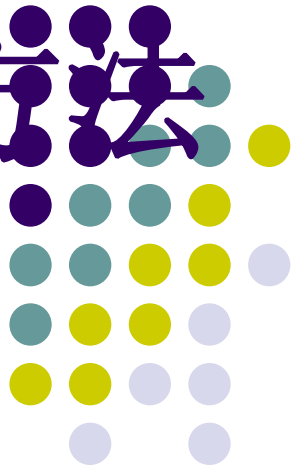
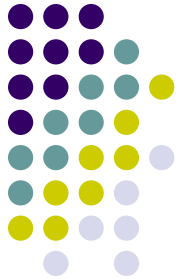
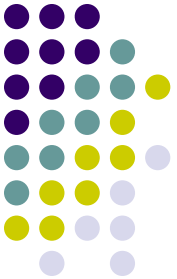
第三章 类的方法



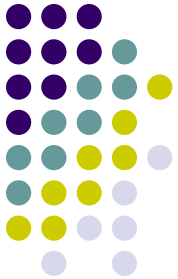


目录



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

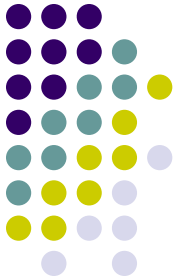
方法的控制流程

- 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 ;
}
```



3.1.1 if选择结构(续)

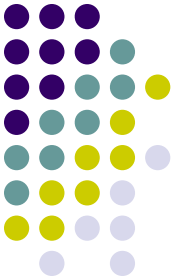
方法的控制流程

- 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



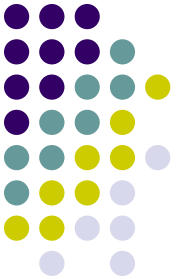
- ex3_1
  - 输入一个年份，判断它是不是闰年。
  - 闰年: 能被4整除但不能被100整除，或者能被400整除。

方法的控制流程

```
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=(new Integer(in.readLine())).intValue();
```

3.1.1 if选择结构(续)

——ex3_1.java

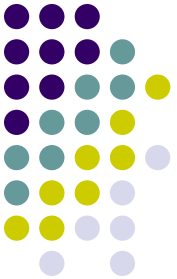


方法的控制流程

```
        IsLeapYear=((year%4==0 && year%100 != 0)||(year%400 == 0));
        if (IsLeapYear)
        {
            System.out.print(year);
            System.out.println( "is a leap year");
        }
        else
        {
            System.out.print(year);
            System.out.println( "is not a leap year");
        }
    }
}
```

3.1.1 if选择结构(续)

——ex3_2.java



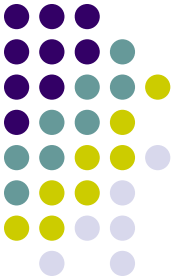
- 输入两个整数比较大小

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=(new Integer(in.readLine())).intValue();
      y=(new Integer(in.readLine())).intValue();
      if (x!=y)
        if (x>y)    System.out.println("x>y");
        else        System.out.println("x<y");
      else   System.out.println("x=y");
   }
}
```

3.1.1 if选择结构(续)

——以条件运算符代替if_else



方法的控制流程

- 例子:

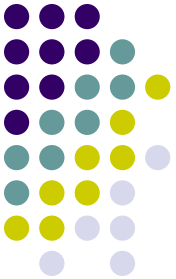
```
If (a>b)
 System.out.println("The larger one is: "+a);
else
 System.out.println("The larger one is: "+b);
```

- 用条件运算符重写:

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

3.1.1 if选择结构(续)

——例3_1



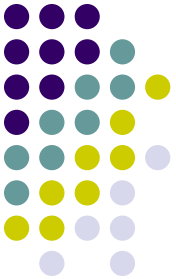
- 计算每个月的天数

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

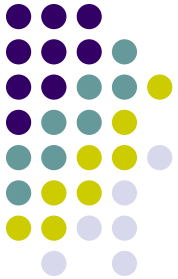


方法的控制流程

- 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

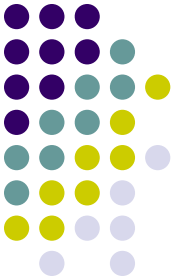
- switch语句是多分支的选择结构

```
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

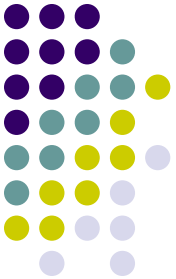


方法的控制流程

