



JAVA语言程序设计

---

## 第三章 类的方法

本课件在清华大学郑莉老师的课件基础上制作



# 目录

---

- 3.1 方法的控制流程
- 3.2 异常处理简介
- 3.3 方法的重载(**overloading**)
- 3.4 本章小结



## 3.1 方法的控制流程

- 方法的控制流程

- ☞ Java 程序通过控制语句来控制方法的执行流程

- ☞ Java 中的流程控制结构主要有三种

- 顺序结构

- 选择结构

- ☞ if 语句（二路选择结构）

- ☞ switch 语句（多路选择结构）

- 循环结构

- ☞ for 语句

- ☞ while 语句

- ☞ do-while 语句



## 3.1.1 if选择结构

方法的  
控制  
流程

- 语法形式

- ☞ 只有if分支，没有else分支

```
if (boolean-expression) {  
    // statement1;  
}
```

- ☞ if-else语句

```
if (boolean-expression) {  
    // statement1 ;  
}  
else {  
    // statement2 ;  
}
```

注意：如果分支中只有一个语句，则不需要花括号；否则分支中的所有语句都要用花括号括起来，构成复合语句

## 3.1.1 if选择结构(续)

方  
法  
的  
控  
制  
流  
程

☞ if-else 语句的特殊形式

```
if (boolean expression) {  
    //statement1;  
}  
else if (boolean expression) {  
    //statement2;  
}  
else if (boolean expression){  
    //statement;  
}  
...  
else {  
    //statement;  
}
```



## 3.1.1 if选择结构(续)

——ex3\_1.java

- ex3\_1

- ☞ 输入一个年份，判断它是不是闰年。
- ☞ 闰年：能被4整除但不能被100整除，或者能被400整除。

```
public class ex3_1
{
    public static void main(String[ ] args) throws IOException{
        int year;
        boolean IsLeapYear;

        System.out.println("Enter the year:");
        BufferedReader in =new BufferedReader(
            new InputStreamReader(System.in));
        year=Integer.parseInt(in.readLine());
```

方  
法  
的  
控  
制  
流  
程



## 3.1.1 if选择结构(续)

——ex3\_1.java

方  
法  
的  
控  
制  
流  
程

```
IsLeapYear=((year%4==0 && year%100 != 0)
|| (year%400 == 0));

if (IsLeapYear)
    System.out.print(year+" is a leap year" );
else
    System.out.println(year+" is not a leap year");
}
}
```



## 3.1.1 if选择结构(续)

——ex3\_2.java

- 输入两个整数比较大小

方法的控制流程

```
import java.io.*;
public class ex3_2
{
    public static void main(String[] args) throws IOException
    {
        int x,y;
        BufferedReader in = new BufferedReader(
            new InputStreamReader(System.in));
        System.out.println("Enter x and y:");
        x=Integer.parseInt(in.readLine());
        y=Integer.parseInt(in.readLine());
        if (x!=y)
            if (x>y) System.out.println("x>y");
            else System.out.println("x<y");
        else System.out.println("x=y");
    }
}
```

if (x>y)  
System.out.println("x>y");  
else if (x<y)  
System.out.println("x<y");  
else  
System.out.println("x=y");

注意：else总是与离它最近的if匹配！



## 3.1.1 if选择结构(续)

### ——以条件运算符代替if\_else

- 例子:

if (a>b)

System.out.println("The larger one is: "+a);

else

System.out.println("The larger one is: "+b);

- 用条件运算符重写:

System.out.println("The larger one is: " + (a>b)?a:b);

方法的  
控制  
流程



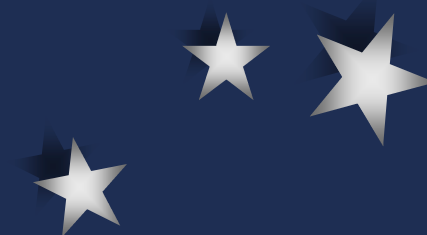
## 3.1.1 if选择结构(续)

### ——例3\_1

- 计算每个月的天数

方  
法  
的  
控  
制  
流  
程

```
static int daysInMonth(int month) {  
    if (month == 2)  
        return(28);  
    else if ((month==4)||(month==6)||(month==9)||(month==11))  
        return(30);  
    else return(31);  
}
```



## 3.1.1 if选择结构(续)

### ——ifElseDemo.java

方  
法  
的  
控  
制  
流  
程

- 已知一个学生的分数，给出其分数等级。90-100分为A级；80-89分为B级；70-79分为C级；60-69分为D级；0-59分为E级

```
public class IfElseDemo {  
    public static void main(String[] args) {  
        int testscore = 76;  
        char grade;  
        if (testscore >= 90) { grade = 'A'; }  
        else if (testscore >= 80) { grade = 'B'; }  
        else if (testscore >= 70) { grade = 'C'; }  
        else if (testscore >= 60) { grade = 'D'; }  
        else { grade = 'F'; }  
        System.out.println("Grade = " + grade);  
    }  
}
```

程序输出：  
**Grade = C**

## 3.1.2 switch选择结构

方  
法  
的  
控  
制  
流  
程

- **switch**语句是多分支的选择结构

```
switch (switch-expression) {  
    case value1: statements for case1; break;  
    case value2: statements for case2; break;  
    ...  
    case valueN: statements for caseN; break;  
    default:    statements for default case; break;  
}
```

- 注意问题

- ✎ **switch-expression**、常量值value1到valueN必须是整形或字符型
- ✎ 如果表达式的值和某个**case**后面的值相同，则从该**case**之后开始执行，直到**break**语句为止
- ✎ **default**是可有可无的，若没有一个常量与表达式的值相同，则从**default**之后开始执行

## 3.1.2 switch选择结构(续)

### ——用switch代替if

方  
法  
的  
控  
制  
流  
程

```
if (i == 1)
    { statementA(); }
else if (i == 2)
    { statementB(); }
else if ((i==3)||(i==4))
    { statementC(); }
else if (i == 5)
    { statementD(); }
else
    { statementF(); }
```

```
switch (i)
{
    case 1:
        statementA();break;
    case 2:
        statementB();break;
    case 3:
    case 4:
        statementC();break;
    case 5:
        statementD();break;
    default: statementF();
}
```

## 3.1.2 switch选择结构(续)

### ——例3\_2

方  
法  
的  
控  
制  
流  
程

- 使用**switch**结构计算每个月的天数

```
static int daysInMonth(int month) {  
    int days;  
    switch(month) {  
        case 2: days = 28; break;  
        case 4:  
        case 6:  
        case 9:  
        case 11: days = 30; break;  
        default: days = 31;  
    }  
    return(days);  
}
```



## 3.1.2 switch选择结构(续)

### ——补充ex3\_3.java

方法的  
控制  
流程

- ex3\_3

☞ 输入0~6之间的某一个整数，然后把它转换成星期 输出。(0对应星期日)

```
import java.io.*;
public class ex3_3
{
    public static void main(String[ ] args)throws IOException
    {
        int day;
        BufferedReader in =new BufferedReader(
            new InputStreamReader(System.in));
        day=(new Integer(in.readLine())).intValue();
```




## 3.1.2 switch 选择结构(续)

### ——补充ex3\_3.java

方法的  
控制  
流程

```
switch (day)
{
    case 0: System.out.println("Sunday"); break;
    case 1: System.out.println("Monday"); break;
    case 2: System.out.println("Tuesday"); break;
    case 3: System.out.println("Wednesday"); break;
    case 4: System.out.println("Thursday"); break;
    case 5: System.out.println("Friday"); break;
    case 6: System.out.println("Saturday"); break;
    default:
        System.out.println("Day out of range Sunday ..Saturday" );
        break;
}
}
```





## 循环结构解题

- 很多题目都需要循环结构进行求解。
- 当一时难以整理出每次循环（迭代）所做的事情时，可以先看一下如果这件事情交给别人做的话，一步一步是怎么做的。
- 在上一步基础上抽象出循环结构的四个方面。
- 在4中，要对出现在2和3中的某些变量进行修改，为下次循环做好准备，并使得循环能最终结束。
- 2和3一般没有绝对的先后顺序。
- 在分析清楚2和3后，才分析4（为什么？）。
- 一般将1放在最后分析

