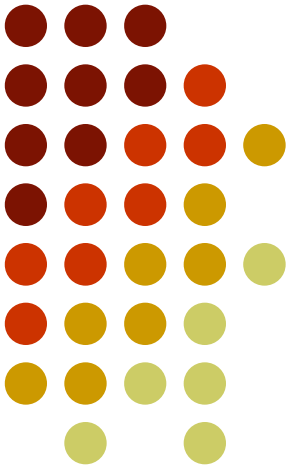
第三章 程序控制结构与异常处理

3.1 选择（分支）结构

3.2 循环结构

3.3 异常处理



基本要求

1. 掌握 Java 的程序控制流程
2. 掌握 Java的异常机制和处理方法

三种基本控制结构

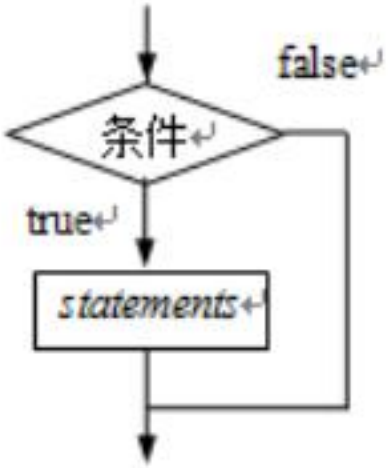
- 结构化程序设计有三种基本控制结构:

  顺序结构——选择结构——循环结构

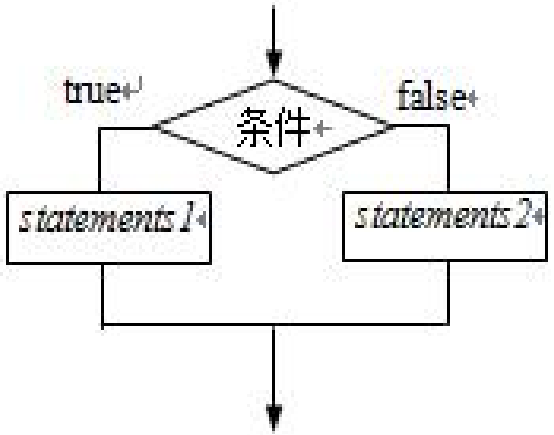
  - 顺序结构：比较简单，程序按语句的顺序依次执行。
  - 选择结构：根据是否满足某个条件确定执行哪些操作。
  - 循环结构：根据是否满足某个条件确定反复执行某些操作。

3.1 选择结构

- 单分支结构
- 双分支结构
- 多分支的if-else结构
- 嵌套的if语句




```
if(条件) {
   // 语句组
}
```




```
if(条件) {
   // 语句组1
}else{
   // 语句组2
}
```

多分支的if-else结构

- 如果程序逻辑需要多个选择，可以在if语句中使用一系列的else if语句，这种结构称为阶梯式if-else结构。

问题描述

- 输入学生的百分制成绩，打印输出等级的成绩。等级规定为，90分（包括）以上的为“优秀”，80分（包括）以上的为“良好”，70分（包括）以上的为“中等”，60分（包括）以上的为“及格”，60分以下为“不及格”。

分支的实现：if-else 选择语句

- if-else 结构的语法为：

```
if (条件)
    语句1;
else
    语句2;
```

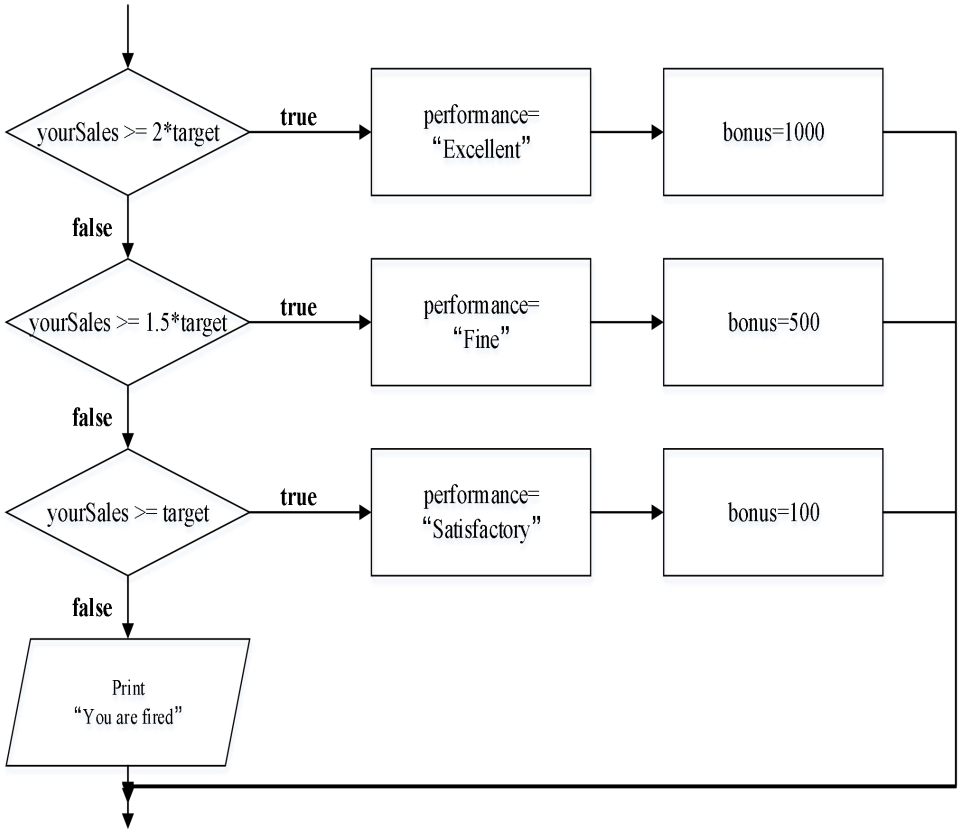
if和else块中只有一条语句

```
if (条件) {
    语句组;
}
else {
    语句组;
}
```

