Java基本语法

Wuhan University

java第一步

- 在Java中，一个源程序文件被称为一个编译单元（compilation unit）。它是一个包含一个或多个类定义的文本文件。Java编译器要求源程序文件使用.java文件扩展名。
- 在Java中，所有的代码都必须驻留在类中。按照约定，类名必须与源程序的文件名相同。

例子

```
import java.io.*;
public class HelloWorld{
   public static void main(String[] arg)
   {
      String test = "Hello World!";
      System.out.println(test);
   }
}
```

Java的标识符

- Java程序设计中，标识符用来对程序中的变量、方法、对象、类、接口、以及包等进行命名
- Java语言的标识符命名必须遵循以下原则：
  - 标识符必须是以字母、下划线(_)、美元符($)开始的一个字符序列；
  - 除第一个字符外，标识符可以由字母、数字、下划线(_)、美元符($)开始的一个字符序列；
  - 标识符对大小写敏感
  - 标识符没有最大长度限制
  - 标识符中间不能有空格和连字符(-)
  - Java语言的关键字不能用作标识符。

```
public class HelloWorld {
    ..........................
}
```

- **public**是关键字，表示访问权限，可以被其他类引用。
- **class** 是关键字，表示定义一个类。
- **HelloWorld**是标识符，给出类的名字。
- **Java**有许多关键字，它们具有特殊意义和用法，不能作为标识符。

Java的关键字

- Java语言中，有一部分标识符是系统定义的，有着专门的意义和用途，不能用于一般的标识符，这些标识符就叫做保留字或关键字。

abstract assert boolean break byte case
catch char class const continue default do
double else extends final finally float for goto
if implements import instanceof int interface
long native new package private protected
public return short static strictfp super
switch synchronized this throw throws
transient try void volatile while

在Java中，true（真）、false（假）和null（空值）都是小写的，它们不是Java的关键字，但是在程序中不能把它们作为名字使用。

关键字

- 所有的关键字都是小写的。如果被大写，就不是关键字了。
- 用于数据类型的关键字
  - **byte short int long float double char boolean**
- 用于流程控制语句的关键字
  - **if else switch case default do while for break continue**
- 方法、类型、变量的修饰关键字
  - **private public protected final static abstract synchronized volatile**
- 异常处理关键字
  - **try catch finally throw throws**

关键字

- 对象相关关键字
  - **new extends implements class instanceof this super**
- 字面值常量关键字
  - **false true null**
- 方法相关关键字
  - **return void**
- 包相关关键字
  - **package import**

关键字

- 对象相关关键字
  - **new extends implements class instanceof this super**
- 字面值常量关键字
  - **false true null**
- 方法相关关键字
  - **return void**
- 包相关关键字
  - **package import**

Java的标识符

- 类名一般是名词，包括大小写字母，每个单词的首字母大写。
- 方法名一般是动词，包括大小写字母，第一个单词的首字母小写，其余单词首字母大写。尽量不要在方法名中使用下划线。
- 常量一般用大写字母表示，单词与单词之间用下划线分割。
- 变量也使用混合大小写形式，第一个单词的首字母大写。变量名中避免使用下划线、美元符号。

Java的标识符

- Java标识符准则
  - 类名、接口名：名词，首字母大写，内含的单词首字母大写AppletInOut
  - 方法名：动词，首字母小写，内含的单词首字母大写actionPerformed
  - 变量名：名词，首字母小写，内含的单词首字母大写connectNumber
  - 常量名：全部大写，单词间用下划线分开HEAD_COUNT

Java的标识符

- Java标识符
  - StudentName
  - get_up
  - _sys_path
  - $pay
  - $9test
  - 测试
  - 7go
  - #super
  - I’am
  - public
  - get-name

不合法的Java标识符

数据类型

- 数据类型就是对内存位置的抽象表达，数据类型指明了变量或者表达式的状态和行为
- **Java**中的数据类型分为简单类型和复合数据类型
  - 简单类型：是不能再简化的、内置的数据类型，由编程语言定义。
  - 复合数据类型：由简单数据类型的组合形成的更大更复杂的数据类型

