《Java 讲义》

教师：王明军

武汉大学资源与环境科学学院

地理信息科学系

Chap1. 绪论

1、Java 历史及发展

Java 是一种解释型的、面向对象的编程语言。历史：1991 年，SUN MicroSystem 公司的 Jame Gosling、Bill Joe 等人为在电视、控制烤箱等家用消费类电子产品上进行交互式操作而开发了一个名为 Oak 的软件。

发展：面向网络应用，类库不断丰富，性能不断提高，应用领域不断拓展。（1994 年以来）

应用：适于开发各种应用，尤其是基于网络的应用、嵌入式应用等。**2、Java 语言特点**

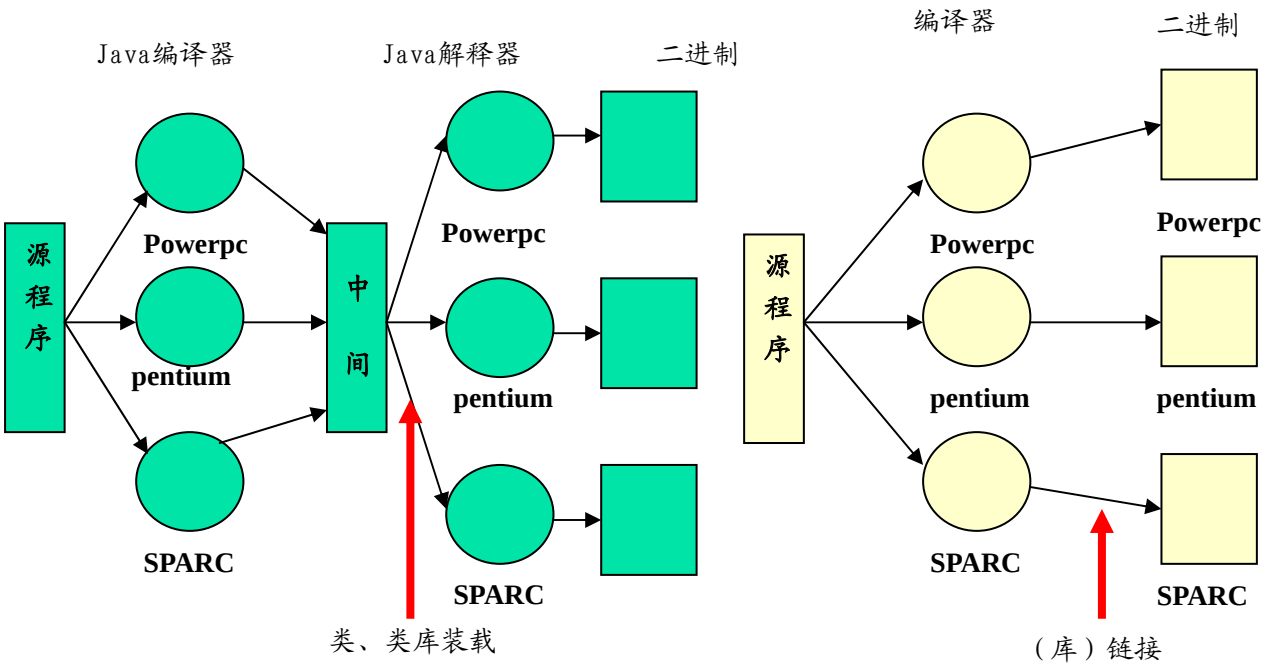
Java= “C++” – “复杂性和奇异性” + “安全性和可移植性”**(1) 面向对象**

Java 语言的设计集中于对象及其接口，它提供了简单的类机制以及动态的接口模型。对象中封装了它的状态变量以及相应的方法，实现了模块化和信息隐藏；而类则提供了一类对象的原型，并且通过继承机制，子类可以使用父类所提供的方法，实现了代码的复用。

(2) 操作平台无关性

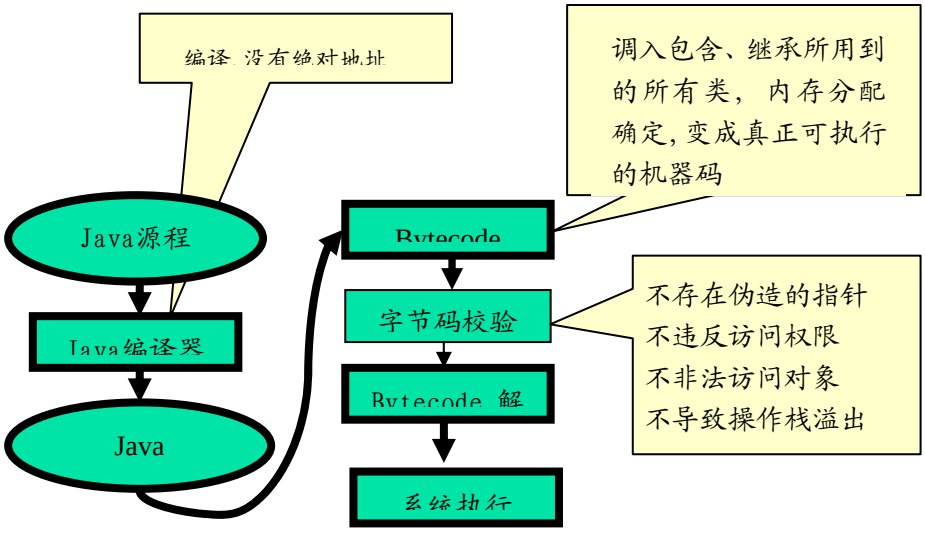
严格的语言定义：没有“依据机器的不同而不同” 或“由编译器决定”等字眼，最后的目标码都是一致的。

编译型和解释型语言的工作模式



Java 解释器生成与体系结构无关的字节码指令，只要安装了 Java 运行时系统，Java 程序就可在任意的处理器上运行。这些字节码指令对应于 Java 虚拟机中的表示，Java 解释器得到字节码后，对它进行转换，使之能够在不同的平台运行。

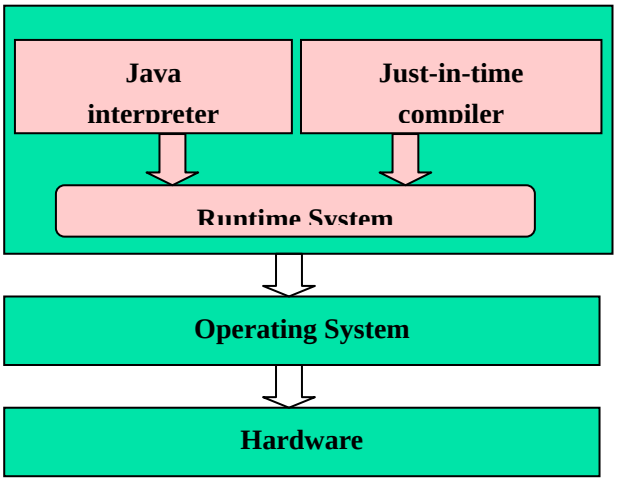
Java的编译与执行



不同的操作系统有不同的虚拟机。它类似一个小巧而高效的 CPU。Bytecode 代码是与平台无关的是虚拟机的机器指令。Java 字节代码运行的两种方式:

interpreter(解释方式)

Just-in-time(即时编译):有代码生成器将字节代码转换成本机的机器代码，然后可以以较高速度执行.