if和else块中有多条语句

教材示例3-3

```
01 if ( yourSales >= 2 * target )
02   {
03     performance = “Excellent”;
04     bonus = 1000;
05   }
06   else if ( yourSales >= 1.5 * target )
07   {
08     performance = “Fine”;
09     bonus = 500;
10   }
11   else if ( yourSales >= target)
12   {
13     performance = “Satisfactory”;
14     bonus = 100;
15   }
16   else
17   {
18     System.out.println(“You are fired!”);
20   }
```



switch语句

选项较多时使用if-else结构显得笨拙,switch语句更简洁明了

```
switch (expression){
   case value1:
      statements  [break;]
   case value2:
      statements  [break;]
   ...
   case valueN:
      statements  [break;]
   [default:
      statements]
}
```

类型为byte,short, int,char,enum,String

常量或常量表达式

语句组

结束switch结构

所有情况都不匹配执行的语句组

switch 选择语句

■ switch 结构的语法为：

```
switch(表达式){
    case '常量值 1':
       语句(组);
       break;
    case '常量值 2':
       语句(组);
       break;
    case '常量值 N':
       语句(组);
       break;
    default:
       语句(组);
}
```

表达式必须为byte，short，int或char类型

- 常量值必须是与表达式类型兼容的特定的一个常量
- 不允许有重复的case值

```
class MySwitch{
    public static void main(String args[]){
        int x=3;
         switch(x){
                default:System.out.println("a");
                case 1:System.out.println("b");
                case 2:
                case 3:System.out.println("c");break;
                case 4:System.out.println("d");
         }
    }
}
```

编程练习：

1. 单分支：从键盘上读取一个整数，若该数是偶数，输出该数。
2. 双分支：在求圆面积问题中，要求只有当radius>=0时才计算圆的面积，否则，程序给出错误提示。
3. 多分支：输入学生的百分制成绩，打印输出等级的成绩。等级规定为，90分（包括）以上的为“优秀”，80分（包括）以上的为“良好”，70分（包括）以上的为“中等”，60分（包括）以上的为“及格”，60分以下为“不及格”。
4. 嵌套的if：求a、b和c中最大值并将其保存到max中,就可以使用一个嵌套的if结构。

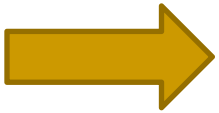
条件运算符——?

- 条件运算符（conditional operator）格式如下：

```
condition ? expression1 : expression2
```

条件运算符可实现if~else结构。若max, a, b是int型变量：

```
if (a > b) {
   max = a;
}else {
   max = b;
}
```



```
max = (a > b)? a : b ;
```

课堂讨论及训练

- 开发一个让学生练习一位数加法的程序。程序开始运行随机生成两个一位数，显示题目让学生输入计算结果，程序给出结果是否正确。

3.2 循环结构

3.2.1 for循环

3.2.2 while循环

3.2.3 循环嵌套

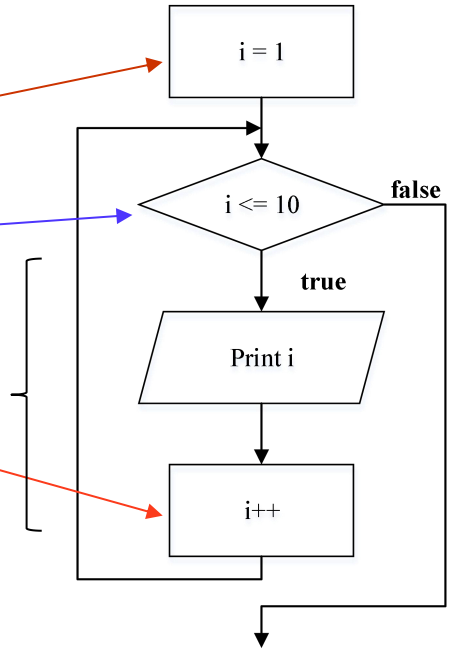
3.2.4 do...while循环

3.2.5 迭代循环

循环结构

- Java语言提供了4种循环结构：while循环、do-while循环、for循环和增强的for循环(for each循环)。

- 一般情况下，一个循环结构包含四部分内容：
  1）初始化部分
  2）循环条件
  3）迭代部分
  4）循环体部分



循环的实现

- for 循环

  for(变量初始化； 条件； 递增或递减变量的值)

  - **{语句组； }**

- while 循环

  while(条件)

  { 语句组； }

- do-while循环

  - **do**
  - **{ 语句组； }**
  - **while(条件)；**

3.2.1 for循环

```
for ([初始化部分];[条件]; [迭代部分]){
    // 循环体
}
```

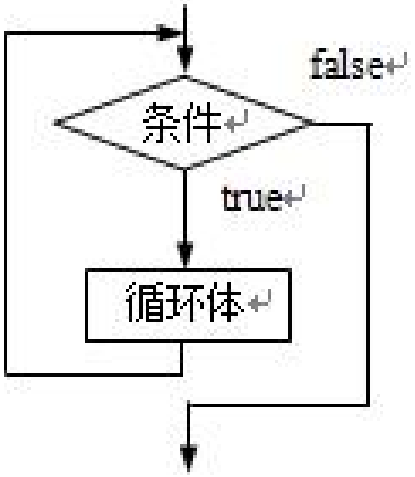
下面的程序将数字1~10输出到屏幕上。

```
for ( int i = 1; i <= 10; i++ )
    System.out.println(i);
```

- 编写程序，显示从100到1000之间所有能被5和6整除的数。

3.2.2 while循环

```
[初始化部分]
while (条件){
    // 循环体
    [迭代部分]
}
```



当条件为true时，while循环执行一条语句（也可以是一个语句块）。常用的格式为：

while (*condition*)

