JAVA 教程 第一讲 Java 语言概述

1.1 Java 语言的发展史

1.1.1Java 语言在互联网时代获得巨大成功

大家想一想，在 PC 下用 windows 编写的程序能够不做修改就直接拿到 UNIX 系统上运行吗？显然是不可以的，因为程序的执行最终必须转换成为计算机硬件的机器指令来执行，专门为某种计算机硬件和操作系统编写的程序是不能够直接放到另外的计算机硬件上执行的，至少要做移植工作。以介绍面向对象编程的基本概念、基本理论为重点，结合 Java 语言的语法规则、编程特点和设计思想、强调容易发生错误和编程应注意的地方，使学生能对 Java 技术有一个总体了解，通过本课程学习，使学生掌握 Java 语言的基础知识，理解和掌握面向对象程序设计的基本思想，熟练地使用 Java 语言进行程序的编写、编译以及调试工作要想让程序能够在不同的计算机上能够运行，就要求程序设计语言是能够跨越各种软件和硬件平台的，而 Java 满足了这一需求。

1995 年，美国 Sun Microsystems 公司正式向 IT 业界推出了 java 语言，该语言具有安全、跨平台、面向对象、简单、适用于网络等显著特点，当时以 web 为主要形式的互联网正在迅猛发展，java 语言的出现迅速引起所有程序员和软件公司的极大关注，程序员们纷纷尝试用 java 语言编写网络应用程序，并利用网络把程序发布到世界各地进行运行。包括 IBM、Oracle、微软、Netscape、Apple、SGI 等大公司纷纷与 Sun Microsystems 公司签订合同，授权使用 java 平台技术。微软公司总裁比尔盖茨先生在经过研究后认为"java 语言是长时间以来最卓越的程序设计语言"。目前，java 语言已经成为最流行的网络编程语言，截止到 2001 年中，全世界大约有 310 万 java 程序员，许多大学纷纷开设 java 课程，Java 正逐步成为世界上程序员最多的编程语言。

在经历了以大型机为代表的集中计算模式和以 PC 机为代表的分散计算模式之后，互联网的出现使得计算模式进入了网络计算时代。网络计算模式的一个特点是计算机是异构的，即计算机的类型和操作系统是不一样的，例如 SUN 工作站的硬件是 SPARC 体系，软件是 UNIX 中的 Solaris 操作系统，而 PC 机的硬件是 INTEL 体系，操作系统是 windows 或者是 Linux，因此相应的编程语言基本上只是适用于单机系统，例如 COBOL、FORTRAN、C、C＋＋等等；网络计算模式的另一个特点是代码可以通过网络在各种计算机上进行迁移，这就迫切需要一种跨平台的编程语言，使得用它编写的程序能够在网络中的各种计算机上能够正常运行，java 就是在这种需求下应运而生的。正是因为 Java 语言符合了互联网时代的发展要求，才使它获得了巨大的成功。

1.1.2 Java 语言的产生

任何事物的产生既有必然的原因也有偶然的因素，Java 语言的出现也验证了这一点。1991 年，美国 Sun Microsystems 公司的某个研究小组为了能够在消费电子产品上开发应用程序，积极寻找合适的编程语言。消费电子产品种类繁多，包括 PDA、机顶盒、手机等等，即使是同一类消费电子产品所采用的处理芯片和操作系统也不相同，也存在着跨平台的问题。当时最流行的编程语言是 C 和 C＋＋语言，Sun 公司的研究人员就考虑是否可以采用 C＋＋语言来编写消费电子产品的应用程序，但是研究表明，对于消费电子产品而言 C＋＋语言过于复杂和庞大，并不适用，安全性也并不令人满意。于是，Bill Joy 先生领导的研究小组就着手设计和开发出一种语言，称之为 Oak。该语言采用了许多 C 语言的语法，提高了安全性，并且是面向对象的语言，但是 Oak 语言在商业上并未获得成功。时间转到了 1995 年，互联网在世界上蓬勃发展，Sun 公司发现 Oak 语言所具有的跨平台、面向对象、安全性高等特点非常符合互联网的需要，于是改进了该语言的设计，要达到如下几个目标：

- ◇ 创建一种面向对象的程序设计语言，而不是面向过程的语言；
- ◇ 提供一个解释执行的程序运行环境，是程序代码独立于平台；
- ◇ 吸收 C 和 C＋＋的优点，使程序员容易掌握；
- ◇ 去掉 C 和 C＋＋中影响程序健壮性的部分，使程序更安全，例如指针、内存申请和释放；
- ◇ 实现多线程，使得程序能够同时执行多个任务；
- ◇ 提供动态下载程序代码的机制；

◇ 提供代码校验机制以保证安全性；

最终，Sun 公司给该语言取名为 Java 语言，造就了一代成功的编程语。

1.2 Java 的工作原理

1.2.1 Java 虚拟机(1)

java 虚拟机是软件模拟的计算机，可以在任何处理器上（无论是在计算机中还是在其它电子设备中）安全并且兼容的执行保存在.class 文件中的字节码。java 虚拟机的"机器码"保存在.class 文件中，有时也可以称之为字节码文件。java 程序的跨平台主要是指字节码文件可以在任何具有 java 虚拟机的计算机或者电子设备上运行，java 虚拟机中的 java 解释器负责将字节码文件解释成为特定的机器码进行运行。java 源程序需要通过编译器编译成为.class 文件（字节码文件），Java 程序的编译和执行过程　　——如图所示——

1.2.1 Java 虚拟机(2)

但是，java 虚拟机的建立需要针对不同的软硬件平台做专门的实现，既要考虑处理器的型号，也要考虑操作系统的种类。如下图所示，目前在 SPARC 结构、X86 结构、MIPS 和 PPC 等嵌入式处理芯片上、在 UNIX、Linux、windows 和部分实时操作系统上都有 Java 虚拟机的实现。 ——如图所示——

1.2.2 无用内存自动回收机制

在程序的执行过程中，部分内存在使用过后就处于废弃状态，如果不及时进行无用内存的回收，就会导致内存泄漏，进而导致系统崩溃。在 C＋＋语言中是由程序员进行内存回收的，程序员需要在编写程序的时候把不再使用的对象内存释放掉；但是这种人为的管理内存释放的方法却往往由于程序员的疏忽而致使内存无法回收，同时也增加了程序员的工作量。而在 Java 运行环境中，始终存在着一个系统级的线程，专门跟踪内存的使用情况，定期检测出不再使用的内存，并进行自动回收，避免了内存的泄露，也减轻了程序员的工作量。

1.2.3 代码安全性检查机制

安全和方便总是相对矛盾的。java 编程语言的出现使得客户端机器可以方便的从网络上下载 java 程序到本机上运行，但是如何保证该 java 程序不携带病毒或者不怀有其它险恶目的呢？如果 java 语言不能保证执行的安全性，那么它就不可能存活到今天。虽然有时候少数程序员会抱怨说 applet 连文件系统也不能访问，但是正是各种安全措施的实行才确保了 Java 语言的生存

字节码的执行需要经过三个步骤，首先由类装载器（class loader）负责把类文件（.class 文件）加载到 java 虚拟机中，在此过程需要检验该类文件是否符合类文件规范；其次字节码校验器（bytecode verifier）检查该类文件的代码中是否存在着某些非法操作，例如 applet 程序中写本机文件系统的操作；如果字节码校验器检验通过，由 java 解释器负责把该类文件解释成为机器码进行执行。java 虚拟机采用的是"沙箱"运行模式，即把 Java 程序的代码和数据都限制在一定内存空间里执行，不允许程序访问该内存空间外的内存，如果是 applet 程序，还不允许访问客户端机器的文件系统。

1.2.4 Java 语言的特点(1)

1. 简单、面向对象和为人所熟悉

java 的简单首先体现在精简的系统上，力图用最小的系统实现足够多的功能；对硬件的要求不高，在小型的计算机上便可以良好的运行。和所有的新一代的程序设计语言一样，java 也采用了面向对象技术并更加彻底，所有的 java 程序和 applet 程序均是对象，封装性实现了模块化和信息隐藏，继承性实现了代码的复用，用户可以建立自己的类库。而且 java 采用的是相对简单的面向对象技术，去掉了运算符重载、多继承的复杂概念，而采用了单一继承、类强制转换、多线程、引用（非指针）等方式。无用内存自动回收机制也使得程序员不必费心管理内存，是程序设计更加简单，同时大大减少了出错的可能。java 语言采用了 C 语言中的大部分语法，熟悉 C 语言的程序员会发现 Java 语言在语法上

与 C 语言极其相似。

2. 鲁棒并且安全

java 语言在编译及运行程序时，都要进行严格的检查。作为一种强制类型语言，java 在编译和连接时都进行大量的类型检查，防止不匹配问题的发生。如果引用一个非法类型、或执行一个非法类型操作，java 将在解释时指出该错误。在 java 程序中不能采用地址计算的方法通过指针访问内存单元，大大减少了错误发生的可能性；而且 java 的数组并非用指针实现，这样就可以在检查中避免数组越界的发生。无用内存自动回收机制也增加了 Java 的鲁棒性。

作为网络语言，java 必须提供足够的安全保障，并且要防止病毒的侵袭。java 在运行应用程序时，严格检查其访问数据的权限，比如不允许网络上的应用程序修改本地的数据。下载到用户计算机中的字节代码在其被执行前要经过一个核实工具，一旦字节代码被核实，便由 java 解释器来执行，该解释器通过阻止对内存的直接访问来进一步提高 java 的安全性。同时 java 极高的鲁棒性也增强了 Java 的安全性。

3. 结构中立并且可以移植

网络上充满了各种不同类型的机器和操作系统，为使 java 程序能在网络的任何地方运行，java 编译器编译生成了与体系结构无关的字节码结构文件格式。任何种类的计算机，只有在其处理器和操作系统上有 java 运行时环境，字节码文件就可以在该计算机上运行。即使是在单一系统的计算机上，结构中立也有非常大的作用。随着处理器结构的不断发展变化，程序员不得不编写各种版本的程序以在不同的处理器上运行，这使得开发出能够在所有平台上工作的软件集合是不可能的。而使用 Java 将使同一版本的应用程序可以运行在所有的平台上。

体系结构的中立也使得 java 系统具有可移植性。java 运行时系统可以移植到不同的处理器和操作系统上，java 的编译器是由 java 语言实现的，解释器是由 Java 语言和标准 C 语言实现的，因此可以较为方便的进行移植工作。

1.2.4 Java 语言的特点(2)

4. 高性能

虽然 java 是解释执行的，但它仍然具有非常高的性能，在一些特定的 CPU 上，java 字节码可以快速的转换成为机器码进行执行。而且 java 字节码格式的设计就是针对机器码的转换，实际转换时相当简便，自动的寄存器分配与编译器对字节码的一些优化可使之生成高质量的代码。随着 java 虚拟机的改进和"即时编译"（just in time）技术的出现使得 Java 的执行速度有了更大的提高。