```
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

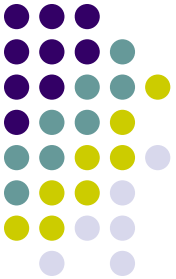


- 使用switch结构计算每个月的天数

方法的控制流程

```
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)
——例3_3



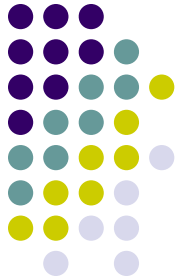
- 用switch语句实现成绩分类的功能

方法的控制流程

```
public class Grade {
    public static char gradeLevel(double g){
      int n = (int)Math.floor(g/10);
      switch (n){
          case 10:
          case 9 : return('A');
          case 8 : return('B');
          case 7 : return('C');
          case 6 : return('D');
          default: return('E');
      }
  }
```

3.1.2 switch选择结构(续)

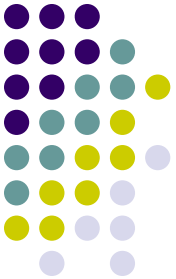
——例3_3



方法的控制流程

```
public static void main(String[] args){
    System.out.println("gradeLevel(100)="+gradeLevel(100));
    System.out.println("gradeLevel(95.5)="+gradeLevel(95.5));
   System.out.println("gradeLevel(88)="+gradeLevel(88));
   System.out.println("gradeLevel(72)="+gradeLevel(72));
   System.out.println("gradeLevel(68.5)="+gradeLevel(68.5));
   System.out.println("gradeLevel(60)="+gradeLevel(60));
   System.out.println("gradeLevel(59.5)="+gradeLevel(59.5));
   System.out.println("gradeLevel(35)="+gradeLevel(35));
}
}
```

3.1.2 switch选择结构(续)
——补充ex3_3.java

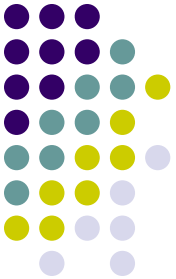


方法的控制流程

- 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）

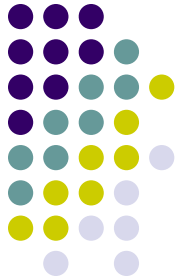
```
import java.io.*;
public class ex3_3
{
 public static void main(String[ ] args)throws IOException
 {
    int day;
    BufferedReader in =new BufferedReader(
                    new InputStreamReader(System.in));
    day=(new Integer(in.readLine())).intValue();
```

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;
         }
     }
}
```



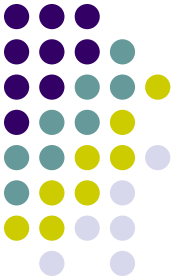
3.1.3 for循环结构

方法的控制流程

- for循环结构
  - 是Java三个循环语句中功能较强、使用较广泛的一个
  - 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;
}
```

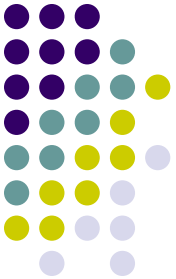
  - 解释
    - start-expression完成循环变量和其他变量的初始化工作
    - check-expression是返回布尔值的条件表达式，用于判断循环是否继续
    - update-expression用来修整循环变量，改变循环条件
    - 三个表达式之间用分号隔开



3.1.3 for循环结构(续)

方法的控制流程

- 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



3.1.3 for循环结构(续)

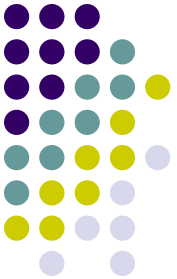
- 打印九九乘数表

方法的控制流程

```
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.print("  "+i+"*"+j+"="+i*j);
            System.out.println();
        }
    }
}
```

3.1.3 for循环结构(续)

——运行结果

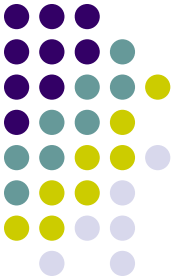


- 运行结果如下：

```
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
```

3.1.3 for循环结构(续)

——补充ex3_7.java



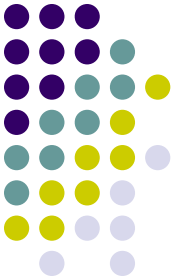
- 输入一个整数，输出它所有的因数

方法的控制流程

