|  |
| --- |
|  |
|  |
| **第一讲 Java语言入门**  **1.1 Java的特点**  **1.1.1面向对象：**   * 与C++相比，JAVA是纯的面向对象的语言       C++为了向下兼容C，保留了很多C里面的特性，而C，众所周知是面向过程的语言，这就使C++成为一个"混血儿"。而JAVA语法中取消了C＋＋里为兼容C所保留的特性，如取消了头文件、指针算法、结构、单元等。  **1.1.2可移植（平台无关性）：**   * 生成中间字节码指令       与其他编程语言不同，Java并不生成可执行文件（.exe文件），而是生成一种中间字节码文件（.class文件）。任何操作系统，只要装有Java虚拟机（JVM），就可以解释并执行这个中间字节码文件。这正是Java实现可移植的机制。   * 原始数据类型存储方法固定，避开移植时的问题       Java的原始数据类型的大小是固定的。比如，在任何机器上，整型都是32位，而C++里整型是依赖于目标机器的，对16位处理器（比如8086），整数用两个字节表示；在像Sun SPARC这样的32位处理器中，整数用4个字节表示。在Intel Pentium处理器上，整数类型由具体的操作系统决定：对于DOS和Win32来说，整数是2个字节；对于Windows 9x 、NT和2000，整数是4个字节。当然，使整数类型平台无关之后，性能必然有所下降，但就Java来说，这个代价是值得的。Java的字符串，则采用标准的Unicode格式保存。可以说，没有这个特性，Java的可移植性也不可能实现。  **1.1.3简单**   * JAVA在语法上与C++类似       JAVA的语法与C++很接近，有过C或者C＋＋编程经验的程序员很容易就可以学会JAVA语法；   * 取消了C++的一些复杂而低效的特性比如：用接口技术代替了C++的多重继承。C＋＋中，一个类允许有多个超类，这个特性叫做"多重继承"，多重继承使得编译器非常复杂且效率不高；JAVA的类只允许有一个超类，而用接口（Interface）技术实现与C＋＋的多继承相类似的功能       其它被取消的特性包括：虚拟基础类、运算符过载等   * JAVA的基本解释器和类支持模块大概仅40K       即使加入基本的标准库和支持线程的模块，也才220K左右。与GUI（图形用户界面）相比，明显小很多  **1.1.4健壮**   * 取消了指针算法       C的最关键的一个特性是它的强大指针算法，比如可以用指针访问字符串、数组甚至文件，很容易产生指针溢出，意外地改写内存和损毁数据。JAVA里有一个指针模型，对于普通的任务，如访问字符串、数组，根本不用指针，而一旦需要，比如访问对象、文件时，也可以获得指针的强大能力，但JAVA同时有高度的指针安全保障，用户永远不可能访问一个坏掉的指针、不可能造成内存分配错误，也用不着专门提防可能出现的内存漏洞。  **1.1.5多线程**   * 强大的多线程能力       多线程可以使程序有更好的交互性和实时性，并可以最大限度地利用多处理器系统。JAVA有着强大的多线程处理能力，因为各主流操作系统平台在线程的实施方面有重大的差异，所以JAVA为不同的操作系统实现了专门的多线程机制。在这点上来说，JAVA没有做到平台无关。 |
| aban |
|  |
| **1.2 Java简史**   * 首次亮相--1995年5月       Java技术起先是作为小型的计算机语言，用于消费类设备，比如有线电视交换盒等。因为这些设备的功能和内存均不足以运行大程序，所以程序要尽可能小；另外，由于不同的厂商可能选用不同的CPU，所以要求这种语言不能受限于任何一种独立的体系结构。1995年5月23日召开的SunWorld '95大会上，一种全新的浏览器（今天的HotJava的前身）亮相，标志着Java的诞生。   * 开始流行--1996年1月       那时网景公司决定在Netscape2.0加入对Java的支持，Netscape2.0在1996年1月发布，从这一版本开始，所有Netscape浏览器均支持Java。注册了Java使用许可证的还有IBM、Symantec、Inprise和其他许多公司。就连微软，也加入了这一行列（尽管他们支持的Java和标准的Java稍有不同）。   * SUN的第一个JAVA版本--1996年初       Sun公司于1996年初发布了Java 1.02，遗憾的是，Java1.02还未摆脱其小型语言的影子，只适合用来做诸如网页上一个随机移动的文字之类的工作，并不适合用来做正规的程序开发。Java1.02作为一种正规的编程语言，可以说，准备得很不充分。   * 成熟--1998年       1998年12月，Java1.2问世了。它是一个功能全面的、具有高度扩展能力的新版本。3天后，Java1.2被改进成Java2，向"一次编写，到处运行"的目标前进了一大步。   * 现状       现在最新的Java版本是JDK1.4，它是一个测试版，稳定的正式版是JDK1.3。用户可以到http://java.sun.com/products/免费下载。微软在它的最新浏览器IE6.0中不再支持Java，为Java的发展带来阴影。 |
| ban |
|  |
| **1.3 JDK的安装与编程环境的设定**  **1.3.1 Java程序的开发过程**  01-3      经典的Java工具包是JDK。尽管可以用任何文本编辑器，如记事本，都可以直接编写Java程序然后用JDK的命令编译和运行，为了调试方便，也为了初学者容易上手，本教程课件将使用一个叫JPadPro的Java编写工具来调用JDK的工具。用户可以到http://www.modelworks.com下载这个工具。  **1.3.2 JDK的安装**  **1.3.3安装运行之后，应该首先设置JDK工具包所在的路径。**      菜单栏的JDK-〉select JDK Directory-〉选择刚才安装的JDK路径-〉OK。  **1.3.4 新建一个project**      菜单栏的File->new projects->选择路径, 输入project的名字。 |
| ban |
|  |
| **1.4两个简单的程序示例**  **1.4.1 Java 应用程序：Hello.java**   * 编写       新建一个.java文件。File-〉new-〉java file-〉输入一个名字：Hello-〉点creat file.出现了一个空的java文件。   * 编译       JDK-〉compile Hello.java 或者点击如演示所示的按钮。实际上是执行JDK工具包里的javac.exe命令，与javac G:\java2jiaocheng\example\teach1\Hello.java是等价的。便已完成之后在同一个文件夹就会有一个同名的.class文件。   * 看编译结果，改错       错误举例：少写一个分号；录入错误；   * 运行       JDK-〉Run classes-〉选择要运行的class，输入参数（这里，没有参数）或者点击如演示所示的按钮。      一个有参数的例子   * 看运行结果   **1.4.2 Java Applet: StarterApplet.java**      新建一个Java Applet文件。File-〉new-〉java Applet-〉输入一个名字：StarterApplet-〉出现了一个空的java文件和一个Html文件。   * 编译   JDK-〉compile StarterApplet.java 或者点击如演示所示的按钮。实际上是执行JDK工具包里的javac.exe命令，与javac G:\java2jiaocheng\example\teach1\ StarterApplet.java是等价的。  注意 不要隐藏文件的扩展名，否则给编译时带来麻烦！可以在工  工具/文件夹先项/查看/中修改！！   * 看编译结果，改错 * 运行       JDK-〉Test Applet-〉选择要运行的class，输入参数（这里，没有参数）      或者点击如演示所示的按钮      或者直接双击Html文件。   * 看运行结果 |
| ban |
|  |
| **1.5 如果获得帮助**  **1.5.1 下载帮助**      JDK并不包含帮助文件，需要到网站上下载，里面有示例和说明。      http://www.confluent.fr/javadoc/jdk13e.html，下载下来时候，双击安装。  **1.5.2 使用帮助**      敲入想查找的关键字，按回车。 |
| ban |
|  |
| **1.6 编程的一些约定**  注意：在一个原文件中只能有一个public 类！且文件名与public类同名！  **1.6.1命名规则**      类名与文件名应该一致，否则编译出错。比如，class Hello的文件名应该是Hello.java      类名首字母大写，每个单词开头大写，其他用小写。比如：Hello.java, StarterApplet.java。      变量名用第一个单词用小写字母，第二个以后的单词首字母大写。比如：int myAge;      方法的名字第一个单词用小写字母，第二个以后的单词首字母大写，比如：getMyAge()；  **1.6.2 注释**  /\* \* Here is a block comment. \*/ 表示完整的一段注释；  if (condition) {  /\* Handle the condition. \*/ ... } 表示注释该行以下部分  if (a == 2) { return TRUE; /\* special case \*/ } else { return isPrime(a); // works only for odd a } 注释一行  **1.6.3 分行**      如果一行太长需要换行。比如：  someMethod(longExpression1, longExpression2, longExpression3,  longExpression4, longExpression5); var = someMethod1(longExpression1, someMethod2(longExpression2, longExpression3));   var = someMethod1(longExpression1, someMethod201-6 (longExpression2, longExpression3));  **讲演计划** |
| ban |
|  |
| **第三讲 语句**      Java的语句可以分为以下4种：   * 表达式语句 * 方法调用语句 * 控制语句 |
| ban |
|  |
| **3.1表达式语句**      一个表达式加上一个分号就构成了一个语句。分号表示一个语句的结束，缺少分号，编译将出错。最典型的表达式语句是赋值语句。  比如：int x; x=23; |
| ban |
|  |
| **3.2方法调用语句**      调用一个类的对象的方法：类名（或对象名）.方法名(参数列表)。  比如： System.out.println("Hello").  如果方法有返回值，还可以把返回值赋值给一个变量。比如： String s="Hello", int len; len=s.length(); |
| ban |
|  |
| **3.3控制语句**      Java语言的控制语句有2种：条件语句、循环语句。  **条件语句：**      条件语句有两种：if语句和switch语句。   * if语句：   if (条件)  {代码块1} else  {代码块2}      如果条件为真，则执行代码块1，否则执行代码块2。      else部分是可选的，也就是说，可以没有else。      如果有else，则与最近的if结合：   * switch语句       switch语句是多分支的开关语句，它的一般格式定义如下：  switch(表达式) { case 常量值1： {代码块1} break;  case 常量值2： {代码块2} break;  …… default: {代码块} }      语句中表达式的值必须是整型或者字符型；常量值1到常量值n必须也是整型或者字符型。switch语句首先计算表达式的值，如果表达式的值和某个case后面的常量值相同，就执行该case里的若干个语句直到break语句为止。如果没有一个常量与表达式的值相同，则执行default后面的若干个语句。default是可有可无的，如果它不存在，并且所有的常量值都和表达式的值不相同，那么switch语句就不会进行任何处理。      需要注意的是，在switch同一个语句中，case后的常量值必须互不相同。 |
| ban |
|  |
| **3.3控制语句**      Java语言的控制语句有2种：条件语句、循环语句。  **条件语句：**      条件语句有两种：if语句和switch语句。   * if语句：   if (条件)  {代码块1} else  {代码块2}      如果条件为真，则执行代码块1，否则执行代码块2。      else部分是可选的，也就是说，可以没有else。      如果有else，则与最近的if结合：   * switch语句       switch语句是多分支的开关语句，它的一般格式定义如下：  switch(表达式) { case 常量值1： {代码块1} break;  case 常量值2： {代码块2} break;  …… default: {代码块} }      语句中表达式的值必须是整型或者字符型；常量值1到常量值n必须也是整型或者字符型。switch语句首先计算表达式的值，如果表达式的值和某个case后面的常量值相同，就执行该case里的若干个语句直到break语句为止。如果没有一个常量与表达式的值相同，则执行default后面的若干个语句。default是可有可无的，如果它不存在，并且所有的常量值都和表达式的值不相同，那么switch语句就不会进行任何处理。      需要注意的是，在switch同一个语句中，case后的常量值必须互不相同。 |
| ban |
|  |
| **循环语句:**      循环语句，顾名思义，是反复执行的语句。比如，计算100的阶乘，1\*2\*3\*...\*100，就需要用到循环语句，不然，就要写一百遍乘法。循环语句需要特别小心，很容易陷入死循环，所以循环体的代码块里需要有能使循环结束的语句。Java有三种循环语句：while语句，do-while语句和for语句。   * while语句：       while语句的格式是：  while(条件) {代码块}      当条件成立的时候，执行代码块，再检查条件，如果还成立，再执行代码块，……直到条件不成立。      比如，计算10的阶乘：   * do-while语句       do-while语句的格式是：  do{ 代码块 }while(条件)      do-while语句和while语句的区别在于：while语句先检查条件，如果条件不成立，则不进入循环体；do-while语句先执行循环体的代码，再检查条件，如果条件成立，则在此执行循环体的代码。所以，do-while语句至少要执行一遍循环体的代码块。      比如，计算10的阶乘：   * for语句       for语句是Java语言中用得最多的循环语句。它的格式如下：  for(表达式1，表达式2，表达式3) {代码块}      其中，表达式1完成变量的初始化，表达式2时布尔类型的表达式，是循环条件，表达式3是党执行了一遍循环之后，修改控制循环的变量值。      for语句的执行过程是这样的：首先计算表达式1，完成必要的初始化工作；然后判断表达式2的值，如果表达式的值为true，则执行循环体；如果为false，则跳出循环。执行完循环体之后紧接着计算表达式3，以便改变循环条件，这样一轮循环就结束了。第二轮循环从计算表达式开始，如果表达式的值仍为true，则继续循环；否则循环结束，执行for语句后面的语句。      比如，计算10的阶乘：   * 嵌套循环       经常可以遇到嵌套循环的例子。所谓嵌套循环，是指一个循环体里还有一个或者更多个循环。比如计算10以内的阶乘之和，或者求50以内的素数，就需要用到嵌套循环。我们以求50以内的素数为例，说明嵌套循环的用法；求10以内的阶乘之和，将留作作业。      循环语句里的break语句和continue语句：      在循环体中，遇到break语句，那么整个循环语句就结束；如果遇到continue语句，那么本次循环就结束，就是说，不再执行本次循环中continue语句后面的语句，而是转入下一次循环。 |
| ban |
|  |
| **3.5 变量的作用域**      变量是由作用域的。所谓变量的作用域，是指一个变量在什么地方生效。总的来说，变量的作用域是声明它的块。所谓块，是指用一对花括号包围起来的任意数量的Java语句。变量只在声明它的块中有效，在这块以外，变量是无效的。      然而，不可以在两个嵌套的块中声明两个完全同名的变量。      注意，在C++中，这是允许的，内部定义会掩盖外部定义。这很容易造成错误，所以Java不允许这样做。 |
| ban |
|  |
| **小结**      这一讲，我们主要学习了Java的语句，主要是流程控制语句，包括条件语句和循环语句等。语句是程序的基本组成部分，是我们进一步学习Java的基础。希望同学们切实掌握好这部分内容，为我们的进一步学习打下坚实的基础。 |
| ban |
|  |
| **习题**   1. 编一个计算1!+2!+…+10!的应用程序   For(int i=1;i<10;i++ )  {for(intj=1;j<=i;j++){int sum=0;  Sum=sum+i\*(i-1)  }}  编写一个应用程序计算100以内的全部质数   1. 一个数如果恰好等于它的因子之和，这个数就称为“完数”，编写一个应用程序，求1000之内的所有完数。 2. 求45和的最大公约数和最小公倍数。 |
| ban |
|  |
| **第四讲 数组和字符串**  **4.1什么是数组？为什么要数组？**      除了基本数据类型，Java还提供一种导出类型：数组。数组是相同类型的数据按顺序组成的一种复合数据类型，通过数组名和下标，可以使用数组中的数据。下标从0开始。数组是所有编程语言中常用的数据结构。      为什么要数组呢？我们来举一个例子。假设我们需要表示一个班50个人的数学成绩，要求求出平均成绩。如果没有数组，我们需要用前面学过的声明变量的方法，声明50个变量，写50次加法运算！数组可以大大地简化类似的问题！我们只要声明一个长度为50的整型数组，结合上一讲我们学过的循环语句，就可以很方便地解决这个问题！      在以前的编程语言比如C或者C++中，字符串也使用数组来表示的：字符串是字符数组！所以字符串与数组有着天然的联系。但是在Java中，提供了一种更方便的表示字符串的方法：用一个String类来表示。类是面向对象的语言特有的概念，对于初次接触面向对象技术的人来说，比较难理解。所以，这一讲，我们将学习表示字符串的String类，也对类的使用有一个粗略的了解，作为下一讲详细学习类的准备。      需要指出的是，C语言里的其他两种导出类型：结构体和共用体，在Java里已经被取消。      下面，我们来学习数组和字符串。 |