5. 解释执行、多线程并且是动态的

如果你了解 C 语言和 C++语言，可以参考下列 Java 与 C/C++语言的比较，如果不了解 C 语言和 C++语言，可以忽略本部分知识。

a. 全局变量

Java 程序不能定义程序的全局变量，而类中的公共、静态变量就相当于这个类的全局变量。这样就使全局变量封装在类中，保证了安全性，而在 C/C++语言中，由于不加封装的全局变量往往会由于使用不当而造成系统的崩溃。

b. 条件转移指令

C/C++语言中用 goto 语句实现无条件跳转，而 Java 语言没有 goto 语言，通过例外处理语句 try、catch、finally 来取代之，提高了程序的可读性，也增强了程序的鲁棒性。

c. 指针

指针是 C/C++语言中最灵活，但也是最容易出错的数据类型。用指针进行内存操作往往造成不可预知的错误，而且，通过指针对内存地址进行显示类型转换后，可以类的私有成员，破坏了安全性。在 java 中，程序员不能进行任何指针操作，同时 Java 中的数组是通过类来实现的，很好的解决了数组越界这一 C/C++语言中不做检查的缺点。

d. 内存管理

在C语言中，程序员使用库函数malloc()和free()来分配和释放内存，C++语言中则是运算符new和delete。再次释放已经释放的内存块或者释放未被分配的内存块，会造成系统的崩溃，而忘记释放不再使用的内存块也会逐渐耗尽系统资源。在Java中，所有的数据结构都是对象，通过运算符new分配内存并得到对象的使用权。无用内存回收机制保证了系统资源的完整，避免了内存管理不周而引起的系统崩溃。

e. 数据类型的一致性

在C/C++语言中，不同的平台上，编译器对简单的数据类型如int、float等分别分配不同的字节数。例如：int在IBM PC上为16位，在VAX-11上就为32位，导致了代码数据的不可移植。在java中，对数据类型的位数分配总是固定的，而不管是在任何的计算机平台上。因此就保证了Java数据的平台无关性和可移植性。

f. 类型转换

在C/C++语言中，可以通过指针进行任意的类型转换，不安全因素大大增加。而在Java语言中系统要对对象的处理进行严格的相容性检查，防止不安全的转换。

g. 头文件

在C/C++语言中使用头文件声明类的原型和全局变量及库函数等，在大的系统中，维护这些头文件是非常困难的。java不支持头文件，类成员的类型和访问权限都封装在一个类中，运行时系统对访问进行控制，防止非法的访问。同时，Java中用import语句与其它类进行通信，以便访问其它类的对象。

h. 结构和联合

C/C++语言中用结构和联合来表示一定的数据结构，但是由于其成员均为公有的，安全性上存在问题。Java不支持结构和联合，通过类把数据结构及对该数据的操作都封装在类里面。

i. 预处理

C/C++语言中有宏定义，而用宏定义实现的代码往往影响程序的可读性，而Java不支持宏定义

为易于实现跨平台性，java设计成为解释执行，字节码本身包含了许多编译时生成的信息，使连接过程更加简单。而多线程使应用程序可以同时进行不同的操作，处理不同的事件。在多线程机制中，不同的线程处理不同的任务，互不干涉，不会由于某一任务处于等待状态而影响了其它任务的执行，这样就可以容易的实现网络上的实时交互操作。Java在执行过程中，可以动态的加载各种类库，这一特点使之非常适合于网络运行，同时也非常有利于软件的开发，即使是更新类库也不必重新编译使用这一类库的应用程序。

1.2.5 Java平台－不断扩展的计算平台

java不仅是编程语言，还是一个开发平台，java技术给程序员提供了许多工具：编译器、解释器、文档生成器和文件打包工具等等。同时java还是一个程序发布平台，有两种主要的"发布环境"，首先java运行时环境（java runtime environment，简称JRE）包含了完整的类文件包，其次许多主要的浏览器都提供了java解释器和运行时环境。目前Sun公司把java平台划分成J2EE、J2SE、J2ME三个平台，针对不同的市场目标和设备进行定位。J2EE是Java2 Enterprise Edition，主要目的是为企业计算提供一个应用服务器的运行和开发平台。J2EE本身是一个开放的标准，任何软件厂商都可以推出自己的符合J2EE标准的产品，使用户可以有多种选择。IBM、Oracle、BEA、HP等29家已经推出了自己的产品，其中尤以BEA公司的weglogic产品和IBM公司的websphare最为著名。J2EE将逐步发展成为可以与微软的.NET战略相对抗的网络计算平台。J2SE是Java2 Standard Edition，主要目的是为台式机和工作站提供一个开发和运行的平台。我们在学习java的过程中，主要是采用J2SE来进行开发。J2ME是Java2 Micro Edition，主要是面向消费电子产品，为消费电子产品提供一个java的运行平台，使得java程序能够在手机、机顶盒、PDA等产品上运行。上述三个Java平台的关系　　如图所示

1.3 一切都是对象

1.3.1 面向过程

面向对象的第一个原则是把数据和对该数据的操作都封装在一个类中，在程序设计时要考虑多个对象及其相互间的关系。有些功能并不一定由一个程序段完全实现，可以让其它对象来实现，在本例中就由类 Max 完成求最大值的功能。而面向对象的另外一个好处是实现代码的重复使用，例如其它的程序中如果需要求最大值的功能，只需要通过类 Max 的对象就可以达到目的。但是如果象面向过程的代码段那样把求最大值的算法都实现在该代码段中，则无法复用

早期的编程语言如 FORTRAN、C 基本上都是面向过程的语言，其编程的主要思路专注于算法的实现。例如下面是一个面向过程的求正整数最大值的程序：

```
int maxSoFar=0,price=1;      //最大值 maxSoFar 的初始值为 0，price 是输入的值
    while(price>0){          //循环输入 price 的值
    if (price>maxSoFar)      //输入的值 price 大于最大值 maxSoFar
    maxSoFar=price;          //则 maxSoFar 的值为 price 的值
String input=JoptionPane.showInputDialog("Enter the next price");
                         //继续输入 price
    price=Double.parseDouble(input);   //把字符串 input 转换成整数 price
    }
System.out.println("The maximum is "+maxSoFar); //打印最大值 maxSoFar
}
```

该程序段主要实现了求最大值的算法，但是，如果考虑用面向对象的编程，可以是另外一种方式：

```
Max max=new Max( );           //max 是类 Max 的一个对象
while(price>0){
      max.updateMax(price);  //对象 max 调用 updateMax 方法，更新最大值
      price=max.getPrice( ); //对象 max 调用 getPrice 获得下一个 price 的值
      }
System.out.println("The maximum is "+max.getMax( ));//对象 max 调用 getMax
                                        方法获得最大值，并打印出来
```

1.3.2 面向对象

纯粹的面向对象程序设计方法是这样的：

1. **所有的东西都是对象**。可以将对象想象成为一种新型变量，它保存着数据，而且还可以对自身数据进行操作。例如类 Max 中保留着数据的最大值，同时还有方法 updateMax 根据新加入的 price 值产生最新的最大值，还有 getMax 方法返回数据的最大值。

2. **程序是一大堆对象的组合**。通过消息传递，各对象知道自己应该做些什么。如果需要让对象做些事情，则须向该对象"发送一条消息"。具体来说，可以将消息想象成为一个调用请求，它调用的是从属于目标对象的一个方法。例如上面面向对象的程序段应该是属于某个类的，比如说是属于类 Shopping，则 Shopping 中就包含了类 Max 的对象 max，调用方法 updateMax 就相当于 Shopping 对象对 max 对象发出一条指令"updateMax"，要求对象 max 重新计算最大值。

3. **每个对象都有自己的存储空间**。可容纳其它对象，或者说通过封装现有的对象，可以产生新型对象。因此，尽管对象的概念非常简单，但是经过封装以后却可以在程序中达到任意高的复杂程度。

4. **每个对象都属于某个类**。根据语法，每个对象都是某个"类"的一个"实例"。一个类的最重要的的特征就是"能将什么消息发给它？"，也就是类本身有哪些操作。例如 max 是类 Max 的实例。

1.4 构建 Java 程序

1.4.1 第一个 Java application

java 程序分为 java application（java 应用程序）和 java applet（java 小应用程序）两种。下面让我们编写一个 java 应用程序，它能够利用来自 java 标准库的 System 对象的多种方法，打印出与当前运行的系统有关的资料。其中"//"代表一种注释方式，表示从这个符号开始到这行结束的所有内容都是注释。在每个程序文件的开头，如果这个文件的代码中用到了系统所提供的额外的类，就必须放置一个 import 语句。说它是额外的是指一个特殊的类库"java.lang"会自动导入到每个 Java 文件

```
//这是我们的第一个 java application，该程序保存在文件 Property.Java 中
import Java.util.*; /*下面我们用到了 Date 和 Properties 这两个类，是
                      属于 Java.util 这个包的；*/
                      /*而 System 和 Runtime 这两个类，是属于
                      Java.lang 这个包的。*/

public class Property { //程序员给这个类取名为 Property
  public static void main(String args[]){ //main 是类的主方法
  System.out.println(new Date( )); //在命令行下面打印出日期
  Properties p=System.getProperties( ); //获得系统的 Properties 对象 p
  p.list(System.out); //在命令行下打印出 p 中的各个系统变量的值
  System.out.println("--- Memory Usage:"); /*打印一行字符串---Memory Usage*/
  Runtime rt=Runtime.getRuntime( ); //获得系统的 Runtime 对象 rt
  System.out.println("Total Memory= "
                  + rt.totalMemory( ) //打印总内存大小
                  +" Free Memory = "
                  +rt.freeMemory( )); //打印空闲内存大小
      }
  }
```

在 java 中，程序都是以类的方式组织的，java 源文件都保存在以 java 为后缀的.java 文件当中。每个可运行的程序都是一个类文件，或者称之为字节码文件，保存在.class 文件中。而作为一个 Java application，类中必须包含主方法，程序的执行是从 main 方法开始的，方法头的格式是确定不变的：

```
public static void main(String args[])
```

其中关键字 public 意味着方法可以由外部世界调用。main 方法的参数是一个字符串数组 args，虽然在本程序中没有用到，但是必须列出来。

程序的第一行非常有意思：

```
System.out.println(new Date());
```

打印语句的参数是一个日期对象 Date，而创建 Date 对象的目的就是把它的值发给 println()语句。一旦这个语句执行完毕，Date 对象就没用了，而后"无用内存回收器"会将其收回。

第二行中调用了 System.getProperties( )。从帮助文档中可知，getProperties( )是 System 类的一个静态方法（static 方法），由于它是"静态"的，所以不必创建任何对象就可以调用该方法。在第三行，Properties 对象有一个名为 list( )的方法，它将自己的全部内容都发给一个 PrintStream 对象，该对象就是 list()方法的参数。

第四行和第六行是典型的打印语句，其中第六行通过运算符"＋"的重载来连接多个字符串对象，在 Java 中只有当"＋"运算符作用于字符串时在能够进行重载。但是让我们仔细观察下述语句：

```
System.out.println("Total Memory= "
                + rt.totalMemory( ) //打印总内存大小
                +" Free Memory = "
                +rt.freeMemory( )); //打印空闲内存大小
```