1-循环初始化

2-循环条件

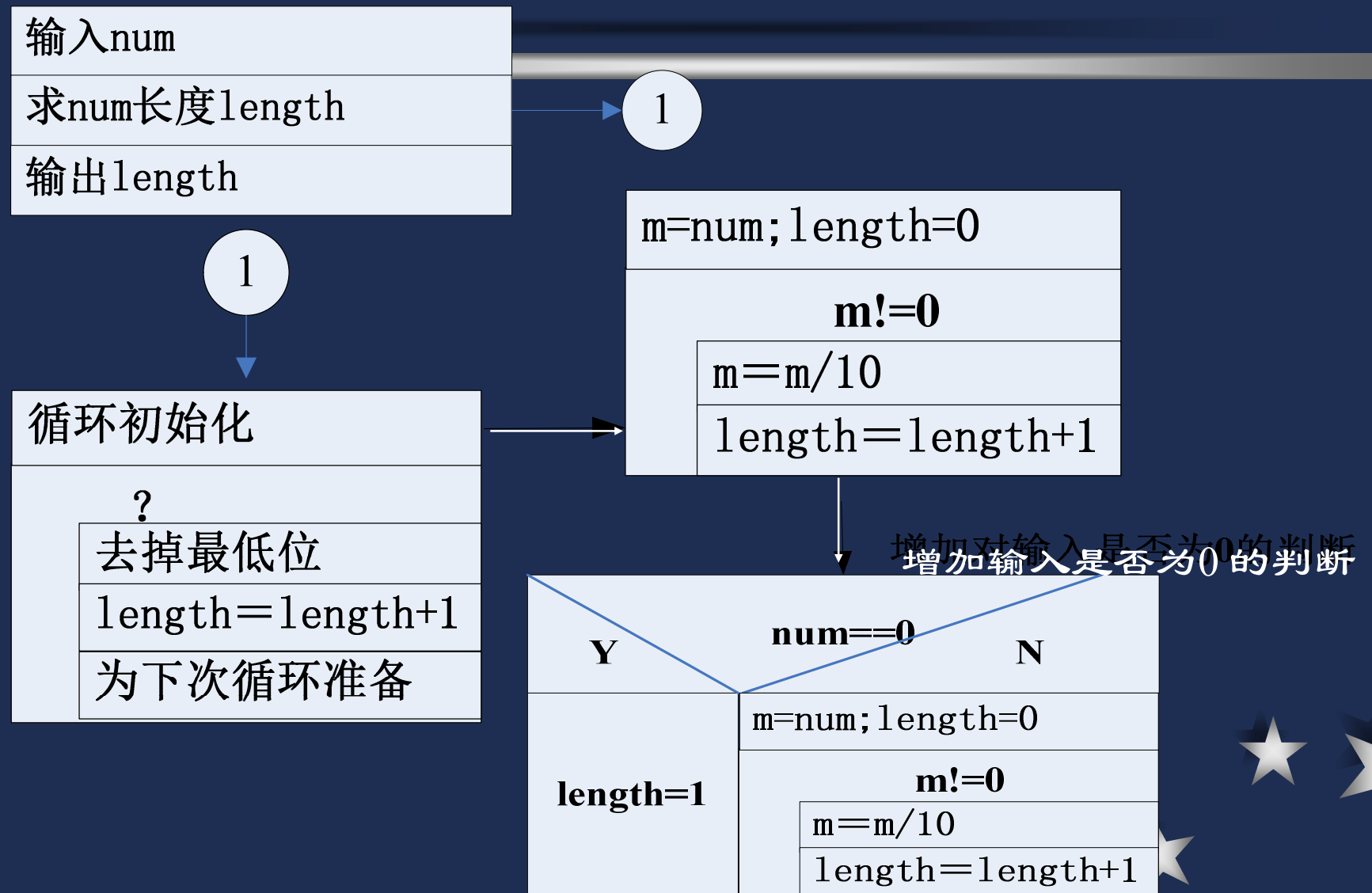
3-本次循环本职工作

4-下次循环的准备工作

## 循环结构解题

- 设计程序，求一个正整数的长度。
- 例如num=12345，求num的长度
- 先看看人是怎么计算长度的：
  - ✎ 第1步：去掉最低位，num=1234，长度len=1
  - ✎ 第2步：去掉最低位，num=123，长度len=2
  - ✎ 第3步：去掉最低位，num=12，长度len=3
  - ✎ 第4步：去掉最低位，num=1，长度len=4
  - ✎ 第5步：去掉最低位，num=0，长度len=5

算法中每一步的核心操作是从num中去掉最低位，长度len加1



## 3.1.4 while 循环结构

方法的  
控制  
流程

- while 语句

- ☞ 实现“当型”循环，其一般语法格式如下：

```
while (check-expression) {  
    //body of the loop;  
}
```

- ☞ 解释

- 条件表达式(check-expression)的返回值为布尔型
    - 循环体可以是单个语句，也可以是复合语句块

- ☞ 执行过程

- 先判断check-expression的值，为真则执行循环体
    - 循环体执行完后再无条件转向条件表达式做计算与判断；当计算出条件表达式的值为假时，跳过循环体执行while语句后面的语句。若为真，则继续执行循环

## 例1：设计程序，求一个正整数的长度。

```
import java.io.*;
public class testLoop {
    public static void main(String[] args)
        throws IOException{
        BufferedReader in = new BufferedReader(
                                new InputStreamReader(System.in));

        int num,m;//存放输入的整数
        int length;//存放长度
        System.out.println("请输入一个正整数");
        num = Integer.parseInt(in.readLine()); //读取一个整数
        length = 0; m = num;
        while(m!=0){
            m= m / 10;
            length++;
        }
        System.out.println("正整数"+ num+"的长度是"+length);
    }
}
```



## 3.1.4 while 循环结构(续)

方法的  
控制  
流程

- 例2：循环接受并输出从键盘输入的字符，直到输入的字符为回车为止

```
char ch;  
  
ch= (char)System.in.read(); // 接收键盘输入  
while (ch!='\n'){  
    System.out.println(ch);  
    ch= (char)System.in.read();  
}
```



## 3.1.4 while循环结构(续)

### ——补充ex3\_4.java

方  
法  
的  
控  
制  
流  
程

- 例3：计算数列1,2,...,10 的和。

```
public class ex3_4
{
    public static void main(String[ ] args)
    {
        int i=1, sum=0;
        while(i<=10) {
            sum+=i;
            i++;
        }
        System.out.println("sum="+sum);
    }
}
```



## 编写一个程序，求一个正整数序列(以-1结束)的和、最小值和最大值

```
import java.io.*;
public class A1
{
    public static void main(String[ ] args) throws IOException
    {
        int max,min,num, sum;
        BufferedReader in = new BufferedReader
            (new InputStreamReader(System.in));
        System.out.println("请输入一批正整数，以-1结束：");

        //读取第一个正整数
        num = Integer.parseInt(in.readLine());
        max = min = sum = num;
```





## 编写一个程序，求一个正整数序列(以-1结束)的和、最小值和最大值（续）

```
//读取后续的正整数
num = Integer.parseInt(in.readLine());
while(num != -1){
    sum = sum + num;
    if (num > max)
        max = num;
    else if(num < min)
        min = num;
    num = Integer.parseInt(in.readLine());
}
System.out.println("最大值是 : "+ max);
System.out.println("最小值是 : "+ min);
System.out.println("和是 : "+ sum);
}
```



## 3.1.5 do-while循环结构

方  
法  
的  
控  
制  
流  
程

- do-while语句

- ✎ 实现“直到型”循环

- ✎ 一般语法结构如下

```
do {  
    //body of the loop;  
} while (check-expression);
```

- ✎ 其使用与while语句很类似，不同的是它首先无条件的执行一遍循环体，再来判断条件表达式的值，若表达式的值为真，则再运行循环体，否则跳出do-while循环，执行下面的语句

- ✎ 特点：它的循环体至少要执行一次



## 3.1.5 do-while循环结构(续)

⑩ 比较这两段程序

方  
法  
的  
控  
制  
流  
程

```
int i=1, sum=0;
while(i<=10)
{
    sum+=i;
    i++;
}
System.out.println
    ("sum="+sum);
```

```
int i=1, sum=0;
do {
    sum+=i;
    i++;
} while(i<=10);
System.out.println
    ("sum="+sum);
```

总结：只要循环体至少要执行一次，则用while和do-while结构效果是等同的

## 3.1.5 do-while循环结构(续)

### ——补充ex3\_5.java

方  
法  
的  
控  
制  
流  
程

- 输入一个整数，然后输出它的翻转形式。如124的翻转形式为421。

```
import java.io.*;
public class ex3_5
{
    public static void main(String[] args)throws IOException{
        int n, right_digit, newnum = 0;
        BufferedReader in = new BufferedReader(
            new InputStreamReader(System.in));
        System.out.println("Enter the number: ");
        n=Integer.parseInt(in.readLine());
        System.out.print("The number in reverse order is ");
        do {
            right_digit = n % 10;
            System.out.print(right_digit);
            n /= 10;
        } while (n != 0);
        System.out.println();
    }
}
```



## ● for语句的执行过程

- 首先根据初始表达式`start-expression`，完成必要的初始化工作；再判断表达式`check-expression`的值，若为真，则执行循环体
- 执行完循环体后再返回表达式`update-expression`，计算并修改循环条件，这样一轮循环就结束了
- 第二轮循环从计算并判断表达式`check-expression`开始，若表达式的值仍为真，则循环继续，否则跳出整个for语句执行for循环下面的句子

## 万 法 的 控 制 流 程

一个

for循环可以嵌套

一般语法格式如下

```
for (start-expression; check-expression; update-expression) {  
    //body of the loop;  
}
```

