc.close();
               }
  }
  }
```

带资源的try语句，使的代码更加简洁。

代码：resourceTry2.java

```
import java.util.*;
import java.io.*;
public class resourceTry2 {
    public static void main(String arg[]){
      int d;
      try(Scanner sc=new Scanner(System.in)){
            d=sc.nextInt();
            System.out.println(25/d);
      }catch(InputMismatchException  e1){
            e1.printStackTrace();

      }catch(NoSuchElementException e2){
            e2.printStackTrace();
      }catch(ArithmeticExceptione3){
            e3.printStackTrace();}
      }
}
```

6.2.4 throw抛出异常及throws声明异常

1、抛出异常

检测到错误的程序可以创建一个合适的异常对象并抛出它，这就称为抛出一个异常。

Java中有两种方法抛出异常。

（1）**Java运行时环境自动抛出异常**：系统定义的Runtime Exception类及其子类和Error都可以由系统自动抛出。

（2）**语句throw抛出异常**：用户程序想在一定条件下显式抛出异常，这必须借助于throw语句抛出。Java用throw语句抛出异常。throw语句的格式如下：

throw 异常对象

设计一个方法，输入半径求圆的面积，如果输入的半径小于0，则抛出异常。

```
import java.util.Scanner;
public class useThrow {
    public static void main(String[] args){
        System.out.println("请输入圆的半径:");
        try(Scanner input=new Scanner(System.in)){
           int radius=input.nextInt();
           double a=area(radius);
           System.out.println("圆的面积是: "+a);
        }catch(Exception e){
           System.out.println(e.getMessage());  }
        }

    public static double area(int r)throws Exception{
        if(r<0){
           throw new Exception("The radius <0");}
        else return 3.14*r*r;
        }
}
```

【运行结果】

```
请输入圆的半径:
-6
The radius can't be negative!
```

2、throws声明异常

定义方法时，如果方法可能出现异常，但该方法不想或不能自己捕获处理这种异常，那就必须在声明方法时用throws声明可能发生的异常。throws语句的格式如下：

返回类型 方法名（[参数列表]） throws 异常类1, 异常类2...
{ … //方法体}

注意：

对于不受查异常（Runtime Exception和Error），Java不要求在方法头中显示声明，但是，其它异常就一定要在方法头中显示声明。

【例】方法IOcopy对输入流和输出流做数据复制工作。在方法定义中声明可能出现的异常，并由调用方法main进行捕获处理。

```
import java.io.*;
class ThrowsDemo{
  static void IOcopy(InputStream in, OutputStream out) throws IOException {
        int s;
        while((s=in.read())!=-1) { //可能抛出IOException
                out.write(s); //可能抛出IOException
        }
    }

   public static void main(String args[]){
        try{
           IOcopy(System.in,System.out);
        }
        catch(IOException e){
            System.out.println("捕获异常: "+e);
        }
    }
}
```

【运行结果】

```
THIS IS A TEST
THIS IS A TEST
```

注意：

（1）当异常需要被方法的调用者处理时，方法应该声明异常。如果能在发生异常的方法中捕获处理异常，那么就不需要声明异常。

（2）在继承结构中，当子类方法覆盖父类方法时，子类方法声明的异常集应该属于父类的异常集，或与父类方法的异常兼容。

例如：

```
class SuperClass {
        public void method() throws IOException, ClassNotFoundException{
         //方法体
         }
        }
class SubClass extends SuperClass {
        public void method() throws FileNotFoundException {
         //方法体
         }
        }
```

上述代码成立，因为FileNotFoundException是IOException的子类。

6.3 自定义异常

编程人员有时需要在满足一定条件的情况下抛出异常。

如果现有的异常类能满足需求，这不存在问题，否则，用户需要自己定义异常类并创建对象。

异常类Exception中的常见方法

- public Exception()
- public Exception(String s):
  - 该参数一般表示该异常对应的错误的描述。
- public String toString():
  - 返回描述当前异常对象信息的字符串。
- public String getMessage():
  - 返回描述当前异常对象信息的详细信息。
- public void printStackTrace():
  - 打印当前异常对象使用栈的轨迹。

创建自定义的异常

- 定义自己的异常
  - 扩展 Throwable 或者 Exception及其子类

```
class InvalidRadiusException extends Exception{
  public InvalidRadiusException(double radius){
      //显示调用父类的带一个字符串参数的构造函数
      super("Invlid radius "+radius);
      }
}
```

- 抛出自己的异常
  - throw new aExcep (“a exception”);

举例

- 定义圆，圆的核心属性是半径，
- 在圆的构造方法中传入半径r，如果r小于等于零，则产生错误。
- 圆的构造方法可以选择输入错误时抛出输入异常，这样生成圆对象的方法可以处理这种异常。

```
//(2) 定义圆，并在其构造方法满足r<=0的情况下抛出InvalidRadiusException异常。
class Circle{
    double r;
    public Circle(double r) throws InvalidRadiusException{
            if(r<=0) {
                    throw new InvalidRadiusException(r);
            }else
                    this.r=r;
    }

    public double getArea() {
            return r*r*3.14;
    }
}
```

```
//（3）使用Circle类的主类

public class SelfException {
    public static void main(String[] args){
        //Scanner input=new Scanner(System.in);
        System.out.println("请输入圆的半径:");
        double  r;
        try(Scanner input=new Scanner(System.in)){

           r=input.nextDouble();
           Circle c=new Circle(r);
           System.out.println("圆的面积是: "+c.getArea());

        }catch(InvalidRadiusException e){
              System.out.println(e.toString());
            }
    }
}
```

小结

异常处理机制提供了一种管理程序中潜在错误的方法，并能警告你留意程序中可能发生的潜在的错误。

1、异常分类

2、try-catch-[finally] 语句，在异常发生时捕获处理它们，并由 finally 语句块来保证程序中必须执行的语句的正确执行。

3 try(资源){}catch语句，自动完成资源的回收

4、throw 语句是用来显式抛出异常。

5、throws 子句的作用是声明方法可能会抛出的异常。