其中，totalMemory( )和 freeMemory( )返回的是数值，并非 String 对象。如果将一个字符串与一个数值相加，结果会如何？在这种情况下，编译器会自动调用一个 toString()方法，将该数值（int 型或者 float 型）转换成字符串。经过这样处理以后，就可以用"＋"进行字符串连接了。

main()的第五行通过调用 Runtime 的 getRuntime()方法创建了一个 Runtime 对象，该对象中包含了内存等信息。

1.4.2 Java 程序的编辑

java 程序的编辑可以使用任何一种文本编辑器，例如 UltraEdit、Notepad、Wordpad 甚至 word，然后只要把编辑好的文件存成.java 文件。当然也可以用一些集成开发环境，例如 Borland 公司的 JBuilder，IBM 公司的 Visualage for Java，此外还有 cafe、kawa 等其它集成开发环境。下面两幅图分别是用 UltraEdit 和 JBuilder 编辑 Property.Java 文件的情况

Sun 公司为全世界的 java 程序员提供了一个免费的 java 程序开发包（Java Develop Kit，简称 JDK），其中包括了 java 编译器命令"javac"，以及 java 执行命令"java"，还有帮助文档生成器命令"javadoc"等等。所有这些命令都可以在命令行下运行，例如我们要编译上述 java 文件 Property.java，如果是在 windows 中进行开发，就可以在"命令提示符"下进行编译，在命令行中敲入"javac Property.Java"，　　--如图--

1.4.4 Java application 的执行

当编译结束以后，在 java 源文件中的每一个类都会生成相应的 .class 文件，例如上图中就会生成一个 Property.class 文件，而 java 程序在执行时调用的是.class 文件。Java application 的执行是在命令行下进行的，如果是在 windows 系统中，就可以"命令提示符"下敲入"java Propery"进行执行，该"java"命令会启动 Java 虚拟机，并读入 Property.class 文件进行执行。如右图 1_4_2 所示：

由于该程序的运行结果直接在命令行下进行输出，其结果如右图 1_4_3 所示

1.4.5 第一个 Java applet

java 程序的另一种形式是 Java applet，applet 没有 main()方法，它必须嵌在超文本文件中，在浏览器中进行运行。右面这个程序将在浏览器中显示一行字符串.

```
//这是我们的第一个 java applet，该程序保存在文件 HelloEducation.Java 中
import Java.awt.Graphics; //在进行显示输出时，需要用到类 Graphics 的对象;
import Java.applet.Applet; //Applet 类是所有的 Java applet 的父类;

public class HelloEducation extends Applet {
                //程序员给这个类取名为 HelloEducation
                //所有的 applet 程序都是 Applet 类的子类
    public String s;
    public void init() {              //
        s=new String("Welcome to Tongfang Education");
```

```
                                //生成一个字符串对象
  }
  public void paint(Graphics g){
          g.drawString(s,25,25);
                                //在浏览器中坐标为（25，25）的位置显示字符串 s
      }
  }
```

applet 程序是从方法 init( )开始执行的，在该方法中完成了对字符串 s 的初始化工作，而显示功能是在方法 paint( )中执行的。paint( )方法是类 Applet 的一个成员方法，其参数是图形对象 Graphics g，通过调用对象 g 的 drawString( )方法就可以显示输出。

1.4.6 Java applet 的执行

java applet 程序也是一个类，其编译方式与 java application 完全一样，HelloEducation.java 程序经过编译以后就生成了 HelloEducation.class 文件。java applet 的执行方式与 java application 完全不同，Java applet 程序必须嵌入到 html 文件中才能够执行，因此必须编写相应的 html 文件。下面为 HelloEducaiton.html 文件的内容：

```
<html>
<applet code=HelloEducation.class width=250 height=250>
</applet>
</html>
```

然后可以通过 JDK 所提供的命令"appletviewer"，在命令行下面执行 Java applet 程序。如果是在 windows 操作系统中，就可以在"命令提示符"下敲入"appletviewer HelloEducation.html"，如图 1_4_4 所示。

此时系统会弹出另外一个窗口运行该 applet 程序，结果如图 1_4_5 所示。

applet 还可以采用另外一种方式运行，那就是直接在浏览器中打开 HelloEducation.html 程序，结果如图 1_4_6 所示。在主流的浏览器如 IE、Netscape 中都包含有 java 虚拟机，负责解释执行 Java applet 程序。

1.5 Java 程序规范

1.5.1 Java 源程序结构

一个完整的 Java 源程序应该包括下列部分：

```
package 语句;  //该部分至多只有一句，必须放在源程序的第一句
import 语句;  /*该部分可以有若干 import 语句或者没有，必须放在所有的
                类定义之前*/
public classDefinition;  //公共类定义部分，至多只有一个公共类的定义
        //java 语言规定该 Java 源程序的文件名必须与该公共类名完全一致
classDefinition; //类定义部分，可以有 0 个或者多个类定义
interfaceDefinition; //接口定义部分，可以有 0 个或者多个接口定义
```

例如一个 java 源程序可以是如下结构，该源程序命名为 HelloWorldApp.Java：

```
  package Javawork.helloworld; /*把编译生成的所有.class 文件放到包
                                  Javawork.helloworld 中*/
  import Java.awt.*;   //告诉编译器本程序中用到系统的 AWT 包
```

```
import Javawork.newcentury; /*告诉编译器本程序中用到用户自定义
                              的包 Javawork.newcentury*/
public class HelloWorldApp{......} /*公共类 HelloWorldApp 的定义,
                                    名字与文件名相同*/
class TheFirstClass{......} //第一个普通类 TheFirstClass 的定义
class TheSecondClass{......} //第二个普通类 TheSecondClass 的定义
                      ..... //其它普通类的定义
interface TheFirstInterface{......} /*第一个接口
                                      TheFirstInterface 的定义*/
                    ..... //其它接口定义
```

package 语句：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，包实际提供了一种命名机制和可见性限制机制。而在 java 的系统类库中，把功能相似的类放到一个包（package）中，例如所有的图形界面的类都放在 java.awt 这个包中，与网络功能有关的类都放到 java.net 这个包中。用户自己编写的类（指.class 文件）也应该按照功能放在由程序员自己命名的相应的包中，例如上例中的 javawork.helloworld 就是一个包。包在实际的实现过程中是与文件系统相对应的，例如 javawork.helloworld 所对应的目录是 path\javawork\helloworld，而 path 是在编译该源程序时指定的。比如在命令行中编译上述 HelloWorldApp.java 文件时，可以在命令行中敲入"javac -d f:\javaproject HelloWorldApp.java"，则编译生成的 HelloWorldApp.class 文件将放在目录 f:\javaproject\javawork\helloworld\目录下面，此时 f:\javaprojcet 相当于 path。但是如果在编译时不指定 path，则生成的.class 文件将放在编译时命令行所在的当前目录下面。比如在命令行目录 f:\javaproject 下敲入编译命令"javac HelloWorldApp.java"，则生成的 HelloWorldApp.class 文件将放在目录 f:\Javaproject 下面，此时的 package 语句相当于没起作用。

但是，如果程序中包含了 package 语句，则在运行时就必须包含包名。例如，HelloWorldApp.java 程序的第一行语句是：package p1.p2；编译的时候在命令行下输入"javac -d path HelloWorldApp.Java"，则 HelloWorldApp.class 将放在目录 path\p1\p2 的下面，这时候运行该程序时有两种方式：

第一种：在命令行下的 path 目录下输入字符"Java p1.p2.HelloWorldApp"。

第二种：在环境变量 classpath 中加入目录 path，则运行时在任何目录下输入"Java p1.p2.HelloWorldApp"即可。

import 语句：如果在源程序中用到了除 java.lang 这个包以外的类，无论是系统的类还是自己定义的包中的类，都必须用 import 语句标识，以通知编译器在编译时找到相应的类文件。例如上例中的 java.awt 是系统的包，而 javawork.newcentury 是用户自定义的包。比如程序中用到了类 Button，而 Button 是属于包 java.awt 的，在编译时编译器将从目录 classpath\java\awt 中去寻找类 Button，classpath 是事先设定的环境变量，比如可以设为：classpath=.；d:\jdk1.3\lib\。 classpath 也可以称为类路径，需要提醒大家注意的是，在 classpath 中往往包含多个路径，用分号隔开。例如 classpath=.；d:\jdk1.3\lib\中的第一个分号之前的路径是一个点，表示当前目录，分号后面的路径是 d:\jdk1.3\lib\，表示系统的标准类库目录。在编译过程中寻找类时，先从环境变量 classpath 的第一个目录开始往下找，比如先从当前目录往下找 java.awt 中的类 Button 时，编译器找不着，然后从环境变量 classpath 的第二个目录开始往下找，就是从系统的标准类库目录 d:\jdk1.3\lib 开始往下找 Java.awt 的 Button 这个类，最后就找到了。如果要从一个包中引入多个类则在包名后加上".*"表示。

如果程序中用到了用户自己定义的包中的类，假如在上面程序中要用到 javawork.newcentury 包中的类 HelloWorldApp，而包 javawork.newcentury 所对应的目录是 f:\javaproject\javawork\newcentury，classpath 仍旧是 classpath=.；d:\jdk1.3\lib\，则编译器在编译时将首先从当前目录寻找包 javawork.newcentury，结果是没有找到；然后又从环境变量 classpath 的第二个目录 d:\jdk1.3\lib\开始往下找，但是仍然没有找到。原因在于包 javawork.newcentury 是放在目录 f:\javaproject 下面。因此，需要重新设定环境变量 classpath，设为 classpath=.； d:\jdk1.3\lib\； f:\javaproject\ 。所以编译器从 f:\javaproject 开始找包 Javawork.newcentury 就可以找到。

源文件的命名规则：如果在源程序中包含有公共类的定义，则该源文件名必须与该公共类的名字完全一致，字母的大小写都必须一样。这是 java 语言的一个严格的规定，如果不遵守，在编译时就会出错。因此，在一个 Java 源程序中至多只能有一个公共类的定义。如果源程序中不包含公共类的定义，则该文件名可以任意取名。如果在一个源程序中有多个类定义，则在编译时将为每个类生成一个.class 文件。

1.5.2 Java 编程规范

软件开发是一个集体协作的过程，程序员之间的代码是经常要进行交换阅读的，因此，java 源程序有一些约定成俗的命名规定，主要目的是为了提高 Java 程序的可读性。

包名：包名是全小写的名词，中间可以由点分隔开，例如：Java.awt.event;

类名：首字母大写，通常由多个单词合成一个类名，要求每个单词的首字母也要大写，例如 class HelloWorldApp;

接口名：命名规则与类名相同，例如 interface Collection;

方法名：往往由多个单词合成，第一个单词通常为动词，首字母小写，中间的每个单词的首字母都要大写，例如: balanceAccount, isButtonPressed;

变量名：全小写，一般为名词，例如：length;

常量名：基本数据类型的常量名为全大写，如果是由多个单词构成，可以用下划线隔开，例如：int YEAR, int WEEK_OF_MONTH; 如果是对象类型的常量，则是大小写混合，由大写字母把单词隔开。

1.5.3 Java 帮助文档

java 中所有类库的介绍都保存在 java 帮助文档中，程序员在编程过程中，必须查阅该帮助文档，了解系统提供的类的功能、成员方法、成员变量等等信息以后，才能够更好的编程。同时，Java 开发工具包（JDK）提供了"java"、"javac"、"Javadoc"、"appletviewer"等命令，在 java 帮助文档中也对此进行了详细的介绍。Java 帮助文档是以 HTML 文件的形式组织，通常是安装在 JDK 目录下的 docs 子目录中的 index.html 文件,所以用浏览器就可以进行查阅。例如 JDK 是安装在 D:\jdk1.3 目录下面，则用浏览器打开 D:\jdk1.3\docs\index.html 文件，就可以看到图 1_5_1 所示的帮助文档。