解释

- `start-expression`完成循环变量和其他变量的初始化工作
- `check-expression`是返回布尔值的条件表达式，用于判断循环是否继续
- `update-expression`用来修整循环变量，改变循环条件
- 三个表达式之间用分号隔开



## 3.1.3 for循环结构(续)

- 例3：计算数列1,2,...,10 的和。使用For循环结构

```
public class ex3_4
{
    public static void main(String[ ] args)
    {
        int i, sum=0;
        for(i = 1; i <= 10; i++)
            sum+=i;
        System.out.println("sum="+sum);
    }
}
```



## 3.1.3 for循环结构(续)

- 打印九九乘法表，要求结果如下：

1\*1=1

2\*1=2 2\*2=4

3\*1=3 3\*2=6 3\*3=9

4\*1=4 4\*2=8 4\*3=12 4\*4=16

5\*1=5 5\*2=10 5\*3=15 5\*4=20 5\*5=25

6\*1=6 6\*2=12 6\*3=18 6\*4=24 6\*5=30 6\*6=36

7\*1=7 7\*2=14 7\*3=21 7\*4=28 7\*5=35 7\*6=42 7\*7=49

8\*1=8 8\*2=16 8\*3=24 8\*4=32 8\*5=40 8\*6=48 8\*7=56 8\*8=64

9\*1=9 9\*2=18 9\*3=27 9\*4=36 9\*5=45 9\*6=54 9\*7=63 9\*8=72 9\*9=81

for (int i=1; i<=9; i++)

打印第i行共i个式子;



for (int j=1; j<=i; j++)

打印第j个式子: i\*j;

## 3.1.3 for循环结构(续)

方  
法  
的  
控  
制  
流  
程

- 打印九九乘数表

```
public class MultiTable {  
    public static void main(String[] args){  
        for (int i=1; i<=9; i++) {  
            for (int j=1; j<=i; j++)  
                System.out.println(" "+i+"*"+j+"="+i*j);  
            System.out.println();  
        }  
    }  
}
```





## 3.1.3 for循环结构(续)

### ——逗号运算符

- 逗号运算符

✎ 可用在 **for** 循环控制表达式的初始化和递增两部分。在这两部分中可以存在多个由逗号分隔的语句，这些语句会被依次计算

```
public class ex3_8
{
    public static void main(String[] args) {
        for(int i = 1, j = i + 10; i < 5; i++, j = i * 2) {
            System.out.println("i= " + i + " j= " + j);
        }
    }
}
```



## 3.1.6 break语句

方法的  
控制  
流程

- 功能
  - ✎ 跳出循环，不再执行剩余部分
- 适用场合
  - ✎ 在switch 结构中，用来终止switch语句的执行
  - ✎ 在for循环及while循环结构中，用于终止break语句所在的最内层循环；与标号一同使用时，将跳出标号所标识的循环
  - ✎ 也可用在代码块中，用于跳出它所指定的块

## 3.1.6 b

- 运行结果

1 2 3 4

Broke out of loop at i = 5

- 解释：执行**break**语句时，程序流程跳出**for**循环

- 简单**break**应用举例

```
public class BreakTest {  
    public static void main( String args[] ) {  
        String output = "";  
        int i;  
        for ( i = 1; i <= 10; i++ ) {  
            if ( i == 5 ) break; // break loop only if count == 5  
            output += i + " ";  
        }  
        output += "\nBroke out of loop at i = " + i;  
        System.out.println(output);  
    }  
}
```



## 3.1.6 break语句(续)

### ——例3\_6

- 在嵌套循环中使用**break**语句：使用下面的程序来实现例3-4的九九乘法表

方  
法  
的  
控  
制  
流  
程

```
public class Examp3_6{
    public static void main(String[] args){

        for (int i=1; i<=9;i++) {
            for (int j=1; j<=9;j++){
                if (j > i) break;
                System.out.print(" "+i+"*"+j+"="+i*j);
            }
            System.out.println();
        }
    }
}
```



## 3.1.6 break语句(续)

- **break与label一同使用**

- ✧ 格式如下

- `break label;`

- ✧ 标号**label**应该定义在程序中某一循环语句前面，用来标志这个循环结构

- ✧ 作用：跳出标号标识的循环结构



- 运行结果

1\*1=1

2\*1=2 2\*2=4

3\*1=3 3\*2=6 3\*3=9

4\*1=4 4\*2=8 4\*3=12 4\*4=16

5\*1=5 5\*2=10 5\*3=15 5\*4=20 5\*5=25

- 说明:第一个**break**语句跳出内层循环;第二个**break outer**语句则跳出标号**outer**所标识的循环,即外重循环

```
public static void main(String[] args){  
    outer:  
    for (int i=1; i<=9;i++) {  
        for (int j=1; j<=9;j++){  
            if (j > i) break;  
            if (i==6) break outer;  
            System.out.print(" "+i+"*"+j+"="+i*j);  
        }  
        System.out.println();  
    }  
}
```

## 3.1.7 continue语句

方  
法  
的  
控  
制  
流  
程

- continue语句

- ✧ 必须用于循环结构中
- ✧ 停止本次迭代，回到循环起始处，开始下一次迭代过程
- ✧ 有两种使用格式
  - 不带标号的continue语句
    - ✧ 终止当前这一轮的循环，跳出本轮循环剩余的语句，直接进入当前循环的下一轮
  - 带标号的continue语句
    - ✧ 使程序的流程直接转入标号标明的循环层次



## 3.1.7 continue语句(续)

### ——不带标号的continue语句

方  
法  
的  
控  
制  
流  
程

- 不带标号的continue语句

- ✎ 在while或do-while循环中，会使流程直接跳转至条件表达式

- ✎ 在for循环中，会使流程跳转至表达式  
update-expression，计算并修改循环变量后再判断循环条件





- 运行结果

1 2 3 4 6 7 8 9 10

Using continue to skip printing 5

i = 11

- 说明: **continue** 语句并没有跳出循环体, 而是跳过本次循环, 进入下一轮循环

- 简单的 continue 循环

```
public class ContinueTest{  
    public static void main( String args[] ) {  
        String output = "";  
        int i;  
        for ( i = 1; i <= 10; i++ ) {  
            if ( i == 5 ) continue; // skip remaining code in this loop  
            output += i + " ";  
        }  
        output += "\nUsing continue to skip printing 5";  
        output += "\ni = " + i;  
        System.out.println(output);  
    }  
}
```

## 3.1.7 continue语句(续)

### ——带标号的continue语句

方  
法  
的  
控  
制  
流  
程

- 带标号的continue语句

- ✎ 格式如下

- continue label;

- ✎ 标号label应该定义在程序中某一循环语句前面，用来标志这个循环结构

- ✎ 作用：使程序的流程直接转入标号标明的循环层次



## 3.1.7 continue语句(续)

### ——例3\_10

- 九九乘法表也可用下面的程序来实现

```
public class Examp3_10{
    public static void main (String args[]) {
        outer:
        for (int i=1; i<10; i++){
            inner:
            for (int j=1; j<10; j++){
                if (i<j){
                    System.out.println();
                    continue outer;
                }
                System.out.print(" "+i+"*"+j+"="+i*j);
            }
        }
    }
}
```

- 当执行到满足条件 $i < j$ 时，跳出inner循环，直接跳到outer循环，计算并修改i的值，进行下一轮的循环

---

```
package exceptions;
public class TestException {
    public static int getInteger(){
        byte[] buffer = new byte[512];
        System.in.read(buffer);
        String s = new String(buffer);
        s=s.trim();
        return Integer.parseInt(s);
    }
}
```

该类无法通过编译。System.in的read方法：  
public int read(byte[] b) throws IOException



## 3.2 异常处理简介

---

- 异常处理

- ✧ 在进行程序设计时，错误的产生是不可避免的。所谓错误，是在程序运行过程中发生的异常事件，这些事件的发生将阻止程序的正常运行
- ✧ 如何处理错误？把错误交给谁去处理？程序又该如何从错误中恢复？
- ✧ 为了加强程序的鲁棒性，Java语言具有特定的运行错误处理机制



## 3.2 异常处理简介

考虑下面的伪代码

**Perform a task**

**If the proceeding task did not execute correctly**

**Perform error processing**

**Perform next task**

**If the proceeding task did not execute correctly**

**Perform error processing**

...