(3) 安全问题

Java 是在网络环境下使用的语言，一个安全的网络至少要防止以下几种破坏的可能性:

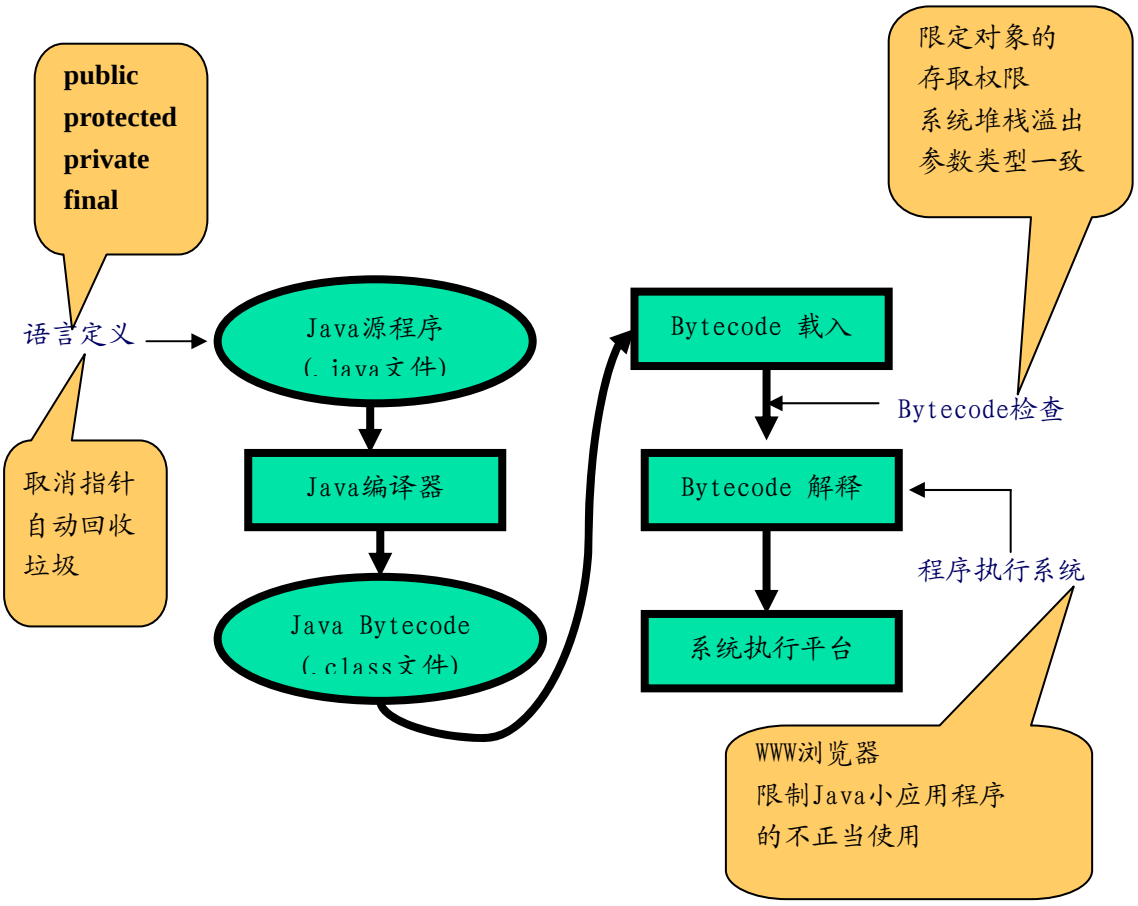
- 毁灭系统资源
- 消耗系统资源
- 挖掘系统或个人机密
- 骚扰正常工作的进行 Bytecode 的运行
- 加载代码

  由 class(Bytecode) loader 完成。
- 校验代码

  由 Bytecode verifier 完成。
- 执行代码

  由 runtime interpreter 完成。

Java的安全措施



(4) 多线程

Java 提供现成的类 Thread，只要继承这个类就可以编写多线程的程序。多线程机制使应用程序能够并行执行，而且同步机制保证了对共享数据的正确操作。通过使用多线程，程序设计者可以分别用不同的线程完成特定的行为，而不需要采用全局的事件循环机制，这样就很容易地实现网络上的实时交互行为。

(5) 可移植性（跨越多个平台）

(6) 分布性（克服空间上的障碍）

(7) 高性能（相对于其他解释型语言）

(8) 健壮性（(9) Java 与 C 及 C++的区别

- 不再有全局变量
- 不再有#include 和#define 等预处理功能
- 不再有 structure、union 及 typedef 等
- 不再有函数、不再有指针、不再有多重继承
- 不再有 goto 语句
- 不再有操作符重载(Operatior Overloading)
- 取消自动类型转换，要求强制转换
- 自动进行内存管理

3. Java 开发工具包括:

▪Javac: Java 编译器，用来将 java 程序编译成 Bytecode。

▪Java: Java 解释器，执行已经转换成 Bytecode 的 java 应用程序。

▪Jdb: Java 调试器， 用来调试 java 程序。

▪Javap: 反编译，将类文件还原回方法和变量。

▪javadoc: 文档生成器,创建 HTML 文件。

▪Appletviwer: Applet 解释器，用来解释已经转换成 Bytecode 的 java 小应用程序。

1. Java 程序结构:

▪ package 语句: 零个或多个，必须放在文件开始

▪ import 语句: 零个或多个，必须放在所有类定义之前▪ public ClassDefinition: 零个或一个▪ ClassDefinition: 零个或多个▪ InterfaceDefinition: 零个或多个

类个数: 至少一个类，最多只能有一个 public 类

源文件命名: 若有 public 类，源文件必须按该类命名

标识符: 区分大小写

2.Java 应用程序（Java Application）

▪ 类库支持: 引用其他类。

▪ 类定义: 定义程序所需的类及接口，包括其内部的变量、方法等。▪ main()方法: 应用程序的入口，与标准 C 中 main()函数的地位是一样的。一个应用程序有且只有一个 main()方法，main()方法必须包含在一个类中，该类即为应用程序的外部标志。

▪ 程序注释: 与 C++类似，/*...*/ //...**3.Java 小程序（Java Applet）**

▪类库支持: 继承 Applet 类，引用其他类。

▪类定义: 定义程序所需的类及接口，包括其内部的变量、方法等。

▪init()方法: 初始化，自动调用，只执行一次。

▪start()方法: 初始化后，重入等都将自动调用。Applet 的主体，在其中可以执行一些任务或启动相关的线程来执行任务，如 paint()方法等。

▪stop()方法: 离开 Applet 所在页面时调用，以停止消耗系统资源。

4.Java Application 举例

```
public class HelloWorldApp
{  public static void main(String args[])
    {
          System.out.println(“Hello World!”);
    }
}
```

■编辑存盘: 文件名和公共类名(用 public 声明)要一致 HelloWorldApp.java

■编译程序: javac HelloWorldApp.java

■运行程序: java HelloWorldApp

■运行结果: Hello World!