***statement*;**

如果开始循环条件的值为false，则while循环体一次也不执行。

While语句举例1

```
// sqrt(99) 开根多少次 归1
1.    public class Sqrto1 {
2.        public static void main(String[] args) {
3.             int  n = 0;
4.            Double x = 99.0;
5.            // java.lang.Math.sqrt(Double.x)
6.          while (x > 1.00) {
7.                x = Math.sqrt(x);
8.                n++;
9.                System.out.println(n + ", x= " + x);
10.         }
11.          System.out.print("\n 99开平方根" + n + "次等于1  ");
12.     }
13.   }
```

While语句举例2

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

        String copyFromMe = "Copy this string until you encounter the
                                                letter 'g'.";
        StringBuffer copyToMe = new StringBuffer();

        int i = 0;
        char c = copyFromMe.charAt(i);

        while (c != 'g') {
            copyToMe.append(c);
            c = copyFromMe.charAt(++i);
         }
        System.out.println(copyToMe);
    }
}
```

结果：

```
Copy this strin
```

- 随机产生一个100~200之间的整数，用户从键盘上输入所猜的数，程序显示是否猜中的消息，如果没有猜中要求用户继续猜，直到猜中为止。

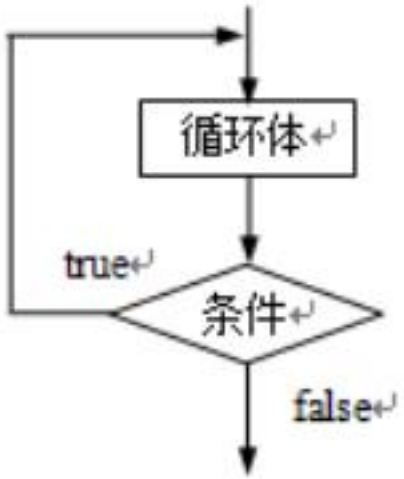
3.2.3 do-while循环

- While循环语句首先检测循环条件。因此，循环体中的代码有可能不被执行。如果希望循环体至少执行一次，则应该将检测条件放在最后。使用do/while循环语句可以实现这种操作方式。它的语法格式为：

```
do
    statement
while (condition);
```

do-while循环

```
[初始化部分]
 do{
   // 循环体
   [迭代部分]
 }while(条件);
```



- 特点：循环体至少被执行一次。

do-while循环示例：累加

```
1.  System.out.println("请输入一个数（输入-1为结束）：");
2.  Scanner scanner = new Scanner(System.in);
3.  int num;
4.  long sum = 0;
5.  int i = 0;
6.  do {
7.        num = scanner.nextInt();
8.        sum += num;
9.        i++;
10.       System.out.print(num + " + ");
11.    } while (num != -1);
12. System.out.println("输入了" + i + "个数，总和为：" +
    sum);
13. scanner.close();
```

有错

do-while语句示例

```
//复制字符串，一直到某个字符为止
public class DoWhileDemo {
    public static void main(String[] args) {
        String s1 = "Copy this string until you encounter the letter 'g'.";
        StringBuffer s2 = new StringBuffer();
        int i = 0;
        char c = s1.charAt(i);
        do {
            s2.append(c);
            c = s1.charAt(++i);
         } while (c != 'g');
        System.out.println(s2);
    }
}
```

结果:

```
Copy this strin
```

3.2.4 循环嵌套

循环嵌套，即多重循环。

（1）for循环结构嵌套

```
for(int i = 0; i <= 10; i++){
    ……
            for(int j = 0; j<=10; j++){
                ……
            }
}
```

（2）while循环结构嵌套

```
int i = 0;
while(i<=10){
    int j = 0;
    While(j<=10){
            j++;
      }
            i++;
}
```

（3）for循环结构与while循环结构嵌套使用

```
for(int i = 0; i <= 10; i++){
    int j = 0;
    while(j <= 10){
        j++;
    }
}
```

（4）while循环结构与for循环结构嵌套使用

```
int i = 0;
        while(i<=10){
            for(int j = 0; j<=10; j++){
            //
             }
            i++;
}
```

循环的嵌套

- 在一个循环的循环体中可以嵌套另一个完整的循环，称为循环的嵌套。

```
while(条件){
    // 外层循环体
     do{
        // 内层循环体
     }while(条件);
    // 其他语句
}
```

```
for(初始化;条件;迭代){
    // 外层循环体
    for(初始化;条件;迭代){
      // 内层循环体
    }
    // 其他语句
}
```

打印输出乘法表

问题描述：

- 编写程序，打印输出如下乘法表

1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81

打印输出乘法表

问题描述：

- 编写程序打印输出如下乘法表

```
1*1=1
1*2=2  2*2=4
1*3=3  2*3=6  3*3=9
1*4=4  2*4=8  3*4=12  4*4=16
1*5=5  2*5=10  3*5=15  4*5=20  5*5=25
1*6=6  2*6=12  3*6=18  4*6=24  5*6=30  6*6=36
1*7=7  2*7=14  3*7=21  4*7=28  5*7=35  6*7=42  7*7=49
1*8=8  2*8=16  3*8=24  4*8=32  5*8=40  6*8=48  7*8=56  8*8=64
1*9=9  2*9=18  3*9=27  4*9=36  5*9=45  6*9=54  7*9=63  8*9=72  9*9=81
```

3.2.5 增强的for循环

```
for (Type var: collectionName){
    // 循环体
}
```

- 主要用于对数组和集合元素迭代。

JDK1.6中for 循环的优化

- 将一个集合作为一个整体放入for循环中，在for循环中可将集合中的元素进行逐个处理。

```
String[] names = {”Wang”,”Zhang”,”Li”,”Wu”};
  for(String option: names) {
      System.out.println(option);
  }
```

infinite loop