简单类型类型	布尔数据类(boolean)	1位
	字符类型	char(2字节)
	整数类型	byte(1字节)
		short(2字节)
		int(4字节)
		long(8字节)
	浮点数	float(4字节)
		double(8字节)
复合数据类型	类	
	接口	
	数组	

整数类型及取值范围

名称	变化范围
byte(1字节)	**–128~127**（-2^7~ 2^7-1 ）
short(2字节)	**–32,768 ~ 32,767**（-2^{15}~ $2^{15}-1$ ）
int(4字节)	**–2,147,483,648 ~ 2,147,483,647**（-2^{31}～ $2^{31}-1$ ）
long(8字节)	**–9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807**

在表示**long**型常量时，需要在数字后面加上后缀**L**或者**l**。例如3L表示一个long型的常量，而不是int型常量。

浮点数类型及取值范围

名称	变化范围
float(4字节)	**-3.403E+038～3.403E+038**
double(8字节)	**-1.798E+308～1.798E+308**

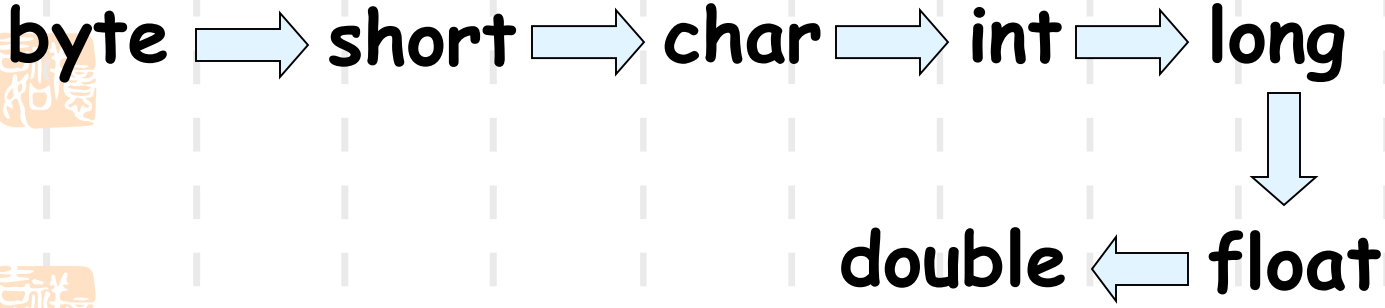
一个浮点数隐含为double型。在一个浮点数后加字母F或f，表示float型。常量值3.45的类型是double；3.45F的类型是float。

数据类型

- **Java**中的所有数字变量都是有符号的
- 每个变量有类型，每个表达式有类型，而且每种类型是严格定义的。
- **Java**不允许数据类型之间随意的转换，只有数字变量之间可以进行类型转换。
- **Java**编译器对所有的表达式和参数都要进行类型相容性的检查以保证类型是兼容的。任何类型的不匹配都是错误的，在编译器完成编译以前，错误必须被改正。

数据类型的转换

- Java中的整型、实型和字符型被视为同一类数据，这些类型由低级到高级的优先关系如下：



- 低优先级的变量可以直接转换为高优先级的变量，编译器会自动进行类型转换

简单数据类型

- **Java提供了Byte, Short, Boolean, Character, Integer, Double, Float和Long等内置的封装类**
- 这些封装类提供了很直观实用的方法

简单数据类型	封装类
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

简单数据类型

- **Java**中，简单数据类型作为类的成员变量声明时自动初始化为默认值。
- 如果简单类型被声明为局部变量时，必须显式地对其进行初始化，否则会导致编译器报错。

简单数据类型	默认值
boolean	false
char	null
byte	0
short	0
int	0
long	0
float	0.0
double	0.0