如果希望查阅 JDK 的命令，则可以选择"Tool Documentation"，如图 1_5_2 红字所示。

此时浏览器就会把 java、javac、Javadoc、appletviewer 等命令列出来，如图 1_5_3。

但是大多时时候，我们需要查阅的是类库的文档，因此需要在"D:\jdk1.3\docs\index.html"文件中选择"Java 2 Platform API Specification"，如图 1_5_4 中红字所示。

然后就进入了详细的类库介绍，如图 1_5_5 所示。

1.5.4 Java 注释

单行注释：从"//"开始到本行结束的内容都是注释，例如：

```
//这是一行单行注释
```

```
        //则是另一行单行注释
```

多行注释：在"/*"和"*/"之间的所有内容都是注释，例如：

```
        /*这是一段注释分布在多行之中*/
```

文档注释：在注释方面 java 提供一种 C/C++所不具有的文档注释方式。其核心思想是当程序员编完程序以后，可以通过 JDK 提供的 javadoc 命令，生成所编程序的 API 文档，而该文档中的内容主要就是从文档注释中提取的。该 API 文档以 HTML 文件的形式出现，与 java 帮助文档的风格与形式完全一致。凡是在"/**"和"*/"之间的内容都是文档注释。例如下面的 DocTest.Java 文件：

```
/** 这是一个文档注释的例子，主要介绍下面这个类 */
public class DocTest{
    /** 变量注释，下面这个变量主要是充当整数计数 */
    public int i;
    /** 方法注释，下面这个方法的主要功能是计数 */
    public void count( ) {}
}
```

通过在命令行下面运行"javadoc -d . DocTest.Java"，就生成了介绍类 DocTest 的 index.html 文件，用浏览器浏览结果如图 1_5_6，注意到 DocTest.Java 文件中的文档注释的内容都出现在该 index.html 文件中。

1.6 建立 Java 开发环境

1.6.1 安装 Java Develop Kit(JDK)

Sun 公司为所有的 java 程序员提供了一套免费的 java 开发和运行环境，取名为 Java2 SDK，可以从 http://sun.com 上进行下载，也可以从同方教育网站上下载。但是最新的消息和版本必须从 Sun 的网站上才能够得到。安装的时候可以选择安装到任意的硬盘驱动器上，例如安装到 D:\jdk1.3 目录下。通常在 JDK 目录下有 bin、demo、lib、jre 等子目录，其中 bin 目录保存了 javac、java、appletviewer 等命令文件，demo 目录保存了许多 java 的例子，lib 目录保存了 java 的类库文件，jre 保存的是 Java 的运行时环境。

1.6.2 安装 Java 帮助文档

由于 JDK 的安装程序中并不包含帮助文档，因此也必须从 Sun 的网站上下载进行安装。通常安装在 JDK 所在目录的 docs 子目录下面。用浏览器打开 docs 子目录下的 index.html 文件就可以阅读所有的帮助文档。

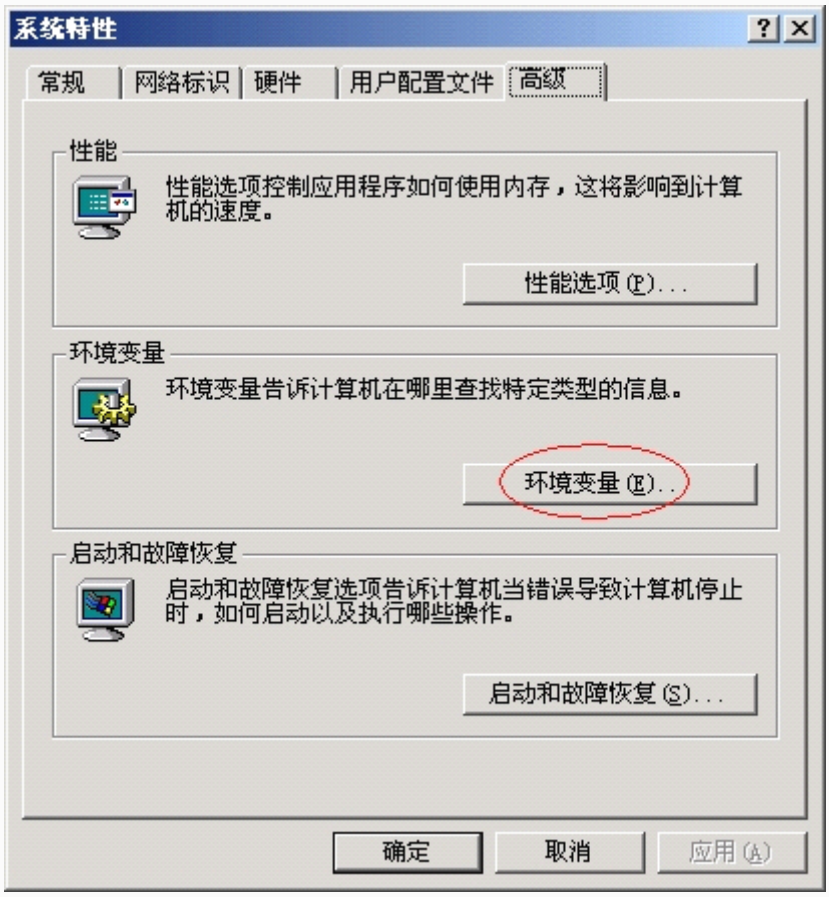
1.6.3 配置类路径

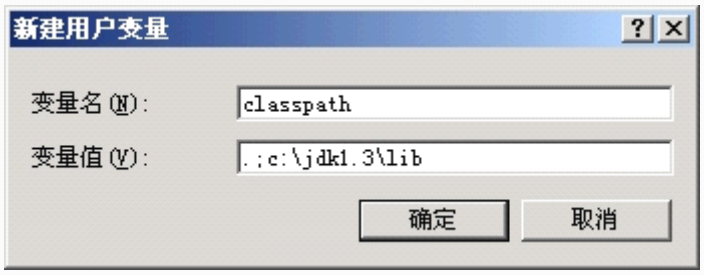
在安装完 JDK 之后，必须配置类路径 classpath 和环境变量 path，JDK 才能够正常运行。如果是在 windows98 中运行，则在

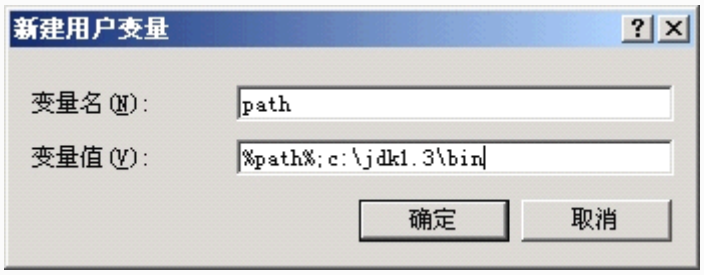
C:\autoexec.bat 文件的末尾添加下列语句：

```
  classpath= .;d:\jdk1.3\lib;
  path=%path%;d:\jdk1.3\bin;
```

如果是在 windows2000 中，则需要用右键单击桌面上"我的电脑"，选择"属性"，则弹出一个名为"系统特性"的窗口，选择"高级"，然后选择"环境变量"，在"环境变量"窗口中编辑 classpath 和 path。







【本讲小结】

java 语言的产生起源于 Sun Microsystems 公司为消费电子产品上应用程序的开发寻找一门编程语言的过程中，而随着互联网时代的到来，原有的 Oak 软件就顺理成章的改造成 java 语言推向了市场，其跨平台、面相对象、安全等特点使其得到广泛的应用。通过在不同的软硬件上实现的 java 虚拟机，java 的字节码文件就可以跨平台的进行运行，无用内存自动回收器也给程序员带来了极大的方便。java 程序以两种方式进行运行，一种是通过 java 虚拟机进行直接运行的 java application，另一种是通过浏览器进行运行的 java applet，但是无论是何种方式，java 都是一门纯粹的面向对象的编程语言。面向对象编程的思路认为程序都是对象的组合，因此要克服面向过程编程的思路，直接按照对象和类的思想去组织程序，面向对象所具有的封装性、继承性、多态性等特点使其具有强大的生命力。Sun 公司为全世界 java 开发人员提供了一套免费的软件开发包 Java2 SDK，也称为 JDK，它不仅是 java 的开发平台，还是 java 的运行平台。java 源程序存放在.java 文件中，可以通过任意一个文本编辑器编辑产生，源程序经过"javac"命令编译过后，就生成了相应的.class 文件，而用"java"命令就可以运行.class 文件。作为面向对象编程人员来说，大体可以分为两

种：类创建者和应用程序员，应用程序员是类的使用者。所以对程序的可读性和 API 帮助文档就有要求，Java 语言本身有一套约定成俗的编程规范，同时程序员首先要学会阅读系统 API 帮助文档，还要学会生成自己编写的程序的 API 帮助文档。

JAVA 教程 第二讲 Java 语言基础知识

2．1 简单数据类型

2．1．1 标识符和保留字

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）。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。例如，identifier，userName，User_Name，_sys_val， $change 为合法的标识符，而 2mail room#，class 为非法的标识符。

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),也称为关键字，下面列出了 Java 语言中的所有保留字：

bstract，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，return，switch，synchronized，short，static，super，try，true，this，throw，throws，threadsafe，transient，void，while 。

Java 语言中的保留字均用小写字母表示。

2．1．2 数据类型概

1． Java 中的数据类型划分

Java 语言的数据类型有简单类型和复合类型：

简单数据类型包括：

整数类型（Integer）：byte， short， int， long

浮点类型（Floating）：float，double

字符类型（Textual）：char

布尔类型（Logical）：boolean

复合数据类型包括：

class

interface

数组

2．常量和变量

常量：用保留字 final 来实现

```
final typeSpecifier varName=value[,varName[=value]…];
                                    如：final int NUM=100;
```

变量：是 Java 程序中的基本存储单元，它的定义包括变量名、变量类型和作用域几个部分。其定义格式如下：

```
typeSpecifier varName[=value[,varName[=value]…];
```

如：int count; char c='a';

变量的作用域指明可访问该变量的一段代码，声明一个变量的同时也就指明了变量的作用域。按作用域来分，变量可以有下面几种：局部变量、类变量、方法参数和例外处理参数。在一个确定的域中，变量名应该是唯一的。局部变量在方法或方法的一个块代码中声明，它的作用域为它所在的代码块（整个方法或方法中的某块代码）。类变量在类中声明，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。例外处理参数传递给例外处理代码，它的作用域就是例外处理部分。

2．1．3 简单数据类型

1．布尔类型--boolean

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;
```

2．字符类型--char

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，其范围为 0～65535。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'*/
```

3．整型数据

整型常量：

◇ 十进制整数

如 123，－456，0

◇ 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。

◇ 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：

数据类型	所占位数	数的范围
byte	8	$-2^7 \sim 2^7-1$
bhort	16	$-2^{15} \sim 2^{15}-1$
int	32	$-2^{31} \sim 2^{31}-1$
long	64	$-2^{63} \sim 2^{63}-1$