```
1. class EnhancedForDemo {
2.     public static void main(String[] args){
3.         int[] numbers =
4.             {1,2,3,4,5,6,7,8,9,10};
5.         for (int item : numbers) {
6.             System.out.println("Count is: " + item);
7.         }
8.     }
9. }
```

特殊跳转语句

- **break [label]**
  从switch语句、循环语句中跳出。

- **continue [label]**
  跳过标号循环体的其余部分，不带label 跳过最内层循环的剩余语句。

- **label: statement**

break 和 continue 语句

- break 语句在循环中用于立即从当前循环终止控制。
- 遇到 break 语句时，将跳出当前循环。
- continue 语句则是从其调用处跳至循环的开始处。
- continue 语句之后的语句将不再执行。

break语句

- break语句是用来跳出while、do、for或switch结构的执行，该语句有两种格式：

  break;

  break *label*;

- 编程计算1+2+3+…之和，当和超过100结束，输出结果。

continue语句

- continue语句与break语句类似，但它只终止执行当前的迭代，导致控制权从下一次迭代开始。该语句有下面两种格式：

  continue;

  continue *label;*

- 编程计算1到10之间的数的和，能被3整除的数除外。

```
① class BreakDemo {
②     public static void main(String[] args) {
③         int[] arrayOfInts = { 32, 87, 3, 589, 12, 1076, 2000,  8, 622, 127 };
④         int searchfor = 12;   // =args[0];
⑤         int i;     boolean foundIt = false;
⑥         for (i = 0; i < arrayOfInts.length; i++) {
⑦             if (arrayOfInts[i] == searchfor) {
⑧                 foundIt = true;
⑨                 break;
⑩             }        }
⑪         if (foundIt) {
⑫                 System.out.println("Found " + searchfor + " at index " + i);
⑬             } else {
⑭                 System.out.println(searchfor + " not in the array");
⑮                   }
⑯       }   }
```

示例学习

例:求2~1000的素数，每行打印15个数字

```
import java.io.*;
public class Prime1  {
    public static void main(String args[])    {
      First_Loop: for(int i=2,k=0;i<1000;i++)
     {    for(int j=2;j<=Math.sqrt(i);j++)
       {   if(i%j==0)
            continue First_Loop;
       }
       System.out.print(i+"\t");
       k++;
       if(k%15==0)
          System.out.println();
      }  //for
    } //main
} //class
```

-continue示例

求最大公约数

问题描述：

- 两个正整数的最大公约数（Greatest Common Divisor，GCD）是能够同时被两个数整除的最大整数。例如，4和2的最大公约数是2，16和24的最大公约数是8。

- 一种方法是，假设求两个整数m和n的最大公约数，显然1是一个公约数，但它可能不是最大的。可以依次检查k（k=2,3,4,…）是否是m和n的最大公约数，直到k大于m或n为止。

打印输出若干素数

问题描述：

- 素数又称质数，有无限个。素数定义为在大于1的正整数中，除了1和它本身以外不再有其他因数的数。

思路：

- 判断一个正整数number是否是素数，从2开始判断number是否能被这个数整除，若能被它整除，则不是素数，一直到number-1为止，若都不能整除，则是素数。

课堂讨论及训练

- 下面的if语句在Java语言中是否合法?

```
if(0 <= n <=10){
    System.out.println("n的值介于0和10之间");
}
```

课堂讨论及训练

- 下面代码的输出结果是什么？解释原因。

```
int x = 80000000;
while (x > 0)
    x ++;
    System.out.println("x is " + x);
```

课堂讨论及训练

- 若实现如下功能：如果年龄（age）大于40岁，并且工资（salary）小于5000元，工资增加1000元；如果年龄不大于40岁，将工资增加500元。讨论该段代码是否正确？

```
if( age > 40)
    if( salary < 5000)
        salary = salary + 1000;
else
    salary = salary + 500;
```

```
int age = 35;
int salary = 4000;
if (age > 40) {
    if (salary < 5000)
        salary = salary + 1000;
} else
    salary = salary + 500;
System.out.println(age+"岁,工资="+salary);
```

编程作业

- 3.2 编写程序，要求用户从键盘上输入一个年份，输出该年是否是闰年。符合下面两个条件之一的年份即为闰年：（1）能被4整除，但不能被100整除；（2）能被400整除。

请输入年份：2017
2017年不是闰年。

- 3.10 编写程序，计算并输出1-1000之间含有7或者是7倍数的整数之和及个数。

总结3-1

- Java 支持下列控制结构：
  - 选择（ **if-else、switch**）
  - 循环（**while、do-while、for**）
  - 跳转（**break、continue**）
- break 语句在循环中用于立即从当前循环终止控制
- continue 语句则是从其调用处跳至循环的开始处
- 数组完全作为对象来使用，可用来将同一类型的数据存储在连续的内存位置

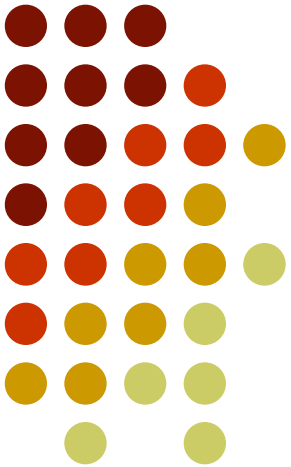
作业（上机3）：

- 上机三：完成一模拟扑克(52张)<洗牌>和<发牌>
- 第2章 Java语言基础 作业
  1. 计算机支持的最大整数值是多少，获取浮点类型的最大最小值
  2. 常见字母的大小写转换，实现基本类型的类型转换
  3. 判断一个数字的奇偶性
  4. 不借助第三者实现两个变量值的互换
  5. 实现输入字符的加密和解密
- Java流程控制
  1. 判断某一年是否为闰年
  2. 根据消费金额计算折扣
  3. 计算1+1／2！+1／3！+…+1／10！的值
  4. 实现输出杨辉三角
  5. 空心菱形如何表示

2021.04.01（#3）

2019.10.8（#4）

3.3 Java异常处理



3.3.1 异常的概念

程序中的错误：

- 语法编译错误：由语言的编译系统负责检测和报告
  - 由于所编写的程序存在语法问题，未能通过由源代码到目标代码的编译过程而产生的错误。
- 运行错误：在程序的运行过程中产生的错误。
  - 严重错误：出现死机、死循环，内存溢出，递归无出口（只能在编程阶段解决）
  - 非致命错误（异常）：被0除，读文件而文件不存在，网络中断等。

对错误的处理

- 程序执行的错误
- 处理错误
  - 向用户通知错误
  - 保存全部工作
  - 让用户适当地退出程序
- 异常处理
  - JAVA检测和报告错误的机制

3.3.1异常的概念

以常规方法处理异常：读文件 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=-2;
            }else  errorCode=-3;
        }else errorCode=-4 ;
   }else   errorCode=-5;
```