| ban |
|  |
| **4.2数组的声明、创建和初始化**   1. 数组的声明：       声明数组，包括声明数组的名字、数组包含的元素的数据类型。数组可以一维的，也可以是二维或者多维的。举例来说：一个班有50个人，我们用一个长度为50的一维数组表示；如果要表示每个同学的五门高考成绩，那我们就需要用一个第一维长度为50，第二维长度为5的二维数组。      声明一维数组有两种格式：   * 数组元素类型 数组名[ ]； * 数组元素类型[ ] 数组名；       比如：int student[ ]; 或者： int[ ] student;      类似地，声明二维数组有两种格式：   * 数组元素类型 数组名[ ][ ]； * 数组元素类型[ ][ ] 数组名；       比如：int score[ ][ ]; 或者： int[ ][ ] score;      下面，我们主要以一维数组为例，学习数组的用法。   1. 数组的创建：       声明数组仅仅给出了数组名字和元素的数据类型，想要真正使用数组还必须为数组分配内存空间，也就是创建数组。在为数组分配内存空间时必须指明数组的长度。为数组分配内存空间的格式如下：   * 数组名 = new 数组元素的类型 [ 数组的长度 ]   例如：student = new int [50]； score = new int [50] [4]；      事实上，数组的声明和创建可以一起完成，比如：  int student [ ] = new int [50];      一旦数组被创建，数组的大小就不能改变。如果在程序运行期间，需要对数组的大小进行扩展，通常需要使用另一种数据对象：Vector。有关向量和扩展数组，有兴趣的同学可以在Java帮助中查找Vector的帮助。      student数组创建之后，其内存模式如图：  04-2   1. 数组的初始化：       创建数组后，系统会给每个元素一个默认的值，比如，整型数组的默认值是0。      我们也可以在声明数组的同时给数组一个初始值，我们称之为初始化。  int num [ ] = {2, 4, 4, 1}。      这个初始化动作相当于：  int num [ ] = new int [ 4]; num [0]=2; num [1]=5; num [2]=4; num [3]=1; |
| ban |
|  |
| **4.3数组的使用**   1. 数组的访问       前面我们已经看到，数组通过下标访问数组元素，比如：student [ 0 ]=70，student [49]=87等。注意，下标从0开始，如果数组长度为n，则下标是0 ~ n -1，如果使用n或者以上的元素，将会发生数组溢出异常。比如：student[ 50 ] = 79，虽然编译的时候能通过，程序运行时将中止。      二维数组的使用与之类似。   1. 数组的复制       可以把一个数组变量复制给另一个，但两个变量引用的都会是同一个内存空间，所以改变一个数组的值，另一个数组变量的值也会改变。  比如： int num [ ] = {2, 3, 5, 1}； int numcopy[ ]=num; numcopy[2 ]=5; 现在，num[2]也变成了5。  04-3      如果真的想把一个数组的所有值都复制到另一个数组中，就需要采用System . arrarycopy()方法：System . arrarycopy ( num, 0, numcopy, 0, 4)。这样，num和numcopy将指向不同的内存空间，numcopy的值的改变，不会在影响num。有兴趣的同学可以参考帮助文件。 |
|  |
|  |
| **4.4字符串**      前面已经说过：Java使用java.lang包中的String类来表示字符串，尽管我们可以用字符数组来实现类似的功能。关于类的详细知识，我们下一讲将会讲到。由于类是面向对象编程语言最核心的概念，学习起来有些难度，我们在这里先通过String类对类的概念有一个粗略的了解，以使我们下面的学习更顺利一些。      字符串，顾名思义，就是一串字符。比如："student100", "China"等。用一对双引号表示字符串。我们也可以把字符串看成字符数组（事实上，C就是把字符串当作字符数组来处理），但是把它视为类更方便一些。为什么？我们下面马上会看到。   * 字符串的声明、创建       字符串声明的格式是： String 字符串名      比如：String s;      字符串的创建格式：字符串名 = new String（字符串常量）或者 字符串名=字符串常量      比如：s = new String ( "student" ); 或者：s = "student"；      声明和创建可以一步完成，比如：String s = new String ( "student" ); 或者 String s = "student";      大家是不是觉得：String这个类的声明，跟前面我们学过的基本数据类型的声明的格式是一样的，比如：整型的声明：int n; 比较一下： String s;      事实上，      类型 变量名      这种格式是类的声明的一般格式，你可以把类当作基本数据类型一样来声明。      另一方面，      变量名= new 类名（参数列表）；比如 s = new String ( "student" );      这是类的一般创建格式。 |
| ban |
|  |
| **4.5与字符串有关的操作**   * 字符串的长度   length()方法： String s = "student"; int len=s.length();      需要指出的是，s.length()这种调用方法，是面向对象编程语言特有的方法，我们把s叫做String类的对象，就像int n，我们把n叫做整型变量一样；把length()叫做String类的方法。下面我们可以看到，String类的方法，都是通过 对象名.方法名（）这种方式调用的。   * 取子串   String s = "I am a Chinese"; String subs; subs = s.substring (7);   * 字符串的比较   String tom = "my name is tom"; String jane = "my name is jane"; tom.equals(jane)；返回false。表示不相等 tom.compareTo(jane)，返回一个负整数，因为第一个不相等的字符t和j相比，t在j的后面；如果返回0，表示相等；如果返回一个正整数，表示tom和jane第一个不相等的字符，tom的在jane的前面。      注意，不要用tom= =jane判断两个字符串是否相等，因为这样的话比较的是它们在内存中的地址是否相同，而不是我们所希望的比较字符串的内容是否相同。   * 字符串连接   String s = " I am"; String s2 = "a Chinese"; String s4, s4; s4 = s+s2; s4=s + 24; 整数型24将会自动转换为字符串。   * 字符串检索       字符串检索是指判断一个字符串是否包含某一个字符或者子字符串，如果有，返回它的位置，如果没有，返回一个负数。  String s = "I am a Chinese"; s.indexOf("Chinese" )， 返回7； s.indexOf('a'); 返回2；   * 字符串转换为数值       如果一个字符串是一串数字，可以把它转换成相应的数值。   * 转换为整型：   String s = "21"; int x; x= Integer.parseInt (s);   * 转换为浮点型   String s = "22.124"; float f; f = Float.valueOf(s).floatValue()；   * 当然，也可以把整数或者浮点数转换为字符串类型：   String s; int n = 24; s = String.valueOf ( n )；   * 其他       与字符串有关的方法还有很多，我们不可能一一讲解。前面我们说过，我们要学会使用帮助。我们查找关于String的帮助，就可以看到，有关于String的所有方法的详细讲解。希望同学们可以学会使用帮助。 |
| ban |
|  |
| **小结**      这一讲，我们学习了数组的一些知识，并通过String类，一方面学习了字符串的使用，另一方面先对类的一些概念和用法，比如类的方法调用，有了一些粗略的了解。      数组是我们经常要用到的数据结构，它是一种导出型的数据结构，对我们处理大规模的数据，有很大帮助。      前面我们说，用String类表示字符串，比我们自己用字符数组来做更方便。因为String类为我们提供了很多现成的可以直接使用的对字符串的操作方法。比如substring( )，equals( )等方法。如果我们要用字符数组来完成与substring类似的功能，就只能自己把子串的每一个字符复制到另外一个字符数组中；与equals()类似的功能，就只能自己一个一个字符地检查两个字符船是否相等。提供对自己的成员数据的完整的操作方法，也就是把数据和对数据的操作封装起来，是类的最大的好处。如果现在还不能理解这句话，不要紧，我们现在只需要知道：使用String类这种方法，非常方便。 |
| ban |
|  |
| **习题**   1. 声明一个数组，存一个学生的五门成绩。求该学生总成绩、平均成绩。 2. 将一个数组中的值按逆序重新存放，例如，原来顺序为：9，5，7，4，8，要求改为：8，4，7，5，9。 3. 编写一个应用程序，连接两个字符串：“I am”和“a student”，并求出第7个字符是什么？找出“am”在该字符串中的位置，最后把所有字符都变成大写字符。 4. 统计一个字符串中字母、空格和数字的个数。 |
| ban |
|  |
| **第五讲：类、对象和接口（1）**  **5.1什么是类**      上一讲我们使用了一个String类，我们说过：类把一类对象的成员函数和方法封装起来，使用起来很方便。那么，什么是类呢？要理解类，我们先来了解"对象"的概念，因为类就是用来创建对象的模板，它包含了被创建的对象的状态描述和方法定义。      我们可以把长虹牌电视机看成是一个对象。我们不必关心电视机里面的集成电路是怎样的，也不用关心电视机的显像管的工作原理，我们只需要知道电视机的遥控器上提供了对这台电视机的什么操作，比如选台、调节色彩、声音等。这样，我们虽然不知道电视机内部是怎么工作的，但我们可以使用这台电视机。听起来这跟编程没什么关系，但面向对象的思想正是与它类似：对象，对外只提供操作该对象的方法，内部的实现对外是隐藏的。我们把这种技术叫做数据隐藏。这样，程序员不必了解一个对象内部是怎么实现的，但可以很方便地使用这个对象。      类，就是对象的模板。比如，电视机是一个类，而某一台电视机就是这个类的一个对象，也叫一个实例。      我们前面使用的String类，虽然我们不知道，也不必知道里面的实现细节，但我们只要知道他的操作方法，就可以很方便地使用它。这样，对别的程序员编写的程序，我们可以很方便地拿来使用，从而提高软件的复用程度和编程的效率。      比如，我们构造一个类：TV，然后对外提供一些对它的操作方法，选台 selectChannel( )等。然后用这个类创建一个对象：TV kongKai；然后就可以这样操作这台电视： kongKai.selectChannel。      类的另一个特性，也是它的一个大好处，是继承。继承的概念类似于这样一种表达：电视是一种电器。如果我们有一个电器类，那么，我们说，电视机类继承了电器类。继承有什么好处呢？设想电器类有一些属性，比如工作电压，功率，有些一些方法，比如：打开 open( )， 关闭：close( )。这时候，如果电视类继承了电器类，电视类就自动拥有了电器类的这些属性和方法，而不用再自己重写一遍（当然，如果想重写，也可以，我们把这种技术叫做重载）。这样做有什么好处呢？最明显的好处就是减轻编程的工作量。假设我们有令一个类：VCD类，它也继承了电器类，对于打开关闭等，同样不用重新编写。      好了，既然面向对象的方法有这么多好处，我们下面就来学习类和对象的一些基本知识。 |
| ban |
|  |
| **5.2 一个例子**      下面是一个完整的类的例子，我们用这个例子详细讲解类的一些用法。  class Employee {  private String name; private double salary; private int hireYear; private static String company=new String("IBM");   public Employee(String n, double s, int d) { name = n; salary = s; hireYear = d; }  public void print() { System.out.println(name + " " + salary + " " + getHireYear() + " "+ getCompany()); return; }  public void raiseSalary(double byPercent) { salary \*= 1 + byPercent / 100; }  public void raiseSalary(int byAbsolute) { salary +=byAbsolute; }  public int getHireYear() { return hireYear; }  public static String getCompany() { return company; } }      为了测试这个类，我们写了一个测试程序：  package teach4; public class EmployeeTest { public static void main(String[] args) { Employee emp = new Employee ("Tony Tester", 38000, 1990); emp.print(); emp.raiseSalary(5.0D); emp.print(); int raise=1000; emp.raiseSalary(raise); emp.print();  } } |
| ban |
|  |
| **5.3 类的结构**      class Employee { } 表示一个类的声明，其中，class是关键字，Employee是类名。在一对大括号中的内容，我们叫做类体，是这个类的实现部分。      前面我们说过，类把数据和在数据上的操作方法封装起来，所以，类体中有两部分，一部分是数据，另一部分是方法。      数据部分是：  private String name; private double salary; private int hireDay;      分别是String类型的name，代表该员工的姓名，double类型的salary，代表该员工的薪水；int类型的hireDay，代表该员工的雇佣年份。      private是一个关键字，是私有的意思，表明这些数据，只能被类里面的方法访问，外部程序是不能直接访问这些数据的。这正是类的一个好处：对外隐藏数据。我们建议，始终保持数据的私用，也就是始终用private来修饰数据，来是我们的程序更安全。      这个类的方法部分是：  public Employee(String n, double s, Day d) public void print() public void raiseSalary(double byPercent) public void raiseSalary(int byAbsolute) public int getHireYear() public static String getCompany()      所谓方法，就是对数据的操作，有过C编程时间的同学，可以把它理解为函数，只是这些函数是放在类里并对类的数据进行操作的。比如这个方法：      public void raiseSalary(double byPercent)就是对数据salary的操作。      public是一个关键字，是公有的意思，表明该方法可以用 对象名.方法名 的方式调用。比如我们的测试程序中: emp.raiseSalary(5.0D)，就是对这个方法的调用。否则，如果是private，该方法就只能被类里的方法调用，像emp.raiseSalary(5.0D)这种调用是非法的。      void表明方法的返回类型为空，也就是什么都不返回。public int getHireYear()这个方法的返回类型是int。一般来说，返回类型这部分是不能缺少的，即使方法什么都不返回，也要用一个void关键字修饰。有一个例外：构造函数。我们后面会讲到。      raiseSalary(double byPercent)是方法名，前面我们已经说过方法的命名规则，第一个单词小写，以后没个单词首字母大写。这样可以增加可读性，根据raiseSalary这个名字就可以知道这个方法的功能是涨工资。括号里是参数列表，类似于我们前面学过的变量的声明。如果有多个参数，用逗号隔开，比如 Employee(String n, double s, int d)。      方法的一对大括号里是方法的实现，就是我们前面学过的语句的组合。比如print( )方法，是打印出该员工的姓名工资等信息；raiseSalary( )方法是根据参数，把员工工资涨相应的比例。注意，如果方法有返回类型且不为void，那么在方法的实现中必须有return语句，返回相应的类型。比如hireYear( )方法，就有return子句。相反，如果返回类型是void，那么不能返回任何数据，但可以是一个return语句，后面不带返回值，比如print( )方法里就有一个空的return语句。 |
| ban |
|  |