4．浮点型（实型）数据

实型常量：

◇ 十进制数形式

由数字和小数点组成，且必须有小数点，如 0.123，1.23，123.0

◇ 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。

◇ float 型的值,必须在数字后加 f 或 F,如 1.23f。

实型变量：

数据类型	所占位数	数的范围
float	32	$3.4e^{-038}$ ～$3.4e^{+038}$
double	64	$1.7e^{-038}$ ～$1.7e^{+038}$

5. 简单数据类型的例子：

【例 2. 1】

```
public class Assign {
  public static void main (String args [ ] ) {
  int x , y ; //定义 x，y 两个整型变量
  float z = 1.234f ; //指定变量 z 为 float 型，且赋初值为 1.234
  double w = 1.234 ; //指定变量 w 为 double 型，且赋初值为 1.234
  boolean flag = true ; //指定变量 flag 为 boolean 型，且赋初值为 true
  char c ; //定义字符型变量 c
  String str ; //定义字符串变量 str
  String str1 = " Hi " ; //指定变量 str1 为 String 型，且赋初值为 Hi
  c = ' A ' ; //给字符型变量 c 赋值'A'
  str = " bye " ; //给字符串变量 str 赋值"bye"
  x = 12 ; //给整型变量 x 赋值为 12
  y = 300; //给整型变量 y 赋值为 300
  }
}
```

2. 1. 4 简单数据类型中各类型数据间的优先关系和相互转换

1. 不同类型数据间的优先关系如下：

低----------->高

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

2. 自动类型转换规则

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；

操作数 1 类型	操作数 2 类型	转换后的类型
byte、short、char	int	int
byte、short、char、int	long	long
byte、short、char、int、long	float	float
byte、short、char、int、long、float	double	double

3．强制类型转换

高级数据要转换成低级数据，需用到强制类型转换，如：

```
int i;
byte b=(byte)i; /*把 int 型变量 i 强制转换为 byte 型*/
```

2．2 运算符和表达式

2．2．1 运算符

对各种类型的数据进行加工的过程成为运算，表示各种不同运算的符号称为运算符，参与运算的数据称为操作数，按操作数的数目来分，可有：

◇ 一元运算符：＋＋，－－，＋，－

◇ 二元运算符：＋，－，>

◇ 三元运算符：？：

基本的运算符按功能划分，有下面几类：

1 算术运算符： +，－，*，/，%，++，－－。

例如：

```
3+2;
a-b;
i++;
--i;
```

2 关系运算符： >，<，>=，<=，==，!=。

例如：

```
count>3;
I==0;
n!=-1;
```

3 布尔逻辑运算符： !，&&，|| 。

例如：

```
flag=true;
!(flag);
flag&&false;
```

4 位运算符： >>，<<，>>>，&，|，^，~。

例如：

a=10011101；b=00111001；则有如下结果：

```
a<<3 =11101000;
a>>3 =11110011 a>>>3=00010011;
a&b=00011001; a|b=10111101;
~a=01100010; a^b=10100100;
```

5 赋值运算符 =，及其扩展赋值运算符如+=，－=，*=，/=等。

例如：

```
i=3;
i+=3;          //等效于 i=i+3;
```

6 条件运算符 ？：

例如：result=(sum= =0 ？ 1 ： num/sum);

7 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。例如：

```
System.out.println("hello world");
int array1[]=new int[4];
```

2．2．2 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。

1．表达式的类型

表达式的类型由运算以及参与运算的操作数的类型决定，可以是简单类型，也可以是复合类型：

布尔型表达式： x&&y||z;

整型表达式： num1+num2;

2．运算符的优先次序

表达式的运算按照运算符的优先顺序从高到低进行,同级运算符从左到右进行：

优先次序	运算符
1	. [] ()
2	++ -- ！ ~ instanceof
3	new (type)
4	* / %
5	+ -
6	>> >>> <<
7	> < >= <=
8	= = !=
9	&
10	^
11	|
12	&&
13	||
14	?:
15	= += -= *= /= %= ^=
16	&= |= <<= >>= >>>=

例如，下述条件语句分四步完成：

```
Result=sum==0?1:num/sum;
```

第 1 步：result=sum==0?1:(num/sum)
第 2 步：result=(sum==0)?1:(num/sum)
第 3 步：result=((sum==0)?1:(num/sum))
第 4 步：result=

2．3 控制语句

Java 程序通过控制语句来执行程序流，完成一定的任务。程序流是由若干个语句组成的，语句可以是单一的一条语句，如 c=a+b,也可以是用大括号{}括起来的一个复合语句。Java 中的控制语句有以下几类：

◇ 分支语句：if-else, switch
◇ 循环语句：while, do-while, for
◇ 与程序转移有关的跳转语句：break, continue, return
◇ 例外处理语句：try-catch-finally, throw
◇ 注释语句：//，/* */， /** */

2．3．1 分支语句

分支语句提供了一种控制机制，使得程序的执行可以跳过某些语句不执行，而转去执行特定的语句。

1. 条件语句 if-else

```
if(boolean-expression)
  statement1;
[else statement2;]
```

2. 多分支语句 switch

```
switch (expression){
  case value1 : statement1;
break;
  case value2 : statement2;
break;
……
  case valueN : statemendN;
break;
  [default : defaultStatement; ]
}
```

◇ 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 valueN 必须是常量，而且所有 case 子句中的值应是不同的。
◇ default 子句是可选的。
◇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（在一些特殊情况下，多个不同的 case 值要执行一组相同的操作，这时可以不用 break）。

2．3．2 循环语句

循环语句的作用是反复执行一段代码，直到满足终止循环的条件为止。Java 语言中提供的循环语句有：
◇ while 语句

◇ do-while 语句
◇ for 语句

1. while 语句

```
[initialization]
while (termination){
    body;
[iteration;]
}
```

2. do-while 语句

```
[initialization]
do {
    body;
[iteration;]
} while (termination);
```

3. for 语句

```
for (initialization; termination; iteration){
    body;
}
```

◇ 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( i=0, j=10; i<j; i++, j--){
  ……
}
```

2．3．3 跳转语句

◇ break 语句
◇ continue 语句
◇ 返回语句 return

1. break 语句

◇ 在 switch 语中，break 语句用来终止 switch 语句的执行。使程序从 switch 语句后的第一个语句开始执行。
◇ 在 Java 中，可以为每个代码块加一个括号，一个代码块通常是用大括号{}括起来的一段代码。加标号的格式如下：

```
BlockLabel: { codeBlock }
```

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

```
break BlockLabel;
break 语句
a:{…… //标记代码块 a
b:{…… //标记代码块 b
c:{…… //标记代码块 c
break b;
  …… //此处的语句块不被执行
}
  …… /此处的语句块不被执行
}
  …… //从此处开始执行
}
```

2. continue 语句

continue 语句用来结束本次循环，跳过循以介绍面向对象编程的基本概念、基本理论为重点，结合 Java 语言的语法规则、编程特点和设计思想、强调容易发生错误和编程应注意的地方，使学生能对 Java 技术有一个总体了解，通过本课程学习，使学生掌握 Java 语言的基础知识，理解和掌握面向对象程序设计的基本思想，熟练地使用 Java 语言进行程序的编写、编译以及调试工作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;
```

例如：

```
outer: for( int i=0; i<10; i++ ){ //外层循环
inner: for( int j=0; j<10; j++ ){ //内层循环
if( i<j ){
    ……
continue outer;

}
    ……
}
    ……
}
```

3. 返回语句 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 ;
return;
```

return 语句通常用在一个方法体的最后,否则会产生编译错误,除非用在 if-else 语句中

2．3．4 例外处理语句

包括 try,catch,finally,throw 语

2．4 数组

Java 语言中，数组是一种最简单的复合数据类型。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。

2．4．1 一维数组

1． 一维数组的定义

type arrayName[ ];
类型(type)可以为 Java 中任意的数据类型，包括简单类型和复合类型。
例如：

```
int intArray[ ];
Date dateArray[];
```

2．一维数组的初始化

◇ **静态初始化**

```
int intArray[]={1,2,3,4};
String stringArray[]={"abc", "How", "you"};
```

◇ **动态初始化**

1）简单类型的数组

```
int intArray[];
intArray = new int[5];
```

2）复合类型的数组

```
String stringArray[ ];
String stringArray = new String[3];/*为数组中每个元素开辟引用
                                    空间(32 位) */
stringArray[0]= new String("How");//为第一个数组元素开辟空间
stringArray[1]= new String("are");//为第二个数组元素开辟空间
stringArray[2]= new String("you");// 为第三个数组元素开辟空间
```

3．一维数组元素的引用

数组元素的引用方式为：

```
arrayName[index]
```

index 为数组下标，它可以为整型常数或表达式，下标从 0 开始。每个数组都有一个属性 length 指明它的长度，例如：intArray.length 指明数组 intArray 的长度。

2．4．2 多维数组

Java 语言中，多维数组被看作数组的数组。

1．二维数组的定义

```
type arrayName[ ][ ];
type [ ][ ]arrayName;
```

2．二维数组的初始化

◇ **静态初始化**

```
int intArray[ ][ ]={{1,2},{2,3},{3,4,5}};
```

Java 语言中，由于把二维数组看作是数组的数组，数组空间不是连续分配的，所以不要求二维数组每一维的大小相同。

◇ **动态初始化**

1）直接为每一维分配空间，格式如下：

```
arrayName = new type[arrayLength1][arrayLength2];
int a[ ][ ] = new int[2][3];
```

2）从最高维开始，分别为每一维分配空间：

```
arrayName = new type[arrayLength1][ ];
arrayName[0] = new type[arrayLength20];
arrayName[1] = new type[arrayLength21];
…
arrayName[arrayLength1-1] = new type[arrayLength2n];
```

3）例：

二维简单数据类型数组的动态初始化如下，

```
int a[ ][ ] = new int[2][ ];
a[0] = new int[3];
a[1] = new int[5];
```

对二维复合数据类型的数组，必须首先为最高维分配引用空间，然后再顺次为低维分配空间。
而且，必须为每个数组元素单独分配空间。

例如：

```
String s[ ][ ] = new String[2][ ];
s[0]= new String[2];//为最高维分配引用空间
s[1]= new String[2]; //为最高维分配引用空间
s[0][0]= new String("Good");// 为每个数组元素单独分配空间
s[0][1]= new String("Luck");// 为每个数组元素单独分配空间
s[1][0]= new String("to");// 为每个数组元素单独分配空间
s[1][1]= new String("You");// 为每个数组元素单独分配空间
```

3．二维数组元素的引用

对二维数组中的每个元素，引用方式为：arrayName[index1][index2]
例如： num[1][0];

4. 二维数组举例：

【例 2. 2】两个矩阵相乘