```
import java.io.*;
public class ex3_7
{
    public static void main(String[ ] args) throws IOException{
       int n,k;
       BufferedReader in =new BufferedReader(
                          new InputStreamReader(System.in));
      System.out.println("Enter a positive integer: ");
      n=(new Integer(in.readLine())).intValue();
      System.out.print("Number  "+n+"   Factors  ");
      for (k=1; k <= n; k++)
           if (n % k == 0)   System.out.print(k + "  ");
     System.out.println();
    }
}
```

3.1.3 for循环结构(续)

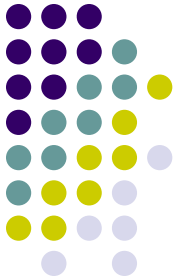
——逗号运算符



方法的控制流程

- 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);
        }
    }
}
```



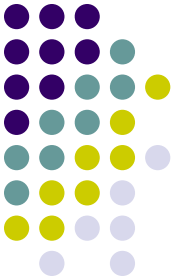
3.1.4 while循环结构

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



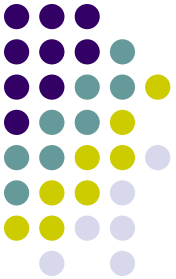
3.1.4 while循环结构(续)

- 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

方法的控制流程

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

3.1.4 while循环结构(续)
——补充ex3_4.java



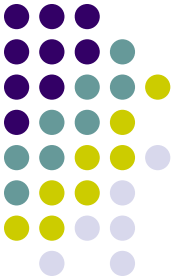
- 计算数列1,2,…,10 的和。

方法的控制流程

```
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);
    }
}
```

3.1.4 while循环结构(续)

——例3_4



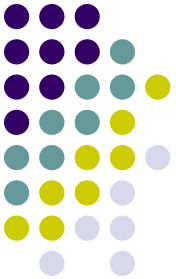
- 计算存款收益：假设银行中存款10000元，按11.25%的利率，一年后连本带利将变为11125元。你若将此款继续存入银行，试问多长时间就会连本带利翻一番

方法的控制流程

```
import java.text.*;
public class Examp3_4 {
    public static void main(String[] args){
        double original,money,interest;
        int years = 0;
        original = money= 10000;
       interest = 11.25 / 100;
       System.out.println("year  money");
```

3.1.4 while循环结构(续)

——例3_4

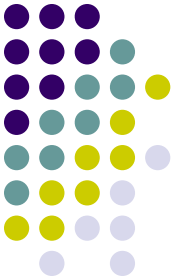


方法的控制流程

```
        while (money < 2 * original) {
                years = years + 1;
                money = money + (interest * money);
                System.out.println(" " + years + "    " +
                                new DecimalFormat("0.00").format(money));
            }
            System.out.println();
            System.out.println(" 第 " + years + " 年存款额达到 " +
                        new DecimalFormat("0.00").format(money) + "元。");
        }
    }
```

3.1.4 while循环结构(续)

——例3_4运行结果

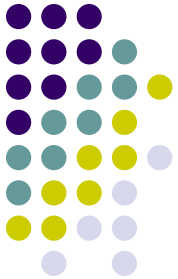


- 运行结果

方法的控制流程

```
year  money
 1    11125.00
 2    12376.56
 3    13768.93
 4    15317.93
 5    17041.20
 6    18958.33
 7    21091.14
```

第 7 年存款额达到 21091.14元。

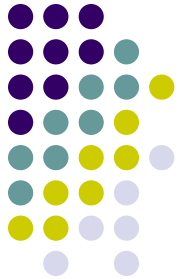


3.1.5 do-while循环结构

- 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)

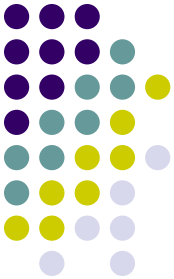
方法的控制流程

•比较这两段程序

```
//…
while(i<=10)
 {
    sum+=i;
    i++;
 }
System.out.println
            ("sum="+sum);
```

```
//…
 do  {
     sum+=i;
     i++;
  } while(i<=10);
System.out.println
            ("sum="+sum);
```

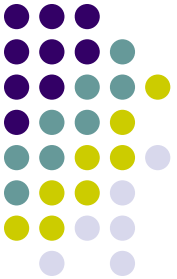
3.1.5 do-while循环结构(续)
——补充ex3_5.java



方法的控制流程

- 输入一个整数，然后输出它的翻转形式

```
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=(new Integer(in.readLine())).intValue();
     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();
  }
}
```



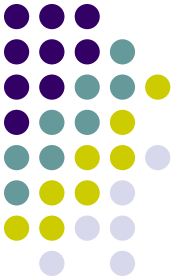
3.1.6 break语句

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 break语句(续)