存在问题:

1. 程序逻辑与错误处理一起混用使得程序难以阅读、修改、维护和调试
2. 如果潜在的问题极少发生，则程序逻辑与错误处理一起混用将会降低程序的性能，因为程序必须测试该错误处理逻辑。

## 3.2 异常处理简介

- Java的异常处理

```
try {  
    //Perform a task  
    result = number1 / number2;  
    //Perform next task  
}  
catch (ArithmeticException e) { //如果产生算术类异常  
    //处理异常  
}  
catch (Exception e) { //如果产生其他异常  
    //处理异常  
}
```

把有风险的代码放到try块中

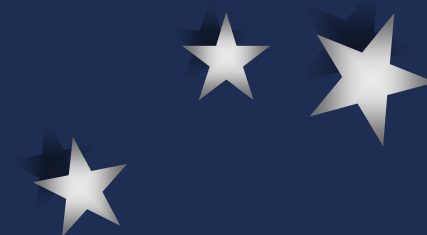
用catch块摆放异常状态的处理程序

程序逻辑和错误处理代码分开，一目了然！

## 3.2.1 异常处理的意义

### 异常处理简介

- 异常的基本概念
  - ✎ 又称为例外，是特殊的运行错误对象
  - ✎ Java中声明了很多异常类，每个异常类都代表了一种运行错误，类中包含了
    - 该运行错误的信息
    - 处理错误的方法
  - ✎ 每当Java程序运行过程中发生一个可识别的运行错误时，即该错误有一个异常类与之相对应时，系统都会产生一个相应的该异常类的对象，即产生一个异常






## 3.2.1 异常处理的意义

---

```
public class Area {  
  
    public static void main(String[] args) {  
  
        BufferedReader in= new BufferedReader(new  
            InputStreamReader(System.in));  
        float itsRadius,itsArea;  
  
        System.out.println("请输入半径: ");  
        itsRadius = Float.parseFloat(in.readLine());//获取半径  
  
        itsArea = itsRadius * itsRadius * 3.14159f;  
        System.out.println("圆的面积是: \n"+itsArea);  
    }  
}
```

该类无法通过编译，readLine调用处提示信息：Unhandled exception type IOException。BufferedReader 中readLine原型：  
public String readLine() throws IOException



## 3.2.1 异常处理的意义

- `BufferedReader` 的 `readLine` 执行时，可能会产生异常，如读取数据时文件被损坏或者失去网络连接。当发生异常时，一定会创建一个异常类的对象。
- `IOException` 就是一个异常类，若在使用标准输入输出设备时产生 IO 异常，则运行时系统会创建该异常对象。Java 的设计者们决定，不在 `readLine` 方法中处理该异常类的对象，而最好是在方法的调用者中处理它。在具体实现上，将该错误发送给方法的调用者(此例中为 `main` 方法)

指示该方法的调用者  
可能会接收到一个异常  
类对象

指示该方法  
生成的异常  
类型

```
public String readLine() throws IOException
```

## 3.2.1 异常处理的意义

---

- 通常，有两种处理异常的方法：
  - ❧ 1. 在方法内处理异常
  - ❧ 2. 将异常传到方法外
- **readLine**的开发者决定将异常传到方法外，因此**main**方法必须决定如何处理异常。目前**main**方法决定也将异常传递到方法外。



## 3.2.1 异常处理的意义

```
public class Area {  
  
    public static void main(String[] args) throws IOException{  
  
        BufferedReader in= new BufferedReader(new  
            InputStreamReader(System.in));  
        float itsRadius,itsArea;  
  
        System.out.println("请输入半径: ");  
        itsRadius = Float.parseFloat(in.readLine());//获取半径  
  
        itsArea = itsRadius * itsRadius * 3.14159f;  
        System.out.println("圆的面积是: \n"+itsArea);  
    }  
}
```

以这种方式处理异常并不是理想做法，这种方法不断地将异常往外抛，直到从程序中抛出到操作系统。



## 3.2.1 异常处理的意义(续)

### 异常处理简介

- java处理异常的方法

- ✧ 抛出(throw)异常

- 在方法的运行过程中，如果发生了异常，则该方法生成一个代表该异常的对象并把它交给运行时系统，运行时系统便寻找相应的代码来处理这一异常

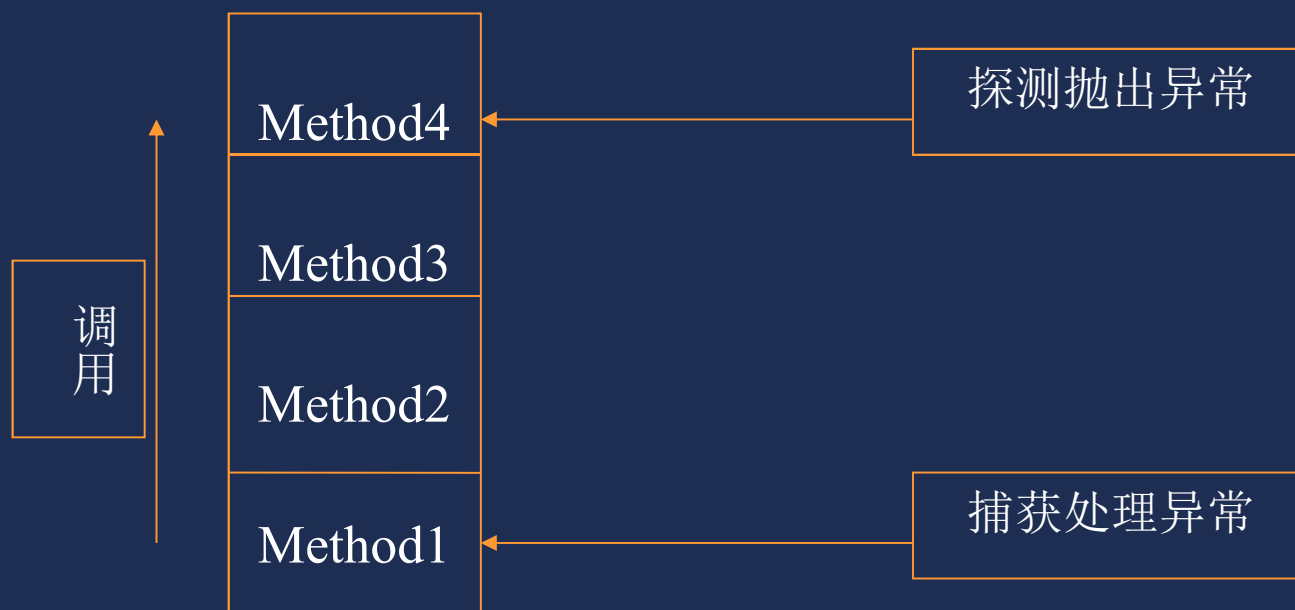
- ✧ 捕获(catch)异常

- 运行时系统在方法的调用栈中查找，从生成异常的方法开始进行回溯，直到找到包含相应异常处理的方法为止

## 3.2.1 异常处理的意义(续)

### ——异常处理示意图

异常处理简介



调用栈：方法1调用方法2，方法2调用方法3，方法3调用方法4。

方法4中产生一个异常，假设方法4对该异常不进行处理，则看方法3中会否对该异常进行处理；如方法3不处理，则看方法2会否处理，如此不断沿方法调用栈回溯。本例中异常在方法1中被处理。



## 3.2.1 异常处理的意义(续)

### 异常处理简介

- **Java异常处理机制的优点**
  - ✎ 将错误处理代码从常规代码中分离出来，提高了程序的可读性和可修改性
  - ✎ 按错误类型和差别分组，程序员可以决定处理他们所选择的任何异常-所有的异常、某种特定类型的所有异常或一组相关类型的异常（如属于同一集成层次结构的异常类型）
  - ✎ 对无法预测的错误的捕获和处理



## 3.2.1 异常处理的意义(续)

- **Java异常处理机制的优点（续）**

☞ 简化软件组合：应用程序中通常由应用程序专用组件调用可重用的预定义组件。当预定义组件出现问题时需要一种将问题汇报给应用程序专用组件的机制，预定义组件事先不可能知道应用程序如何处理发生的问题。异常处理使得预定义组件能够将问题通报给应用程序专用组件，然后应用程序专用组件再以应用程序特有的方式来处理问题，简化软件的组合。