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

```
        System.out.println(“Hello World!”);
    }
}
```

▪声明一个类: public class HelloWorldApp{}，类名第一个字母大写。

▪一个类中可有很多方法， main 方法是运行程序的第一个方法，方法名的第一个字母小写。

▪System.out.println 是向屏幕输出，相当于 C 中的 printf()。

```
class CommArg
{
     public static void main(String args[])
     {
          // Display command arguments
          int i;
          if( args.length > 0 )  //have some command arguments
          {
               for( i=0; i<args.length; i++ )
               {
                    System.out.println("arg["+i+"] = "+args[i]);
               }
          }
          else   //no command argument
          {
               System.out.println("No arguments!");
          }
    }
}
```

■命令行参数: main(String args[])，与标准 C 中 main(int argc, char* argv[])相似。

■字符串的拼接: "arg[" + i + "] = " + args[i]。

■编译程序: javac HelloWorldApp.java

■运行程序（命令行参数获取）: java CommArg first second third

■运行结果:

```
       arg[0] = first
       arg[1] = second
       arg[2] = third
import java.applet.*;
import java.awt.*;
public class HelloApplet extends Applet{
    public String s;
    public void init()
    {
        s = new String(“Hello World!”);
    }
    public void paint(Graphics g)
    {
        g.drawString(s,25,25);
```

```
    }
}
```

编辑存盘：文件名和主类名一致

编译代码：javac HelloApplet.java

编写 HTML 文件： HelloApplet.html

```
<HTML>
  <HEAD>
     <TITLE>Hello World</TITLE>
  </HEAD>
    <applet code="HelloApplet.class" width=300 height=300>
    </applet>
</HTML>
```

Java 小应用程序不能直接执行和使用,必须要在浏览器中执行。

▪运行 applet 程序:

1. appletviewer MyTest.html
2. 在浏览器中运行

▪ 运行结果:

Hello World!

理解程序:

■import 语句相当于 C 语言中的 include。

■每一个 applet 都是 java.applet.Applet 的子类，用 extends 继承。

■applet 中没有 main()方法。当 applet 被浏览器运行时， init() 、start()方法等自动执行，再调用 paint()方法。

■在 applet 中与屏幕输出有关的操作通过 Graphics 对象来实现。

■一个 Java 源文件内最多只能有一个 public 类，称为主类，且文件名必须和它同名。

```
import java.util.*;
import java.awt.*;
import java.applet.*;
import java.text.*;
public class MyTest extends Applet
{
    String s1,s2,s3,s4;
    public void init()
    {
        s1 = getParameter("p1");
        s2 = getParameter("p2");
        s3 = getParameter("p3");
        s4 = getParameter("p4");
    }
    public void paint(Graphics g)
    {
        g.drawString(s1,10,10);
        g.drawString(s2,10,30);
        g.drawString(s3,10,50);
```

```
        g.drawString(s4,10,70);
    }
}
```

编辑存盘：文件名和主类名一致

编译代码：javac MyTest.java

编写 HTML 文件： MyTest.html

```
<HTML>
  <HEAD>
    <TITLE>Applet Parameter Test</TITLE>
  </HEAD>
    <applet code="MyTest.class" width=300 height=300>
    <param name=p1 value="1111111">
    <param name=p2 value="2222222">
    <param name=p3 value="3333333">
    <param name=p4 value="4444444">
    </applet>
</HTML>
```

▪运行 applet 程序：

1. appletviewer MyTest.html
2. 在浏览器中运行

▪ 运行结果：

1111111
2222222
3333333
4444444

理解程序：

■从页面中获取 Applet 参数：

```
s1 = getParameter("p1");
```

■在页面中设置 Applet 参数：

<param name=p1 value="1111111">▪包、类、变量、方法等命名：要体现各自的含义。

包名全部小写，io，awt

类名第一个字母要大写，HelloWorldApp

变量名第一个字母要小写，userName

方法名第一个字母要小写，setName

▪程序书写格式：保证良好的可读性，使程序一目了然。

大括号{}的使用与对齐　　语句段的对齐

在语句段之间适当空行

▪ 程序注释：帮助了解程序的功能。

类注释　　变量注释

方法注释　　语句注释

语句段注释

Chap2. Java 语言基础

任何程序设计语言，都是由语言规范和一系列开发库组成的。如标准 C，除了语言规范外，还有很多函数库；MS Visual C++更是提供了庞大的 APIs 和 MFC。

Java 语言也不例外，也是由 Java 语言规范和 Java 开发类库（JFC）组成的。

学习任何程序设计语言，都是要从这两方面着手，尤其是要能够熟练地使用后者。

1、Java 数据类型
2、Java 运算符和表达式
3、Java 控制语句
4、Java 类定义规范
5、Java 数组
6、Java 开发类库组成

1、Java 数据类型

（1）标识符

程序员对程序中的各个元素加以命名时使用的命名记号称为标识符(identifier)包括：类名、变量名、常量名、方法名、... Java 语言中，标识符是以字母，下划线（_），美元符($)开始的一个字符序列，后面可以跟字母，下划线，美元符，数字。

合法的标识符

identifier userName User_Name

_sys_value $change

非法的标识符

2mail room# class

（2）保留字

具有专门的意义和用途，不能当作一般的标识符使用，这些标识符称为保留字(reserved word)。

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

（3）常量

用文字串来表示，具有不同的类型，其定义格式为：

final Type varName = value [, varName [=value] ...];

（4）变量

程序中的基本存储单元，其定义包括变量名、变量类型和作用域几个部分，定义格式为：

Type varName = value [, varName [=value] ...];

作用域：指可访问变量的一段代码，在程序中不同的地方声明的变量具有不同的作用域：局部变量、类变量、方法参数、例外处理参数。在一定的作用域内，变量名必须唯一。

（5）数据类型基本类型：所有基本类型所占的位数都是确定的，并不因操作系统的不同而不同。

数据类型	所占位数	数的范围
char	16	0 ~65535
byte	8	$-2^7 \sim 2^7-1$
short	16	$-2^{15} \sim 2^{15}-1$
int	32	$-2^{31} \sim 2^{31}-1$
long	64	$-2^{63} \sim 2^{63}-1$
float	32	$3.4e^{-038} \sim 3.4e^{+038}$
double	64	$1.7e^{-308} \sim 1.7e^{+308}$

引用类型:

§ 在 Java 中“引用”是指向一个对象在内存中的位置，在本质上是一种带有很强的完整性和安全性的限制的指针。

§ 当你声明某个类，接口或数组类型的一个变量时，那个变量的值总是某个对象的引用或者是 null 引用。

§ 指针就是简单的地址而已，引用除了表示地址而外，还象被引用的数据对象的缩影，还提供其他信息。

§ 指针可以有++、--运算，引用不可以运算。**布尔型数据**只有两个值 true 和 false，且它们不对应于任何整数值

布尔型变量的定义如: boolean b=true;

字符常量

字符常量是用单引号括起来的一个字符，如‘a’，‘A’;

字符型变量

类型为 char，它在机器中占 16 位。字符型变量的定义如: char c=‘a’; //指定变量 c 为 char 型，且赋初值为′a′

整型常量

1. 十进制整数

如 123，－456，0

2. 八进制整数

以 0 开头，如 0123 表示十进制数 83，－011 表示十进制数－9。

3. 十六进制整数

以 0x 或 0X 开头，如 0x123 表示十进制数 291，－0X12 表示十进制数－18。

整型变量

类型为 byte、short、int 或 long，byte 在机器中占 8 位，short 占 16 位，int 占 32 位，long 占 64 位。整型变量的定义如:

```
int x=123;          //指定变量 x 为 int 型，且赋初值为 123
byte b = 8;   short s = 10;   long y = 123L;   long z = 123l;
```