| **5.4 构造方法**      我们注意到，有一个方法是与类名同名的，而且没有返回类型，比如这个例子中的public Employee(String n, double s, int d)，我们把它叫做构造方法。构造方法是做什么用的呢？构造方法是在声明对象的时候，自动调用的方法，其作用是为对象的数据做一些必要的初始化工作。比如，这里的public Employee(String n, double s, int d)方法，就是初始化这个员工的姓名，工资和雇佣年份。在声明emp对象的时候，调用的就是构造函数。      Employee emp = new Employee ("Tony ", 10000, 1990);      如果没有定义构造函数，Java会自动提供一个默认的构造函数，把所有成员数据初始化为默认值，比如数字类型（包括整型、浮点型）将是0，布尔类型将是false等。注意在没有构造方法时，new后面的（）中不能有数字！！      需要注意的是，与构造函数相对的是析构函数，目的是在对象不再使用的时候回收对象使用的内存。C++里就支持析构函数。但是，由于Java提供自动回收"垃圾"的机制，所以不需要进行内存回收，所以Java没有析构函数。 |
| ban |
|  |
| **5.5 方法的重载**      方法的名字相同但参数的类型或个数不同，我们叫做方法的重载。      类允许有相同名字的方法，比如我们这个例子中的raiseSalary方法。  public void raiseSalary(double byPercent) public void raiseSalary(int byAbsolute)      第一个的参数是double型，表明工资上涨的百分比；第二个的参数使int型，表明工资上涨的数额。这两个方法参数不同。调用的时候，根据参数的不同，决定使用哪一个方法。比如，我们的例子中，emp.raiseSalary(5.0D)的参数是double型，所以将会调用第一个，工资上涨5%，emp.raiseSalary(raise)的参数raise是一个int型，所以将会调用第二个，是工资上涨1000元。      方法的重载的好处是使我们的程序处理更方便。比如，我们的例子中，涨工资，我们提供了统一的raiseSalary( ) 方法，我们不用自己判断涨的是百分比还是绝对数额，由程序自己判断，是程序更好用，可读性更强。 |
| ban |
|  |
| **5.6 静态方法和静态成员变量**      所谓静态方法和静态成员变量，是指那些用static关键字修饰的方法或者变量，比如我们例子中的private static String company就是一个静态成员变量，而public static String getCompany()和public static void setCompany(String s)都是静态方法。      静态的意思，是指该方法或者变量在整个类中只有一份。我们说，类是用来创建对象的模板，就是说，我们创建对象的时候，每个对象都会有类中所声明的成员变量的一个副本。但是，对于静态成员变量，在内存中只有一个副本，所有的对象将共享这个副本。比如我们的例子中，所有Employee所在的公司就只有一个，所以我们没有必要为每一个Employee的对象都保留一个company的副本，所以我们把它声明为静态的成员变量。比如我们例子中有两个Employee类的对象：emp和emp2，他们的成员变量内存模式如下图所示：  05-6      所以，只要改变了company的值，对所有Employee的对象都是起作用的。      另一方面，静态方法只能访问静态成员变量，比如我们的例子中，setCompany( )只能访问company，如果它访问name之类的变量，编译器将会报错。而且静态方法的调用是 类名.方法名的方式来调用的，也可以用一般的 对象名.方法名 的方式来调用。      实际上，我们先前用的System.out.println( )方法，就是一个静态的方法，所以我们可以直接用 类名.方法名 的方式调用。而一个类里如果有main函数都要声明为静态方法，因为一个程序只能有一个main函数入口。 |
| ban |
|  |
| **小结**      这一讲我们学习了类和对象的一些基本概念和知识，包括类的结构，构造方法 ，方法的重载 ，静态方法和静态成员变量。类和对象，是面向对象编程语言的最重要的特征，也是精髓所在。希望大家努力掌握好这部分知识。下一讲，我们继续学习Java的继承、接口等概念。 |
| ban |
|  |
| **习题**   1. 类的成员变量和局部变量有什么区别？从内存考虑。 2. 编个程序检验一下public方法和private方法有什么区别。 3. 编一个程序检验静态方法和静态成员变量的用法。 4. 在我们的例子中再重载raiseSalary()方法，按工龄涨工资，比如，如果工龄为5年，那么增长1%\*5； |
| ban |
|  |
| **第六讲 类、对象和接口（2）**  **6.1 类的继承**      继承是类的另一个特性。继承的意义在于：我们重复使用或更改现成的类的方法，也可以加入新的数据成员以及新的方法，以满足新环境的需要。这种技术是所有面向对象的编程语言的一个基本特征。      让我们来看一个例子：前面我们定义了一个Employee类，这只是普通员工，现在我们要定义一个经理类。经理也是员工的一种，所以Employee类中的数据和方法他也应该有；但经理又不同于普通员工，经理有秘书，而且涨工资的时候还要加上分红。怎么办？我们要不要从头开始写一个经理类？      有了继承的技术，我们可以在Employee类的基础上，编写我们的Manager类。程序如下：  package teach4; import java.util.Date;  class Manager extends Employee {  private String secretaryName;  public Manager(String n, double s, int d) { super(n, s, d);必须在构造方法的第一个语句。 secretaryName = ""; }  public void raiseSalary(double byPercent) { // add 1/2% bonus for every year of service Date today = new Date(2001,1,1); double bonus = 0.5 \* (today.getYear() - getHireYear()); super.raiseSalary(byPercent + bonus); }  public void setSecretaryName(String n) { secretaryName = n; }  public String getSecretaryName() { return secretaryName; } }      我们以这个例子为例，学习继承的用法。      首先，请注意这个类的头部有些不同：class Manager extends Employee；其中，关键字extends是扩展的意思，表明Manager类是从Employee类继承而来。我们把Employee叫做父类或者超类，把Manager叫做子类或者衍生类。一般来说，子类比父类有更多的功能。      Manager的构造方法中有个语句：super(n, s, d)，super是一个关键字，意思是调用父类的方法，在这里是父类，也就是Employee的构造方法；类似地，super.raiseSalary(byPercent + bonus)表示调用父类Employee的raiseSalary方法。所以，如果要在子类中调用父类的方法，使用super。 Manage的构造方法调用了Employee的构造方法。事实上，一般要求子类用super语句调用父类的构造方法并附上恰当的参数。如果没有用super调用，将默认地调用父类的默认构造方法，这时，如果父类没有没有默认的构造方法，编译器将报错。在子类的构造方法中，如果有super语句，要求super语句在第一行。      子类自动拥有父类的标志为public的成员变量和方法，比如：虽然我们在Manager类中没有定义print( )方法，但是boss.print()是合法的，因为print( )是Employee类的一个方法。如果希望改变父类中的方法，使之适合子类，我们也可以覆盖它。比如，因为经理的提薪方式是：除了上涨百分比，还要加上工龄\*0.5的奖金，与普通员工的提薪方式就有不同。所以，我们要覆盖掉这个方法。实际上，只要重写这个方法就可以了。boss.raiseSalary(1.0)，将自动调用Manager类里定义的raiseSalary()方法，而不是Employeee里的方法。      但是，如果是私有的成员变量或者方法，也就是用private关键字修饰的那些，那么子类是不能访问的。如果希望子类能访问而别的类不能访问，我们可以用protected关键字，比如：protected String name；这样，name对于Manager来说是可以可见和可访问的，而对于不是Employee的子类的其他类，则是不能访问的。      总结一下访问权限关键字：   * public:对全世界来说都是可见的； * private:仅对类来说是可见的； * protected:对所有子类和同一个包（package）来说是可见的； * 不声明：如果这三个关键字都没有，那么默认为对整个包可见。       Manager类里定义的setSecretaryName()和getSecretaryName()方法，都只能为Manager类的对象调用，Employee类的对象是不能调用这两个方法的。      继承可以是几层的。比如，我们可以以Manager为父类，再衍生出Executive类来。我们也可以从Employee类衍生出Programmer类来，它与Manager类没有什么关系。如果希望防止别人从自己编写的一个类中衍生出另一个类，可以加上关键字final，比如，不希望从Manager类中再衍生出别的类，我们可以把Manager类的声明改为：final class Manager extends Employee。这样可以提高程序的安全性，但可扩展性会受到影响。      我们可以画出类的继承结构如下：  06-1 |
| ban |
|  |
| **6.2造型**      也叫强制类型转换。回忆一下，我们在第二讲的Java的基本语法中，说到基本数据类型的强制类型转换，可以把一个浮点数强制转换为整型，比如：double x=3.14; int nx = (int)x;这样，x=3，把小数部分丢掉了。我们同样可以把类强制转换为另一个类。但不是什么时候都可以转换的，只有在继承结构图内部，才能进行强制类型转换。      比如：Manager boss; Employee emp; 下面这个语句：emp=boss，相当于emp=(Employee)boss；因为emp是Employee类型的，而boss是Manager类型的，但是，这时候，emp将不能调用Manager类的方法，比如getSecretaryName()，如果要调用，必须把emp转换回来。比如：Manager manager; manager=(Manager)emp；这样，manager就可以调用getSecretaryName()了。只允许子类转换成父类的调用父类的方法不允许调过来。      但是，如果一个对象确实是Employee类型，现在要把它转换为Manager类型而且试图访问Manager的方法，编译将会通过，但是运行时将会出现异常，程序将中止运行。 |
| ban |
|  |
| **6.3接口**      前面我们介绍Java的特点的时候说过，Java不允许多重继承，即一个类不允许有多个父类，至多只能有一个父类。在Java中，取代多重继承的技术是接口。Java是用接口技术的原因，是因为多重继承要么使编译器非常复杂，要么效率不高。      那么，接口是什么呢？我们可以把接口理解为一个描述框架，里面定义了一些方法，但并不实现这些方法，而是由继承这个接口的类来实现。这样，如果某一个类继承了一个接口，意思是说：这个类实现了接口所定义的所有方法。      比如，Java的标准库里定义了一个名为Comparable的接口。只要某个类是可对比的，就可以继承并实现这个接口。数字型和字符串型都是可比的，所以String, Float, Integer, Double等类都继承并实现了这个接口。如果我们定义我们的例子中的Employee类业是可比的，以进公司的年份来比较，那么我们也可以继承这个类，并实现它。      注意，如果继承了一个接口，就必须实现这个接口所声明的所有方法。 |
| ban |
|  |
| **6.4 this对象引用**      有时会有这种情况，想全面地访问当前的对象，而不是某一个特殊的实例变量。This关键字引用方法所运作的那个对象。      比如，我们可能经常会遇到这样的构造方法：this( )。这个方法会调用这个类中的另一个构造方法。我们来看看程序示例。 |
| ban |
|  |
| **6.5 类的包装和引入**      package和import语句      包(Package) 由一组类(class)和接口(interface)组成。它是管理大型名字空间，避免名字冲突的工具。每一个类和接口的名字都包含在某个包中。按照一般的习惯，它的名字是由"."号分隔的单词构成，第一个单词通常是开发这个包的组织的名称。      定义一个包由package语句定义。如果使用package语句，编译单元的第一行必须无空格，也无注释。其格式如下：package packageName;若编译单元无package语句，则该单元被置于一个缺省的无名的包中。即：如果源文件中定义的类在放在同一个包中，但该包没有名字。      import语句用来引入所需要的类。      为了引用已有的类，使用import语句，import语句必须放在package语句（如果有的华）和类的定义之间。import语句的格式：      import 包名.类名  比如： import java.applet.Applet; import java.awt.Label;  如果要引用一个包里所有的类，用\*表示，比如： import java.awt.\*;      package语句用来指明该源文件定义的类所在的包，可以把包粗略理解为存放类的文件夹。比如，java.lang.\*，就是java提供的一个程序包，里面放置了诸如数学方法等很多有用的程序，供大家调用。我们把package语句叫做包装语句。为了引用一个类，我们需要用引入语句：import。比如，我们要使用数学方法（比如求绝对值、求开方等），就要用import java.lang.Math，然后就可以调用Math里的方法了，非常方便。      包装的作用是方便管理，避免重名。比如，我们的例子，通过 package teach5包装在teach4里的：以后如果谁要用我们的程序中的类，只要用import teach4就可以引用了。注意，同在一个package里的类，可以直接相互引用，不用import语句。比如我们例子中，EmployeeTest类使用了Employee类，但没有用 import Employee之类的语句，就是因为它们都被包装在teach5里。 |
| ban |
|  |
| **6.5 类的包装和引入**      package和import语句      包(Package) 由一组类(class)和接口(interface)组成。它是管理大型名字空间，避免名字冲突的工具。每一个类和接口的名字都包含在某个包中。按照一般的习惯，它的名字是由"."号分隔的单词构成，第一个单词通常是开发这个包的组织的名称。      定义一个包由package语句定义。如果使用package语句，编译单元的第一行必须无空格，也无注释。其格式如下：package packageName;若编译单元无package语句，则该单元被置于一个缺省的无名的包中。即：如果源文件中定义的类在放在同一个包中，但该包没有名字。      import语句用来引入所需要的类。      为了引用已有的类，使用import语句，import语句必须放在package语句（如果有的话）和类的定义之间。import语句的格式：      import 包名.类名  比如： import java.applet.Applet; import java.awt.Label;  如果要引用一个包里所有的类，用\*表示，比如： import java.awt.\*;      package语句用来指明该源文件定义的类所在的包，可以把包粗略理解为存放类的文件夹。比如，java.lang.\*，就是java提供的一个程序包，里面放置了诸如数学方法等很多有用的程序，供大家调用。我们把package语句叫做包装语句。为了引用一个类，我们需要用引入语句：import。比如，我们要使用数学方法（比如求绝对值、求开方等），就要用import java.lang.Math，然后就可以调用Math里的方法了，非常方便。      包装的作用是方便管理，避免重名。比如，我们的例子，通过 package teach5包装在teach4里的：以后如果谁要用我们的程序中的类，只要用import teach4就可以引用了。注意，同在一个package里的类，可以直接相互引用，不用import语句。比如我们例子中，EmployeeTest类使用了Employee类，但没有用 import Employee之类的语句，就是因为它们都被包装在teach5里。 |
| ban |
|  |
| **小结**      这一讲，我们主要学习了类的继承和接口。至此，我们学习了完Java语言的主要的必备的基础知识。同学们现在应该可以编写自己的面向对象的应用程序了。希望同学们多加练习，熟练掌握。下一讲开始，我们开始面向应用，学习编写小应用程序，也就是Java Applet。 |
| ban |
|  |
| **习题**   1. 子类和父类的构造方法调用顺序是怎么样的，举例说明。 2. 编程实现"人"类、"学生"类和"大学生"类。"人"类是"学生"类的父类，"学生"类是"大学生"类的父类。"人"类的成员变量有：年龄，性别，籍贯；"学生"类的成员变量有：就读学校，成绩考评；"大学生"类的变量有：导师，每月补助。 3. 要求实现Comparable接口，按年龄比较两个人的大小；  * 类的数组； * 类作为参数 * this指针 * 子类 * 抽象类 |
| ban |
|  |
| **习题**   1. 子类和父类的构造方法调用顺序是怎么样的，举例说明。 2. 编程实现"人"类、"学生"类和"大学生"类。"人"类是"学生"类的父类，"学生"类是"大学生"类的父类。"人"类的成员变量有：年龄，性别，籍贯；"学生"类的成员变量有：就读学校，成绩考评；"大学生"类的变量有：导师，每月补助。 3. 要求实现Comparable接口，按年龄比较两个人的大小；  * 类的数组； * 类作为参数 * this指针 * 子类 * 抽象类 |