## 3.2.2 错误的概念

异常处理简介

- 错误

- ✧ 程序运行过程中发生的异常事件
- ✧ 根据错误的严重程度不同，可分为两类

- 错误

- ✧ 致命性的，用户程序无法处理
    - ✧ `Error`类是所有错误类的父类

- 异常

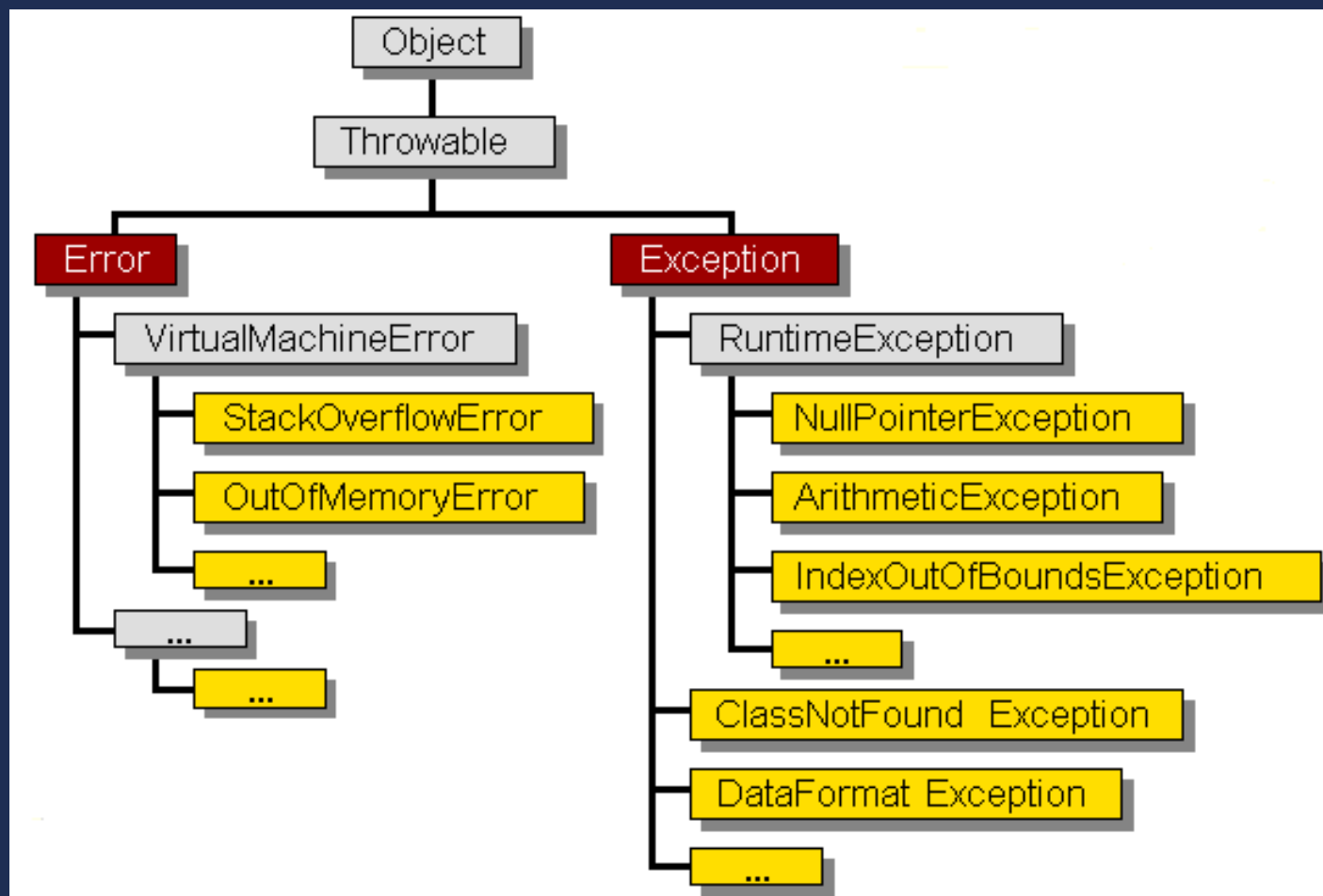
- ✧ 非致命性的，可编制程序捕获和处理
    - ✧ `Exception`类是所有异常类的父类



## 3.2.2 错误的概念(续)

### ——异常和错误类的层次结构

异常处理简介



## 3.2.2 错误的概念(续)

### 异常处理简介

- 再对异常进行分类

- ✎ 非检查型异常

- 不能期望程序捕获的异常(例如数组越界, 除零等)
- 均继承自 RuntimeException
- 在方法中不需要声明, 编译器也不进行检查

- ✎ 检查型异常

- 其他类型的异常
- 如果被调用的方法抛出一个类型为 E 的检查型异常, 那么调用者必须捕获 E 或者也声明抛出 E (或者 E 的一个父类), 对此编译器要进行检查

## 3.2.2 错误的概念(续)

- 问：为什么编译器不管那些运行时异常？它们不会让程序死掉么？
- 答：大部分的运行时异常都是因为程序书写的问题（如除零错误、数据越界访问），而非我们无法预测和防止的问题（如文件不存在、服务器死机）。程序书写的问题在测试阶段就应该被发现，**try/catch**块是用来处理真正的异常。



## 3.2.2 错误的概念(续)

### ——预定义的一些常见异常

异常处理简介

- Java预定义的一些常见异常

- ✧ ArithmeticException

- 整数除法中除数为0

- ✧ NullPointerException

- 访问的对象还没有实例化

- ✧ NegativeArraySizeException

- 创建数组时元素个数是负数

- ✧ ArrayIndexOutOfBoundsException

- 访问数组元素时，数组下标越界

- ✧ ArrayStoreException

- 程序试图向数组中存取错误类型的数据

- ✧ FileNotFoundException

- 试图存取一个并不存在的文件

- ✧ IOException

- 通常的I/O错误

非检查型异常

检查型异常

## 3.2.2 错误的概念(续)

### ——例3\_11

- 测试系统定义的运行异常——数组越界出现的异常

异常  
处理  
简介

```
import java.io.*;
public class HelloWorld {
    public static void main (String args[ ]) {
        int i = 0;
        String greetings [ ] = {"Hello world!", "No, I mean it!",
                                "HELLO WORLD!!"};

        while (i < 4) {
            System.out.println (greetings[i]);
            i++;
        }
    }
}
```



## 3.2.2 错误的概念(续)

### ——例3\_11运行结果

异常处理简介

- 运行结果

Hello world!

No, I mean it!

HELLO WORLD!!

Exception in thread "main"

java.lang.ArrayIndexOutOfBoundsException

at HelloWorld.main(HelloWorld.java:7)

- 说明

❧ 访问数组下标越界，导致ArrayIndexOutOfBoundsException 异常

❧ 该异常是系统定义好的类，对应系统可识别的错误，所以Java虚拟机会自动中止程序的执行流程，并新建一个该异常类的对象，即抛出数组出界异常

## 3.2.3 异常的处理

异常处理简介

- 对于检查型异常，Java强迫程序必须进行处理。处理方法有两种：

- ✧ 声明抛出异常

- 不在当前方法内处理异常，而是把异常抛出到调用方法中——偷懒！！

- ✧ 捕获异常

- 在当前方法内使用try{}catch(){}块，捕获到所发生的异常，并进行相应的处理——负责！





## 3.2.3 异常的处理(续)

### 方式1——声明抛出异常

#### 异常处理简介

- 声明抛出异常

- ❧ 如果程序员不想在当前方法内处理异常，可以使用throws子句声明将异常抛出到调用方法中。
- ❧ 如果方法抛出了并未在其throws子句中列出的检查型异常，则是编译错误。
- ❧ 子类方法重写超类某个方法时，子类方法的throws子句中列出的异常只能是重写的超类方法中列出的异常的子集。
- ❧ 如果所有的方法都选择了抛出此异常，最后 JVM 将捕获它，输出相关的错误信息，并终止程序的运行。在异常被抛出的过程中，任何方法都可以捕获它并进行相应的处理。
- ❧ 如果在一个方法中抛出了异常但是却没有捕获，则该方法终止执行，控制将返回到最初调用该方法的语句。

## 3.2.3 异常的处理(续)

### ——一个例子

#### 异常处理简介

```
public void openThisFile(String fileName)
    throws java.io.FileNotFoundException {
    //code for method
}

public void getCustomerInfo()
    throws java.io.FileNotFoundException {
    // do something
    this.openThisFile("customer.txt");
    // do something
}
```

openThisFile可能会产生异常，因为它选择抛出异常，所以调用者getCustomerInfo会收到该异常，其要不进行处理，要不继续抛出

- 如果在openThisFile中抛出了FileNotFoundException异常，getCustomerInfo将停止执行，并将此异常传送给它的调用者



```
public class UsingException {  
    public static void main(String args[]) {  
        try {  
            throwException();
```

Method throw Exception  
finally is always executed  
Exception handled in main

```
        }  
        catch(Exception exception){  
            System.err.println("Exception handled in main");  
        }  
    }  
}
```

## 方式2——捕获异常

```
public static void throwException() throws Exception  
{
```

```
    try{  
        System.out.println("Method throw Exception");  
        throw new Exception();//创建并抛出异常  
    }  
    catch(RuntimeException runtimeException){  
        System.err.println("Exception handled in method  
        throwException");  
    }  
    finally{  
        System.err.println("finally is always executed");  
    }  
}
```

throwException方法选择对  
Exception异常不  
进行处理，将其抛  
给调用者main。  
main对异常进行  
捕获处理。



## 3.2.3 异常的处理(续)

### 方式2——捕获异常

异常  
处理  
简介

- 语法格式

```
try {  
    statement(s) //可能抛出异常  
} catch (exceptiontype1 name1) {  
    statement(s) //处理异常  
} catch (exceptiontype2 name2) {  
    statement(s) //处理异常  
}  
finally {  
    statement(s)  
}
```