实型常量

1. 十进制数形式

由数字和小数点组成，且必须有小数点，如 0.123，.123，123.,123.0

2. 科学计数法形式

如: 123e3 或 123E3，其中 e 或 E 之前必须有数字，且 e 或 E 后面的指数必须为整数。

实型变量

类型为 float 或 double，float 在机器中占 32 位，double 占 64 位。实型变量的定义如:

```
float x=0.123;          //指定变量 x 为 float 型，且赋初值为 0.123
```

```
    double y = 0.123F;    double z = 0.123f;
public class Assign
{
     public  static  void  main (String args[ ] )
    {
        int  x ,  y ;
        byte  b = 6;
        float  z = 1.234f ;
        double  w = 1.234 ;
        boolean  flag = true ;
        char  c ;
        c = ‘ A ‘ ;
        x = 12 ;
        y = 300;
        ......
    }
```

}**自动类型转换**

整型、实型、字符型数据可以混合运算。运算中，不同类型的数据先转化为同一类型，然后进行运算，转换从低级到高级:

低----------->高

byte,short,char—> int —> long—> float —> double

如果从高级转换成低级，则需要强制类型转换，但会导致溢出或精度下降。

如: int i = 8; byte b=(byte)i;

2、Java 运算符和表达式（1）运算符

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: =，及其扩展赋值运算符如+=，－=，*=，/=等。条件运算符: ?: 其它: 包括分量运算符 ·，下标运算符 [] ，实例运算符instanceof，内存分配运算符new，强制类型转换运算符 (类型)，方法调用运算符 () 等。

由于数据类型的长度是确定的，所以没有长度运算符sizeof。

2）表达式

表达式是由操作数和运算符按一定的语法形式组成的符号序列。

一个常量或一个变量名字是最简单的表达式，其值即该常量或变量的值;

表达式的值还可以用作其他运算的操作数，形成更复杂的表达式。

例:

x	num1+num2	a*(b+c)+d
3.14	x<=(y+z)	x&&y||z

（3）运算符的优先次序

1） . , [] , ()　　9） &

2） ++ , -- , ! , ~ , instanceof　　10）　^

3） new (type)　　11）　|

4） * , / , %　　12）　&&

5） + , -　　13）　||
6） >> , >>> , <<　　14）　?:
7） > , < , >= , <=　　15）　= , += , -= , *= , /= , %= , ^=
8） == , !=　　16）　&= , |= , <<= , >>= , >>>=

3、Java 控制语句

§分支语句：if-else, switch
§ 循环语句：while, do-while, for
§ 与程序转移有关的其它语句：
break, continue, return
§ 例外处理语句：try-catch-finally, throw条件语句 if-else

```
if(boolean-expression1)
{
      statements1;
}
else if(boolean-expression2)
{
     statements2;
 }
 ...    else
{
     statementsN;
}
```

布尔表达式 boolean-expression 是任意一个返回布尔数据类型的表达式，而且必须是（比 C 或 C ++要严格）。

与 C 或 C++一样，每个单一语句后面都要有分号。为了增强程序的可读性，应将 if 或 else 后的语句用{}括起来。

else 子句是任选的，不能单独作为语句使用，它必须和 if 语句配对使用，并且总是与离它最近的 if 配对。

n 多分支语句 switch

```
switch (expression)
{
case value1 :
{
                    statements1;
        break;
}
.....    case valueN :
{
        statementsN;
        break;
}
[default  :
{
        defaultStatements;
```

}]
}

§表达式 expression 的返回值类型必须是这几种类型之一: int、byte、char、short。

§ case 子句中的值 valueI 必须是常量，而且所有 case 子句中的值应是不同的。

§ default 子句是任选的。

§ break 语句用来在执行完一个 case 分支后，使程序跳出 switch 语句，即终止 switch 语句的执行。如果某个 case 分支后没有 break 语句，程序将不再做比较而执行下一个分支。

§ switch 语句的功能可以用 if-else 语句来实现，但某些情况下，使用 switch 语句更简炼。

循环语句 while（当型循环）

```
[initialization]              //初始化条件
while (termination){   //循环条件
body;                 //循环体
[iteration;]         //迭代，变更循环条件
 }
```

当表达式 termination 为 true 时，执行{}中的语句，否则终止循环。

循环语句 do-while（直到型循环）

```
[initialization]              //初始化条件
do{
body;                 //循环体
[iteration;]           //迭代，变更循环条件
 } while (termination) ;    //循环条件
```

首先执行一遍{}中的语句，当表达式 termination 为 true 时，继续执行{}中的语句，否则终止循环。

循环语句 for（另一种当型循环）

```
for (initialization; termination; iteration)
{
    body;        //循环体
}
    initialization    //初始化条件
    termination      //循环条件
    iteration           //迭代，变更循环条件
```

§for 语句执行时，首先执行初始化操作，然后判断终止条件是否满足，如果满足，则执行循环体中的语句，最后执行迭代部分。完成一次循环后，重新判断终止条件。

§ 初始化、终止以及迭代部分都可以为空语句(但分号不能省)，三者均为空的时候，相当于一个无限循环。

§ 在初始化部分和迭代部分可以使用逗号语句，来进行多个操作。逗号语句是用逗号分隔的语句序列。

```
for( int i=0, int j=10; i<j; i++, j--)
{
.....    }
```

程序转移相关语句 break

在 switch 语中，break 语句用来终止 switch 语句的执行，使程序从整个 switch 语句后的第一条语句开始执行。n 在 Java 中，可以为每个代码块加一个标号，一个代码块通常是用

大括号{}括起来的一段代码。加标号的格式为:

```
BlockLabel: {
                codeBlock;
            }
```

nbreak 语句的第二种使用情况就是跳出它所指定的块，并从紧跟该块后的第一条语句处执行。 break BlockLabel;

程序转移相关语句 continue

§continue 语句用来结束本次循环，跳过循环体中下面尚未执行的语句，接着进行终止条件的判断，以决定是否继续循环。对于 for 语句，在进行终止条件的判断前，还要先执行迭代语句。它的格式为: continue; §也可以用 continue 跳转到括号指明的外层循环中，这时的格式为 continue outerLable;

程序转移相关语句 return§ return 语句从当前方法中退出，返回到调用该方法的语句处，并从紧跟该语句的下一条语句继续程序的执行。返回语句有两种格式:

```
return expression;     //当方法需要返回某种类型数据时
return;                //当方法的返回类型为 void 时
```

§ 单独一条 return 语句放在方法中间时,会产生编译错误,因为其后的语句将不会执行到。若真需要退出方法，可以通过将 return 语句嵌入某些语句（如 if-else）来使程序在未执行完方法中所有语句时退出。

例外处理语句: try-catch-finally, throw

在进行程序设计时，错误的产生是不可避免得。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？这是任何程序设计语言都必须面对和解决的问题。Java 语言中是通过例外（Exception）来处理错误的。我们将在第六讲中详细介绍例外及其处理。

4、Java 类定义规范

Java 是一种面向对象的程序设计语言，具备面向对象技术的基本属性。类是 Java 中体现面相对象特征的主要内容，它是 Java 中的一种重要数据类型，是组成 Java 程序的基本要素。我们将在下一节课中详细介绍类的定义以及与之相关的对象、包、接口等概念。