补充部分

**竖**

***诚信、创新、开放、合作***

**JAVA的面向对象编程--------课堂笔记**

**面向对象主要针对面向过程。**

**面向过程的基本单元是函数。**

**什么是对象：EVERYTHING IS OBJECT（万物皆对象）**

**所有的事物都有两个方面：**

**有什么（属性）：用来描述对象。**

**能够做什么（方法）：告诉外界对象有那些功能。**

**后者以前者为基础。**

**大的对象的属性也可以是一个对象。**

**为什么要使用面向对象：**

**首先，面向对象符合人类看待事物的一般规律。**

**对象的方法的实现细节是屏蔽的，只有对象方法的实现者了解细节。**

**方法的定义非常重要。方法有参数，也可能有返回值。**

**注意区分：对象（本身）、对象的实现者、对象的调用者。**

**分析对象主要从方法开始。**

**我们通过类来看待对象，类是对象的抽象。**

**其次，采用面向对象方法可以使系统各部分各司其职、各尽所能。**

**对象之间的耦合性一定要低（比如不同硬盘和不同主板之间的关系）。这样才能使每个对象本身做成最好的。**

**对于对象的要求：高内聚、低耦合，这样容易拼装成为一个系统。**

**实现高内聚就是要最大限度低提高复用性（复用性好是因为高内聚）。**

**可复用性是OOP的基础。**

**比较面向过程的思想和面向对象的思想：**

**面向过程的思想：由过程、步骤、函数组成，以过程为核心；**

**面向对象的思想：以对象为中心，先开发类，得到对象，通过对象之间相互通信实现功能。**

**面向过程是先有算法，后有数据结构。**

**面向对象是先有数据结构，然后再有算法。**

**在用面向对象思想开发的过程中，可以复用对象就进行复用，如无法进行复用则开发新的对象。**

**开发过程是用对个简单的对象的多个简单的方法，来实现复杂的功能 。**

**从语法上来看，一个类是一个新的数据类型。**

**在面向对象编程中，除了简单数据类型，就是对象类型。**

**定义类的格式：**

**class Student{**

**代码**

**}**

**注意类名中单词的首字母大写。**

**实例变量：定义在类中但在任何方法之外。（New出来的均有初值）**

**局部变量：定义在方法之中的变量。**

**局部变量要先赋值，再进行运算，而实例变量均已经赋初值。这是局部变量和实例变量的一大区别。**

**实例变量的对象赋值为null。**

**局部变量不允许范围内定义两个同名变量。实例变量的作用域在本类中完全有效，当被其他的类调用的时候也可能有效。**

**实例变量和局部变量允许命名冲突。**

**书写方法的格式：**

**修饰符 返回值 方法名 调用过程中 方法体**

**可能出现的例外**

**public int/void addNumber(参数) throw Excepion {}**

**例：**

**public int addNumber(int a,int b){**

**}**

**注：方法名中的参数int a,int b为局部变量**

**类方法中的一类特殊方法：构造方法。**

**构造方法是当用类生成对象时，系统在生成对象的过程中利用的方法。**

**注意：构造方法在生成对象的时候会被调用，但并不是构造方法生成了对象。**

**构造方法没有返回值。格式为：public 方法名。**

**构造方法的方法名与类名相同。**

**构造方法是在对象生成的过程中自动调用，不可能利用指令去调用。**

**在一个对象的生成周期中构造方法只用一次，一旦这个对象生成，那么这个构造方法失效。**

**用类来生成对象的语句：**

**Student s=new Student()。**

**第一个Student表示这是用Student类进行定义。“Student()”表示调用一个无参数的构造方法。**

**如果()中有参数，则系统构造对象的过程中调用有参的方法。**

**此时S称为一个对象变量。**

**Student s的存储区域存放的是地址：一个对象在硬盘上占有一个连续地址，首地址赋予s空间。**

**S称为对象Student的引用。**

**注意：在对象变量中存放的是引用（地址）；在简单变量中存放的是数值。**

**可以构造多个构造方法，但多个构造方法的参数表一定不同，参数顺序不同即属于不同的构造方法：**

**public student(string name,int a){**

**}**

**public student(int a,string name){**

**}**

**为两个不同的构造方法。**

**如果我们未给系统提供一个构造方法，那么系统会自动提供一个为空的构造方法。**

**练习：写一个类，定义一个对象，定义两个构造方法：一个有参，一个无参。**

**（编写一个程序验证对象的传递的值为地址）**

**注意下面这种形式：**

**static void changename(student stu){stu.setName “LUCY”}**

**注意生成新的对象与旧对象指向无关，生成新对象生命消亡与旧对象无关。**

**面向对象方法的重载（overloading）和覆盖（overriding）。**

**在有些JAVA书籍中将overriding称为重载，overloading称为过载。**

**Overloading在一个类中可以定义多个同名方法，各个方法的参数表一定不同。但修饰词可能相同，返回值也可能相同。**

**在程序的编译过程中根据变量类型来找相应的方法。因此也有人认为 overloading是编译时的多态，以后我们还会学到运行时多态。**

**为什么会存在overloading技术呢？作为应对方法的细节。**

**利用类型的差异来影响对方法的调用。**

**吃（）可以分为吃肉，吃菜，吃药，在一个类中可以定义多个吃方法。**

**构造方法也可以实现overloading。例：**

**public void teach(){};**

**public void teach(int a){};**

**public void teach(String a){}为三种不同的方法。**

**Overloading方法是从低向高转。**

**Byte—short—float—int—long—double。**

**在构造方法中，this表示本类的其他构造方法：**

**student(){};**

**student(string n){**

**this();//表示调用student()**

**}**

**如果调用student(int a)则为this(int a)。**

**特别注意：用this调用其他构造方法时，this必须为第一条语句，然后才是其他语句。**

**This表示当前对象。**

**Public void printNum(){**

**Int number=40；**

**System.out.println(this.number);**

**}**

**此时打印的是实例变量，而非局部变量，即定义在类中而非方法中的变量。**

**This.number表示实例变量。**

**谁调用this.number那么谁即为当前(this)对象的number方法。**

**封装：使对象的属性尽可能私有，对象的方法尽可能的公开。用private表示此成员属性为该类的私有属性。**

**Public表示该属性（方法）公开；**

**Private表示该属性（方法）为只有本类内部可以访问（类内部可见）。**

**（想用private还要用set和get方法供其他方法调用，这样可以保证对属性的访问方式统一，并且便于维护访问权限以及属性数据合法性）**

**如果没有特殊情况，属性一定私有，方法该公开的公开。**

**如果不指明谁调用方法，则默认为this。**

**区分实例变量和局部变量时一定要写this。**

**11.29**

**继承：**

**父类（SuperClass）和 子类（SonClass）。**

**父类的非私有化属性和方法可以默认继承到子类。**

**Class Son extends Father{**

**}**

**而如果父类中的私有方法被子类调用的话，则编译报错。**

**父类的构造方法子类不可以继承，更不存在覆盖的问题。（非构造方法可以）**

**如果子类访问父类的构造方法，则在编译的时候提示访问不到该方法。**

**JAVA中不允许多继承，一个类有且只有一个父类（单继承）。**

**JAVA的数据结构为树型结构，而非网状。（JAVA通过接口和内部类实现多继承）**

**方法的覆盖（overriding）**

**方法的重载并不一定是在一个类中：子类可以从父类继承一个方法，也可以定义一个同名异参的方法，也称为overloading。**

**当子类从父类继承一个无参方法，而又定义了一个同样的无参方法，则子类新写的方法覆盖父类的方法，称为覆盖。（注意返回值类型也必须相同，否则编译出错。）**

**如果方法不同，则成重载。**

**对于方法的修饰词，子类方法要比父类的方法范围更加的宽泛。**

**父类为public，那么子类为private则出现错误。**

**之所以构造方法先运行父类再运行子类是因为构造方法是无法覆盖的。**

**以下范围依次由严到宽：**

**private ：本类访问；**

**default ：表示默认，不仅本类访问，而且是同包可见。**

**Protected：同包可见+不同包的子类可见**

**Public ：表示所有的地方均可见。**

**当构造一个对象的时候，系统先构造父类对象，再构造子类对象。**

**构造一个对象的顺序：（注意：构造父类对象的时候也是这几步）**

1. **递归地构造父类对象；**
2. **顺序地调用本类成员属性赋初值语句；**
3. **本类的构造方法。**

**Super()表示调用父类的构造方法。**

**Super()也和this一样必须放在第一行。**

**This()用于调用本类的构造方法。**

**如果没有定义构造方法，那么就会调用父类的无参构造方法，即super()。**

**要养成良好的编程习惯：就是要加上默认的父类无参的构造方法。**

**思考：可是如果我们没有定义无参的构造方法，而在程序中构造了有参的构造方法，那么如果方法中没有参数，那么系统还会调用有参的构造方法么？应该不会。**

**多态：多态指的是编译时类型变化，而运行时类型不变。**

**多态分两种：**

1. **编译时多态：编译时动态重载；**
2. **运行时多态：指一个对象可以具有多个类型。**

**对象是客观的，人对对象的认识是主观的。**

**例：**

**Animal a=new Dog()；查看格式名称；**

**Dog d=(Dog)a。声明父类来引用子类。**

**（思考上面的格式）**

**运行时多态的三原则：（应用时为覆盖）**

1. **对象不变；（改变的是主观认识）**
2. **对于对象的调用只能限于编译时类型的方法，如调用运行时类型方法报错。**

**在上面的例子中：Animal a=new Dog()；对象a的编译时类型为Animal，运行时类型为dog。**

**注意：编译时类型一定要为运行时类型的父类（或者同类型）。**

**对于语句：Dog d=(Dog)a。将d强制声明为a类型，此时d为Dog()，此时d就可以调用运行时类型。注意：a和d指向同一对象。**

1. **在程序的运行时，动态类型判定。运行时调用运行时类型，即它调用覆盖后的方法。**

**关系运算符：instanceof**

**a instanceof Animal;(这个式子的结果是一个布尔表达式)**

**a为对象变量，Animal是类名。**

**上面语句是判定a是否可以贴Animal标签。如果可以贴则返回true，否则返回false。**

**在上面的题目中： a instanceof Animal返回 True，**

**a instanceof Dog也返回 True，**

**instanceof用于判定是否将前面的对象变量赋值后边的类名。**

**Instanceof一般用于在强制类型转换之前判定变量是否可以强制转换。**

**如果Animal a=new Animal()；**

**Dog d=Dog()a;**

**此时编译无误，但运行则会报错。**

**Animal a=new Dog()相当于下面语句的功能：**

**Animal a=getAnimal()；**

**Public static Animal.getAnimal;**

**Return new Dog()；**

**封装、继承、多态为面向对象的三大基石（特性）。**

**运行时的动态类型判定针对的是方法。运行程序访问的属性仍为编译时属性。**

**Overloading针对的是编译时类型，不存在运行时的多态。**

**习题：建立一个shape类，有circle和rect子类。**

**Shape类有zhouchang()和area()两种方法。**

**（正方形）squ为rect子类，rect有cha()用于比较长宽的差。**

**覆盖时考虑子类的private及父类的public（考虑多态），之所以这样是避免调用A时出现实际调用B的情况。而出现错误。**

**11.29下午讲的是教程上的Module6**

**Module6-7包括：面向对象高级、内部类、集合、反射（暂时不讲）、例外。**

**面向对象高级、集合和例外都是面向对象的核心内容。**

**面向对象高级： 修饰符：**

**static:①可修饰变量（属性）；②可修饰方法；③可修饰代码块。**

**Static int data语句说明data为类变量，为一个类的共享变量，属于整个类。**

**Int data为实例变量。**

**例：**

**static int data;**

**m1.data=0;**

**m1.data++的结果为1,此时m2.data的结果也为1。**

**Static定义的是一块为整个类共有的一块存储区域，其发生变化时访问到的数据都时经过变化的。**

**其变量可以通过类名去访问：类名.变量名。与通过访问对象的编译时类型访问类变量为等价的。**

**Public static void printData(){}**

**表明此类方法为类方法（静态方法）**

**静态方法不需要有对象，可以使用类名调用。**

**静态方法中不允许访问类的非静态成员，包括成员的变量和方法，因为此时是通过类调用的，没有对象的概念。This.data是不可用的。**

**一般情况下，主方法是静态方法，所以可调用静态方法，主方法为静态方法是因为它是整个软件系统的入口，而进入入口时系统中没有任何对象，只能使用类调用。**

**覆盖不适用于静态方法。**

**静态方法不可被覆盖。（允许在子类中定义同名静态方法，但是没有多态，严格的讲，方法间没有多态就不能称为覆盖）**

**当static修饰代码块时（注：此代码块要在此类的任何一个方法之外），那么这个代码块在代码被装载进虚拟机生成对象的时候可被装载一次，以后再也不执行了。**

**一般静态代码块被用来初始化静态成员。**

**Static通常用于Singleton模式开发：**

**Singleton是一种设计模式，高于语法，可以保证一个类在整个系统中仅有一个对象。**

*-**--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------*

*-------------------------------------------------------------------------------------------------------------*

*-------------------------------------------------------------------------------------------------------------*