复合数据类型

- 类
  - 定义了属性和方法的数据类型
  - 定义class类型的变量只能引用类的实例或者null

```
Catfish fish = new Catfish();    //合法
Catfish fish = null;             //合法
Catfish fish = 0;                //不合法
```

复合数据类型

- 数组
  - 是动态创建的引用对象
  - 数组只能引用数组的实例或者null
  - 数组对象由元素构成，元素的个数可以为0，这种情况下称数组为空
  - 所有的数组都是从0开始对元素编号

```
int[] array1 = new int[64];

int[] array1 = null;
```

```
class Array {
    public static void main(String args[]) {
        int month_days[];
        month_days = new int[12];
        month_days[0] = 31;
        month_days[1] = 28;
        month_days[2] = 31;
        month_days[3] = 30;
        month_days[4] = 31;
        month_days[5] = 30;
        month_days[6] = 31;
        month_days[7] = 31;
        month_days[8] = 30;
        month_days[9] = 31;
        month_days[10] = 30;
        month_days[11] = 31;
        System.out.println("April has " + month_days[3] + "
        days.");
        }
}
```

```
class Matrix {
    public static void main(String args[]) {
        double m[][] = {
            { 0*0, 1*0, 2*0, 3*0 },
            { 0*1, 1*1, 2*1, 3*1 },
            { 0*2, 1*2, 2*2, 3*2 },
            { 0*3, 1*3, 2*3, 3*3 }
        };
        int i, j;
        for(i=0; i<4; i++) {
            for(j=0; j<4; j++)
                System.out.print(m[i][j] + " ");
            System.out.println();
        }
    }
}
```

```
0.0 0.0 0.0 0.0
0.0 1.0 2.0 3.0
0.0 2.0 4.0 6.0
0.0 3.0 6.0 9.0
```

常量

- **Java中的常量值是用文字表示的，它区分不同的数据类型，如：**
  - **整型常量：123**
  - **实型常量：1.23**
  - **字符常量：‘a’**
  - **布尔常量：true, false**
  - **字符串常量：“Hello Java”**

变量 (Variables)

- 程序执行过程中，**值可以改变的量**
  - 整型变量、实型变量、字符型变量、字符串变量、布尔变量等
- **变量是Java的基本存储单元，它的定义包括变量名、变量类型和作用域几个部分**
  - **变量名是一个合法的标识符**
  - **变量类型可以为任一种数据类型**
  - **变量的作用域指明可访问该变量的代码段，声明一个变量的同时也就是指明了变量的作用域。**

变量 (Variables)

- 变量定义（变量的声明）格式
  - **type identifier[=value][, identifier[=value]…];**
    - 类型 变量名[=初值][, 变量名[=初值] …];
  - 类型: 基本数据类型或引用类型

```
int          x, y, z;
float        a, b;
char         c1, c2, c3;
double       d1;
boolean   mycom;
int a, b=1, c;
String str1, str2 = "Hello String!";
```

变量 (Variables)

- 变量赋初值/初始化
- 在变量声明时赋值

1. 类型 变量名 [=初值][, 变量名[=初值] …]
2. int x=1, y=2, z=3;
3. float e = 2.718281828f;

- 也可以在声明后采用赋值语句

1. float pi, y;
2. pi = 3.1415926f;
3. y = 2.71828f;

变量

- 按照作用域分，变量可以分为：
  - 局部变量：在方法或方法的一块代码中声明，它的作用域为它所在的代码块
  - 类成员变量：在类中声明，而不是在类的某个方法中声明，它的作用域是整个类
  - 方法参数：传递给方法，它的作用域就是这个方法
  - 异常处理参数：传递给异常处理代码，其作用域为异常处理部分

运算符

- 运算符指明对操作数进行的运算
- 按照运算符的功能分
  - 算术运算符(+,-,*,/,%,++,--)
  - 关系运算符(>,<,>=,<=,==,!=)
  - 布尔逻辑运算符(!,&&,||)
  - 位运算符(>>,<<,>>>,&,|,^,~)
  - 赋值运算符(=,+=等)
  - 条件运算符(?:)
  - 其他(包括下标运算符[ ], 内存分配运算符new等)