——例3_5



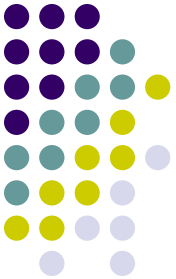
方法的控制流程

- 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_5运行结果



- 运行结果

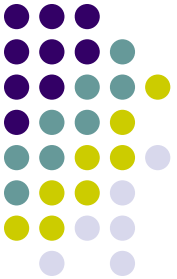
1 2 3 4

Broke out of loop at i = 5

- 解释
  - 执行break语句时，程序流程跳出for循环

3.1.6 break语句(续)

——例3_6



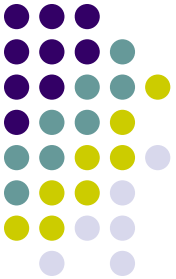
- 在嵌套循环中使用break语句：使用下面的程序来实现例3-4的九九乘法表

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)

——例3_7



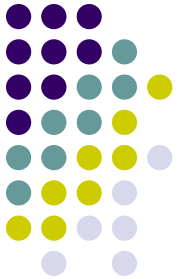
方法的控制流程

- break与label一同使用举例

```
public class Examp3_7{
    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();
      }
    }
}
```

3.1.6 break语句(续)

——例3_7运行结果

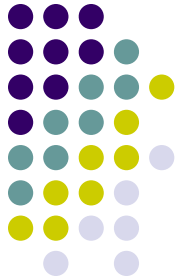


方法的控制流程

- 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



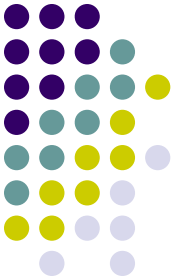
3.1.7 continue语句

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)

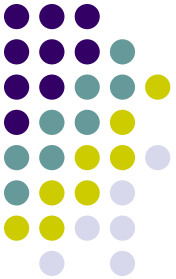
——不带标号的continue语句



- 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

方法的控制流程

3.1.7 continue语句(续)
——例3_8



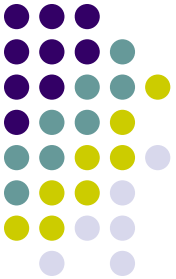
方法的控制流程

- 简单的continue语句举例

```
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)

——例3_8运行结果



方法的控制流程

- 运行结果

  1 2 3 4 6 7 8 9 10

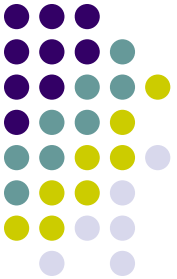
  Using continue to skip printing 5

  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

3.1.7 continue语句(续)

——例3_9



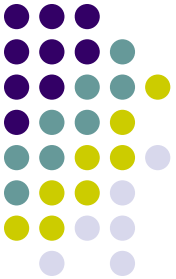
- 打印2到9之间的偶数的平方，但是不包括偶数6的平方

方法的控制流程

```
public class Examp3_9{
    public static void main (String args[])
    {
      for (int i=2; i<=9; i+=2)
        { if (i==6)
            continue;
            System.out.println(i*i);
         }
    }
}
```

3.1.7 continue语句(续)

——带标号的continue语句



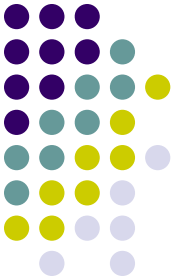
- 带标号的continue语句
  - 格式如下

    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

方法的控制流程

3.1.7 continue语句(续)

——例3_10

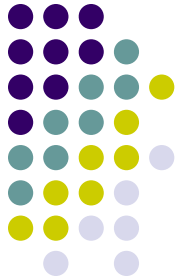


方法的控制流程

- 九九乘法表也可用下面的程序来实现

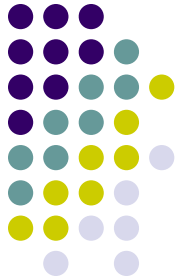
```
public class Examp3_10{
    public static void main (String args[]) {
        outer:
        for (int i=1; i<10; i++){
          inner:
          for (int j=1; j<10; j++){
              if (i<j){
                 System.out.println();
                continue outer;
              }
            System.out.print("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