常规方法处理异常的缺点：

（1）大部分精力花在出错处理上了.

（2）只把能够想到的错误考虑到，
对以外的情况无法处理

（3）程序可读性差

（4）出错返回信息量太少

处理错误的方式

- 函数式编程中，直接检测调用函数返回值以确定错误内容，存在的问题
  - 检测错误返回的编码工作量大,逻辑上复杂
  - 返回的错误信息有限，无详细的位置、性质等
- 异常处理
  - throws，使不能正常结束的方法抛出一个封装了错误信息的对象——异常
  - 使用异常处理程序处理异常错误

file:///D:/Java/tutorial/essential/exceptions/index.html

示例：MyFrame.java

```
public static void main(String args[]) {
   EventQueue.invokeLater(new Runnable() {
      public void run() {
         try {
               MyFrame frame = new MyFrame();
               frame.setVisible(true);
            } catch (Exception e) {
              e.printStackTrace();
   }   }   });
}
```

异常处理机制

- Java中定义了很多异常类，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和处理错误的方法等内容。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。一旦一个异常对象产生了，系统中就一定有相应的机制来处理它，确保不会产生死机、死循环或其他对操作系统的损害，从而保证了整个程序运行的安全性。这就是Java的异常处理机制。

用异常的形式处理错误：read-File()

生成、抛出

```
try {
    openTheFile();      //如果读文件失败抛出fileopenFailed异常对象
    determine_its_size();
  //如果确定文件size失败,抛出sizeDetermineFailed异常对象
    allocate_that_much_memory();
  //如果分配内存失败抛出memoryAllocateFailed异常对象
    closeTheFile();}
```

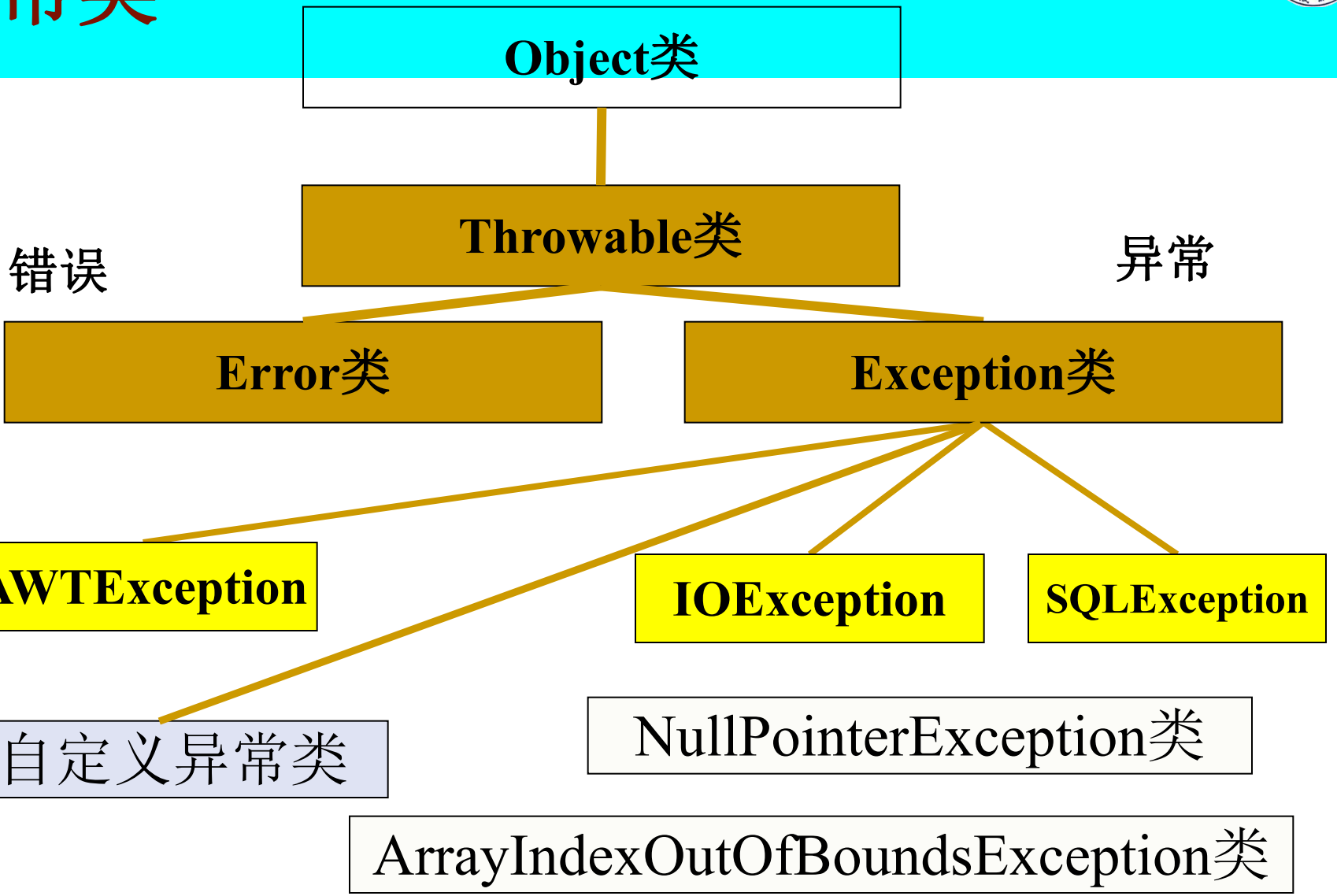
捕获、处理

```
catch(fileopenFailed          e)   {dosomething;}
catch(sizeDetermineFailed     e)   {dosomething;}
catch(memoryAllocateFailed    e)   {dosomething;}
catch(readFailed              e)   {dosomething;}
catch(fileCloseFailed         e)   {dosomething;}
```