```
public class MatrixMultiply{
  public static void main(String args[]){
  int i,j,k;
  int a[][]=new int [2][3]; //动态初始化一个二维数组
  int b[][]={{1,5,2,8},{5,9,10,-3},{2,7,-5,-18}};//静态初始化
                                                  一个二维数组
  int c[][]=new int[2][4]; //动态初始化一个二维数组
  for (i=0;i<2;i++)
      for (j=0; j<3 ;j++)
        a[i][j]=(i+1)*(j+2);
  for (i=0;i<2;i++){
      for (j=0;j<4;j++){
        c[i][j]=0;
  for(k=0;k<3;k++)
      c[i][j]+=a[i][k]*b[k][j];
        }
      }
  System.out.println("*******Matrix C********");//打印 Matrix C 标记
  for(i=0;i<2;i++){
      for (j=0;j<4;j++)
        System.out.println(c[i][j]+" ");
      System.out.println();
        }
      }
  }
```

2. 5 字符串的处理

2. 5. 1 字符串的表示

Java 语言中，把字符串作为对象来处理，类 String 和 StringBuffer 都可以用来表示一个字符串。(类名都是大写字母打头)

1. 字符串常量

字符串常量是用双引号括住的一串字符。

```
"Hello World!"
```

2. String 表示字符串常量

用 String 表示字符串：

```
String( char chars[ ] );
String( char chars[ ], int startIndex, int numChars );
String( byte ascii[ ], int hiByte );
String( byte ascii[ ], int hiByte, int startIndex, int numChars );
```

String 使用示例：

String s=new String() ; 生成一个空串

下面用不同方法生成字符串"abc"：

```
char chars1[]={'a','b','c'};
char chars2[]={'a','b','c','d','e'};
String s1=new String(chars1);
String s2=new String(chars2,0,3);
byte ascii1[]={97,98,99};
byte ascii2[]={97,98,99,100,101};
String s3=new String(ascii1,0);
String s4=new String(ascii2,0,0,3);
```

3. 用 StringBuffer 表示字符串

```
StringBuffer( ); /*分配 16 个字符的缓冲区*/
StringBuffer( int len ); /*分配 len 个字符的缓冲区*/
StringBuffer( String s ); /*除了按照 s 的大小分配空间外,再分配 16 个
                            字符的缓冲区*/
```

2．5．2 访问字符串

1. 类 String 中提供了 length( )、charAt( )、indexOf( )、lastIndexOf( )、getChars( )、getBytes( )、toCharArray( )等方法。

◇ public int length() 此方法返回字符串的字符个数

◇ public char charAt(int index) 此方法返回字符串中 index 位置上的字符，其中 index 值的 范围是 0~length-1

◇ public int indexOf(int ch)
public lastIndexOf(in ch)

返回字符 ch 在字符串中出现的第一个和最后一个的位置

◇ public int indexOf(String str)
public int lastIndexOf(String str)

返回子串 str 中第一个字符在字符串中出现的第一个和最后一个的位置

◇ public int indexOf(int ch,int fromIndex)
public lastIndexOf(in ch ,int fromIndex)

返回字符 ch 在字符串中位置 fromIndex 以后出现的第一个和最后一个的位置

◇ public int indexOf(String str,int fromIndex)
public int lastIndexOf(String str,int fromIndex)

返回子串 str 中的第一个字符在字符串中位置 fromIndex 后出现的第一个和最后一个的位置。

◇ public void getchars(int srcbegin,int end ,char buf[],int dstbegin)

srcbegin 为要提取的第一个字符在源串中的位置， end 为要提取的最后一个字符在源串中的位置，字符数组 buf[]存放目的字符串， dstbegin 为提取的字符串在目的串中的起始位置。

◇public void getBytes(int srcBegin, int srcEnd,byte[] dst, int dstBegin)
参数及用法同上，只是串中的字符均用 8 位表示。

2. 类 StringBuffer 提供了 length( )、charAt( )、getChars( )、capacity()等方法。

方法 capacity()用来得到字符串缓冲区的容量，它与方法 length()所返回的值通常是不同的。

2. 5. 3 修改字符串

修改字符串的目的是为了得到新的字符串，类 String 和类 StringBuffer 都提供了相应的方法。有关各个方法的使用，参考 Java 2 API。

1. String 类提供的方法：

concat( )
replace( )
substring( )
toLowerCase( )
toUpperCase( )

◇ public String contat(String str);
用来将当前字符串对象与给定字符串 str 连接起来。
◇ public String replace(char oldChar,char newChar);
用来把串中出现的所有特定字符替换成指定字符以生成新串。
◇ public String substring(int beginIndex);
public String substring(int beginIndex,int endIndex);
用来得到字符串中指定范围内的子串。
◇ public String toLowerCase();
把串中所有的字符变成小写。
◇ public String toUpperCase();
把串中所有的字符变成大写。

2. StringBuffer 类提供的方法：

append( )
insert( )
setCharAt( )

如果操作后的字符超出已分配的缓冲区,则系统会自动为它分配额外的空间。
◇ public synchronized StringBuffer append(String str);
用来在已有字符串末尾添加一个字符串 str。
◇ public synchronized StringBuffer insert(int offset, String str);
用来在字符串的索引 offset 位置处插入字符串 str。
◇ public synchronized void setCharAt(int index,char ch);
用来设置指定索引 index 位置的字符值。

注意：String 中对字符串的操作不是对源操作串对象本身进行的，而是对新生成的一个源操作串对象的拷贝进行的，其操作的结果不影响源串。

相反，StringBuffer 中对字符串的连接操作是对源串本身进行的，操作之后源串的值发生了变化，变成连接后的串。

2．5．4 其它操作

1．字符串的比较

String 中提供的方法：
equals( )和 equalsIgnoreCase( )
它们与运算符'= ='实现的比较是不同的。运算符'= ='比较两个对象是否引用同一个实例，而 equals( )和 equalsIgnoreCase( )则比较　　两个字符串中对应的每个字符值是否相同。

2．字符串的转化

Java.lang.Object 中提供了方法 toString( )把对象转化为字符串。

3．字符串"+"操作

运算符'+'可用来实现字符串的连接：
String s = "He is "+age+" years old.";
其他类型的数据与字符串进行"+"运算时，将自动转换成字符串。具体过程如下：
String s=new StringBuffer("he is").append(age).append("years old").toString();

注意：除了对运算符"+"进行了重载外，Java 不支持其它运算符的重载。

【本讲小结】

java 中的数据类型有简单数据类型和复合数据类型两种，其中简单数据类型包括整数类型、浮点类型、字符类型和布尔类型；复合数据类型包含类、接口和数组。表达式是由运算符和操作数组成的符号序列，对一个表达式进行运算时，要按运算符的优先顺序从高向低进行，同级的运算符则按从左到右的方向进行。条件语句、循环语句和跳转语句是 Java 中常用的控制语句。

数组是最简单的复合数据类型，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。Java 中，对数组定义时并不为数组元素分配内存，只有初始化后，才为数组中的每一个元素分配空间。已定义的数组必须经过初始化后，才可以引用。数组的初始化分为静态初始化和动态初始化两种，其中对复合数据类型数组动态初始化时，必须经过两步空间分配：首先，为数组开辟每个元素的引用空间；然后，再为每个数组元素开辟空间。Java 中把字符串当作对象来处理， Java.lang.String 类提供了一系列操作字符串的方法，使得字符串的生成、访问和修改等操作容易和规范。

JAVA 教程 第三讲 Java 语言中的面向对象特性

3．1 面向对象技术基础

3．1．1 面向对象的基本概念

面向对象的基本思想
面向对象是一种新兴的程序设计方法,或者是一种新的程序设计规范(paradigm),其基本思想是使用对象、类、继

承、封装、消息等基本概念来进行程序设计。从现实世界中客观存在的事物（即对象）出发来构造软件系统，并且在系统构造中尽可能运用人类的自然思维方式。开发一个软件是为了解决某些问题，这些问题所涉及的业务范围称作该软件的问题域。其应用领域不仅仅是软件，还有计算机体系结构和人工智能等。

1. 对象的基本概念

对象是系统中用来描述客观事物的一个实体，它是构成系统的一个基本单位。一个对象由一组属性和对这组属性进行操作的一组服务组成。从更抽象的角度来说，对象是问题域或实现域中某些事物的一个抽象，它反映该事物在系统中需要保存的信息和发挥的作用；它是一组属性和有权对这些属性进行操作的一组服务的封装体。客观世界是由对象和对象之间的联系组成的。

主动对象是一组属性和一组服务的封装体，其中至少有一个服务不需要接收消息就能主动执行（称作主动服务）。

2. 类的基本概念

把众多的事物归纳、划分成一些类是人类在认识客观世界时经常采用的思维方法。分类的原则是抽象。类是具有相同属性和服务的一组对象的集合，它为属于该类的所有对象提供了统一的抽象描述，其内部包括属性和服务两个主要部分。在面向对象的编程语言中，类是一个独立的程序单位，它应该有一个类名并包括属性说明和服务说明两个主要部分。类与对象的关系就如模具和铸件的关系，类的实例化结果就是对象，而对一类对象的抽象就是类。

3. 消息

消息就是向对象发出的服务请求，它应该包含下述信息：提供服务的对象标识、服务标识、输入信息和回答信息。服务通常被称为方法或函数。

3．1．2 面向对象的基本特征

1．封装性

封装性就是把对象的属性和服务结合成一个独立的相同单位，并尽可能隐蔽对象的内部细节，包含两个含义：

◇ 把对象的全部属性和全部服务结合在一起，形成一个不可分割的独立单位（即对象）。

◇ 信息隐蔽，即尽可能隐蔽对象的内部细节，对外形成一个边界〔或者说形成一道屏障〕，只保留有限的对外接口使之与外部发生联系。

封装的原则在软件上的反映是：要求使对象以外的部分不能随意存取对象的内部数据（属性），从而有效的避免了外部错误对它的"交叉感染"，使软件错误能够局部化，大大减少查错和排错的难度。

2．继承性

特殊类的对象拥有其一般类的全部属性与服务，称作特殊类对一般类的继承。例如，轮船、客轮；人、大人。一个类可以是多个一般类的特殊类，它从多个一般类中继承了属性与服务，这称为多继承。例如，客轮是轮船和客运工具的特殊类。在 Java 语言中，通常我们称一般类为父类（superclass,超类），特殊类为子类(subclass)。

3．多态性

对象的多态性是指在一般类中定义的属性或服务被特殊类继承之后，可以具有不同的数据类型或表现出不同的行

为。这使得同一个属性或服务在一般类及其各个特殊类中具有不同的语义。例如："几何图形"的"绘图"方法，"椭圆"和"多边形"都是"几何图"的子类，其"绘图"方法功能不同。

3．1．3 面向对象程序设计方法

OOA－Object Oriented Analysis　　面向对象的分析

OOD－Object Oriented Design　　面向对象的设计

OOI－Object Oriented Implementation　　面向对象的实现

3．2 Java 语言的面向对象特性

3．2．1 类

类是 java 中的一种重要的复合数据类型，是组成 Java 程序的基本要素。它封装了一类对象的状态和方法，是这一类对象的原形。一个类的实现包括两个部分：**类声明和类体**。

1．类声明：

```
[public][abstract|final] class className [extends superclassName] [implements interfaceNameList]
{……}
```

其中，修饰符 public,abstract,final 说明了类的属性，className 为类名，superclassName 为类的父类的名字，interfaceNameList 为类所实现的接口列表。

2．类体

类体定义如下：