3.2 异常处理简介

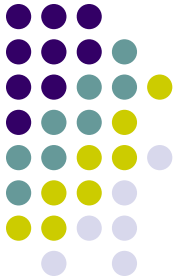
- 异常处理
  - 在进行程序设计时，错误的产生是不可避免的。所谓错误，是在程序运行过程中发生的异常事件，这些事件的发生将阻止程序的正常运行
  - 如何处理错误？把错误交给谁去处理？程序又该如何从错误中恢复？
  - 为了加强程序的鲁棒性，Java语言具有特定的运行错误处理机制



3.2.1 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
  - 是面向对象规范的一部分，是异常类的对象
  - 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



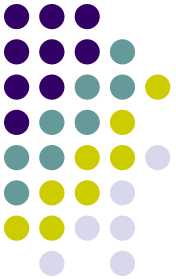
3.2.1 异常处理的意义(续)

异常处理简介

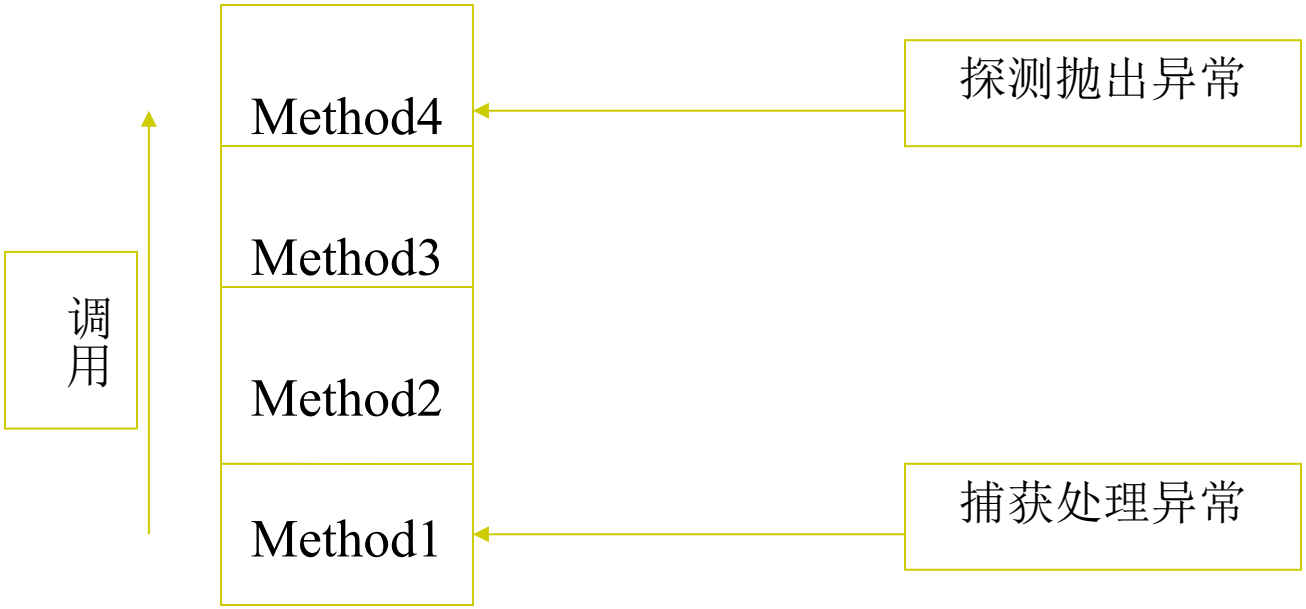
- java处理错误的方法
  - 抛出(throw)异常
    - 在方法的运行过程中，如果发生了异常，则该方法生成一个代表该异常的对象并把它交给运行时系统，运行时系统便寻找相应的代码来处理这一异常
  - 捕获(catch)异常
    - 运行时系统在方法的调用栈中查找，从生成异常的方法开始进行回溯，直到找到包含相应异常处理的方法为止

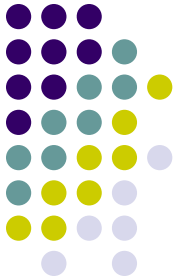
3.2.1 异常处理的意义(续)

——异常处理示意图



异常处理简介



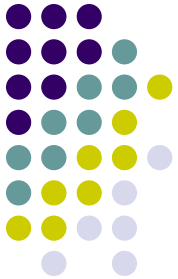


3.2.1 异常处理的意义(续)

- Java异常处理机制的优点

异常处理简介

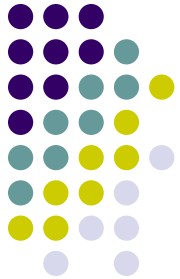
- 将错误处理代码从常规代码中分离出来
- 按错误类型和差别分组
- 对无法预测的错误的捕获和处理
- 克服了传统方法的错误信息有限的问题
- 把错误传播给调用堆栈