**11.30**

**final可以修饰类、属性、方法。**

**当用final修饰类的时候，此类不可被继承，即final类没有子类。这样可以用final保证用户调用时动作的一致性，可以防止子类覆盖情况的发生。**

**当利用final修饰一个属性（变量）的时候，此时的属性成为常量。**

**JAVA利用final定义常量（注意在JAVA命名规范中常量需要全部字母都大写）：**

**Final int AGE=10；**

**常量的地址不可改变，但在地址中保存的值（即对象的属性）是可以改变的。**

**Final可以配合static使用。 ？**

**Static final int age=10；**

**在JAVA中利用public static final的组合方式对常量进行标识（固定格式）。**

**对于在构造方法中利用final进行赋值的时候，此时在构造之前系统设置的默认值相对于构造方法失效。**

**常量（这里的常量指的是实例常量：即成员变量）赋值：**

**①在初始化的时候通过显式声明赋值。Final int x=3；**

**②在构造的时候赋值。**

**局部变量可以随时赋值。**

**利用final定义方法：这样的方法为一个不可覆盖的方法。**

**Public final void print(){}；**

**为了保证方法的一致性（即不被改变），可将方法用final定义。**

**如果在父类中有final定义的方法，那么在子类中继承同一个方法。**

**如果一个方法前有修饰词private或static，则系统会自动在前面加上final。即private和static方法默认均为final方法。**

**注：final并不涉及继承，继承取决于类的修饰符是否为private、default、protected还是public。也就是说，是否继承取决于这个方法对于子类是否可见。**

**Abstract(抽象)可以修饰类、方法**

**如果将一个类设置为abstract，则此类必须被继承使用。此类不可生成对象，必须被继承使用。**

**Abstract可以将子类的共性最大限度的抽取出来，放在父类中，以提高程序的简洁性。**

**Abstract虽然不能生成对象，但是可以声明，作为编译时类型，但不能作为运行时类型。**

**Final和abstract永远不会同时出现。**

**当abstract用于修饰方法时，此时该方法为抽象方法，此时方法不需要实现，实现留给子类覆盖，子类覆盖该方法之后方法才能够生效。**

**注意比较：**

**private void print(){}；此语句表示方法的空实现。**

**Abstract void print()； 此语句表示方法的抽象，无实现。**

**如果一个类中有一个抽象方法，那么这个类一定为一个抽象类。**

**反之，如果一个类为抽象类，那么其中可能有非抽象的方法。**

**如果让一个非抽象类继承一个含抽象方法的抽象类，则编译时会发生错误。因为当一个非抽象类继承一个抽象方法的时候，本着只有一个类中有一个抽象方法，那么这个类必须为抽象类的原则。这个类必须为抽象类，这与此类为非抽象冲突，所以报错。**

**所以子类的方法必须覆盖父类的抽象方法。方法才能够起作用。**

只有将理论被熟练运用在实际的程序设计的过程中之后，才能说理论被完全掌握！

**为了实现多态，那么父类必须有定义。而父类并不实现，留给子类去实现。此时可将父类定义成abstract类。如果没有定义抽象的父类，那么编译会出现错误。**

**Abstract和static不能放在一起，否则便会出现错误。（这是因为static不可被覆盖，而abstract为了生效必须被覆盖。）**

**例：（本例已存在\CODING\abstract\TestClass.java文件中）**

**public class TestClass{**

**public static void main(String[] args){**

**SuperClass sc=new SubClass();**

**Sc.print();**

**}**

**Abstract class SuperClass{**

**Abstract void print();}**

**}**

**class SubClass extends SuperClass(){**

**void print(){**

**System.out.println(“print”);}**

**}**

**JAVA的核心概念：接口（interface）**

**接口与类属于同一层次，实际上，接口是一种特殊的抽象类。**

**如:**

**interface IA{**

**}**

**public interface：公开接口**

**与类相似，一个文件只能有一个public接口，且与文件名相同。**

**在一个文件中不可同时定义一个public接口和一个public类。**

**一个接口中，所有方法为公开、抽象方法；所有的属性都是公开、静态、常量。**

**一个类实现一个接口的格式：**

**class IAImple implements IA{**

**};**

**一个类实现接口，相当于它继承一个抽象类。**

**类必须实现接口中的方法，否则其为一抽象类。**

**实现中接口和类相同。**

**接口中可不写public，但在子类中实现接口的过程中public不可省。**

**（如果剩去public则在编译的时候提示出错：对象无法从接口中实现方法。）**

**注：**

1. **一个类除继承另外一个类，还可以实现接口；**

**class IAImpl extends java.util.Arrylist implement IA{}**

**继承类 实现接口**

**这样可以实现变相的多继承。**

1. **一个类只能继承另外一个类，但是它可以继承多个接口，中间用“，”隔开。**

**Implements IA,IB**

**所谓实现一个接口，就是指实现接口中的方法。**

1. **接口和接口之间可以定义继承关系，并且接口之间允许实现多继承。**

**例：interface IC extends IA,IB{};**

**接口也可以用于定义对象**

**IA I=new IAImpl();**

**实现的类从父类和接口继承的都可做运行时类型。**

**IAImple extends A implement IA,IB**

**IB I=new IAImple();**

**I instance of IAImple;**

**I instance of A;**

**I instance of IA;**

**I instance of IB;**

**返回的结果均为true.**

**接口和多态都为JAVA技术的核心。**

**接口往往被我们定义成一类XX的东西。**

**接口实际上是定义一个规范、标准。**

1. **通过接口可以实现不同层次、不同体系对象的共同属性；**

**通过接口实现write once as anywhere.**

**以JAVA数据库连接为例子：JDBC制定标准；数据厂商实现标准；用户使用标准。**

**接口通常用来屏蔽底层的差异。**

**②接口也因为上述原因被用来保持架构的稳定性。**

**JAVA中有一个特殊的类： Object。它是JAVA体系中所有类的父类（直接父类或者间接父类）。**

**此类中的方法可以使所的类均继承。**

**以下介绍的三种方法属于Object:**

1. **finalize方法：当一个对象被垃圾回收的时候调用的方法。**
2. **toString():是利用字符串来表示对象。**

**当我们直接打印定义的对象的时候，隐含的是打印toString()的返回值。**

**可以通过子类作为一个toString()来覆盖父类的toString()。**

**以取得我们想得到的表现形式，即当我们想利用一个自定义的方式描述对象的时候，我们应该覆盖toString()。**

**(3)equal**

**首先试比较下例：**

**String A=new String(“hello”);**

**String A=new String(“hello”);**

**A==B(此时程序返回为FALSE)**

**因为此时AB中存的是地址，因为创建了新的对象，所以存放的是不同的地址。**

**附加知识：**

**字符串类为JAVA中的特殊类，String中为final类，一个字符串的值不可重复。因此在JAVA VM（虚拟机）中有一个字符串池，专门用来存储字符串。如果遇到String a=”hello”时（注意没有NEW，不是创建新串），系统在字符串池中寻找是否有”hello”，此时字符串池中没有”hello”，那么系统将此字符串存到字符串池中，然后将”hello”在字符串池中的地址返回a。如果系统再遇到String b=”hello”，此时系统可以在字符串池中找到 “hello”。则会把地址返回b，此时a与b为相同。**

**String a=”hello”;**

**System.out.println(a==”hello”);**

**系统的返回值为true。**

**故如果要比较两个字符串是否相同（而不是他们的地址是否相同）。可以对a调用equal:**

**System.out.println(a.equal(b));**

**equal用来比较两个对象中字符串的顺序。**

**a.equal(b)是a与b的值的比较。**

**注意下面程序：**

**student a=new student(“LUCY”,20);**

**student b=new student(“LUCY”,20);**

**System.out.println(a==b);**

**System.out.println(a.equal(b));**

**此时返回的结果均为false。**

**以下为定义equal（加上这个定义，返回ture或false）**

**public boolean equals(Object o){**

**student s=(student)o;**

**if (s.name.equals(this.name)&&s.age==this.age)**

**else return false;**

**}如果equals()返回的值为**

**以下为实现标准equals的流程：**

**public boolean equals(Object o){**

**if (this==o) return trun; //此时两者相同  
 if (o==null) return false;**

**if (! o instanceof strudent) return false; //不同类**

**studeng s=(student)o; //强制转换**

**if (s.name.equals(this.name)&&s.age==this.age) return true;**

**else return false;**

**}**

**以上过程为实现equals的标准过程。**

**练习：建立一个employee类，有String name,int id,double salary.运用get和set方法，使用toString，使用equals。**

**封装类：**

**JAVA为每一个简单数据类型提供了一个封装类，使每个简单数据类型可以被Object来装载。**

**除了int和char，其余类型首字母大写即成封装类。**

**转换字符的方式：**

**int I=10;**

**String s=I+” ”;**

**String s1=String.valueOf(i);**

**Int I=10;**

**Interger I\_class=new integer(I);**

**竖**

**看javadoc的帮助文档。**

**附加内容：**

**“==”在任何时候都是比较地址，这种比较永远不会被覆盖。**

**程序员自己编写的类和JDK类是一种合作关系。（因为多态的存在，可能存在我们调用JDK类的情况，也可能存在JDK自动调用我们的类的情况。）**

**注意：类型转换中double\interger\string之间的转换最多。**

**12.01**

**内部类：**

**（注：所有使用内部类的地方都可以不用内部类，使用内部类可以使程序更加的简洁，便于命名规范和划分层次结构）。**

**内部类是指在一个外部类的内部再定义一个类。**

**内部类作为外部类的一个成员，并且依附于外部类而存在的。**

**内部类可为静态，可用PROTECTED和PRIVATE修饰。（而外部类不可以：外部类只能使用PUBLIC和DEFAULT）。**

**内部类的分类：**

**成员内部类、**

**局部内部类、**

**静态内部类、**

**匿名内部类（图形是要用到，必须掌握）。**

1. **成员内部类：作为外部类的一个成员存在，与外部类的属性、方法并列。**

**内部类和外部类的实例变量可以共存。**

**在内部类中访问实例变量：this.属性**

**在内部类访问外部类的实例变量：外部类名.this.属性。**

**成员内部类的优点：**

**⑴内部类作为外部类的成员，可以访问外部类的私有成员或属性。（即使将外部类声明为PRIVATE，但是对于处于其内部的内部类还是可见的。）**

**⑵用内部类定义在外部类中不可访问的属性。这样就在外部类中实现了比外部类的private还要小的访问权限。**

**注意：内部类是一个编译时的概念，一旦编译成功，就会成为完全不同的两类。**

**对于一个名为outer的外部类和其内部定义的名为inner的内部类。编译完成后出现outer.class和outer$inner.class两类。**

**（编写一个程序检验：在一个TestOuter.java程序中验证内部类在编译完成之后，会出现几个class.）**

**成员内部类不可以有静态属性。（为什么？）**

**如果在外部类的外部访问内部类，使用out.inner.**

**建立内部类对象时应注意：**

**在外部类的内部可以直接使用inner s=new inner();（因为外部类知道inner是哪个类，所以可以生成对象。）**

**而在外部类的外部，要生成（new）一个内部类对象，需要首先建立一个外部类对象（外部类可用），然后在生成一个内部类对象。**

**Outer.Inner in=Outer.new.Inner()。**

**错误的定义方式：**

**Outer.Inner in=new Outer.Inner()。**

**注意：当Outer是一个private类时，外部类对于其外部访问是私有的，所以就无法建立外部类对象，进而也无法建立内部类对象。**

1. **局部内部类：在方法中定义的内部类称为局部内部类。**

**与局部变量类似，在局部内部类前不加修饰符public和private，其范围为定义它的代码块。**

**注意：局部内部类不仅可以访问外部类实例变量，还可以访问外部类的局部变量（但此时要求外部类的局部变量必须为final）？？**

**在类外不可直接生成局部内部类（保证局部内部类对外是不可见的）。**

**要想使用局部内部类时需要生成对象，对象调用方法，在方法中才能调用其局部内部类。**

1. **静态内部类：（注意：前三种内部类与变量类似，所以可以对照参考变量）**

**静态内部类定义在类中，任何方法外，用static定义。**

**静态内部类只能访问外部类的静态成员。**

**生成（new）一个静态内部类不需要外部类成员：这是静态内部类和成员内部类的区别。静态内部类的对象可以直接生成：**

**Outer.Inner in=new Outer.Inner()；**

**而不需要通过生成外部类对象来生成。这样实际上使静态内部类成为了一个顶级类。**

**静态内部类不可用private来进行定义。例子：**

**对于两个类，拥有相同的方法：**

**People**

**{**

**run();**

**}**

**Machine{**

**run();**

**}**

**此时有一个robot类：**

**class Robot extends People implement Machine.**

**此时run()不可直接实现。**

**注意：当类与接口（或者是接口与接口）发生方法命名冲突的时候，此时必须使用内部类来实现。**

**用接口不能完全地实现多继承，用接口配合内部类才能实现真正的多继承。**

1. **匿名内部类（必须掌握）：**

**匿名内部类是一种特殊的局部内部类，它是通过匿名类实现接口。**

**IA被定义为接口。**

**IA I=new IA(){};**

**注：一个匿名内部类一定是在new的后面，用其隐含实现一个接口或实现一个类，没有类名，根据多态，我们使用其父类名。**

**因其为局部内部类，那么局部内部类的所有限制都对其生效。**

**匿名内部类是唯一一种无构造方法类。**

**匿名内部类在编译的时候由系统自动起名Out$1.class。**

**如果一个对象编译时的类型是接口，那么其运行的类型为实现这个接口的类。**

**因匿名内部类无构造方法，所以其使用范围非常的有限。**

**（下午：）Exception（例外/异常）（教程上的MODEL7）**

**对于程序可能出现的错误应该做出预案。**

**例外是程序中所有出乎意料的结果。（关系到系统的健壮性）**

**JAVA会将所有的错误封装成为一个对象，其根本父类为Throwable。**

**Throwable有两个子类：Error和Exception。**

**一个Error对象表示一个程序错误，指的是底层的、低级的、不可恢复的严重错误。此时程序一定会退出，因为已经失去了运行所必须的物理环境。**

**对于Error错误我们无法进行处理，因为我们是通过程序来应对错误，可是程序已经退出了。**

**我们可以处理的Throwable对象中只有Exception对象（例外/异常）。**

**Exception有两个子类：Runtime exception（未检查异常）**

**非Runtime exception（已检查异常）**

**（注意：无论是未检查异常还是已检查异常在编译的时候都不会被发现，在编译的过程中检查的是程序的语法错误，而异常是一个运行时程序出错的概念。）**

**在Exception中，所有的非未检查异常都是已检查异常，没有另外的异常！！**

**未检查异常是因为程序员没有进行必要的检查，因为他的疏忽和错误而引起的异常。一定是属于虚拟机内部的异常（比如空指针）。**

**应对未检查异常就是养成良好的检查习惯。**

**已检查异常是不可避免的，对于已检查异常必须实现定义好应对的方法。**

**已检查异常肯定跨越出了虚拟机的范围。（比如“未找到文件”）**

**如何处理已检查异常（对于所有的已检查异常都要进行处理）：**

**首先了解异常形成的机制：**

**当一个方法中有一条语句出现了异常，它就会throw（抛出）一个例外对象，然后后面的语句不会执行返回上一级方法，其上一级方法接受到了例外对象之后，有可能对这个异常进行处理，也可能将这个异常转到它的上一级。**

**对于接收到的已检查异常有两种处理方式：throws和try方法。**

**注意：出错的方法有可能是JDK，也可能是程序员写的程序，无论谁写的，抛出一定用throw。**

**例：public void print() throws Exception.**

**对于方法a，如果它定义了throws Exception。那么当它调用的方法b返回异常对象时，方法a并不处理，而将这个异常对象向上一级返回，如果所有的方法均不进行处理，返回到主方法，程序中止。（要避免所有的方法都返回的使用方法，因为这样出现一个很小的异常就会令程序中止）。**