异常处理机制

- 当方法执行过程中出现错误而干扰了程序流程时，会抛出一个异常，即构造出一个异常类的对象。
- 异常类对象代表当前出现的一个具体异常，该对象封装了异常的有关信息。
- 异常分为系统定义异常和用户自定义异常。
- 异常抛出方式：
  - 自动抛出（系统定义异常）
  - 用throw语句抛出（用户定义异常）
- 方法中的异常处理：
  - 捕获异常，就地解决，并使程序继续执行 。
  - 将异常向外转移，即将异常抛出方法之外，由调用该方法的环境去处理。

异常类


Object类
Throwable类
错误
异常
Error类
Exception类
AWTException
IOException
SQLException
自定义异常类
NullPointerException类
ArrayIndexOutOfBoundsException类

java.lang.Throwable

- java.lang.Error
  - java.lang.AssertionError
  - java.lang.LinkageError
  - java.lang.ThreadDeath
  - java.lang.VirtualMachineError
- java.lang.Exception
  - java.lang.ClassNotFoundException
  - java.lang.CloneNotSupportedException
  - java.lang.IllegalAccessException
  - java.lang.InstantiationException
  - java.lang.NoSuchFieldException
  - java.lang.InterruptedException
  - java.lang.NoSuchMethodException
  - java.lang.RuntimeException

异常类

- Exception类定义的是较轻的错误，你可以编写代码来处理这类错误，并继续程序的执行。
- Exception原因：
  - ✓ 打开的文件不存在。
  - ✓ 网络连接中断。
  - ✓ 操作数超过允许范围。
  - ✓ 想要加载的类文件不存在。
  - ✓ 试图通过空的引用型变量访问对象。
  - ✓ 数组下标越界。
- Error类定义的错误是致命性错误，一般会导致程序停止执行。

Exception类

- 构造方法
  - Exception()
  - Exception(String 异常描述)
- Exception类的方法均继承自Throwable类
- 方法
  - String getMessage()
    返回异常描述。
  - String toString()
    返回异常对象详细信息。
  - void printStackTrace()
    打印异常发生的路径，即引起异常的方法调用嵌套序列。

系统定义的异常类

- ClassNotFoundException
  未找到要加载的类
- ArrayIndexOutOfBoundsException
  数组越界使用
- FileNotFoundException
  未找到指定的文件或目录
- IOException
  输入、输出错误
- NullPointerException
  引用空的尚无内存空间的对象

- 系统将常见错误予定义为若干异常类
- 当出现异常时自动抛出

系统定义的异常类

- ArithmeticException
  算术错误，如除数为0
- InterruptedException
  一线程在睡眠、等待或其他原因暂停时被其他线程打断
- UnknownHostException
  无法确定主机的IP地址
- SecurityException
  安全性错误，如Applet欲读写文件
- MalformedURLException
  URL格式错误

例:包含算术异常, 字符串越界,数组越界三种异常的程序

Ex0601.java

```
public class Exp01 {
public static void main(String args[]){
    String s="abcdefg";  char c;
    int a,b=0;
    int[] array=new int[7];
    try {a=1/b;
     }catch(ArithmeticException ae) {
        System.out.println("Catch "+ae);    }
    try {array[8]=0;
    }catch(ArrayIndexOutOfBoundsException ai){
        System.out.println("Catch "+ai);    }
    try { c=s.charAt(8);
        }catch(StringIndexOutOfBoundsException se){
            System.out.println("Catch "+se);
        }
    }
}
```

运行结果：

Catch java.lang.ArithmeticException: / by zero
Catch java.lang.ArrayIndexOutOfBoundsException: 8
Catch java.lang.StringIndexOutOfBoundsException: String index out of range: 8

用户定义的异常类

- 用户自定义异常主要用来处理用户程序中特定的逻辑、运行错误。
- 定义异常类

```
class MyExp    extends Exception {  //或继承其他异常类
//定义新的属性
//重载构造方法
   例：  MyExp(String msg)    { super(msg)  }
// 重载原方法，或定义新方法   }
```

- 抛出异常类： throw 异常类对象；

例： throw new MyExp(“余额不足”) ;

例： throw new Exception() ;

3.3.3 异常处理——捕获异常

- 捕获并处理异常

```
try{
    //接受监视的程序块,在此区域内发生
    //的异常,由catch中指定的程序处理;
}catch(要处理的异常种类和标识符) {
    //处理异常;
}catch(要处理的异常种类和标识符) {
    //处理异常;
}
```

异常的产生

- 自动抛出异常
  - 运行异常

    程序运行中可自动由Java解释器引发并处理的异常；编程时不需捕获或声明。

    如：除以0、下标越界、指针例外等
  - 非运行异常

    可在编译期由编译器确定某方法是否会发生此类异常；必须编程时捕获或声明

    如：IOException, InterruptedException

数据连接例外

```
public Connection  OpenConn() throws Exception {

try {
    java.util.Properties props = new java.util.Properties();
      props.put("user",     dbUser);
      props.put("password", dbPsw);
      props.put("db", dbName);
      props.put("server", dbServer);
      props.put("port","1433");
      props.put("charset","GB2312");
     java.sql.Driver drv =
     (java.sql.Driver)Class.forName("weblogic.jdbc.mssqlserver4.Driver").newInstance
     ();
      theConnection = drv.connect("jdbc:weblogic:mssqlserver4:", props);
      }
  catch (SQLException ex) {
       System.err.println("SQL Connectting Error: " + ex.getMessage());
       }
      return theConnection;
  }
```