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)

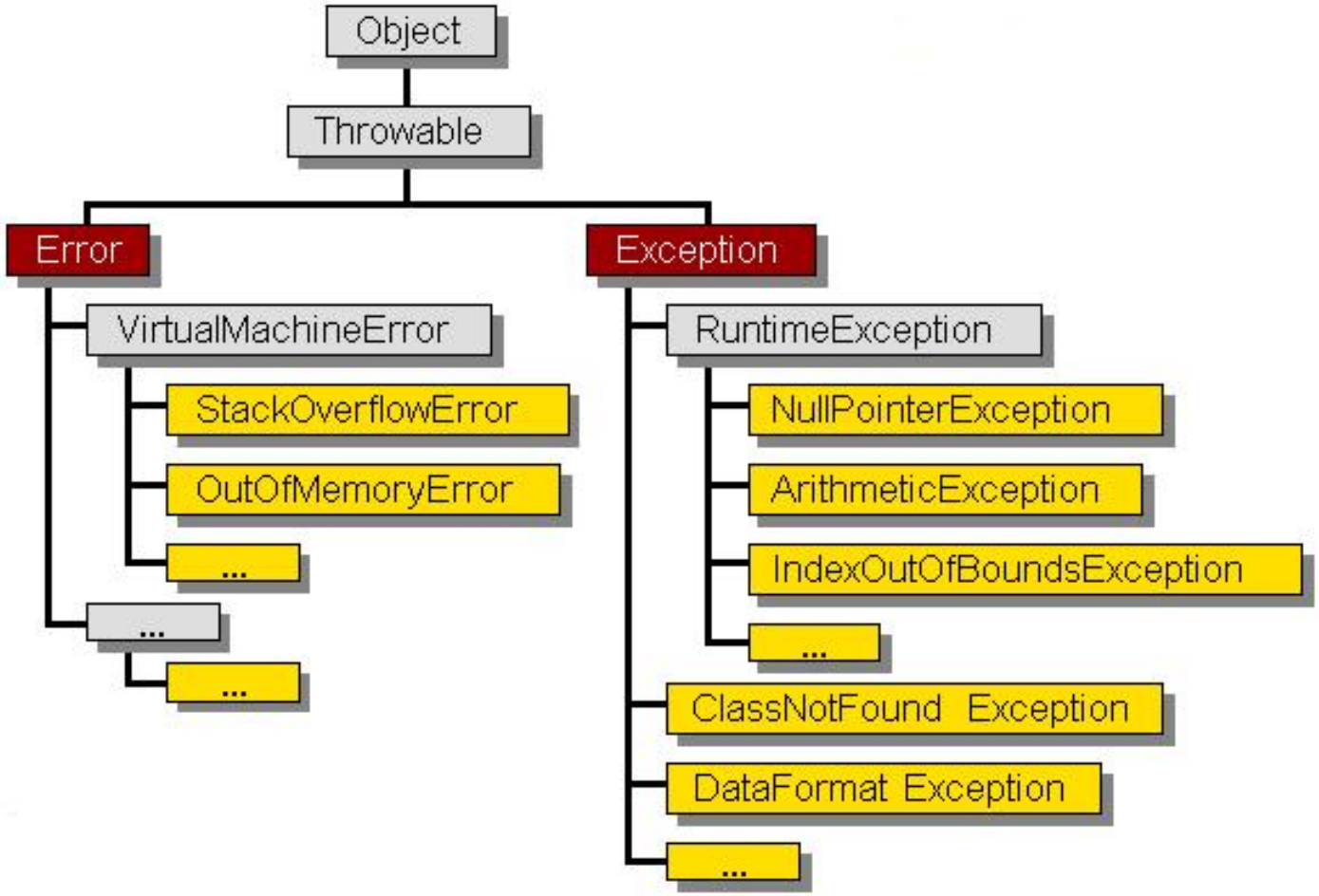
- 再对异常进行分类
  - 非检查型异常
    - 不能期望程序捕获的异常(例如数组越界，除零等)
    - 继承自RuntimeException
    - 在方法中不需要声明，编译器也不进行检查
  - 检查型异常
    - 其他类型的异常
    - 如果被调用的方法抛出一个类型为E的检查型异常，那么调用者必须捕获E或者也声明抛出E（或者E的一个父类），对此编译器要进行检查