**如果在方法的程序中有一行throw new Exception()，返回错误，那么其后的程序不执行。因为错误返回后，后面的程序肯定没有机会执行，那么JAVA认为以后的程序没有存在的必要。**

**对于try……catch格式：**

**try {可能出现错误的代码块} catch(exception e){进行处理的代码} ；**

**对象变量的声明**

**用这种方法，如果代码正确，那么程序不经过catch语句直接向下运行；**

**如果代码不正确，则将返回的异常对象和e进行匹配，如果匹配成功，则处理其后面的异常处理代码。（如果用exception来声明e的话，因为exception为所有exception对象的父类，所有肯定匹配成功）。处理完代码后这个例外就完全处理完毕，程序会接着从出现异常的地方向下执行（是从出现异常的地方还是在catch后面呢？利用程序进行验证）。最后程序正常退出。**

**Try中如果发现错误，即跳出try去匹配catch，那么try后面的语句就不会被执行。**

**一个try可以跟进多个catch语句，用于处理不同情况。当一个try只能匹配一个catch。**

**我们可以写多个catch语句，但是不能将父类型的exception的位置写在子类型的excepiton之前，因为这样父类型肯定先于子类型被匹配，所有子类型就成为废话。JAVA编译出错。**

**在try，catch后还可以再跟一子句finally。其中的代码语句无论如何都会被执行（因为finally子句的这个特性，所以一般将释放资源，关闭连接的语句写在里面）。**

**如果在程序中书写了检查（抛出）exception但是没有对这个可能出现的检查结果进行处理，那么程序就会报错。**

**而如果只有处理情况（try）而没有相应的catch子句，则编译还是通不过。**

**如何知道在编写的程序中会出现例外呢**

1. **调用方法，查看API中查看方法中是否有已检查错误。**
2. **在编译的过程中看提示信息，然后加上相应的处理。**

**Exception有一个message属性。在使用catch的时候可以调用：**

**Catch(IOException e){System.out.println(e.message())};**

**Catch(IOException e){e.printStackTrace()};**

**上面这条语句回告诉我们出错类型所历经的过程，在调试的中非常有用。**

**开发中的两个道理：**

**①如何控制try的范围：根据操作的连动性和相关性，如果前面的程序代码块抛出的错误影响了后面程序代码的运行，那么这个我们就说这两个程序代码存在关联，应该放在同一个try中。**

1. **对已经查出来的例外，有throw(积极)和try catch（消极）两种处理方法。**

**对于try catch放在能够很好地处理例外的位置（即放在具备对例外进行处理的能力的位置）。如果没有处理能力就继续上抛。**

**当我们自己定义一个例外类的时候必须使其继承excepiton或者RuntimeException。**

**Throw是一个语句，用来做抛出例外的功能。**

**而throws是表示如果下级方法中如果有例外抛出，那么本方法不做处理，继续向上抛出。**

**Throws后跟的是例外类型。**

**断言是一种调试工具（assert）**

**其后跟的是布尔类型的表达式，如果表达式结果为真不影响程序运行。如果为假系统出现低级错误，在屏幕上出现assert信息。**

**Assert只是用于调试。在产品编译完成后上线assert代码就被删除了。**

**方法的覆盖中，如果子类的方法抛出的例外是父类方法抛出的例外的父类型，那么编译就会出错：子类无法覆盖父类。**

**结论：子类方法不可比父类方法抛出更多的例外。子类抛出的例外或者与父类抛出的例外一致，或者是父类抛出例外的子类型。或者子类型不抛出例外。**

**如果父类型无throws时，子类型也不允许出现throws。此时只能使用try catch。**

**练习：写一个方法：int add(int a,int b)**

**{**

**return a+b；**

**}**

**当a+b=100;抛出100为异常处理。**

**12.02**

**集合（从本部分开始涉及API）**

**集合是指一个对象容纳了多个对象，这个集合对象主要用来管理维护一系列相似的对象。**

**数组就是一种对象。（练习：如何编写一个数组程序，并进行遍历。）**

**java.util.\*定义了一系列的接口和类，告诉我们用什么类NEW出一个对象，可以进行超越数组的操作。**

**（注：JAVA1.5对JAVA1.4的最大改进就是增加了对范型的支持）**

**集合框架接口的分类：（分collection接口 和 map接口）**

**Collection接口 Map接口**

**List接口 Set接口 SortedMap接口**

**SortedSet接口**

**JAVA中所有与集合有关的实现类都是这六个接口的实现类。**

**Collection接口：集合中每一个元素为一个对象，这个接口将这些对象组织在一起，形成一维结构。**

**List接口代表按照元素一定的相关顺序来组织（在这个序列中顺序是主要的），List接口中数据可重复。**

**Set接口是数学中集合的概念：其元素无序，且不可重复。（正好与List对应）**

**SortedSet会按照数字将元素排列，为“可排序集合”。**

**Map接口中每一个元素不是一个对象，而是一个键对象和值对象组成的键值对（Key-Value）。**

**Key-Value是用一个不可重复的key集合对应可重复的value集合。（典型的例子是字典：通过页码的key值找字的value值）。**

**例子：**

**key1—value1;**

**key2—value2;**

**key3—value3.**

**SortedMap：如果一个Map可以根据key值排序，则称其为SortedMap。（如字典）**

**!!注意数组和集合的区别：数组中只能存简单数据类型。Collection接口和Map接口只能存对象。**

**以下介绍接口：**

**List接口：（介绍其下的两个实现类：ArrayList和LinkedList）**

**ArrayList和数组非常类似，其底层①也用数组组织数据，ArrayList是动态可变数组。**

1. **底层：指存储格式。说明ArrayList对象都是存在于数组中。**

**注：数组和集合都是从下标0开始。**

**ArrayList有一个add(Object o)方法用于插入数组。**

**ArrayList的使用：（完成这个程序）**

**先import java.util.\*；**

**用ArrayList在一个数组中添加数据，并遍历。**

**ArrayList中数组的顺序与添加顺序一致。**

**只有List可用get和size。而Set则不可用（因其无序）。**

**Collection接口都是通过Iterator()（即迭代器）来对Set和List遍历。**

**通过语句：Iterator it=c.iterator(); 得到一个迭代器，将集合中所有元素顺序排列。然后可以通过interator方法进行遍历，迭代器有一个游标（指针）指向首位置。**

**Interator有hasNext()，用于判断元素右边是否还有数据，返回True说明有。然后就可以调用next动作。Next()会将游标移到下一个元素，并把它所跨过的元素返回。（这样就可以对元素进行遍历）**

**练习：写一个程序，输入对象信息，比较基本信息。**

**集合中每一个元素都有对象，如有字符串要经过强制类型转换。**

**Collections是工具类，所有方法均为有用方法，且方法为static。**

**有Sort方法用于给List排序。**

**Collections.Sort()分为两部分，一部分为排序规则；一部分为排序算法。**

**规则用来判断对象；算法是考虑如何排序。**

**对于自定义对象，Sort不知道规则，所以无法比较。这种情况下一定要定义排序规则。方式有两种：**

1. **java.lang下面有一个接口：Comparable（可比较的）**

**可以让自定义对象实现一个接口，这个接口只有一个方法comparableTo(Object o)**

**其规则是当前对象与o对象进行比较，其返回一个int值，系统根据此值来进行排序。**

**如 当前对象>o对象，则返回值>0；（可将返回值定义为1）**

**如 当前对象=o对象，则返回值=0；**

**如 当前对象<o对象，则返回值〈0。（可将返回值定义为-1）**

**看TestArraylist的java代码。**

**我们通过返回值1和-1位置的调换来实现升序和降序排列的转换。**

1. **java.util下有一个Comparator(比较器)**

**它拥有compare()，用来比较两个方法。**

**要生成比较器，则用Sort中Sort（List,List(Compate)）**

**第二种方法更灵活，且在运行的时候不用编译。**

**注意：要想实现comparTo()就必须在主方法中写上implement comparable.**

**练习：生成一个EMPLOYEE类，然后将一系列对象放入到ArrayList。用Iterator遍历，排序之后，再进行遍历。**

**集合的最大缺点是无法进行类型判定（这个缺点在JAVA1.5中已经解决），这样就可能出现因为类型不同而出现类型错误。**

**解决的方法是添加类型的判断。**

**LinkedList接口（在代码的使用过程中和ArrayList没有什么区别）**

**ArrayList底层是object数组，所以ArrayList具有数组的查询速度快的优点以及增删速度慢的缺点。**

**而在LinkedList的底层是一种双向循环链表。在此链表上每一个数据节点都由三部分组成：前指针（指向前面的节点的位置），数据，后指针（指向后面的节点的位置）。最后一个节点的后指针指向第一个节点的前指针，形成一个循环。**

**双向循环链表的查询效率低但是增删效率高。所以LinkedList具有查询效率低但增删效率高的特点。**

**ArrayList和LinkedList在用法上没有区别，但是在功能上还是有区别的。**

**LinkedList经常用在增删操作较多而查询操作很少的情况下：队列和堆栈。**

**队列：先进先出的数据结构。**

**堆栈：后进先出的数据结构。**

**注意：使用堆栈的时候一定不能提供方法让不是最后一个元素的元素获得出栈的机会。**

**LinkedList提供以下方法：（ArrayList无此类方法）**

**addFirst();**

**removeFirst();**

**addLast();**

**removeLast();**

**在堆栈中，push为入栈操作，pop为出栈操作。**

**Push用addFirst()；pop用removeFirst()，实现后进先出。**

**用isEmpty()--其父类的方法，来判断栈是否为空。**

**在队列中，put为入队列操作，get为出队列操作。**

**Put用addFirst()，get用removeLast()实现队列。**

**List接口的实现类（Vector）（与ArrayList相似，区别是Vector是重量级的组件，使用使消耗的资源比较多。）**

**结论：在考虑并发的情况下用Vector（保证线程的安全）。**

**在不考虑并发的情况下用ArrayList（不能保证线程的安全）。**

**面试经验（知识点）：**

**java.util.stack（stack即为堆栈）的父类为Vector。可是stack的父类是最不应该为Vector的。因为Vector的底层是数组，且Vector有get方法（意味着它可能访问到并不属于最后一个位置元素的其他元素，很不安全）。**

**对于堆栈和队列只能用push类和get类。**

**Stack类以后不要轻易使用。**

**！！！实现堆栈一定要用LinkedList。**

**（在JAVA1.5中，collection有queue来实现队列。）**

**Set-HashSet实现类：**

**遍历一个Set的方法只有一个：迭代器（interator）。**

**HashSet中元素是无序的（这个无序指的是数据的添加顺序和后来的排列顺序不同），而且元素不可重复。**

**在Object中除了有final()，toString()，equals()，还有hashCode()。**

**HashSet底层用的也是数组。**

**当向数组中利用add(Object o)添加对象的时候，系统先找对象的hashCode：**

**int hc=o.hashCode(); 返回的hashCode为整数值。**

**Int I=hc%n;（n为数组的长度），取得余数后，利用余数向数组中相应的位置添加数据，以n为6为例，如果I=0则放在数组a[0]位置，如果I=1,则放在数组a[1]位置。如果equals()返回的值为true，则说明数据重复。如果equals()返回的值为false，则再找其他的位置进行比较。这样的机制就导致两个相同的对象有可能重复地添加到数组中，因为他们的hashCode不同。**

**如果我们能够使两个相同的对象具有相同hashcode，才能在equals()返回为真。**

**在实例中，定义student对象时覆盖它的hashcode。**

**因为String类是自动覆盖的，所以当比较String类的对象的时候，就不会出现有两个相同的string对象的情况。**

**现在，在大部分的JDK中，都已经要求覆盖了hashCode。**

**结论：如将自定义类用hashSet来添加对象，一定要覆盖hashcode()和equals()，覆盖的原则是保证当两个对象hashcode返回相同的整数，而且equals()返回值为True。**

**如果偷懒，没有设定equals()，就会造成返回hashCode虽然结果相同，但在程序执行的过程中会多次地调用equals()，从而影响程序执行的效率。**

**我们要保证相同对象的返回的hashCode一定相同，也要保证不相同的对象的hashCode尽可能不同（因为数组的边界性，hashCode还是可能相同的）。例子：**

**public int hashCode(){**

**return name.hashcode()+age;**

**}**

**这个例子保证了相同姓名和年龄的记录返回的hashCode是相同的。**

**使用hashSet的优点：**

**hashSet的底层是数组，其查询效率非常高。而且在增加和删除的时候由于运用的hashCode的比较开确定添加元素的位置，所以不存在元素的偏移，所以效率也非常高。因为hashSet查询和删除和增加元素的效率都非常高。**

**但是hashSet增删的高效率是通过花费大量的空间换来的：因为空间越大，取余数相同的情况就越小。HashSet这种算法会建立许多无用的空间。**

**使用hashSet接口时要注意，如果发生冲突，就会出现遍历整个数组的情况，这样就使得效率非常的低。**

**练习：new一个hashset，插入employee对象，不允许重复，并且遍历出来。**

**添加知识点：**

**集合对象存放的是一系列对象的引用。**

**例：**

**Student S**

**Al.add(s);**

**s.setName(“lucy”);**

**Student s2=(Student)(al.get(o1));**

**可知s2也是s。**

**12.05**

**SortedSet可自动为元素排序。**

**SortedSet的实现类是TreeSet:它的作用是字为添加到TreeSet中的元素排序。**

**练习：自定义类用TreeSet排序。**

**与HashSet不同，TreeSet并不需要实现HashCode()和equals()。**

**只要实现compareable和compareTo()接可以实现过滤功能。**

**（注：HashSet不调用CompareTo()）。**

**如果要查询集合中的数据，使用Set必须全部遍历，所以查询的效率低。使用Map，可通过查找key得到value，查询效率高。**

**集合中常用的是：ArrayList，HashSet，HashMap。其中ArrayList和HashMap使用最为广泛。**

**使用HashMap，put()表示放置元素，get()表示取元素。**

**遍历Map，使用keySet()可以返回set值，用keySet()得到key值，使用迭代器遍历，然后使用put()得到value值。**

**上面这个算法的关键语句：**

**Set s=m.keySet();**

**Interator it=new interator();**

**Object key=it.next();**

**Object value=m.get(key);**

**注意：HashMap与HashCode有关，用Sort对象排序。**

**如果在HashMap中有key值重复，那么后面一条记录的value覆盖前面一条记录。**

**Key值既然可以作为对象，那么也可以用一个自定义的类。比如：**

**m.put(new sutdent(“Liucy”,30),”boss”)**

**如果没有语句来判定Student类对象是否相同，则会全部打印出来。**

**当我们用自定义的类对象作为key时，我们必须在程序中覆盖HashCode()和equals()。**

**注：HashMap底层也是用数组，HashSet底层实际上也是HashMap，HashSet类中有HashMap属性（我们如何在API中查属性）。HashSet实际上为(key.null)类型的HashMap。有key值而没有value值。**