算术运算符

运算符	含义	例子
+	加	3+4
-	减	5-7
*	乘	4*4
/	除	14/8
%	模	14%8 37.2%10

算术表达式

表达式	等价式
x+=y	x=x+y
x-=y	x=x-y
x*=y	x=x*y
x/=y	x=x/y
x%=y	x=x%y

关系表达式

运算符	含义	例子
==	等于	x==3
!=	不等于	x！=3
<	小于	x<3
>	大于	x>3
<=	小于等于	x<=3
>=	大于等于	x>=3

逻辑表达式

运算符	含义
!	逻辑非
&&	逻辑与
||	逻辑或

逻辑表达式

- 逻辑运算符(Logical Operators)
  - 操作数的逻辑关系，计算结果“true”或“false”
    - 逻辑与 && “op1 && op2”
      1. 操作数都为真“true”，结果为真“true”
      2. 否则结果为假“false”
    - 逻辑或 || “op1 || op2”
      1. 有一个操作数为真“true”，结果为真“true”
      2. 否则结果为假“false”
    - 逻辑非 ！ “! op”
      1. 取反，操作数为真“true”，结果为假“false”，反之……
  - 优先级
    - (!) > (&&) > (||)
    - (!)>算术运算符>关系运算符>(&&) > (||)

自增、自减运算符

- 变量赋值，一元运算符
- 自增运算符(++)、自减运算符(--)
  - int i=5; i++; ++i; i--; --i;
  - “赋值”和“运算”的先后顺序

```
float x =7, y=15, v1, v2;
v1 = x++;
v2 = ++y;
```

```
x=8    y=16
v1=7  v2=16
```

```
int i = 10;
int n = i++%5;
```

i = 11, n = 0

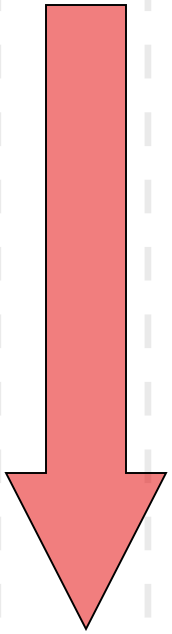
```
int i = 10;
int n = ++i%5;
```

i = 11, n = 1

```
int i=3;
j=++i; i的值先变成4，再赋给j，j的值为4
j=i++; 先将i的值3赋给j，j的值为3，然后i变成4
```

运算符优先级

- ()
- []
- ++, --
- *, /, %, +, -
- ==, !=
- &&, ||, ? :,
- =, +=, -=, *=, /=, %=