异常处理简介

3.2.2 错误的概念(续)

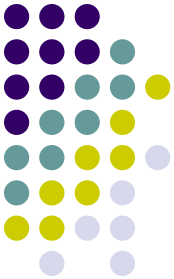
——异常和错误类的层次结构

异常处理简介



3.2.2 错误的概念(续)

——预定义的一些常见异常



异常处理简介

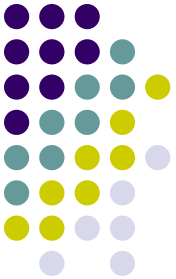
- Java预定义的一些常见异常
  - ArithmeticException
    - 整数除法中除数为0
  - NullPointerException
    - 访问的对象还没有实例化
  - NegativeArraySizeException
    - 创建数组时元素个数是负数
  - ArrayIndexOutOfBoundsException
    - 访问数组元素时，数组下标越界
  - ArrayStoreException
    - 程序试图向数组中存取错误类型的数据
  - FileNotFoundException
    - 试图存取一个并不存在的文件
  - 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++;
       }
    }
}
```

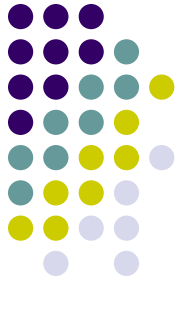
3.2.2 错误的概念(续)

——例3_11运行结果

- 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

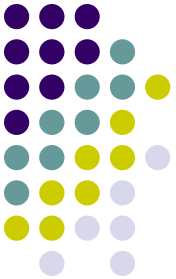
- 对于检查型异常，Java强迫程序必须进行处理。处理方法有两种：

异常处理简介

  - 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
  - 捕获异常
    - 使用try{}catch(){}块，捕获到所发生的异常，并进行相应的处理

3.2.3 异常的处理(续)

——声明抛出异常

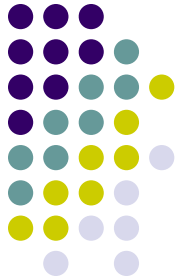


异常处理简介

- 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

3.2.3 异常的处理(续)

——一个例子



异常处理简介

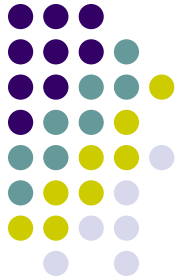
```
public void openThisFile(String fileName)
    throws java.io.FileNotFoundException {
        //code  for method
 }

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

- 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

3.2.3 异常的处理(续)

——捕获异常



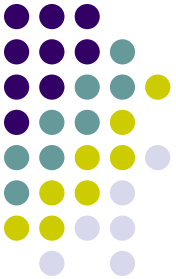
- 语法格式

异常处理简介

```
try {
    statement(s)
} catch (exceptiontype name) {
    statement(s)
} finally {
    statement(s)
}
```

3.2.3 异常的处理(续)

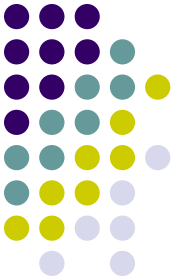
——捕获异常



异常处理简介

- 说明
  - try 语句
    - 其后跟随可能产生异常的代码块
  - catch语句
    - 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
  - finally语句
    - 不论在try代码段是否产生异常，finally 后的程序代码段都会被执行。通常在这里释放内存以外的其他资源
- 注意事项
  - 在类层次树中，一般的异常类型放在后面，特殊的放在前面

3.2.3 异常的处理(续)
——例3_12

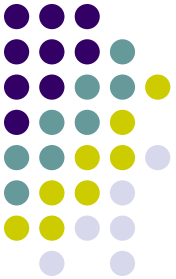


- 读入两个整数，第一个数除以第二个数，之后输出

异常处理简介

```
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



异常处理简介

- 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";}
}
}
```

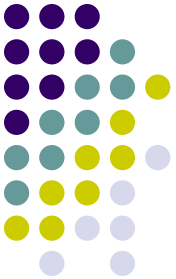
3.2.3 异常的处理(续)

——例3_12运行结果

- 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)
```

3.2.3 异常的处理(续)
——例3_13



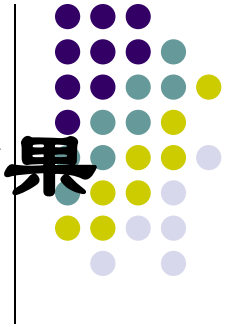
异常处理简介

- 捕获 NumberFormatException类型的异常