## 3.2.3 异常的处理

### 异常处理简介

#### ● 说明

##### ☞ try 子句

- 其后跟随可能产生异常的代码块
- 产生异常后，位于try内的产生异常的代码块之后的代码不会再运行

##### ☞ catch子句

- 其后跟随异常处理语句，通常用到两个方法
  - ☞ getMessage() – 返回一个字符串对发生的异常进行描述。
  - ☞ printStackTrace() – 给出方法的调用序列，一直到异常的  
产生位置，通常在测试和调试时有用
- 在异常发生时，程序将查找第1个能处理异常的catch子句：抛出的异常类型和异常参数类型相同或者是后者的子类
- catch子句处理完毕后，程序将忽略任何与try块相关的其他子句，并从try/catch后的第1行代码开始运行。

```
try {  
    statement(s) //可能抛出异常  
}  
catch (exceptiontype1 name1) {  
    statement(s) //处理异常  
}
```



## 3.2.3 异常的

### finally子句

```
try {  
    statement(s) //可能抛出异常  
} catch (exceptiontype1 name1) {  
    statement(s) //处理异常  
} finally {  
    statement(s)  
}
```

- 不论在try代码段是否产生异常，finally 子句的程序代码段都会被执行。通常在这里释放相应的try块所获取的内存以外的其他资源
- 如果使用return、break或continue语句退出try块，java也将保证执行finally子句,然后再回到return、break或continue。
- finally 子句执行后，程序控制将从finally 子句后的第1条语句开始执行。



## 3.2.3 异常的处理(续)

### ——捕获异常

- 代码执行路径

```
try {  
    statement1(s) //有风险代码  
    statement2(s) //有风险代码  
} catch (exceptiontype1 name) {  
    statement3(s) //处理异常  
} catch (exceptiontype2 name) {  
    statement4(s) //处理异常  
} finally {  
    statement5(s)  
}  
// 异常处理完了，此处代码继续运行  
statement5(6)
```

```
try {  
    statement1(s)  
    statement2(s)  
} catch (exceptiontype1 name) {  
    statement3(s) //处理异常  
} catch (exceptiontype2 name) {  
    statement4(s) //处理异常  
} finally {  
    statement5(s)  
}  
statement6(s)
```

若try块中没有抛出异常，则执行statement1(s)、statement2(s)、statement5(s)和statement6(s)

## 3.2.3 异常的处理(续)

### ——捕获异常

- 代码执行路径

```
try {  
    statement1(s) //有风险代码  
    statement2(s) //有风险代码  
} catch (exceptiontype1 name) {  
    statement3(s) //处理异常  
} catch (exceptiontype2 name) {  
    statement4(s) //处理异常  
} finally {  
    statement5(s)  
}  
// 异常处理完了，此处代码继续运行  
statement6(s)
```

```
try {  
    statement1(s)  
    statement2(s)  
} catch (exceptiontype1 name) {  
    statement3(s) //处理异常  
} catch (exceptiontype2 name) {  
    statement4(s) //处理异常  
} finally {  
    statement5(s)  
}  
Statement6(s)
```

若statement1(s)处抛出了异常，且抛出的异常类型与exceptiontype1匹配，则跳出try块转去执行statement3(s)，然后执行statement5(s)和statement6(s)



## 3.2.3 异常的处理(续)

### ——捕获异常

- 代码执行路径

```
try {  
    statement1(s) //有风险代码  
    statement2(s) //有风险代码  
} catch (exceptiontype1 name) {  
    statement3(s) //处理异常  
} catch (exceptiontype2 name) {  
    statement4(s) //处理异常  
} finally {  
    statement5(s)  
}  
// 异常处理完了，此处代码继续运行  
statement6(s)
```

```
try {  
    statement1(s)  
    statement2(s)  
} catch (exceptiontype1 name) {  
    statement3(s) //处理异常  
} catch (exceptiontype2 name) {  
    statement4(s) //处理异常  
} finally {  
    statement5(s)  
}  
Statement6(s)
```

若statement1(s)处抛出了异常，则跳出try块，但未找到与该异常匹配的catch块，则执行statement5(s)，然后异常被抛出该方法。

---

```
public class TestTryCatchFinally {
    public static void main(String[] args) {
        try
        {
            System.out.println("Start try");
            String[] colours={"RED","BLUE","GREEN"};
            System.out.print("which color?(1,2,3) : ");
            String pos =EasyScanner.nextString();

            // next line could throw NumberFormatException
            int i = Integer.parseInt(pos);
            // next line could throw
            // ArrayIndexOutOfBoundsException
            System.out.println(colours[i-1]);
            System.out.println(" End Try");

        }
    }
}
```



---

```
catch(ArrayIndexOutOfBoundsException e)
{
    System.out.println("Enter Catch");
    System.out.println(e);
}
finally
{
    System.out.println("Enter Finally");
    System.out.println("Good Bye");
}

System.out.println("After Finally");
}
```



运行情况1：  
没有产生异常，  
try块全部执行，  
catch块不执行

```
Start try
which color?(1,2,3) : 1
RED
End Try
Enter Finally
Good Bye
After Finally
```


运行情况2：  
发生异常，try  
块中止执行，  
catch块捕获异常

```
Start try
which color?(1,2,3) : 4
Enter Catch
java.lang.ArrayIndexOutOfBoundsException: 3
Enter Finally
Good Bye
After Finally
```



运行情况3:  
catch未能  
捕获异常

```
Start try
which color?(1,2,3) : 3c
Enter Finally
Good Bye
Exception in thread "main"
java.lang.NumberFormatException: For input string: "3c"
    at
    java.lang.NumberFormatException.forInputString(Number
    FormatException.java:48)
        at java.lang.Integer.parseInt(Integer.java:456)
        at java.lang.Integer.parseInt(Integer.java:497)
    at
    exceptions.TestTryCatchFinally.main(TestTryCatchFinally
    .java:14)
```



## 3.2.3 异常的处理(续)

### ——捕获异常

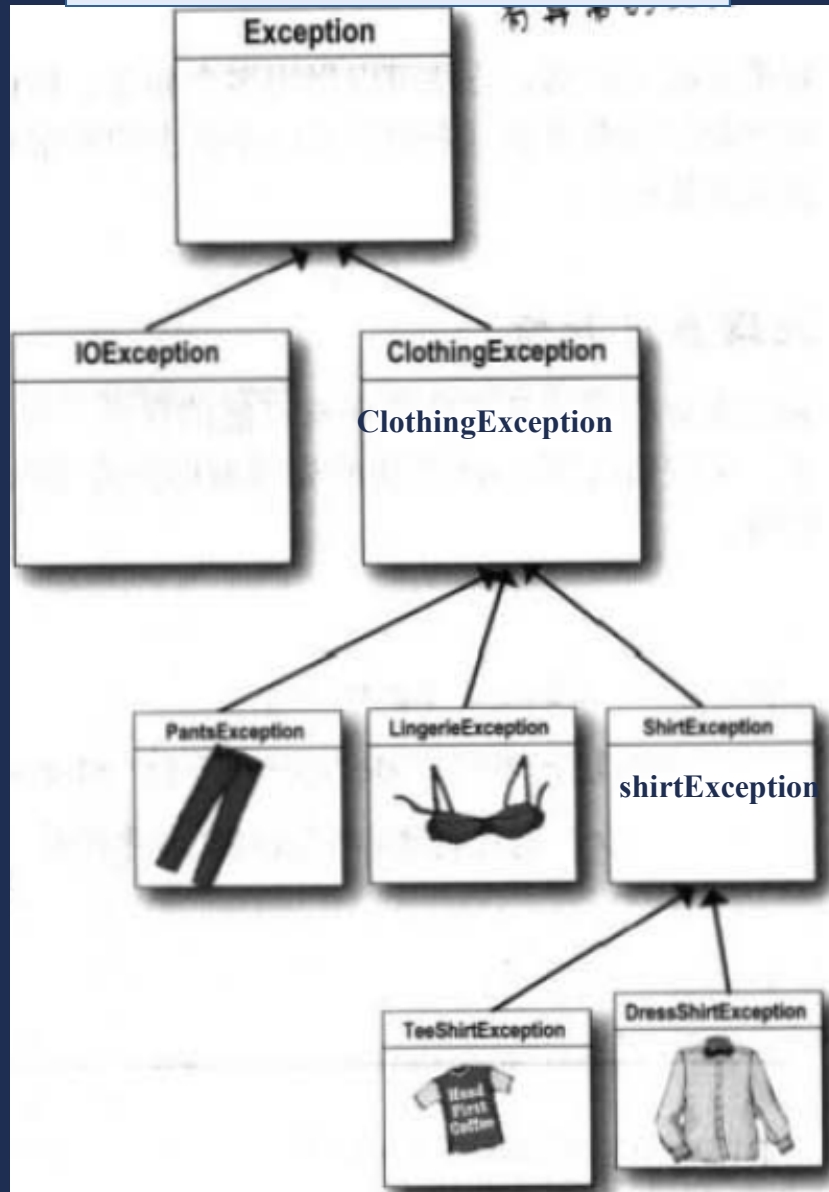
- 一个方法可以抛出多个异常

```
public class Laundry{  
    public void doLaundry() throws  
        PantsException, LingerieException{  
        //有可能抛出两个异常的代码  
    }  
}
```

```
public class Foo {  
    public void go() {  
        Laundry laundry = new Laundry();  
        try {  
            laundry.doLaundry();  
        } catch (PantsException pex) {  
            // 恢复程序代码  
        } catch (LingerieException lex) {  
            // 恢复程序代码  
        }  
    }  
}
```