高

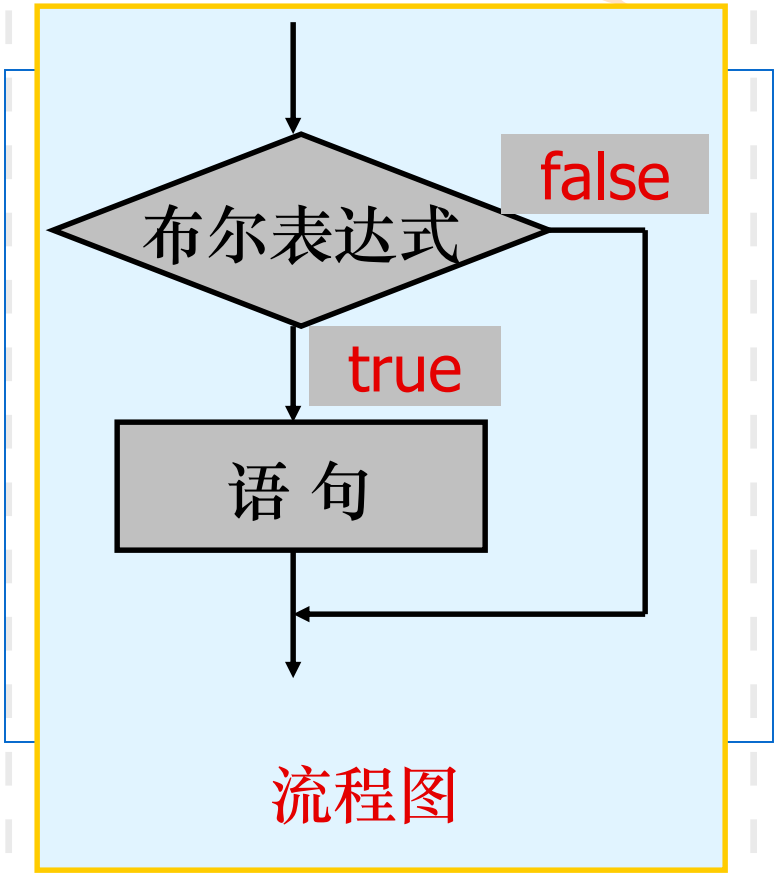
低

流程控制

- **Java中的流控制语句**
  - 分支语句：**if…else, switch, break, return**
  - 循环语句：**while, do…while, for, continue**
  - 异常处理语句：**try…catch…finally, throw**

if语句

- 布尔表达式是任意一个返回布尔型数据的表达式
- 每个单一的语句后面必须有分号
- 语句**statement1**和**statement2**可以为复合语句



if语句

```
if(i>1)
   System.out.println("i>1");
else
   System.out.println("i<=1");
```

```
if(i>1){
   i=0;
   System.out.println("i>1");
}
else
   System.out.println("i<=1");
```

嵌套if语句

- 条件表达式从上而下被求值，一旦找到为真的条件，则执行与它关联的语句，而其他的部分被忽略了。
- 如果所有的条件都不为真，则执行最后的**else**语句。

```
if(条件)
        statement1;
else if(条件)
        statement2;
……
else statementn;
```

```
class IfElse {
    public static void main(String args[]) {
         int month = 4; // April
        String season;
        if(month == 12 || month == 1 || month == 2)
                season = "Winter";
        else if(month == 3 || month == 4 || month == 5)
                 season = "Spring";
        else if(month == 6 || month == 7 || month == 8)
                season = "Summer";
        else if(month == 9 || month == 10 || month == 11)
                season = "Autumn";
        else
                season = "Bogus Month";
        System.out.println("April is in the " + season + ".");
      }
}
```

switch语句

- switch语句是java的多路分支语句
  - 它提供一种基于一个表达式的值来使程序执行不同部分的简单方法
  - 它提供了一个比一系列if-else-if语句更好的选择

switch语句

```
switch(expression){
    case value1:
        //statement1
    break;
      ……
    case valueN:
        //statementN
     break;
    default:
        //default statement
}
```

表达式必须为byte、short、int或char类型

每个case语句后的值value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 SampleSwitch {
    public static void main(String ar
        for(int i=0; i<6; i++)
            switch(i){
                case 0:
                  System.out.println("i
                    break;
                case 1:
                      System.out.println("i is one");
                    break;
                case 2:
                    System.out.println("i is two");
                    break;
                default:
                      System.out.println("i is greater
                     than two");
                    break;}    }
}
```