```
import java.io.*;
public class ExceptionTester2  {
    public static void main(String args[]) {
        int number1=0, number2=0;
        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();
        }
        catch (NumberFormatException e) {
            System.out.println("Those were not proper integers!  I quit!");
            System.exit(-1);
        }
          System.out.print(number1 + " / " + number2 + "=");
        int result = number1 / number2;
        System.out.println(result);
    }
}
```

3.2.3 异常的处理(续)

——例3_13运行结果

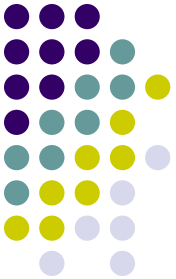


- 运行结果

异常处理简介

Enter the first number:
abc
Those were not proper integers! I quit!

3.2.3 异常的处理(续)
——例3_14



异常处理简介

- 捕获被零除的异常（ArithmeticException类型的异常）

```
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) {
    System.out.println("Invalid integer entered!");
    System.exit(-1);
}
catch (ArithmeticException e) {
    System.out.println("Second number is 0, cannot do division!");
    System.exit(-1);
}
```

3.2.3 异常的处理(续)

——例3_14运行结果

- 运行结果

异常处理简介

```
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

```
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.");
         }
      }
   }
```

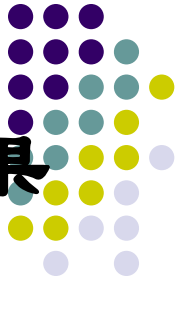
3.2.3 异常的处理(续)

——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!");
     }
  }
}
```

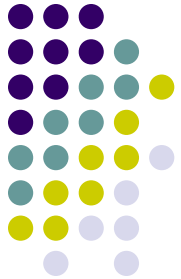
3.2.3 异常的处理(续)

——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
    - throw new ThrowableObject();
    - ArithmeticException e = new ArithmeticException();
      throw e;

3.2.4 生成异常对象(续)

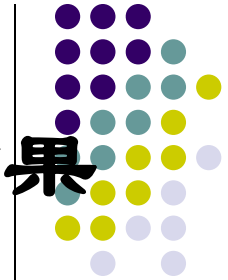
——例3_16

- 生成异常对象举例

```
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);
        }
    }
}
```

3.2.4 生成异常对象(续)

——例3_16运行结果

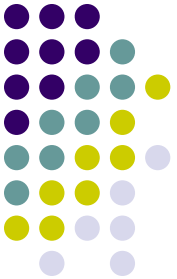


- 运行结果

java.lang.ArithmeticException

java.lang.ArrayIndexOutOfBoundsException

java.lang.StringIndexOutOfBoundsException



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);
    }
```

3.2.5 声明自己的异常类(续)

——例3_17

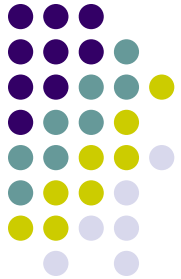
```
public static void main(String args[]) {
        int number1=0, number2=0, result=0;
        try {
          System.out.println("Enter the first number:");
          number1 = Integer.valueOf(Keyboard.getString()).intValue();
          System.out.println("Enter the second number:");
          number2 = Integer.valueOf(Keyboard.getString()).intValue();
          result = 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);
    }
}
```

3.2.5 声明自己的异常类(续)

——例3_17运行结果

- 运行结果如下

```
Enter the first number:
140
Enter the second number:
0
DivideByZeroException: Attempted to divide by
    zero
```

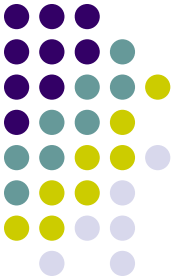


3.3 方法重载

- 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

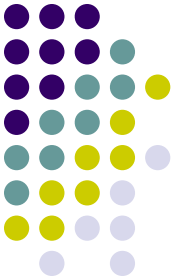


- 通过方法重载分别接收一个或几个不同数据类型的数据

```
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)

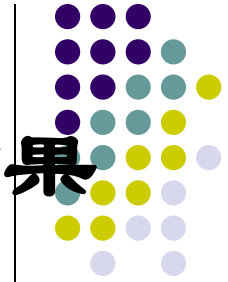
——例3_18



```
    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?");
    }
}
```

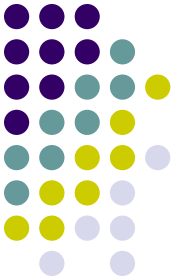
3.3 方法重载(续)

——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