5、Java 数组

在 Java 语言中，数组是一种最简单的复合数据类型（引用数据类型）。数组是有序数据的集合，数组中的每个元素具有相同的数据类型，可以用一个统一的数组名和下标来唯一地确定数组中的元素。数组有一维数组和多维数组。我们将在后面的课程中介绍。

6、Java 开发类库组成

Java 提供了丰富的标准类来帮助程序设计者更方便快捷地编写程序，这些标准类组成了类包，主要有:

```
java.lang                    java.awt
java.applet                  java.awt.image
java.awt.peer                java.io
java.net                     java.util
```

除了 java.lang 之外，其余类包都不是 java 语言所必须的。

1）java.lang

本类包中包含了各种定义 java 语言时必须的类，这些类能够以其他类不能使用的方式访问 java 的内部。任何 java 程序都将自动引入这个包。其中的类包括:

§ Object 类: java 中最原始、最重要的类，每个 java 类都是它的子类，它实现了每个类都必须具有的基本方法。

§ 基本类型包装器: Boolean, Character, Number, Double, Float, Integer, Long。

§ **String 类：字符串类。**

§ **Math 类：数学函数的集合。**

§ **执行线程：类 Thread，ThreadGroup，接口 Runable。**

§ **异常和错误：类 Exception，Error，接口 Throwable。**

（1）java.lang

§ 运行环境：可以通过类 Runtime 和 System 访问外部系统环境。System 类的两个常用功能就是访问标准输入/输出流和错误流、退出程序。

§ 其他类：接口 Cloneable、运行时的类等。

（2）java.applet

Java Applet 是 Java 编程的一个主要魅力，java.applet 类包提供了 Applet 的运行机制以及一些编写 Applet 非常有用的方法。

（3） java.awt

本类包是各种窗口环境的统一界面（AWT 代表 Abstract Windows Toolkit，即抽象窗口工具包），其中的类使得创建诸如窗口、菜单、滚动条、文本区、按钮以及复选框等图形用户界面（GUI）的元素变得非常容易。

（4） java.awt.image

类包能够以独立于设备的方式加载并过滤位图图象。

（5） java.awt.peer

java.awt.peer 是全部 awt 组件的对等对象接口的集合，每个接口都提供了机器相关基本的方法，awt 使用这些方法来实现 GUI，而不必关心是何种机器或操作系统。

（6） java.io

Java 的输入/输出模式是完全建立在流的基础之上的。流是一种字节从一个地方到另一个地方的单向流动，可以把流附加于文件、管道和通信链路等。 java.io 类包中定义的许多种流类通过继承的方式进行组织，其中也包括一些用来访问本地文件系统上的文件的流类。

（7） java.net

java.net 类包用来完成与网络相关的功能：URL、WWW 连接以及更为通用的 Socket 网络通信。

（8） java.util

java.util 类包包含了一些实用类和有用的数据结构，如字典(Dictionary)、散列表(Hashtable)、堆栈(Stack)、向量(Vectror)以及枚举类(Enumeration)等。

Chap3. Java 与面向对象技术

1、面向对象的概念
2、Java 中的类、方法和变量
3、Java 名字空间及访问规则
4、Java 中的抽象类、接口和程序包
5、对象的构造方法

1、面向对象的概念

所谓面向对象的方法学，就是使我们分析、设计和实现一个系统的方法尽可能地接近我们认识一个系统的方法。包括:

- 面向对象的分析（OOA, Object-Oriented Analysis）
- 面向对象的设计（OOD, Object-Oriented Design）
- 面向对象的程序设计(OOPL, Object-Oriented Program)

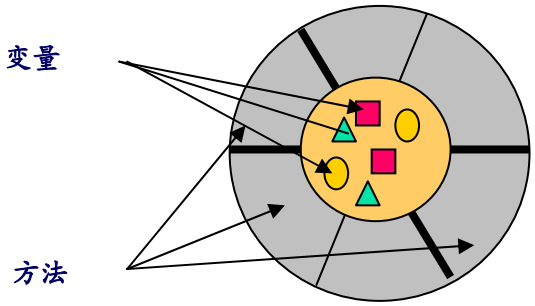
面向对象技术主要围绕以下几个概念:

对象、抽象数据类型、类、类型层次（子类）、继承性、多态性。

▪对象

对象有两个层次的概念，现实生活中对象指的是可观世界的实体；而程序中对象就是一组变量和相关方法的集合，其中变量表明对象的状态，方法表明对象所具有的行为。

可以将现实生活中的对象经过抽象，映射为程序中的对象。对象在程序中是通过一种抽象数据类型来描述的，这种抽象数据类型称为类（Class）。




```
Class Car
{
    int color_number;
    int door_number;
    int speed;

    void brake() { … }
    void speedUp() {…}
    void slowDown() { … }
}
```

▪类（Class）

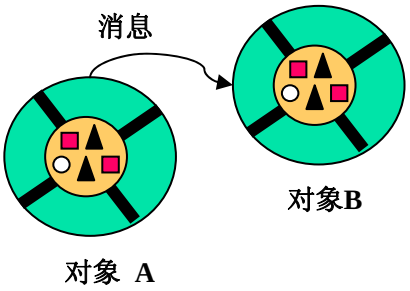
类是描述对象的“基本原型”，它定义一类对象所能拥有的数据和能完成的操作。在面向对象的程序设计中，类是程序的基本单元。

相似的对象可以归并到同一个类中去，就像传统语言中的变量与类型关系一样。

程序中的对象是类的一个实例，是一个软件单元，它由一组结构化的数据和在其上的一组操作构成。

■变量：指对象的所知道的状态。
■方法：指对象的功能单元。
■消息
软件对象通过相互间传递消息来相互作用和通信，一个消息由三部分组成:

1. 接受消息的对象
2. 接收对象要采取的方法
3. 方法需要的参数



■一个例子

```
class Hello
{
  private String s;
  public void showString()
  {
      System.out.println(s);
  }
  public void changeString(String str)
  {
      s = str;
  }
}
```

■在程序中操作对象是类的一个实例。
■创建一个对象: Hello obj=new Hello();
■调用方法: obj.showString();▪为什么使用类
采用简单数据类型表示现实世界中概念的局存在一些限性。例如：采用 int 型数据表示一个日期概念，需要使用 3 个变量:

```
int day,month,year;
```

如果要表示 2 个人的生日，就要使用 6 个变量:

```
int mybirthday, mybirthmonth, mybirthyear;
int yourbirthday,yourbirthmonth,yourbirthyear;
```

类中不但有变量，还有与之相关的操作所定义的方法。将变量和方法封装在一个类中，可以对成员变量进行隐藏,外部对类成员的访问都通过方法进行,能够保护类成员不被非法修改。

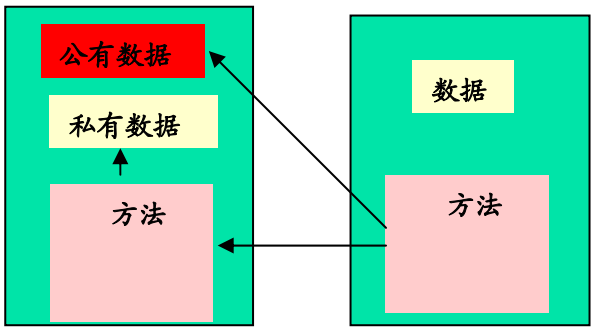
```
class BirthDate
{
      public int day,month,year;
      public int tomorrow()
      {
          ……      }
```