```
class className
{[public | protected | private ] [static]
[final] [transient] [volatile] type
variableName;                                    //成员变量
[public | protected | private ] [static]
[final | abstract] [native] [synchronized]
returnType methodName([paramList]) [throws exceptionList]
  {statements}                                   //成员方法
}
```

3．成员变量

成员变量的声明方式如下：

```
[public | protected | private ] [static]
[final] [transient] [volatile] type
variableName;                                    //成员变量
```

其中，

static：静态变量（类变量）；相对于实例变量

final：常量

transient：暂时性变量，用于对象存档
volatile：贡献变量，用于并发线程的共享

4．成员方法

方法的实现包括两部分内容：方法声明和方法体。

```
[public | protected | private ] [static]
[final  | abstract] [native] [synchronized]
returnType methodName([paramList])
[throws exceptionList]                          //方法声明
  {statements}                                  //方法体
```

方法声明中的限定词的含义：
static：类方法，可通过类名直接调用
abstract：抽象方法，没有方法体
final：方法不能被重写
native：集成其它语言的代码
synchronized：控制多个并发线程的访问

◇ 方法声明

方法声明包括方法名、返回类型和外部参数。其中参数的类型可以是简单数据类型，也可以是复合数据类型（又称引用数据类型）。

对于简单数据类型来说，Java 实现的是值传递，方法接收参数的值，但不能改变这些参数的值。如果要改变参数的值，则用引用数据类型，因为引用数据类型传递给方法的是数据在内存中的地址，方法中对数据的操作可以改变数据的值。

例 3-1 说明了简单数据类型与引用数据的区别。

【例 3-1】

```
import Java.io.*;
public class PassTest{
float ptValue;
public static void main(String args[]) {
int val;
PassTest pt=new PassTest();
val=11;
System.out.println("Original Int Value is:"+val);
pt.changeInt(val);                                      //值参数
System.out.println("Int Value after Change is:" +val); /*值参数
                                  值的修改，没有影响值参数的值*/
pt.ptValue=101f;
System.out.println("Original ptValue is:"+pt.ptValue);
pt.changeObjValue(pt); //引用类型的参数
System.out.println("ptValue after Change is:"+pt.ptValue);/* 引用参数值的修改，改变了引用参数的值*/

}
public void changeInt(int value){
```

```
    value=55;                          //在方法内部对值参数进行了修改
    }
    public void changeObjValue(PassTest ref){
    ref.ptValue=99f;                   //在方法内部对引用参数进行了修改
        }
    }
```

运行结果

```
c:\>Java PassTest
Original Int Value is : 11
Int Value after Change is: 11
Original ptValue is: 101.0
ptValue after Change is : 99.0
```

◇ **方法体**

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。方法体中声明的局部变量的作用域在该方法内部。若局部变量与类的成员变量同名，则类的成员变量被隐藏。

例 3-2 说明了局部变量 z 和类成员变量 z 的作用域是不同的。

【例 3-2】

```
    import Java.io.*;
    class Variable{
    int x=0,y=0,z=0;                          //类的成员变量
    void init(int x,int y) {
    this.x=x; this.y=y;
    int z=5;                                  //局部变量
    System.out.println("** in init**");
    System.out.println("x="+x+" y="+y+" z="+z);
        }
    }
    public class VariableTest{
    public static void main(String args[]){
    Variable v=new Variable();
    System.out.println("**before init**");
    System.out.println("x="+v.x+" y="+ v.y+" z="+v.z);
    v.init(20,30);
    System.out.println("**after init**");
    System.out.println("x="+v.x+ " y="+ v.y+" z="+v.z);
        }
    }
```

运行结果

```
c:\>Java VariableTest
```

```
**before init**
x=0 y=0 z=0
** in init **
x=20 y=30 z=5
**after init**
x=20 y=30 z=0
```

上例中我们用到了 this，这是因为 init()方法的参数名与类的成员变量 x,y 的名字相同，而参数名会隐藏成员变量，所以在方法中，为了区别参数和类的成员变量，我们必须使用 this。this-----用在一个方法中引用当前对象,它的值是调用该方法的对象。返回值须与返回类型一致，或者完全相同，或是其子类。当返回类型是接口时，返回值必须实现该接口。

5. 方法重载

方法重载是指多个方法享有相同的名字，但是这些方法的参数必须不同，或者是参数的个数不同，或者是参数类型不同。返回类型不能用来区分重载的方法。

参数类型的区分度一定要足够，例如不能是同一简单类型的参数，如 int 与 long。

【例 3-3】

```
import Java.io.*;
class MethodOverloading{
void receive(int i) {
System.out.println("Receive one int data");
System.out.println("i="+i);
}
void receive(int x, int y) {
System.out.println("Receive two int datas");
System.out.println("x="+x+" y="+y);
    }
}
public class MethodOverloadingTest{
public static void main(String args[]) {
MethodOverloading mo=new MethodOverloading();
mo.receive(1);
mo.receive(2,3);

    }
}
```

运行结果(编译器会根据参数的个数和类型来决定当前所使用的方法)

```
c:\>Java MethodOverloadingTest

Receive one int data
```

```
i=1
Receive two int datas
x=2 y=3
```

6. 构造方法

◇ 构造方法是一个特殊的方法。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。
◇ 重载经常用于构造方法。
◇ 构造方法只能由 new 运算符调用

【例 3-4】

```
class Point{
int x,y;
Point(){
x=0; y=0;
}
Point(int x, int y){
this.x=x;
this.y=y;
    }

}
```

3．2．2 对象

类实例化可生成对象，对象通过消息传递来进行交互。消息传递即激活指定的某个对象的方法以改变其状态或让它产生一定的行为。一个对象的生命周期包括三个阶段：生成、使用和消除。

1. 对象的生成

对象的生成包括声明、实例化和初始化。
格式为：

```
type objectName=new type([paramlist]);
```

◇ **声明**：type objectName
声明并不为对象分配内存空间，而只是分配一个引用空间；对象的引用类似于指针，是 32 位的地址空间，它的值指向一个中间的数据结构，它存储有关数据类型的信息以及当前对象所在的堆的地址，而对于对象所在的实际的内存地址是不可操作的，这就保证了安全性。

◇ **实例化**：运算符 new 为对象分配内存空间，它调用对象的构造方法，返回引用；一个类的不同对象分别占据不同的内存空间。

◇ **生成**：执行构造方法，进行初始化；根据参数不同调用相应的构造方法。

2. 对象的使用

通过运算符"."可以实现对变量的访问和方法的调用。变量和方法可以通过设定访问权限来限制其它对象对它的访问。

◇调用对象的变量

格式：objectReference.variable

objectReference 是一个已生成的对象，也可以是能生成对象的表达式

例： p.x= 10;

tx=new Point( ).x;

◇调用对象的方法

格式：objectReference.methodName([paramlist]);

例如：p.move(30,20);

new Point( ).move(30,20);

3. 对象的清除

当不存在对一个对象的引用时，该对象成为一个无用对象。Java 的垃圾收集器自动扫描对象的动态内存区，把没有引用的对象作为垃圾收集起来并释放。

System.gc( );

当系统内存用尽或调用 System.gc( )要求垃圾回收时，垃圾回收线程与系统同步运行。

3．2．3 面向对象特性

Java 语言中有三个典型的面向对象的特性：封装性、继承性和多态性，下面将详细阐述。

1． 封装性

Java 语言中，对象就是对一组变量和相关方法的封装，其中变量表明了对象的状态，方法表明了对象具有的行为。通过对象的封装，实现了模块化和信息隐藏。通过对类的成员施以一定的访问权限，实现了类中成员的信息隐藏。

◇ 类体定义的一般格式：

```
class className
{     [public | protected | private ] [static]
      [final] [transient] [volatile] type
      variableName;                          //成员变量
      [public | protected | private ] [static]
      [final | abstract] [native] [synchronized]
      returnType methodName([paramList])
      [throws exceptionList]
      {statements} //成员方法
}
```

◇ Java 类中的限定词

Java 语言中有四种不同的限定词，提供了四种不同的访问权限。

1） private

类中限定为 private 的成员，只能被这个类本身访问。

如果一个类的构造方法声明为 private,则其它类不能生成该类的一个实例。

2） default

类中不加任何访问权限限定的成员属于缺省的（default）访问状态，可以被这个类本身和同一个包中的类所访问。

3） protected

类中限定为 protected 的成员，可以被这个类本身、它的子类（包括同一个包中以及不同包中的子类）和同一个包中的所有其他的类访问。

4） public

类中限定为 public 的成员，可以被所有的类访问。

表 3-1 列出了这些限定词的作用范围。

【表 3-1】 Java 中类的限定词的作用范围比较

	同一个类	同一个包	不同包的子类	不同包非子类
private	*			
default	*	*		
protected	*	*	*	
public	*	*	*	*

2. 继承性

通过继承实现代码复用。Java 中所有的类都是通过直接或间接地继承 java.lang.Object 类得到的。继承而得到的类称为子类，被继承的类称为父类。子类不能继承父类中访问权限为 private 的成员变量和方法。子类可以重写父类的方法，及命名与父类同名的成员变量。但 Java 不支持多重继承，即一个类从多个超类派生的能力。

◇ 创建子类

格式：

```
class SubClass extends SuperClass {
…
}
```

◇ 成员变量的隐藏和方法的重写

子类通过隐藏父类的成员变量和重写父类的方法，可以把父类的状态和行为改变为自身的状态和行为。

例如：

```
class SuperClass{
    int x; …
    void setX( ){ x=0; } …
}
class SubClass extends SuperClass{
    int x;      //隐藏了父类的变量 x
    …
    void setX( ) { //重写了父类的方法 setX()
    x=5; } ….
}
```

注意：子类中重写的方法和父类中被重写的方法要具有相同的名字，相同的参数表和相同的返回类型，只是函数体不同。

◇ super

Java 中通过 super 来实现对父类成员的访问，super 用来引用当前对象的父类。Super 的使用有三种情况：

1）访问父类被隐藏的成员变量，如：

```
super.variable;
```

2）调用父类中被重写的方法，如：

```
super.Method([paramlist]);
```

3）调用父类的构造函数，如：

```
super([paramlist]);
```

【例 3-5】

```
import Java.io.*;
class SuperClass{
    int x;
    SuperClass( ) {
      x=3;
      System.out.println("in SuperClass : x=" +x);
    }
      void doSomething( ) {
      System.out.println("in SuperClass.doSomething()");
    }
}
class SubClass extends SuperClass {
    int x;
    SubClass( ) {
      super( );        //调用父类的构造方法
      x=5;             //super( ) 要放在方法中的第一句
      System.out.println("in SubClass :x="+x);
    }
      void doSomething( ) {
      super.doSomething( ); //调用父类的方法
      System.out.println("in SubClass.doSomething()");
      System.out.println("super.x="+super.x+" sub.x="+x);
    }
}
public class Inheritance {
      public static void main(String args[]) {
      SubClass subC=new SubClass();
      subC.doSomething();
    }
}
```

运行结果