```
i is zero
i is one
i is two
i is greater than two
i is greater than two
i is greater than two
```

while语句

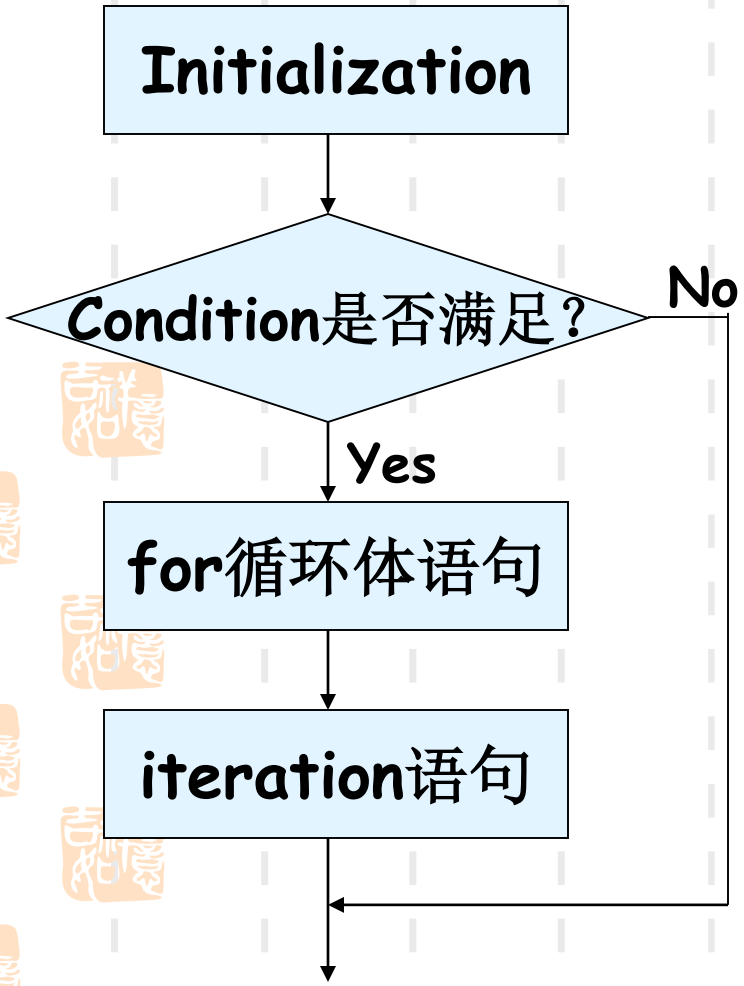
- 条件可以是任何布尔表达式。只要条件表达式为真，循环体就被执行。当条件为假时，程序控制就传递到循环后面紧跟的语句行。
- 如果只有单个语句需要重复，则不需要大括号，否则需要大括号。

```
while(条件)
        statement;
```

```
class While {
    public static void main(String args[]) {
        int n = 10;
        while(n > 0) {
            System.out.println("Number" + n);
            n--;
         }
        n = 10;
        while(n-- > 0);   //while体可以为空语句
    }
}
```

```
Number10
Number9
Number8
Number7
Number6
Number5
Number4
Number3
Number2
Number1
```

for语句




```
for( initialization;
   condition;
   iteration) {
    // body
}
```

for (初始化表达式；循环条件表达式；循环后的操作表达式) {

执行语句;

}

```
int n;
for(n=10; n>0; n--)
            System.out.println("number
```

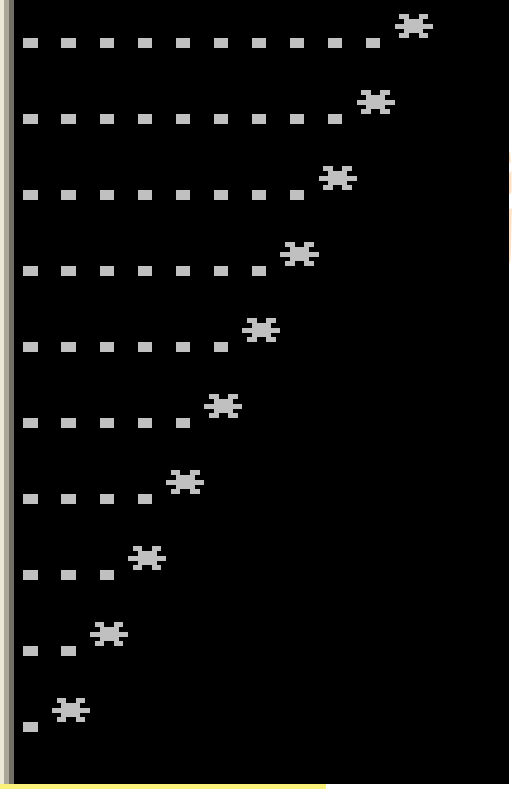
```
tick 10
tick 9
tick 8
tick 7
tick 6
tick 5
tick 4
tick 3
tick 2
tick 1
```