}

BirthDate mybirth, yourbirth;BirthDate date;

已知当前日期对象，求第 2 天的日期对象：date.day =date.day+1;

如果 date.day 已经为 31，操作结果是非法状态。可以定义一个成员方法 tomorrow()，求第 2 天的日期对象。外部要获得当前日期的后一天时，只要调用： date.tomorrow();**■封装**



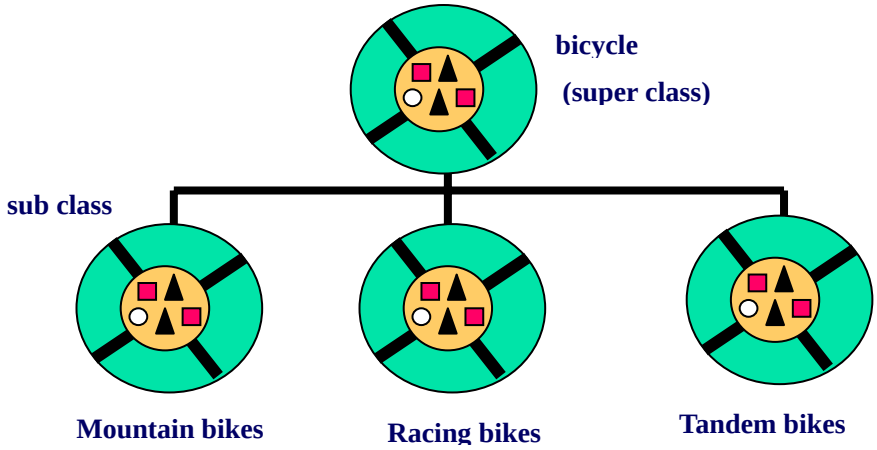
封装把对象的所有组成部分组合在一起，封装定义程序如何引用对象的数据，封装实际上使用方法将类的数据隐藏起来，控制用户对类的修改和访问数据的程度。

■子类

子类是作为另一个类的扩充或修正而定义的一个类。

▪继承

继承是子类利用父类中定义的方法和变量，就像它们属于子类本身一样。




```
class Car
{   int color_number;
    int door_number;
    int speed;
    public void push_break()
    {
        …     }
    public void add_oil() { …   }
}
class Trash_Car  extends  Car
{
    double amount;
```

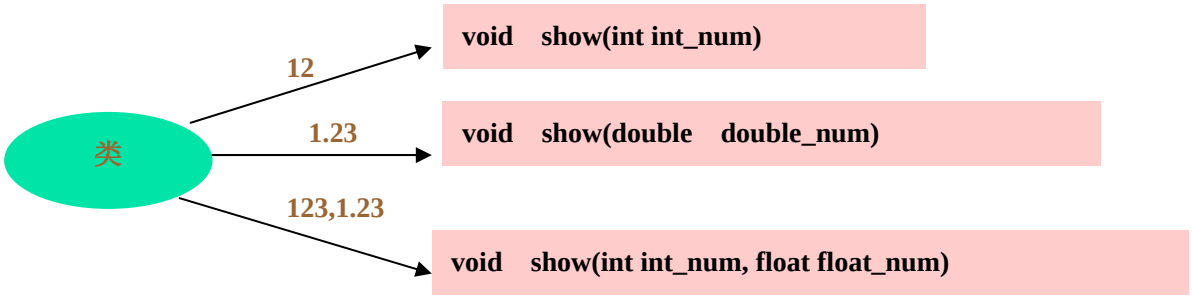
```
    public void fill_trash()
    {
        …    }
}
```

■方法的覆盖

在子类中重新定义父类中已有的方法。

```
class Car
{   int color_number;
    int door_number;
    int speed;
    public void push_break()
    {
          speed = 0;
     }
    public void add_oil() { …  }
}  class Trash_Car  extends  Car
{
    double amount;
    public void fill_trash() { … }
    public void push_break()
    {
          speed = speed – 10;
     }
}
```

■方法的重载(多态性)在同一个类中至少有两个方法用同一个名字，但有不同的参数。



2、Java 中的类、方法和变量

■类的严格定义及修饰字

```
[类的修饰字] class 类名称 [extends 父类名称][implements 接口名称列表]
 {
       变量定义及初始化;
       方法定义及方法体;
}
```

类的修饰字: [public] [abstract | final]

缺省方式为 friendly

▪ 成员变量的定义及修饰字

```
[变量修饰字] 变量数据类型 变量名 1,变量名 2[=变量初值]…;
[public | protected | private ] [static] [final ] [transient ][volatile]
```

成员变量的类型可以是 Java 中任意的数据类型，包括简单类型，类，接口，数组。在一个类中的成员变量应该是唯一的。

▪方法和变量的定义及修饰字

```
[方法修饰字] 返回类型  方法名称(参数 1,参数 2,...) [throws exceptionList]
{
    ...(statements;)    //方法体: 方法的内容
}
```

[public | protected | private] [static] [final | abstract] [native] [synchronized]

返回类型可以是任意的 Java 数据类型，当一个方法不需要返回值时，返回类型为 void。

参数的类型可以是简单数据类型，也可以是引用数据类型（数组、类或接口），参数传递方式是值传递。

方法体是对方法的实现。它包括局部变量的声明以及所有合法的 Java 指令。局部变量的作用域只在该方法内部。

```
class CarDemo
{
    public static void main(String args[])
    {
        Car Democar=new Car();
        DemoCar.set_number(3838);
        DemoCar.show_number();
    }
}
class Car
{
   int car_number;
   void set_number(int car_num)
   {
        car_number=car_num;
    }
    void show_number()
    {
        System.out.println ("My car No. is : "+car_number);
    }
}
```

▪对象

（1）对象的生成

通过 new 操作符生成一个对象; 例如:

```
Car    demoCar;
demoCar = new Car();
```

（2）对象的构造过程

✓ 为对象开辟空间，并对对象的成员变量进行缺省的初始化;

✓ 对成员变量进行指定的初始化;

✓ 调用构造方法。

▪对象

（3）对象的使用

对象的使用是通过一个引用类型的变量来实现，包括引用对象的成员变量和方法，通过运算符 . 可以实现对变量的访问和方法的调用。例如:

```
BirthDate date;
int day;
day = date.day; //引用 date 的成员变量 day
date.tomorrow();     //调用 date 的方法 tomorrow()
```

- **类的继承**

```
class Car
{
     int car_number;
     void set_number(int car_num)
     {
          car_number=car_num;
     }
     void show_number()
     {
          System.out.println ("My car No. is :"+car_number);
     }
}
class TrashCar extends Car
{
     int capacity;
     void set_capacity(int trash_car_capacity)
     {
           capacity=trash_car_capacity;
     }
     void show_capacity()
     {
          System.out.println("My capacity is: " + capacity);
     }
}
class CarDemo
{
     public static void main(String args[])
     {
         TrashCar DemoTrashCar = new TrashCar();
         DemoTrashCar.set_number(4949);
         DemoTrashCar.show_number();
         DemoTrashCar.set_capacity(20);
         DemoTrashCar.show_capacity();
     }
}
```

Car 是父类，TrashCar 是子类。TrashCar 中继承了 Car 中的两个方法，同时又增加了两个新方法。

继承性是面向对象程序设计语言的另一基本特征，通过继承可以实现代码的复用。继承而得到的类为子类，被继承的类为父类，父类包括所有直接或间接被继承的类。Java 中不支持多重继承。通过在类的声明中加入 extends 子句来创建一个类的子类:

```
class SubClass extends SuperClass
{.....}
```

如果缺省 extends 子句，则该类为 java.lang.Object 的子类。子类可以继承父类中访问权限设定为 public、 protected、 default 的成员变量和方法，但是不能继承访问权限为 private 的成员变量和方法。

■何时选择继承?