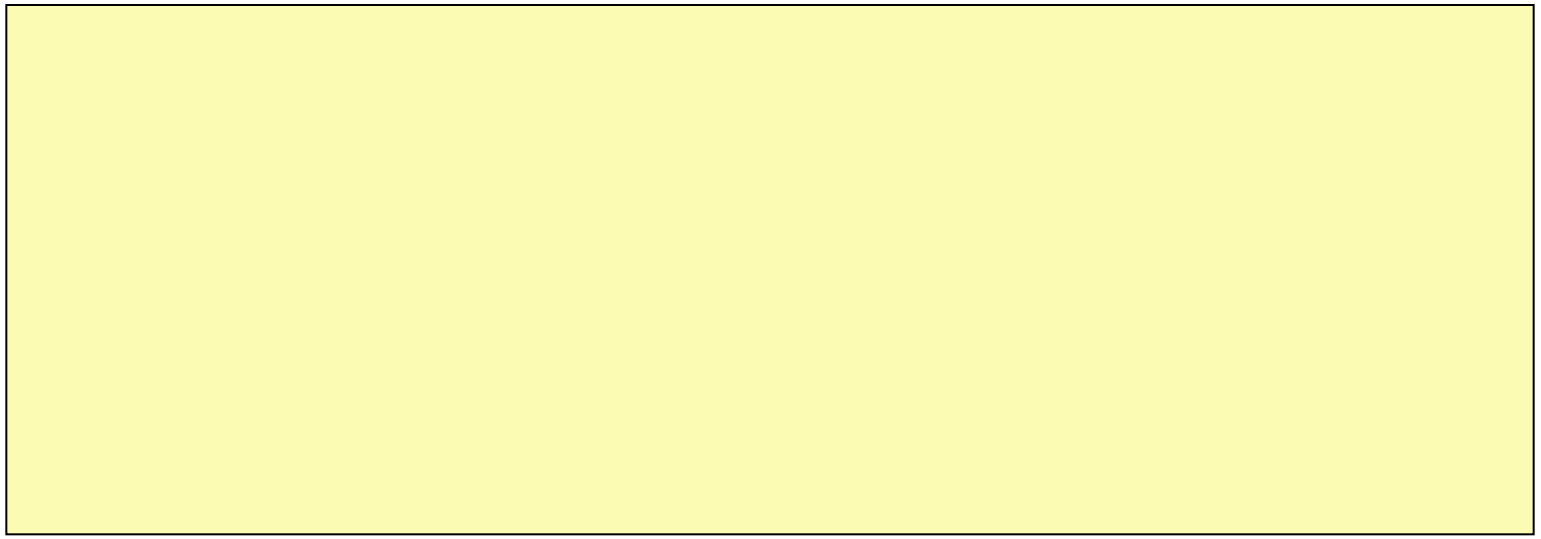
异常的产生

- 编程抛出异常：
  - 在程序的相应部分抛出异常：
  - throw new XXXException();
  - throw new XXXException(String describe);
- 例:String readData() throws EOFException

```
{.....
    throw new EOFException("error");
.....}
```

异常处理

- 警戒区：可能会引起异常的代码段



- 若try块中出现异常，则控制转向下面的异常处理部分，然后执行后续语句。
- 若try块中没有异常，则try块执行完，控制转向后续语句。

多个异常的处理

- 可用一组并列的catch块处理多异常情况

```
try { . . . }
catch( 异常类1  e1) { . . . }
catch( 异常类2  e2) { . . . }
. . .
```

- 匹配规则：
  - ✓ 抛出对象与catch参数类型相同
  - ✓ 抛出对象为catch参数类的子类
  - ✓ 按先后顺序捕获(注意catch块书写时的排列顺序:先具体、后一般)，但只捕获一次。

异常处理过程

- 当遇到关键字 **throw** 时就抛出一个异常
- 将控制转移到相关的 catch 块中处理之
- 如果产生异常的方法本身没有相应catch 语句块，则应有**throws**语句声明例外
- 退出当前方法并转向上一级调用此方法的方法的 catch 语句块
- 若始终没有catch块来处理则由运行系统处理

例 ExceptionMethods.java

```
public class ExceptionMethods {

    public static void main(String[] args) {
   // int a=0,b=2;   结果是？
     int a=2,b=0;    结果是？
    try {  //抛出例外
        a=1/b;
        System.out.println("a="+ a);
        if(b==0)        throw new Exception(“Here‘s my Exception”); //不执行
        if(a==0)        throw new Exception("Here's my Exception");
    } catch(Exception e) {  //捕获例外
         //处理例外
         System.out.println("Caught Exception");
         System.out.println("e.getMessage(): " + e.getMessage());
         System.out.println("e.toString(): " + e.toString());
         System.out.println("e.printStackTrace():");
         e.printStackTrace();
    }
     } //end of main()
} //end of class
```

```
----------Configuration: <Default>----------
Caught Exception
e.getMessage(): / by zero
e.toString(): java.lang.ArithmeticException: / by zero
e.printStackTrace():
java.lang.ArithmeticException: / by zero
    at ExceptionMethods.main(ExceptionMethods.java:15)
```

```
try { //抛出例外
throw new Exception("Here's my Exception");
```

```
----------Configuration: <Default>----------
Caught Exception
e.getMessage(): Here's my Exception
e.toString(): java.lang.Exception: Here's my Exception
e.printStackTrace():
java.lang.Exception: Here's my Exception
    at ExceptionMethods.main(ExceptionMethods.java:17)
```