## 异常也是多态的



```

public void doLaundry() throws
ClothingException{
    //此处可以抛出ClothingException的
    任何子类对象
}
    
```



```

try{
    laundry.doLaundry();
}
catch(ClothingException e){
    //此处可以catch ClothingException的
    任何子类对象
}
    
```

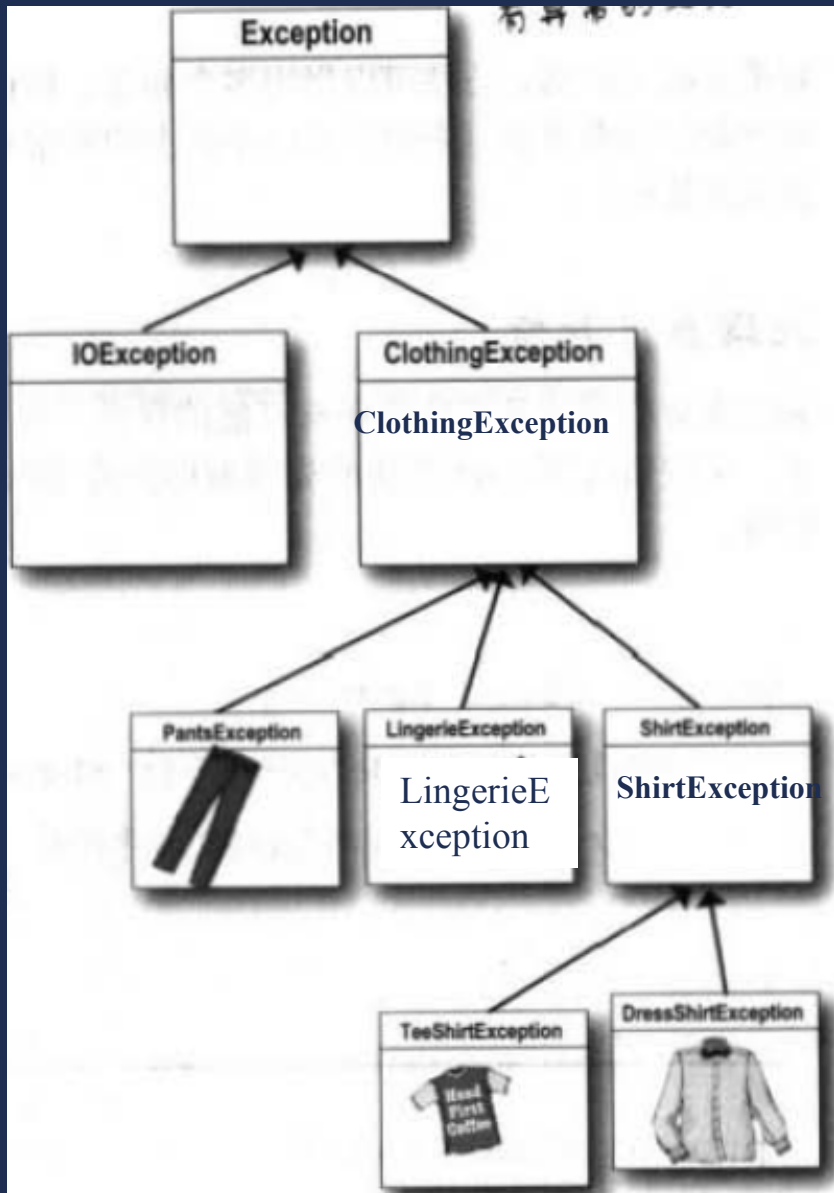


```

try{
    laundry.doLaundry();
}
catch(ShirtException e){
    //此处只能catch ShirtException的子类
    对象
}
    
```



## 为不同的异常书写不同的处理



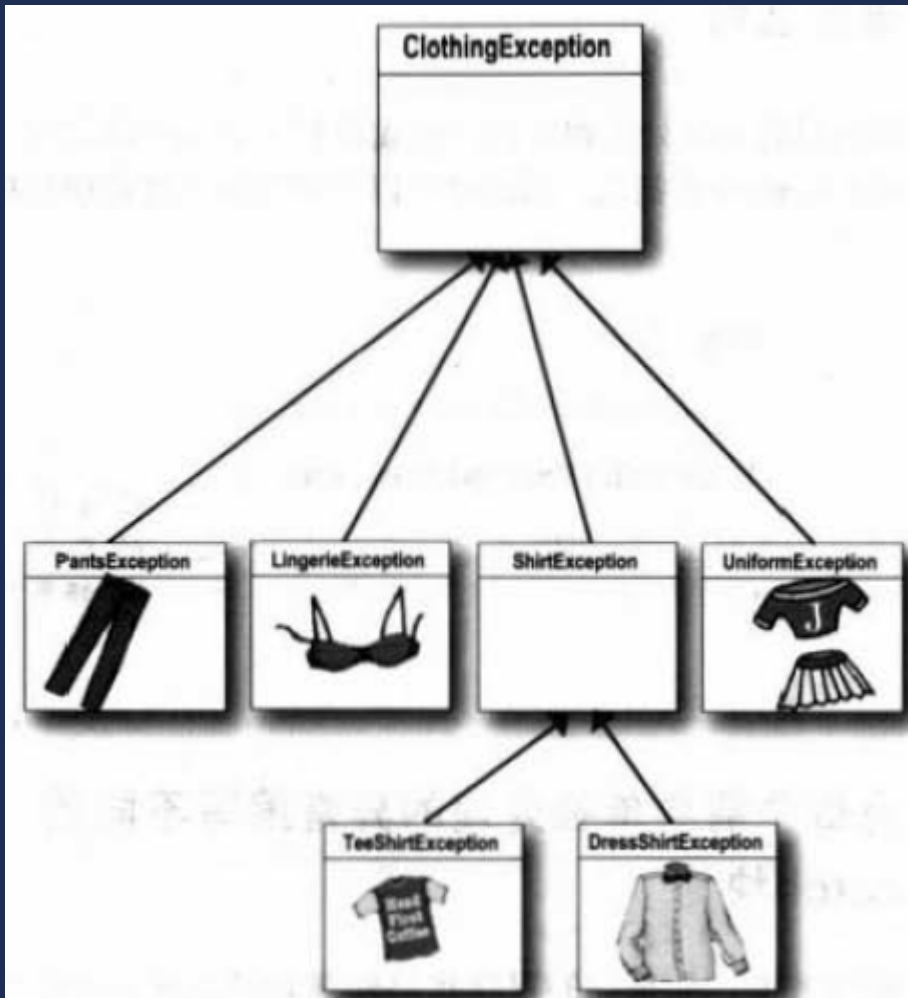
```
try{
    laundry.doLaundry();
}
catch(Exception e){
    //使用Exception超类，这样会捕获所有的异常，你会搞不清楚哪里出错
}
```

↓

```
try{
    laundry.doLaundry();
}
catch(ShirtException e){
    //处理此问题
}
catch(LingerieException e){
    //处理此问题
}
catch(ClothingException e){
    //处理其他问题
}
```



有多个catch块时，一般的异常类型放在后面，特殊的放在前面



```
try{
    laundry.doLaundry();
}
catch(ClothingException e){
    //处理其他问题
}
catch(ShirtException e){
    //处理此问题
}
catch(LingerieException e){
    //处理此问题
}
```

ClothingException放在最前面不合适！

ClothingException应该放到最后，ShirtException和LingerieException次序调换也没有问题，因为是同级的

## 3.2.3 异常的处理(续)

### ——例3\_12

异常  
处理  
简介

- 读入两个整数，第一个数除以第二个数，之后输出

```
import java.io.*;
public class ExceptionTester {
    public static void main(String args[]) {
        System.out.println("Enter the first number:");
        int number1 = Keyboard.getInteger();
        System.out.println("Enter the second number:");
        int number2 = Keyboard.getInteger();
        System.out.print(number1 + " / " + number2 + "=");
        int result = number1 / number2;
        System.out.println(result);
    }
}
```



## 3.2.3 异常的处理(续)

### ——例3\_12

- 其中，**Keyboard**类的声明如下

```
import java.io.*;
public class Keyboard{
    static BufferedReader inputStream = new BufferedReader
        (new InputStreamReader(System.in));
    public static int getInteger() {
        try {
            return (Integer.valueOf(inputStream.readLine().trim()).intValue());
        } catch (Exception e) {
            e.printStackTrace();
            return 0;
        }
    }
    public static String getString() {
        try{
            return (inputStream.readLine());
        }catch (IOException e)
        { return "0";}
    }
}
```