一个很好的经验: “B 是一个 A 吗?”如果是则让 B 做 A 的子类.

▪类方法的覆盖

方法覆盖即指在子类中重新定义父类中已有的方法。

```
class Car
{   int color_number;
    int door_number;
    int speed;
    public void push_break()
    {
          speed = 0;
     }
    public void add_oil() {  ...   }
}
class Trash_Car  extends  Car
{
    double amount;
    public void fill_trash() {  ...  }
    public void push_break()
    {
           speed = speed – 10;
     }
}
```

▪覆盖方法的调用

对于重写的方法，Java 运行时系统根据调用该方法的实例的类型来决定选择哪个方法调用。

```
public class DemoCar
{
public static void main( String args[ ] )
{
    Car  aCar = new Trash_Car( );
    aCar. push_break( );
}
}
```

在这里， 类 Trash_Car 中的 push_break( )方法将被调用。

▪方法覆盖时应遵循的原则

（1）覆盖后的方法不能比被覆盖的方法有更严格的访问权限。

（2）覆盖后的方法不能比被覆盖的方法产生更多的例外。

▪类方法的重载

方法重载即指多个方法可以享有相同的名字。但是这些方法的参数必须不同，或者是参数个数不同，或者是参数类型不同。

例如，要打印不同类型的数据，int，float，String,不需要定义不同名的方法:

printInt(int);　printFloat(float);　printString(String)。

利用方法重载，只需要定义一个方法名: println()，接收不同的参数:

println(int); println(float); println(String);

▪多态性

类方法的重载是一种多态性。除此之外，多态性还可以是指在程序中需要使用父类对象的地方，都可以用子类对象来代替。　例如:

```
public class Employee extends Object
{.....}
public class Manager extends Employee
{.....}
```

则:

```
Employee e = new Manager();   //合法语句
```

▪成员变量的隐藏

可以用方法来实现对成员变量的隐藏: 设置变量方法 setVariable()， 获取变量方法 getVariable()。

```
class Sample
{
    int x;
    ......    void setX(int var )
    {
        x = var;
    }
    int getX()
    {
            return x;
    }
    .....}▪对象状态的确定
```

在 Java 语言中，提供了操作符 instanceof 用来判断对象是否属于某个类的实例。

```
public  void  method (Employee  e)
{
        if  ( e  instanceof  Manager )
        {
             ...//do something as a Manager
         }
        else  if  ( e  instanceof  Contractor )
        {
            ...//do something as a Contractor
```

```
        }
        else
        {
        ... //do something else
       }
}
```

3、Java 名字空间及访问规则

■每个类都拥有自己的名字空间，即指类及其方法和变量可以在一定的范围内知道彼此的存在，可以使用。对类而言:

■abstract 类不能直接产生属于这个类的对象;

■final 类不能被其他任何类所继承(安全的考虑);

■public 类不但可以被同一程序包中的其它类使用，别的程序包中的类也可以使用;

■friendly(缺省)类只能被本包中的其它类使用。

■对类的成员变量和成员方法而言，其应用范围可以通过施以一定的访问权限来限定。

同一个类中　同一个包中　不同包中的子类　不同包中的非子类　　private ★

default ★　★　　protected　★　★　　★　　public

★　★　　★　　★　▪public: 任何其它类、对象只要可以看到这个类的话，那么它就可以存取变量的数据，或使用方法。

▪ protected: 同一类，同一包可以使用。不同包的类要使用，必须是该类的子类。

▪ private: 不允许任何其他类存取和调用。

▪friendly(前边没有修饰字的情况): 在同一程序包中出现的类才可以直接使用它的数据和方法.

■当子类中的变量名与父类的相同，父类的变量被遮盖。

```
class A              class B extends A
{                {
    int data_a=3;                    int data_a = 5;
}                }
class C extends B
{
    void print_out()
    {
         System.out.println("data_a= "+data_a);
         System.out.println("A.data_a= "+A.data_a);
         System.out.println("B.data_a= "+B.data_a);
     }
}
class demo
{
    public static void main(String args[])
    {
         C  c = new C();
         c.println_out();
    }
}
```

▪到此为止——final

▪ final 在类之前，标是该类不能被继承。

▪ final 在方法之前，防止该方法被覆盖。

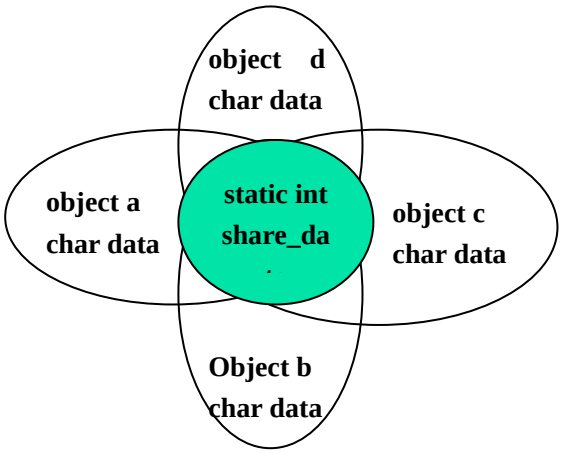
▪final 在变量之前，定义一个常量。

▪属于类的变量和方法——static

static 在变量或方法之前，表明它们是属于类的，称为类方法（静态方法）或类变量（静态变量）。若无 static 修饰，则是实例方法和实例变量。

类变量在各实例间共享

```
class ABCD
{
     char data;
     static int share_data;
}
class Demo
{
   ABCD  a,b,c,d;
}
```



§类变量的生存期不依赖于对象，相当于 C 语言中全局变量的作用。其它类可以不通过实例化访问它们。

```
public class StaticVar
{
    public static int number = 5;
}
public class OtherClass
{
    public void method()
    {
          int x = StaticVar.number;
    }
}
```

§类方法相当于 C 语言中的全局函数，其他的类不用实例化即可调用它们。

```
public class GeneralFunction
{
```

```
    public static int addUp(int x,int y)
    {
        return x+y;
    }
}
public class UseGeneral
{
    public void method()
    {
        int a = 9;
        int b =10;
        int c = GeneralFunction.addUp(a,b);
    }
}
```

§同一个类中的方法可以访问该类的成员变量;

§一个类的方法只能访问自己的局部变量。

§不正确的引用

```
class StaticError
{
    String mystring=“hello”;
    public static void main(String args[])
    {
        System.out.println(mystring);
    }
}
```

编译时错误信息: nonstatic variable mystring cannot be referenced from a static context “System.out.println(mystring);”。