自定义异常类

- 声明一个新的异常类，该异常类必须从Java已有定义的异常类继承，如Exception、IOException等
- 为新的异常类定义属性和方法，或重载父类的属性和方法，使这些属性和方法能够体现该类所对应的错误的信息。
- 例：Inheriting.java

```
class MyException extends Exception { //自定义例外
        public MyException() {}
        public MyException(String msg) {  super(msg);      }
}

public class Inheriting {
        public static void f( ) throws MyException {
                System.out.println("Throwing MyException from f()");
                throw new MyException();  //抛出自定义例外
        }
       // … … …
        public static void ma
                try {   f(); }  //
                catch(MyExcep
    //     … … …
        } //end of main()
} //end of class Inheriting
```

```
--------------------Configuration: <Default>--------------------
Throwing MyException from f()
MyException
    at Inheriting.f(Inheriting.java:9)
    at Inheriting.main(Inheriting.java:13)
```

异常转移

- 异常总是发生在方法执行过程中。
- 当异常不处理时会向方法外转移。
- 系统定义的异常自动向外转移。
- 用户自定义的异常要转移需要在方法头声明一下

```
String readData() throws MyExp
{……….
        throw new MyExp(“年龄超过规定”);
}
```

Throw和Throws的使用

```
package ch03;
import java.util.Scanner;

public class ThrowThrowsDemo {
    String name;    String sex;    int age;
        public static void main(String[] args) {
        ThrowThrowsDemo thr = new ThrowThrowsDemo();
        try {
            thr.stdInput();
            thr.compute(20, 4);
            thr.compute(5, 0);
        } catch (Exception e) {
            System.out.print("发生异常：  ");
            e.printStackTrace();
        }
    }
```

```
void stdInput() { // throws Exception{
    try {
        Scanner input = new Scanner(System.in);
            System.out.println("请输入你的姓名：");
            name = input.next();
            System.out.println("请输入你的年龄：");
            age = input.nextInt();
            if (age < 15 || age > 70)
                throw new Exception("年龄必须为（16~70）！"); // 语句
            System.out.println("请输入你的性别：");
            sex = input.next();
            if ("男".equals(sex) || "女".equals(sex)) {
                System.out.println("我的名字叫" + name + "，年龄为：" + age + "，性
别为：" + sex);
            } else {
                throw new Exception("性别只能是男/女！"); // 语句
            }
            input.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void compute(int a, int b) throws Exception {
        // System.out.print("我可能发生异常：  ");
        System.out.println("a/b的值为" + a / b);

    }
}
```

finally

```
try{
   ……
   }catch(ExceptionType  e){
   …….
   }finally{
     ……
   }
```

- finally: 不论是否有异常抛出，均执行
- 只有当try块中执行System.exit()时，会立即结束程序。
- 用处：一般用来进行一些“善后”操作，如系统资源的释放、文件的关闭等
- 例：FinallyWorks.java

```
public class FinallyWorks {
        static int count = 0;
        public static void main(String[] args) {
                while(true) {
                        try {
                                // count is zero first time:
                                if(count++ == 0)
                                        throw new Exception();
                                System.out.println("No exception");
                        } catch(Exception e) {
                                System.out.println("Exception
thrown“+e.getMessage());
                        } finally { //是否抛出例外都会执行
                                System.out.println("in finally clause");
                                if(count == 2) break; // out of "while"
                        }
                } //end of while
        } //end of main()
} //end of class
```

综合练习：ConfigException.java

```
// 打开当前目录下的dbconfig.ini文件 ( **程序不完整，自己课后修改)
public class ConfigException {
 String configDir;
public boolean configFile() throws MyException {
   Properties  props = new Properties(System.getProperties());
   configDir=   props.getProperty("user.dir");
    try{
    File config = new File(configDir,"dbconfig.ini");
    if (config.exists() ){
       props.load(new FileInputStream(config));
       return true;
    }else{
        throw new MyException(configDir+"dbconfig.ini, 配置文件不存在!");}
     } catch(IOException ex) {
        System.err.println("Login SQLException: " + ex.getMessage());
     }
```

```
public void openConfig() {
  try{
     if (configFile()){
         System.out.println("DBServer: "+dbServer +"\n DB Name:"+
    dbName +"\n User:"+ dbUser+"\n PSW:"+ dbPsw);
       }else{
         System.out.println("无法找到数据库配置文件 dbconfig.ini");
       }
     }catch(MyException ex) {
       System.err.println("Open config file Exception: " +
    ex.getMessage());
    }
}
```

```
class MyException extends Exception {
        //自定义例外
        public MyException() {  }
        public MyException(String msg)  {
           super(msg);
           }
}
```

作业：

. 阅读下面的程序错找出误？

```
class MyException {
     public String toString() {return "自定义异常";}
 }
public class Quiz1 {
    public static void main(String args［］) {
        myMathod();
    }
    myMathod() {     throw new MyException();}
}
```

改过后的程序：

```
class MyException  extends Exception {
   public String toString() {return "自定义异常";}
}
public class Quiz1 {
  public static void main(String args[]) {
    try {myMethod();     }
    catch (MyException e){
       System.out.println(e.toString());    }
  }
  static void myMethod() throws MyException  {
     throw new MyException(); }
}
```

习题和上机练习

1、解释throws和throw的区别

```
public Connection  OpenConn() throws Exception {
```

用户程序自定义的异常和应用程序特定的异常,必须借助于 throws 和 throw 语句来定义抛出异常。

1、throw是语句抛出一个异常。

语法：throw (异常对象);

throw e;

2、throws是方法可能抛出异常的声明。(用在声明方法时，表示该方法可能要抛出异常)

语法：[(修饰符)](返回值类型)(方法名)([参数列表])[throws(异常类)]{.....}

如：public void doA(int a) throws Exception1,Exception3{.....}

区别：

(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。