异常  
处理  
简介

## 3.2.3 异常的处理(续)

### ——例3\_12运行结果

#### 异常处理简介

- 运行结果

```
Enter the first number:
```

```
140
```

```
Enter the second number:
```

```
abc
```

```
java.lang.NumberFormatException: abc
```

```
at java.lang.Integer.parseInt(Integer.java:426)
```

```
at java.lang.Integer.valueOf(Integer.java:532)
```

```
at Keyboard.getInteger(Keyboard.java:10)
```

```
at ExceptionTester.main(ExceptionTester.java:7)
```

```
140 / 0=Exception in thread "main" java.lang.ArithmeticException:  
/ by zero
```

```
at ExceptionTester.main(ExceptionTester.java:10)
```

- 说明

- ✎ 在 `Keyboard.getInteger()` 方法中，捕获任何 `Exception` 类的异常，并输出相关信息

- ✎ 为了处理输入的字母（而非数字），可以使用方法 `Keyboard.getString()` 取得字符串后，再做转换

## 3.2.3 异常

### 异常处理简介

- 捕获异常

```
try {  
    System.out.println("Enter the first number:");  
    number1 = Integer.valueOf(Keyboard.getString()).intValue();  
    System.out.println("Enter the second number:");  
    number2 = Integer.valueOf(Keyboard.getString()).intValue();  
    result = number1 / number2;  
}  
catch (NumberFormatException e) { //格式异常  
    System.out.println("Invalid integer entered!");  
    System.exit(-1);  
}  
catch (ArithmeticException e) { //被0除异常  
    System.out.println("Second number is 0, cannot do division!");  
    System.exit(-1);  
}
```

- 运行结果

Enter the first number:

143

Enter the second number:

0

Second number is 0, cannot do division!



## 3.2.3 异常的处理(续)

### ——例3\_14改进

- 对程序进行改进：重复提示输入，直到输入合法的数据。为了避免代码重复，可将数据存入数组

异常  
处理  
简介

```
import java.io.*;
public class ExceptionTester4 {
    public static void main(String args[]) {
        int    result;
        int    number[] = new int[2];
        boolean valid;
        for (int i=0; i<2; i++) {
            valid = false;
            while (!valid) {
                try {
                    System.out.println("Enter number "+(i+1));
                    number[i]=Integer.valueOf(Keyboard.getString()).intValue();
                    valid = true;
                } catch (NumberFormatException e) {
                    System.out.println("Invalid integer entered. Please tryagain.");
                }
            }
        }
    }
}
```

## 3.2.3 异常的处理(续)

——例3\_14改进

```
try {  
    result = number[0] / number[1];  
    System.out.print(number[0] + " / " +  
                      number[1] + "=" + result);  
} catch (ArithmeticException e) {  
    System.out.println("Second number is 0,  
                      cannot do division!");  
}  
}  
}
```



## 3.2.3 异常的处理(续)

### ——例3\_14运行结果

- 运行结果

```
Enter number 1
abc
Invalid integer entered. Please try again.
Enter number 1
efg
Invalid integer entered. Please try again.
Enter number 1
143
Enter number 2
abc
Invalid integer entered. Please try again.
Enter number 2
40
143 / 40=3
```





## 3.2.4 生成异常对象

- 生成异常对象

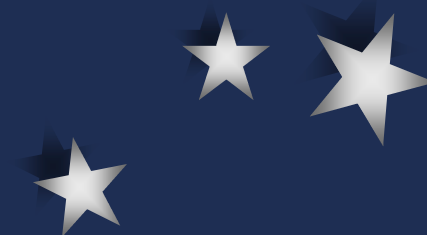
- ☞ 三种方式

- 由Java虚拟机生成
    - 由Java类库中的某些类生成
    - 在程序中生成自己的异常对象，也即是异常可以不是出错产生，而是人为地抛出

- ☞ 生成异常对象都是通过throw语句实现，生成的异常对象必须是Throwable或其子类的实例

- 方式1: `throw new ThrowableObject();`
    - 方式2:

- ```
ArithmeticException e = new ArithmeticException();  
throw e;
```



## 3.2.4 生

## ● 运行结果

java.lang.ArithmeticException

java.lang.ArrayIndexOutOfBoundsException

java.lang.StringIndexOutOfBoundsException

## ● 生成异常对象

```
class ThrowTest
{
    public static void main(String args[])
    {
        try { throw new ArithmeticException();
        } catch(ArithmeticException ae){
            System.out.println(ae);
        }
        try { throw new ArrayIndexOutOfBoundsException();
        } catch(ArrayIndexOutOfBoundsException ai){
            System.out.println(ai);
        }
        try { throw new StringIndexOutOfBoundsException();
        } catch(StringIndexOutOfBoundsException si){
            System.out.println(si);
        }
    }
}
```

## 3.2.5 声明自己的异常类

- 声明自己的异常类

- ✎ 除使用系统预定义的异常类外，用户还可声明自己的异常类
- ✎ 自定义的所有异常类都必须是 **Exception** 的子类
- ✎ 一般的声明方法如下

```
public class MyExceptionName extends SuperclassOfMyException {  
    public MyExceptionName() {  
        super("Some string explaining the exception");  
    }  
}
```

## 3.2.5 声明自己的异常类(续)

### ——例3\_17

- 声明当除数为零时抛出的异常类 **DivideByZeroException**

```
public class DivideByZeroException extends ArithmeticException{  
    public DivideByZeroException() {  
        super("Attempted to divide by zero");  
    }  
}
```

```
import java.io.*;  
public class Examp3_16 {  
    private static int quotient(int numerator, int denominator)  
        throws DivideByZeroException {  
        if (denominator == 0) throw new DivideByZeroException();  
        return(numerator / denominator);  
    }  
}
```



- 运行结果如下

Enter the first number:

140

Enter the second number:

0

DivideByZeroException: Attempted to divide by zero

```
public static void main(String[] args) {
    int number1, number2, result;
    try {
        number1 = Integer.valueOf(Keyboard.getString()).intValue();
        System.out.println("Enter the second number:");
        number2 = Integer.valueOf(Keyboard.getString()).intValue();
        result = quotient(number1, number2);
    }
    catch (NumberFormatException e) {
        System.out.println("Invalid integer entered!");
        System.exit(-1);
    }
    catch (DivideByZeroException e) {
        System.out.println(e.toString());
        System.exit(-1);
    }
    System.out.println(number1 + " / " + number2 + "=" + result);
}
```

## 3.3 方法重载

---

- 方法重载

- ✧ 一个类中名字相同的多个方法
- ✧ 这些方法的参数必须不同，Java 可通过参数列表的不同来辨别重载的方法
  - 或者参数个数不同
  - 或者参数类型不同
- ✧ 返回值可以相同，也可以不同
- ✧ 重载的价值在于它允许通过使用一个方法名来访问多个方法




## 3.3 方法重载(续)

### ——例3\_18

- 通过方法重载分别接收一个或几个不同数据类型的数据

```
class MethodOverloading {  
    public void receive(int i){  
        System.out.println("Receive one int parameter. ");  
        System.out.println("i="+i);  
    }  
    public void receive(double d){  
        System.out.println("Receive one double parameter. ");  
        System.out.println("d="+d);  
    }  
    public void receive(String s){  
        System.out.println("Receive one String parameter. ");  
        System.out.println("s="+s);  
    }  
}
```




## 3.3 方法重载(续)

### ——例3\_18

```
public void receive(int i,int j){
    System.out.println("Receive two int parameters. ");
    System.out.println("i=" + i + " j=" + j);
}
public void receive(int i,double d){
    System.out.println("Receive one int parameter and one double parameter. ");
    System.out.println("i=" + i + " d=" + d);
}
}

public class Examp3_17 {
    public static void main(String args[]){
        MethodOverloading m = new MethodOverloading();
        m.receive(2);
        m.receive(5.6);
        m.receive(3,4);
        m.receive(7,8.2);
        m.receive("Is it fun?");
    }
}
```





## 3.3 方法重载(续)

### ——例3\_18运行结果

- 运行结果

Receive one int parameter.

i=2

Receive one double parameter.

d=5.6

Receive two int parameters.

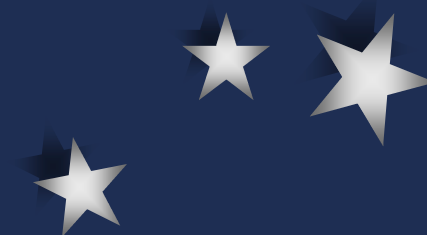
i=3 j=4

Receive one int parameter and one double parameter.

i=7 d=8.2

Receive one String parameter.

s=Is it fun?



## 3.4 本章小结

---

- 本章内容

- ✎ Java程序中类方法的控制结构，包括顺序、分支及循环三种基本结构
- ✎ Java的异常处理机制，包括对错误的分类方法，如何抛出异常、捕获异常
- ✎ 方法的重载

- 本章要求

- ✎ 掌握三种流程控制语法，并熟练应用
- ✎ 了解Java的异常处理机制，会编写相应程序
- ✎ 掌握方法重载的含义，并熟练应用