为什么不正确: 只有对象的方法可以访问对象的变量。

解决的办法

1. 将变量改成类变量

```
class StaticError
{
    static String mystring=“hello”;
    public static void main(String args[])
    {
        System.out.println(mystring);
    }
}
```

2. 先创建一个类的实例

```
class NoStaticError
{
     String mystring=“hello”;
     public static void main(String args[])
     {
```

```
        NoStaticError  noError;
        noError              =              new              NoStaticError();
System.out.println(noError.mystring);
    }
}
```

4、Java 中的抽象类、接口和程序包

§抽象类与抽象方法

用 abstract 关键字来修饰一个类时,该类叫做抽象类;用 abstract 来修饰一个方法时,该方法叫做抽象方法。

抽象类必须被继承，抽象方法必须被重写。

抽象类不能被直接实例化。因此它一般作为其它类的超类,与 final 类正好相反。

抽象方法只需声明，而不需实现。定义了抽象方法的类必须是抽象类。

```
            abstract returnType abstractMethod( [paramlist] );
```

§两个类 Circle 和 Rectangle，完成相关参数的计算

```
class Circle
{
    public float r;
    Circle(float r)
    {
        this.r = r;       //this 指"这个对象的"
    }
    public float area()
    {
       return 3.14*r*r;
    }
}class Rectangle
{
    public float width,height;
    Rectangle (float w, float h)
    {
        width = w; //这里不需"this"
        height = h;
    }
    public float area()
    {
        return width*height;
    }
}
```

假设有若干个 Circle，以及若干个 Rectangle，希望计算它们的总面积，直截了当的做法是将它们分别放到两个数组中，用两个循环，加上一个加法，这种做法是不漂亮的。

如果还有其它形状: triangle，ellipses 等，上述方法显得“累赘”。我们希望有一种统一的表示，例如用一个数组 shape[]，接受所有的形状，然后用:

```
for (i=0; i<shape.length; i++)
{
  area_total += shape[i].area
```

```
abstract class Shape
{
    abstract float area();
}class Circle extends Shape
{
    public float r;
    Circle(float r)
    {
          this.r = r;       //this 指"这个对象的"
     }
     public float area()
     {
         return 3.14*r*r;
      }
}class Rectangle  extends Shape
{
     public float width,height;
     Rectangle (float w, float h)
     {
           width = w; //这里不需"this"
           height = h;
     }
     public float area()
     {
           return width*height;
     }
}
```

接口（interface）

接口就是方法定义和常量值的集合。从本质上讲，接口是一种特殊的抽象类，这种抽象类中只包含常量和方法的定义，而没有方法的实现。

ü 通过接口可以实现不相关类的相同行为，而不需要考虑这些类之间的层次关系。

ü 通过接口可以指明多个类需要实现的方法。

ü 通过接口可以了解对象的交互界面，而不需了解对象所对应的类。

接口的定义:

```
[public] interface interfaceName [extends SuperInterfaceList]
{
    ......    //常量定义和方法定义}
```

用 implements 子句来表示一个类使用某个接口。

在类体中可以使用接口中定义的常量，而且必须实现接口中定义的所有方法。

利用接口可实现多重 继承，即一个类可以实现多个接口，在 implements 子句中用逗号分隔。

接口的作用和抽象类相似，只定义原型，不直接定义方法的内容。

接口中的方法和变量必须是 public 的。

```
interface Collection
{
    int MAX_NUM=100;
    void add (Object obj);
    void delete (Object obj);
    Object find (Object obj);
    int currentCount ( );
}
class FIFOQueue implements Collection
{
    void add ( Object obj )
    {
        ......    }
    void delete( Object obj )
    {
        ......    }
    Object find( Object obj )
    {
        ......    }
    int currentCount()
    {
    }
```

}**程序包（package）**

由于 Java 编译器为每个类生成一个字节码文件，且文件名与类名相同，因此同名的类有可能发生冲突。为了解决这一问题，Java 提供包来管理类名空间。

程序包相当于其它语言中的库函数。

打包

Java 中用 package 语句来将一个 Java 源文件中的类打成一个包。package 语句作为 Java 源文件的第一条语句，指明该文件中定义的类所在的包。(若缺省该语句，则指定为无名包)。它的格式为:

package pkg1[.pkg2[.pkg3...]]; Java 编译器把包对应于文件系统的目录管理，package 语句中，用 . 来指明目录的层次。

```
package myclass.graphics;
class Square {...;}
class Circle {...;}
class Triangle {...;}
```

package myclass.graphics; 这条语句指定这个包中的文件存储在目录 *path*/myclass/graphics 下。

包层次的根目录 *path* 是由环境变量 CLASSPATH 来确定的。

为了能使用 Java 中已提供的类，我们需要用 import 语句来引入所需要的类。

import package1[.package2...]. (classname |*);

例如:

```
import   myclass.graphics.*;
```

```
import    java.io.*;
```

编译和生成包

如果在程序 Test.java 中已定义了包 p1，编译时采用如下方式:

javac -d destpath Test.java

则编译器会自动在 destpath 目录下建立一个子目录 p1,并将生成的.class 文件都放到 destpath/p1 下。 destpath 可以是环境变量 CLASSPATH 中的一个。

5、对象的构造方法

构造方法(Constructor)是一种特殊的方法。Java 中的每个类都有构造方法，用来初始化该类的一个新的对象。构造方法具有和类名相同的名称，而且不返回任何数据类型。系统在产生对象时会自动执行。

构造方法应包含的内容:

§ 定义一些初值或内存配置工作;

§ 一个类可以有多个构造方法(重载)，根据参数的不同决定执行哪一个;

§ 如果程序中没有定义构造方法，则创建实例时使用的是缺省构造方法，它是一个无内容的空方法。

```
public class Employee{
      private String name;
      private int salary;
      public Employee(String n,int s)
      {
    name = n;
    salary = s;
      }
      public Employee(String n){
    this(n,0);
       }
       public Employee()
      {
    this(“Unknown”);
      }
}
```

this

this 指自己这个对象，它的一个主要作用是要将自己这个对象当作参数，传送给别的对象中的方法。

```
class ThisClass
{
    public static void main()
    {
         Bank bank=new Bank();
         bank.someMethod(this);
     }
}class Circle
{
```

```
    int  r;
    Circle(int r)
    {
        this.r=r;
    }
    public  area()
    {
        return r*r*3.14;
    }
```
}**super**

super 指这个对象的父类。super 可以用来引用父类中的(被覆盖的)方法、(被隐藏的)变量及构造方法。

```
public  class apple extends fruits
{
    public apple(int price)
    {
        super(price);
        super.var = value;
        super.method(paraList);
    }
}
```

以上程序表示使用父类的构造方法生成实例，super 必须是子类构造方法的第一条语句。

Finalizer

在对对象进行垃圾收集前,Java 运行时系统会自动调用对象的 finalize() 方法来释放系统资源。该方法必须按如下方式声明:

```
protected void finalize() throws throwable
{.....}
```

finalize()方法是在 java.lang.Object 中实现的，在用户自定义的类中，它可以被覆盖，但一般在最后要调用父类的 finalize()方法来清除对象所使用的所有资源。

```
protected void finalize() throws throwable
{
    .....  //释放本类中使用的资源
    super.finalize();
}
```