```
c:\> Java Inheritance
in SuperClass: x=3
in SubClass: x=5
in SuperClass.doSomething()

in SubClass.doSomething()
super.x=3 sub.x=5
```

3. 多态性

在 Java 语言中，多态性体现在两个方面：由方法重载实现的静态多态性（编译时多态）和方法重写实现的动态多态性（运行时多态）。

1） 编译时多态

在编译阶段，具体调用哪个被重载的方法，编译器会根据参数的不同来静态确定调用相应的方法。

2） 运行时多态

由于子类继承了父类所有的属性（私有的除外），所以子类对象可以作为父类对象使用。程序中凡是使用父类对象的地方，都可以用子类对象来代替。一个对象可以通过引用子类的实例来调用子类的方法。

◇ **重写方法的调用原则**：Java 运行时系统根据调用该方法的实例，来决定调用哪个方法。对子类的一个实例，如果子类重写了父类的方法，则运行时系统调用子类的方法；如果子类继承了父类的方法（未重写），则运行时系统调用父类的方法。

在例 3-6 中，父类对象 a 引用的是子类的实例，所以，Java 运行时调用子类 B 的 callme 方法。

【例 3-6】

```
import Java.io.*;
class A{
      void callme( ) {
        System.out.println("Inside A's callme()method");
      }
}
class B extends A{
      void callme( ) {
        System.out.println("Inside B's callme() Method");
      }
}
public class Dispatch{
      public static void main(String args[]) {
        A a=new B();
        a.callme( );
      }
```

```
}
```

运行结果

```
c:\> Java Dispatch
Inside B's callme() method
```

◇ **方法重写时应遵循的原则：**

1）改写后的方法不能比被重写的方法有更严格的访问权限（可以相同）。

2）改写后的方法不能比重写的方法产生更多的例外。

4. 其它

◇ **final 关键字**

final 关键字可以修饰类、类的成员变量和成员方法，但 final 的作用不同。

1） final 修饰成员变量：

final 修饰变量,则成为常量，例如

```
final type variableName;
```

修饰成员变量时，定义时同时给出初始值，而修饰局部变量时不做要求。

2） final 修饰成员方法：

final 修饰方法，则该方法不能被子类重写

```
final returnType methodName(paramList){
…
}
```

3） final 类：

final 修饰类，则类不能被继承

```
final class finalClassName{
…
}
```

◇ **实例成员和类成员**

用 static 关键字可以声明类变量和类方法，其格式如下：

```
static type classVar;
static returnType classMethod({paramlist}) {
…
}
```

如果在声明时不用 static 关键字修饰，则声明为实例变量和实例方法。

1） 实例变量和类变量

每个对象的实例变量都分配内存，通过该对象来访问这些实例变量，不同的实例变量是不同的。

类变量仅在生成第一个对象时分配内存，所有实例对象共享同一个类变量，每个实例对象对类变量的改变都会影响到其它的实例对象。类变量可通过类名直接访问，无需先生成一个实例对象，也可以通过实例对象访问类变量。

2） 实例方法和类方法

实例方法可以对当前对象的实例变量进行操作，也可以对类变量进行操作，实例方法由实例对象调用。

但类方法不能访问实例变量，只能访问类变量。类方法可以由类名直接调用，也可由实例对象进行调用。类方法中不能使用 this 或 super 关键字。

例 3-7 是关于实例成员和类成员的例子。

【例 3-7】

```
class Member {
    static int classVar;
    int instanceVar;
    static void setClassVar(int i) {
      classVar=i;
      // instanceVar=i; // 类方法不能访问实例变量
    }
    static int getClassVar()
      { return classVar; }
    void setInstanceVar(int i)
      { classVar=i; //实例方法不但可以访问类变量，也可以实例变量
      instanceVar=i; }
      int getInstanceVar( )
      { return instanceVar; }
    }
    public class MemberTest{
      public static void main(String args[]) {
             Member m1=new member();
             Member m2=new member();
             m1.setClassVar(1);
             m2.setClassVar(2);
             System.out.println("m1.classVar="+m1.getClassVar()+"
                                 m2.ClassVar="+m2.getClassVar());
             m1.setInstanceVar(11);
             m2.setInstanceVar(22);
             System.out.println("m1.InstanceVar="+m1.getInstanceVar
                        ()+" m2.InstanceVar="+m2.getInstanceVar());
      }
    }
```

运行结果

```
c:\> Java MemberTest
m1.classVar=2 m2.classVar=2
m1.InstanceVar=11 m2.InstanceVar=22
```

◇ 类 Java.lang.Object

类 java.lang.Object 处于 Java 开发环境的类层次的根部，其它所有的类都是直接或间接地继承了此类。该类定义了一些最基本的状态和行为。下面，我们介绍一些常用的方法。

equals() ：比较两个对象(引用)是否相同。
getClass()：返回对象运行时所对应的类的表示，从而可得到相应的信息。
toString()：用来返回对象的字符串表示。
finalize()：用于在垃圾收集前清除对象。
notify(),notifyAll(),wait()：用于多线以介绍面向对象编程的基本概念、基本理论为重点，结合 Java 语言的语法规则、编程特点和设计思想、强调容易发生错误和编程应注意的地方，使学生能对 Java 技术有一个总体了解，通过本课程学习，使学生掌握 Java 语言的基础知识，理解和掌握面向对象程序设计的基本思想，熟练地使用 Java 语言进行程序的编写、编译以及调试工作程处理中的同步。

3．2．4 抽象类和接口

1． 抽象类

Java 语言中，用 abstract 关键字来修饰一个类时，这个类叫做抽象类，用 abstract 关键字来修饰一个方法时,这个方法叫做抽象方法。格式如下：

```
abstract class abstractClass{ …} //抽象类
abstract returnType abstractMethod([paramlist]) //抽象方法
```

抽象类必须被继承，抽象方法必须被重写。抽象方法只需声明，无需实现；抽象类不能被实例化，抽象类不一定要包含抽象方法。若类中包含了抽象方法，则该类必须被定义为抽象类。

2． 接口

接口是抽象类的一种，只包含常量和方法的定义，而没有变量和方法的实现，且其方法都是抽象方法。它的用处体现在下面几个方面：

◇ 通过接口实现不相关类的相同行为,而无需考虑这些类之间的关系。
◇ 通过接口指明多个类需要实现的方法。
◇ 通过接口了解对象的交互界面,而无需了解对象所对应的类。

1）接口的定义

接口的定义包括接口声明和接口体。
接口声明的格式如下：

```
[public] interface interfaceName[extends listOfSuperInterface] { … }
```

extends 子句与类声明的 extends 子句基本相同，不同的是一个接口可有多个父接口，用逗号隔开，而一个类只能有一个父类。

接口体包括常量定义和方法定义
常量定义格式为：type NAME=value; 该常量被实现该接口的多个类共享; 具有 public ,final, static 的属性。

方法体定义格式为：(具有 public 和 abstract 属性)

```
returnType methodName([paramlist]);
```

2）接口的实现

在类的声明中用 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。一个类可以实现多个接口,在 implements 子句中用逗号分开。

3） 接口类型的使用

接口作为一种引用类型来使用。任何实现该接口的类的实例都可以存储在该接口类型的变量中，通过这些变量可以访问类所实现的接口中的方法。

3．2．5 内部类

1． 内部类的定义和使用：

内部类是在一个类的内部嵌套定义的类，它可以是其它类的成员，也可以在一个语句块的内部定义，还可以在表达式内部匿名定义。

内部类有如下特性：

◇ 一般用在定义它的类或语句块之内,在外部引用它时必须给出完整的名称.名字不能与包含它的类名相同。
◇ 可以使用包含它的类的静态和实例成员变量,也可以使用它所在方法的局部变量。
◇ 可以定义为 abstract。
◇ 可以声明为 private 或 protected。
◇ 若被声明为 static,就变成了顶层类,不能再使用局部变量。
◇ 若想在 Inner Class 中声明任何 static 成员,则该 Inner Class 必须声明为 static。

例 3-8 是一个说明内部类如何使用的例子，其中，定义了两个内部类：MouseMotionHandler 和 MouseEventHandler，分别用来处理鼠标移动事件和鼠标点按事件。

【例 3-8】

```
import Java.awt.*;
import Java.awt.event.*;
public class TwoListenInner {
    private Frame f;
    private TextField tf;
        public static void main(String args[]) {
          TwoListenInner that=new TwoListenInner();
          that.go();
    }

        public void go() {
          f=new Frame("Two listeners example");
          f.add("North",new Label("Click and drag the mouse"));
          tf=new TextField(30);
          f.add("South",tf);
          f.addMouseMotionListener(new MouseMotionHandler());
          f.addMouseListener(new MouseEventHandler());
          f.setSize(300,300);
          f.setVisible(true);
    }
    public class MouseMotionHandler extends MouseMotionAdapter {
        public void mouseDragged(MouseEvent e){
          String s="Mouse dragging:X="+e.getX()+"Y="+e.getY();
          tf.setText(s);
        }
    }
```

```
    public class MouseEventHandler extends MouseAdapter {
        public void mouseEntered(MouseEvent e){
          String s="The mouse entered";
          tf.setText(s);
        }
        public void mouseExited(MouseEvent e){
          String s="The mouse left the building";
          tf.setText(s);
        }
    }
}
```

同学们可以运行一下这个程序，看一看它的运行结果。当你将鼠标移入 frame 时，文本框中会出现："The mouse entered"；当你在 frame 中拖曳鼠标时，文本框中会出现："Mouse dragging:X=64 Y=117"；当鼠标离开文本框时，文本框中出现："The mouse left the building"。

2. 匿名类的定义和使用：

匿名类是一种特殊的内部类，它是在一个表达式内部包含一个完整的类定义。通过对例 6-7 中 go()部分语句的修改，我们可以看到匿名类的使用情况。

```
public void go() {
            f=new Frame("Two listeners example");
            f.add("North",new Label("Click and drag the mouse"));
            tf=new TextField(30);
            f.add("South",tf);
            f.addMouseMotionListener(new MouseMotionHandler(){
            /*定义了一个匿名类，类名没有显式地给出，只是该类是
              MouseMotionHandler 类的子类*/
                public void mouseDragged(MouseEvent e){
                    String s="Mouse dragging:X="+e.getX()+"Y
                    ="+e.getY();
                    tf.setText(s);
                }
            });
            f.addMouseListener(new MouseEventHandler());

            f.setSize(300,300);
            f.setVisible(true);
            }
```

3. 内部类的优缺点：

◇ 优点:节省编译后产生的字节码文件的大小
◇ 缺点:使程序结构不清楚

【本讲小结】

类是 Java 语言面向对象编程的基本元素，它定义了一个对象的结构和功能。 Java 类中包含成员变量和成员方法。成员变量有两种，用 static 关键字修饰的变量为类变量，无 static 修饰的变量为实例变量。相应地，成员方法也有两种，用 static 修饰的为类方法，无 static 修饰的为实例方法。实例方法不仅可以对当前对象的实例变量进行操作，也可以对类变量进行操作；但类方法只能访问类变量。实例变量和实例方法必须由实例对象来调用，而类变量和类方法不仅可由实例对象来调用，还可由类名直接调用。Java 通过在类定义的大括号里声明变量来把数据封装在一个类里，这里的变量称为成员变量。为了解决类名可能相同的问题，Java 中提供包来管理类名空间。

封装性、继承性和多态性是 java 语言中面向对象的三个特性。接口是 java 语言中特有的数据类型，由于接口的存在，解决了 Java 语言不支持多重继承的问题。内部类是指在一个类的内部嵌套定义的类。