**正因为以上的原因，TreeSet和TreeMap的实现也有些类似的关系。**

**注意：TreeSet和TreeMap非常的消耗时间，因此很少使用。**

**我们应该熟悉各种实现类的选择——非常体现你的功底。**

**HashSet VS TreeSet：HashSet非常的消耗空间，TreeSet因为有排序功能，因此资源消耗非常的高，我们应该尽量少使用，而且最好不要重复使用。**

**基于以上原因，我们尽可能的运用HashSet而不用TreeSet，除非必须排序。**

**同理：HashMap VS TreeMap:一般使用HashMap，排序的时候使用TreeMap。**

**HashMap VS Hashtable（注意在这里table的第一个字母小写）之间的区别有些类似于ArrayList和Vector，Hashtable是重量级的组件，在考虑并发的情况，对安全性要求比较高的时候使用。**

**Map的运用非常的多。**

**使用HashMap()，如果使用自定义类，一定要覆盖HashCode()和equals()。**

**重点掌握集合的四种操作：增加、删除、遍历、排序。**

**Module8—12利用两天的时间完成。**

**Module8：图形界面**

**Module9：事件模型（在本部分最重要）**

**Module10：AWT**

**Module11：Swing**

**Module12：Applet（这个技术基本已经被淘汰）**

**软件应用的三个发展阶段：**

**单机应用**

**网络应用（C/S结构）**

**BS结构：B表示浏览器，S表示server端。即利用浏览器作为客户端，因此对于图形界面的要求已经不高，现在的发展趋势是不使用安装，即不用任何的本地应用，图形很快就会被服务器构件开发所取代。**

**经验之谈：Swing的开发工作会非常的累，而且这项技术正在走向没落。避免从事有这种特征的工作。**

**AWT也即将被取代。**

**Module8—Module11所使用的技术都将被JSF技术所取代。**

**JSF是服务器端的Swing：目前技术已经成熟，但是开发环境（工具）还不成熟。**

**Module12的Applet技术也将被WebStart所取代。**

**Module9为重点，所谓事件模型是指观察者设计模式的JAVA应用。事件模型是重点。**

**Module8：图形界面（java.awt.\*）**

**Awt：抽象窗口工具箱，它由三部分组成：**

**①组件：界面元素；**

**②容器：装载组件的容器（例如窗体）；**

**③布局管理器：负责决定容器中组件的摆放位置。**

**图形界面的应用分四步：**

1. **选择一个容器：**

**⑴window:带标题的容器（如Frame）；**

**⑵Panel:面板**

**通过add()想容器中添加组件。**

**Java的图形界面依然是跨平台的。但是在调用了一个窗体之后只生成一个窗体，没有事件的处理，关闭按钮并不工作。此时只能使用CTRL+C终止程序。**

**②设置一个布局管理器：用setLayout()；**

**③向容器中添加组件；**

1. **添加组件的事务处理。P198**

**P204：Panel也是一种容器：但是不可见的。在设置容易的时候不要忘记设置它们的可见性。**

**Panel pan=new Panel;**

**Fp.setLayout(null);表示不要布局管理器。**

**五种布局管理器：**

**P206：Flow Layout(流式布局)：按照组件添加到容器中的顺序，顺序排放组件位置。默认为水平排列，如果越界那么会向下排列。排列的位置随着容器大小的改变而改变。**

**Panel默认的布局管理器为Flow Layout。**

**Border Layout：会将容器非常五个区域：东西南北中。**

**语句：**

**Button b1=new Botton(“north”);//botton上的文字**

**f.add(b1,”North”);//表示b1这个botton放在north位置**

**注：一个区域只能放置一个组件，如果想在一个区域放置多个组件就需要使用Panel来装载。**

**Frame和Dialog的默认布局管理器是Border Layout。**

**Grid Layout：将容器生成等长等大的条列格，每个块中放置一个组件。**

**f.setLayout GridLayout(5,2,10,10)//表示条列格为5行2类，后面为格间距。**

**CardLayout:一个容器可以放置多个组件，但每次只有一个组件可见（组件重叠）。**

**使用first()，last()，next()可以决定哪个组件可见。可以用于将一系列的面板有顺序地呈现给用户。**

**重点：GridBag Layout:在Grid中可指定一个组件占据多行多列，GridBag的设置非常的烦琐。**

**Module9:AWT:事件模型**

**事件模型指的是对象之间进行通信的设计模式。**

**对象1给对象2发送一个信息相当于对象1引用对象2的方法。**

**模型即是一种设计模式（约定俗成）**

**对象对为三种：**

**①事件源：发出事件者；**

**②事件对象：发出的事件本身；**

1. **事件监听器：提供处理事件指定的方法。**

**Java AWT事件模型也称为授权事件模型，指事件可以和监听器之间事先建立一种关系：约定那些事件如何处理，由谁去进行处理。这种约定称为授权。**

**一个事件源可以授权多个监听者（授权也称为监听者的注册）；**

**多个事件源也可以注册多个事件监听器。**

**监听者对于事件源的发出的事件作出响应。**

**竖**

**在java.util中有EventListener接口：所有事件监听者都要实现这个接口。**

**java.util中有EventObject类：所有的事件都为其子类。**

**事件范例在\CoreJava\Girl.java文件中。(文件已加注释)**

**注意：接口因对不同的事件监听器对其处理可能不同，所以只能建立监听的功能，而无法实现处理。**

**下面程序建立监听功能：**

**//监听器接口要定义监听器所具备的功能，定义方法**

**{**

**void WhatIdoWhenGirlHappy(EmotionEvent e);**

**void WhatIdoWhenGirlSad(EmotionEvent e);**

**}**

**注意查看参考书：事件的设置模式，如何实现授权模型。**

**事件模式的实现步骤：**

**开发事件对象（事件发送者）——接口——接口实现类——设置监听对象**

**一定要理解透彻Gril.java程序。**

**重点：学会处理对一个事件源有多个事件的监听器（在发送消息时监听器收到消息的排名不分先后）。**

**事件监听的响应顺序是不分先后的，不是谁先注册谁就先响应。**

**事件监听由两个部分组成（接口和接口的实现类）。**

**事件源 事件对象 事件监听**

**gril EmotinEvent EmotionListener(接口)、Boy(接口的实现类)**

**鼠标事件：MouseEvent，接口：MouseListener。**

**P235 ActionEvent。**

**注意在写程序的时候：import java.awt.\*;以及import java.awt.event.\*注意两者的不同。**

**在生成一个窗体的时候，点击窗体的右上角关闭按钮激发窗体事件的方法：窗体Frame为事件源，WindowsListener接口调用Windowsclosing()。**

**为了配合后面的实现，我们必须将WindowsListener所有的方法都实现，除了Windowsclosing方法，其余的方法均为空实现。**

**（练习：写一个带button窗体，点关闭按钮退出。）**

**上面程序中实现了许多不必要的实现类，虽然是空实现。**

**为了避免上面那些无用的实现，可以利用WindowEvent的一个WindowEvent类，还是利用windowsListener。还有WindowAdapter类，它已经实现了WindowsListener。它给出的全部都是空实现，那就可以只写想要实现的类，去覆盖其中的类，就不用写空实现。**

**注意：监听过多，会抛tooManyListener例外。**

**12.06**

**Module 10**

**Canvas组件：画布，可以实现动画操作。**

**TextArea：文本域。**

**在单行文本域中回车会激发ActionEvent。**

**用CheckBoxGroup实现单选框功能。**

**Java中，单选框和复选框都是使用CheckBox实现。**

**菜单：new MenuBar()，MenuBar表示菜单条。**

**菜单中的每一项为MenuItem，一般级联菜单不应该超过三级。**

**练习：**

**设计一个计算器：注意设置一个boolean值（append）来判断输入数字是位于第一个数的后面还是属于输入的第二个数。**

**设置一个变量来存放“+”，点完运算符后，将append设置为false。**

**String number1**

**Char operator 存放运算符。**

**Module 11 Swing**

**AWT是Java最早出现的图形界面，但很快就被Swing所取代。**

**Swing才是一种真正的图形开发。**

**AWT在不同平台所出现的界面可能有所不同：因为每个OS都有自己的UI组件库，java调用不同系统的UI。**

**注意AWT为重量级组件，相当消耗资源，且不同系统的组件可能不同。因为这个问题使得AWT开发的软件难以作到跨平台。  
更为要命的是：不同OS的组件库都存在BUG。必须多种平台进行测试，并且AWT的组件库并不丰富。**

**为解决以上问题，SUN和IBM以及NETSCAPE联合开发出JAVA基础类包Swing：注意JAVA的基础类以Swing为核心。**

**注意引用：javax.swing.\*;javax表示JAVA的扩展。**

**我们在学习JDBC的时候会过度到J2EE。**

**在Swing的组件中，基本上都是在AWT组件的名称前面加“J”。**

**一般情况下，除了Choise等组件:**

**import javax.swing.\*;好要加上：import java.awt.\*以及import java.awt.event.\*。**

**Swing与AWT的最大区别是Swing为JAVA自身的组件。已经不是对等实体，与底层的OS无关。**

**（JBUILDER就是使用Swing写的）**

**Swing与AWT在事件模型处理上是一致的。**

**Jframe实际上是一堆窗体的叠加。**

**Swing比AWT更加复杂且灵活。**

**在JDK1.4中，给JFRAME添加Button不可用jf.add(b)。而是使用jf.getContentPane().add(b)。**

**content是先申请面板。不过在JDK1.5中可以使用add.。**

**Jpanel支持双缓冲技术。**

**在Jbutton中可以添加图标。**

**JscrollPane可以管理比屏幕还要大的组件。**

**TextArea只有装入JscrollPane中才能实现滚动条。**

**JeditorPane用于显示浏览器。**

**注意：Tabbed Panel与Border的比较。**

**进度条：ProgressBar。**

**JcomboBox：下拉菜单：在AWT中同类组件是choice。**

**JlistPanel：选择列表**

**BorderPanel：设置边框**

**JsplitPanel：可将容器分为两个部分，其中一个部分有Jtree。**

**TextBox：也是一种新的容器，可以设置组件的间距。**

**TextFileChoose：文件选择器。**

**ColorChoose：颜色选择器**

**Module 12 Applet**

**Applet为Panel的子类**

**Applet是java的自动执行方式（这是它的优势，主要用于HTML）。**

**工作四种语法：init()，start()，stop()，destory()。**

**Swing中有一个Japplet，如使用Swing组件。**

**Applet消亡的原因：**

**①java为安全起见对Applet有所限制：Applet不允许访问本地文件信息、敏感信息，不能执行本地指令（比如FORMAT），不能访问初原服务器之外的其他服务器。**

1. **IE不支持新版本的Applet。**

**Applet的优势：**

**网络传输，自动下载。**

**Application的优势：没有执行限制。**

**WebStart：可在网络传输，并且在本地无限制。因此前景光明。**

**练习：**

**使用Swing实现一个界面，分为上下两个部分，南边为JtextField组件，可编辑，上面为JtextArea组件，不可编辑，在JtextField组件输入字符，按回车，就可以将内容输入到JtextArea组件。(AREA区域可以滚动)**