```
//在for循环中声明循环控制变量
for(int n=10; n>0; n--)
    System.out.println("tick " + n);
}
```

```
//使用逗号
for(a=1, b=4; a<b; a++, b--) {
    System.out.println("a = " + a);
    System.out.println("b = " + b);
}
```

```
a = 1
b = 4
a = 2
b = 3
```

for语句的嵌套循环



```
// Loops may be nested.
class Nested {
    public static void main(String a
        int i, j;
        for(i=0; i<10; i++) {
            for(j=i; j<10; j++)
                System.out.print(".");
            System.out.println("*");
        }
    }
}
```

跳转语句

- Java 支持三种跳转语句
  - break
  - continue
  - return

  这些语句把控制转移到程序的其他部分
- Java还支持另一种能改变程序执行流程的方法:通过异常处理。异常处理提供了一种结构化的方法，通过该方法可以使你的程序捕获并处理运行时刻错误。它由五个关键字来控制：try，catch，throw，throws和 finally。

使用break语句

- 使用**break**语句可以强行退出循环，忽略循环体中的任何其他语句和循环的条件测试。在循环中遇到**break**语句时，循环被终止，程序控制在循环后面的语句重新开始。
- **break**语句能用于任何 **Java**循环中，包括人们有意设置的无限循环。

使用break语句

```
class BreakLoop {
      public static void main(String arg
            for(int i=0; i<100; i++) {
                  if(i == 10) break;
                  System.out.println("i: " + i);
             }
            System.out.println("Loop complete.");
       }
}
```

```
i: 0
i: 1
i: 2
i: 3
i: 4
i: 5
i: 6
i: 7
i: 8
i: 9
Loop complete.
```

```
//在一系列嵌套循环中使用b
//的循环。
class BreakLoop {
    public static void main(String args[]) {
        for(int i=0; i<3; i++) {
            System.out.print("Pass " + i + ": ");
            for(int j=0; j<100; j++) {
                if(j == 10) break;
                System.out.print(j + " ");
            }
            System.out.println("!");
        }
        System.out.println("Loops complete.");
}
}
```

```
Pass 0: 0 1 2 3 4 5 6 7 8 9 !
Pass 1: 0 1 2 3 4 5 6 7 8 9 !
Pass 2: 0 1 2 3 4 5 6 7 8 9 !
Loops complete.
```

使用break语句

- 注意
  - 首先，一个循环中可以有一个以上的**break**语句。但要小心，太多的**break**语句会破坏代码结构。
  - 其次，**switch**语句中的**break**仅仅影响该**switch**语句，而不会影响其中的任何循环。
  - **break**不是被设计来提供一种正常的循环终止的方法。循环的条件语句是专门用来终止循环的。只有在某类特殊的情况下，才用**break**语句来取消一个循环。

continue语句

- continue强迫一个循环提早反复，也即，继续运行循环，但是要忽略这次重复剩余的循环体的语句。
- continue语句是break语句的补充
  - 在while循环中，continue语句使控制直接转移给控制循环的条件表达式，然后继续循环过程。
  - 在for循环中，循环的反复表达式被求值，然后执行条件表达式，循环继续执行。

continue语句

```
class ContinueTest {
    public static void main(String args[]) {
        for(int i=0; i<10; i++) {
            System.out.print(i + " ");
            if (i%2 == 0) continue;
            System.out.println("*");
            }
    }
}
```

return语句

- return语句用来明确地从一个方法返回，也就是，return语句使程序控制返回到调用它的方法。
- 在一个方法的任何时间，return语句可被用来使正在执行的分支程序返回到调用它的方法。

return语句

```
class ReturnStatement {
    public static void main(String args[]) {
        boolean t = true;
        System.out.println("Before the
                return.");
        if(t) return; // return to caller
        System.out.println("This won't
                execute.");
    }
}
```