**12.07**

**多线程**

**进程：任务**

**任务并发执行是一个宏观概念，微观上是串行的。**

**进程的调度是有OS负责的（有的系统为独占式，有的系统为共享式，根据重要性，进程有优先级）。**

**由OS将时间分为若干个时间片。**

**JAVA在语言级支持多线程。**

**分配时间的仍然是OS。**

**参看P377**

**线程由两种实现方式：**

**第一种方式：**

**class MyThread extends Thread{**

**public void run(){**

**需要进行执行的代码，如循环。**

**}**

**}**

**public class TestThread{**

**main(){**

**Thread t1=new Mythread();**

**T1.start();**

**}**

**}**

**只有等到所有的线程全部结束之后，进程才退出。**

**第二种方式：**

**Class MyThread implements Runnable{**

**Public void run(){**

**Runnable target=new MyThread();**

**Thread t3=new Thread(target);**

**Thread.start();//启动线程**

**}**

**}**

**P384:通过接口实现继承**

**练习：写两个线程：**

1. **输入200个“###”②输入200个“\*\*\*”**

**下面为线程中的7中非常重要的状态：（有的书上也只有认为前五种状态：而将“锁池”和“等待队列”都看成是“阻塞”状态的特殊情况：这种认识也是正确的，但是将“锁池”和“等待队列”单独分离出来有利于对程序的理解）**

**① ⑴**

**② ⑵**

**③ ⑶ run()结束**

**Start()**

**OS分配CPU**

**CPU时间片结束**

**yield() o.wait()**

**等待锁标记**

**notify()**

**注意：图中标记依次为**

**①输入完毕；②wake up③t1退出**

**⑴如等待输入（输入设备进行处理，而CUP不处理），则放入阻塞，直到输入完毕。**

**⑵线程休眠sleep（）**

**⑶t1.join()指停止main()，然后在某段时间内将t1加入运行队列，直到t1退出，main()才结束。**

**特别注意：①②③与⑴⑵⑶是一一对应的。**

**进程的休眠：Thread sleep(1000);//括号中以毫秒为单位**

**当main()运行完毕，即使在结束时时间片还没有用完，CPU也放弃此时间片，继续运行其他程序。**

**Try{Thread.sleep(1000);}**

**Catch(Exception e){e.printStackTrace(e);}**

**T1.join()表示运行线程放弃执行权，进入阻塞状态。**

**当t1结束时，main()可以重新进入运行状态。**

**T1.join实际上是把并发的线程编程并行运行。**

**线程的优先级：1-10，越大优先级越高，优先级越高被OS选中的可能性就越大。（不建议使用，因为不同操作系统的优先级并不相同，使得程序不具备跨平台性，这种优先级只是粗略地划分）。**

**注：程序的跨平台性：除了能够运行，还必须保证运行的结果。**

**一个使用yield()就马上交出执行权，回到可运行状态，等待OS的再次调用。**

**下午：**

**程序员需要关注的线程同步和互斥的问题。**

**多线程的并发一般不是程序员决定，而是由容器决定。**

**多线程出现故障的原因：**

**两个线程同时访问一个数据资源（临界资源），形成数据发生不一致和不完整。**

**数据的不一致往往是因为一个线程中的两个关联的操作只完成了一步。**

**避免以上的问题可采用对数据进行加锁的方法**

**每个对象除了属性和方法，都有一个monitor（互斥锁标记），用来将这个对象交给一个线程，只有拿到monitor的线程才能够访问这个对象。**

**Synchronized:这个修饰词可以用来修饰方法和代码块**

**Object obj;**

**Obj.setValue(123);**

**Synchronized用来修饰方法，表示当某个线程调用这个方法之后，其他的事件不能再调用这个方法。只有拿到obj标记的线程才能够执行代码块。**

**注意：Synchronized一定使用在一个方法中。**

**锁标记是对象的概念，加锁是对对象加锁，目的是在线程之间进行协调。**

**当用Synchronized修饰某个方法的时候，表示该方法都对当前对象加锁。**

**给方法加Synchronized和用Synchronized修饰对象的效果是一致的。**

**一个线程可以拿到多个锁标记，一个对象最多只能将monitor给一个线程。**

**Synchronized是以牺牲程序运行的效率为代价的，因此应该尽量控制互斥代码块的范围。**

**方法的Synchronized特性本身不会被继承，只能覆盖。**

**线程因为未拿到锁标记而发生的阻塞不同于前面五个基本状态中的阻塞，称为锁池。**

**每个对象都有自己的一个锁池的空间，用于放置等待运行的线程。**

**这些线程中哪个线程拿到锁标记由系统决定。**

**锁标记如果过多，就会出现线程等待其他线程释放锁标记，而又都不释放自己的锁标记供其他线程运行的状况。就是死锁。**

**死锁的问题通过线程间的通信的方式进行解决。**

**线程间通信机制实际上也就是协调机制。**

**线程间通信使用的空间称之为对象的等待队列，则个队列也是属于对象的空间的。**

**Object类中又一个wait()，在运行状态中，线程调用wait()，此时表示着线程将释放自己所有的锁标记，同时进入这个对象的等待队列。**

**等待队列的状态也是阻塞状态，只不过线程释放自己的锁标记。**

**Notify()**

**如果一个线程调用对象的notify()，就是通知对象等待队列的一个线程出列。进入锁池。如果使用notifyall()则通知等待队列中所有的线程出列。**

**注意：只能对加锁的资源进行wait()和notify()。**

**释放锁标记只有在Synchronized代码结束或者调用wait()。**

**注意锁标记是自己不会自动释放，必须有通知。**

**注意在程序中判定一个条件是否成立时要注意使用WHILE要比使用IF要严密。**

**WHILE会放置程序饶过判断条件而造成越界。**

**补充知识：**

**suspend（）是将一个运行时状态进入阻塞状态（注意不释放锁标记）。恢复状态的时候用resume()。Stop()指释放全部。**

**这几个方法上都有Deprecated标志，说明这个方法不推荐使用。**

**一般来说，主方法main()结束的时候线程结束，可是也可能出现需要中断线程的情况。对于多线程一般每个线程都是一个循环，如果中断线程我们必须想办法使其退出。**

**如果主方法main()想结束阻塞中的线程（比如sleep或wait）**

**那么我们可以从其他进程对线程对象调用interrupt()。用于对阻塞（或锁池）会抛出例外Interrupted Exception。**

**这个例外会使线程中断并执行catch中代码。**

**多线程中的重点：实现多线程的两种方式，Synchronized,以及生产者和消费者问题（ProducerConsumer.java文件）。**

**练习：**

1. **存车位的停开车的次序输出问题；**
2. **写两个线程，一个线程打印1-52，另一个线程答应字母A-Z。打印顺序为12A34B56C……5152Z。通过使用线程之间的通信协调关系。**

**注：分别给两个对象构造一个对象o，数字每打印两个或字母每打印一个就执行o.wait()。在o.wait()之前不要忘了写o.notify()。**

**补充说明：通过Synchronized，可知Vector较ArrayList方法的区别就是Vector所有的方法都有Synchronized。所以Vector更为安全。**

**同样：Hashtable较HashMap也是如此。**

**12.08**

**Module 10：I/O流（java如何实现与外界数据的交流）**

**Input/Output：指跨越出了JVM的边界，与外界数据的源头或者目标数据源进行数据交换。**

**输出**

数据源

JMV

**输入**

**注意：输入/输出是针对JVM而言。**

**File类（java.io.\*）可表示一个文件，也有可能是一个目录（在JAVA中文件和目录都属于这个类中，而且区分不是非常的明显）。**

**Java.io下的方法是对磁盘上的文件进行磁盘操作，但是无法读取文件的内容。**

**注意：创建一个文件对象和创建一个文件在JAVA中是两个不同的概念。前者是在虚拟机中创建了一个文件，但却并没有将它真正地创建到OS的文件系统中，随着虚拟机的关闭，这个创建的对象也就消失了。而创建一个文件才是在系统中真正地建立一个文件。**

**例如：File f=new File(“11.txt”);//创建一个名为11.txt的文件对象**

**f.CreateNewFile(); //真正地创建文件**

**f.CreateMkdir()：创建目录**

**f.delete()；删除文件**

**f.deleteOnExit();在进程退出的时候删除文件，这样的操作通常用在临时文件的删除。**

**对于命令：File f2=new file(“d:\\abc\\789\\1.txt”)**

**这个命令不具备跨平台性，因为不同的OS的文件系统很不相同。**

**如果想要跨平台，在file类下有separtor()，返回锁出平台的文件分隔符。**

**File.fdir=new File(File.separator);**

**String str=”abc”+File.separator+”789”;**

**使用文件下的方法的时候一定注意是否具备跨平台性。**

**List()：显示文件的名（相对路径）**

**ListFiles()：返回Files类型数组，可以用getName()来访问到文件名。**

**使用isDirectory()和isFile()来判断究竟是文件还是目录。**

**练习：**

**写一个javaTest程序，列出所有目录下的\*.java文件，把子目录下的JAVA文件也打印出来。**

**使用I/O流访问file中的内容。**

**JVM与外界通过数据通道进行数据交换。**

**分类：**

**按流分为输入流和输出流；**

**按传输单位分为字节流和字符流；**

**还可以分为节点流和过滤流。**

**节点流：负责数据源和程序之间建立连接；**

**过滤流：用于给节点增加功能。**

**过滤流的构造方式是以其他流位参数构造（这样的设计模式称为装饰模式）。**

**字节输入流：io包中的InputStream为所有字节输入流的父类。**

**Int read();读入一个字节（每次一个）；**

**可先使用new byte[]=数组，调用read(byte[] b)**

**read (byte[])返回值可以表示有效数；read (byte[])返回值为-1表示结束。**

**字节输出流：io包中的OutputStream位所有字节输入流的父类。**

**Write和输入流中的read相对应。**

**在流中close()方法由程序员控制。因为输入输出流已经超越了VM的边界，所以有时可能无法回收资源。**

**原则：凡是跨出虚拟机边界的资源都要求程序员自己关闭，不要指望垃圾回收。**

**以Stream结尾的类都是字节流。**

**如果构造FileOutputStream的同时磁盘会建立一个文件。如果创建的文件与磁盘上已有的文件名重名，就会发生覆盖。**

**用FileOutputStream中的boolean，则视，添加情况，将数据覆盖重名文件还是将输入内容放在文件的后面。（编写程序验证）**

**DataOutputStream:输入数据的类型。**

**因为每中数据类型的不同，所以可能会输出错误。**

**所有对于：DataOutputStream**

**DataInputStream**

**两者的输入顺序必须一致。**

**过滤流：**

**bufferedOutputStream**

**bufferedInputStream**

**用于给节点流增加一个缓冲的功能。  
在VM的内部建立一个缓冲区，数据先写入缓冲区，等到缓冲区的数据满了之后再一次性写出，效率很高。**

**使用带缓冲区的输入输出流的速度会大幅提高，缓冲区越大，效率越高。（这是典型的牺牲空间换时间）**

**切记：使用带缓冲区的流，如果数据数据输入完毕，使用flush方法将缓冲区中的内容一次性写入到外部数据源。用close()也可以达到相同的效果，因为每次close都会使用flush。一定要注意关闭外部的过滤流。**

**（非重点）管道流：也是一种节点流，用于给两个线程交换数据。**

**PipedOutputStream**

**PipedInputStream**

**输出流：connect(输入流)**

**RondomAccessFile类允许随机访问文件**

**GetFilepoint()可以知道文件中的指针位置，使用seek()定位。**

**Mode(“r”:随机读；”w”：随机写；”rw”：随机读写)**

**练习：写一个类A，JAVA A file1 file2**

**file1要求是系统中已经存在的文件。File2是还没有存在的文件。**

**执行完这个命令，那么file2就是file1中的内容。**

**字符流：reader\write只能输纯文本文件。**

**FileReader类：字符文件的输出**

**字节流与字符流的区别：**

**字节流的字符编码：**

**字符编码把字符转换成数字存储到计算机中，按ASCii将字母映射为整数。**

**把数字从计算机转换成相应的字符的过程称为解码。**

**编码方式的分类：**

**ASCII（数字、英文）:1个字符占一个字节（所有的编码集都兼容ASCII）**

**ISO8859-1（欧洲）：1个字符占一个字节**

**GB-2312/GBK：1个字符占两个字节**

**Unicode: 1个字符占两个字节（网络传输速度慢）**

**UTF-8：变长字节，对于英文一个字节，对于汉字两个或三个字节。**

**原则：保证编解码方式的统一，才能不至于出现错误。**

**Io包的InputStreamread称为从字节流到字符流的桥转换类。这个类可以设定字符转换方式。**

**OutputStreamred:字符到字节**

**Bufferread有readline()使得字符输入更加方便。**

**在I/O流中，所有输入方法都是阻塞方法。**

**Bufferwrite给输出字符加缓冲，因为它的方法很少，所以使用父类printwrite，它可以使用字节流对象，而且方法很多。**

**练习：做一个记事本**

**swing/JfileChoose: getSelect file()**

**InputStreeamReader：把字节变为字符**

**JAVA中对字符串长无限制 bufferedReader（ir）**

**12.09**

**class ObjectOutputStream也是过滤流，使节点流直接获得输出对象。**

**最有用的方法：WriteObject(Object b)**

**用流传输对象称为对象的序列化，但并不使所有的对象都可以进行序列化的。只有在实现类时必须实现一个接口：IO包下的Serializable(可序列化的)。此接口没有任何的方法，这样的接口称为标记接口。**

**Class Student implements Serializable**

**把对象通过流序列化到某一个持久性介质称为对象的可持久化。**

**Hibernate就是研究对象的可持久化。**

**ObuectInputStream in =new ObjectInputStream;**

**Object o1=in.readObuect();**

**Student s1=(Student)o1；**

**注意：因为o1是一个对象，因为需要对其进行保存。**

**Transient用来修饰属性。**

**Transient int num;**

**表示当我们对属性序列化时忽略这个属性（即忽略不使之持久化）。**

**所有属性必须都是可序列化的，特别是当有些属性本身也是对象的时候，要尤其注意这一点。**

**判断是否一个属性或对象可序列化：Serialver。**

**Serialver TestObject（TestObject必须为已经编译）**

**执行结果：如果不可序列化；则出现不可序列化的提示。如果可以序列化，那么就会出现序列化的ID：UID。**

**java.until.\*有**

**StringTokenizer（参数1，参数2）按某种符号隔开文件**

**StringTokenizer(s,”:”) 用“：”隔开字符，s为对象。**

**练习：将一个类序列化到文件，然后读出。下午：**

1. **网络基础知识**
2. **JAVA网络编程**

**网络与分布式集群系统的区别：每个节点都是一台计算机，而不是各种计算机内部的功能设备。**

**Ip:具有全球唯一性，相对于internet，IP为逻辑地址。**

**端口(port)：一台PC中可以有65536个端口，进程通过端口交换数据。连线的时候需要输入IP也需要输入端口信息。**

**计算机通信实际上的主机之间的进程通信，进程的通信就需要在端口进行联系。**

**192.168.0.23:21**

**协议：为了进行网络中的数据交换（通信）而建立的规则、标准或约定。**

**不同层的协议是不同的。**

**网络层：寻址、路由（指如何到达地址的过程）**

**传输层：端口连接**

**TCP模型：应用层/传输层/网络层/网络接口**

**端口是一种抽象的软件结构，与协议相关：TCP23端口和UDT23端口为两个不同的概念。**

**端口应该用1024以上的端口，以下的端口都已经设定功能。**

**套接字(socket)的引入：**

**Ip+Port=Socket（这是个对象的概念。）**

**Socket为传输层概念，而JSP是对应用层编程。例：**

**java.net.\*;**

**(Server端定义顺序)**

**ServerSocket(intport)**

**Socket.accept()；//阻塞方法，当客户端发出请求是就恢复**

**如果客户端收到请求：**

**则Socket SI=ss.accept()；**

**注意客户端和服务器的Socket为两个不同的socket。**

**Socket的两个方法：**

**getInputStream()：客户端用**

**getOutputStream() 服务器端用**

**使用完毕后切记Socket.close()，两个Socket都关，而且不用关内部的流。**

**在client端，Socket s=new Socket(“127.0.0.1”,8000);**

**127.0.0.1为一个默认本机的地址。**

**练习：**

1. **客户端向服务器发出一个字符串，服务器转换成大写传回客户端。**

**大写的函数：String.toUpperCase()**

1. **服务器告诉客户端：“自开机以来你是第n 个用户”。**

**12.12**

**UDP编程：**

**DatagramSocket（邮递员）：对应数据报的Socket概念，不需要创建两个socket，不可使用输入输出流。**

**DatagramPacket（信件）：数据包，是UDP下进行传输数据的单位，数据存放在字节数组中。**

**UDP也需要现有Server端，然后再有Client端。**

**两端都是DatagramPacket（相当于电话的概念），需要NEW两个DatagramPacket。**

**InetAddress:网址**

**这种信息传输方式相当于传真，信息打包，在接受端准备纸。**

**模式：**

**发送端：Server:**

**DatagramPacket inDataPacket=new DatagramPacket ((msg,msg.length); InetAdress.getByName(ip),port);**

**接收端：**

**clientAddress=inDataPack.getAddress();//取得地址**

**clientPort=inDataPack.getPort();//取得端口号**

**datagramSocket.send; //Server**

**datagramSocket.accept; //Client**

[**URL:在应用层的编程**](file:///C:\Users\Administrator\Downloads\在应用层的编程)

**注意比较：**

[**http://Localhost:8080/directory**](http://Localhost:8080/directory) **//查找网络服务器的目录**

**file://directory //查找本地的文件系统**

**java的开发主要以http为基础。**

**反射：主要用于工具和框架的开发。**

**反射是对于类的再抽象；通过字符串来抽象类。**

**JAVA类的运行：classLoader:加载到虚拟机（vm）**

**Vm中只能存储对象（动态运行时的概念），.class文件加载到VM上就成为一个对象，同时初始静态成员及静态代码（只执行一次）。**

**Lang包下有一个类为Class：在反射中使用。此类中的每个对象为VM中的类对象，每个类都对应类类的一个对象（class.class）。**

**例：对于一个Object类，用getClass()得到其类的对象，获得类的对象就相当于获得类的信息，可以调用其下的所有方法，包括类的私有方法。**

**注意：在反射中没有简单数据类型，所有的编译时类型都是对象。**

**反射把编译时应该解决的问题留到了运行时。**