SCJP 认证考试指南

拥有 SCJP，职业开端与众不同

SCJP － Sun Certified Java Programmer (Sun Java 程序员认证)

Sun 公司作为 Java 语言的发明者，对全球的 Java 开发人员进行技术水平认证。该认证在国际上获得了 IT 公司的普遍认可，是目前国际上最热门的 IT 认证之一。《认证杂志》（Certificate Magazine）的权威调查结果表明：

- 持有 SCJP 认证者能够迅速获得面试机会
- 持有 SCJP 认证者的平均资薪比持有其他认证的开发人员高 21.7％
- 持有 SCJP 认证者在公司更容易获得晋升的机会

第 1 章 声明和访问控制

目标一 创建数组

数组

Java 中的数组跟 C/C++这些语言中的数组的语法结构很相似。但是，Java 去掉了 C/C++中的可以通过[]或者使用指针来访问元素的功能。这种在 C/C++中被普遍接受的功能虽然强大，但是也让 Bug 横行的软件更容易出现。因为 Java 不支持这种直接通过指针来操纵数据，这类的 Bug 也被消除了。

数组是一类包含被称为元素的值的对象。这就为你在程序中移动或保存一组数据以很方便的支持，并且允许你根据需要访问和改变这些值。用一个小例子来说：你可以创建一个 String 类型的数组，每一个都包含一个运动队队员名字。数组可以传送给一个需要访问每个队员名字的方法。如果一个新队员加入，其中一个老队员的名字可以被修改成新队员的名字。这就显得比 player1、player2、player3 等等很随意的不相关的变量方便很多。跟变量通过变量名来访问不同的是，元素通过从 0 开始的数字来访问。因此，你可以一个个的访问数组的每个元素。

数组跟对象很相似，它们都是用 new 关键字来创建，并且有属于主要父对象类的方法。数组可能存储简单类型或者对象的引用。

数组的每个元素必须是同一类型的。元素的类型在数组被声明时确定。如果你需要存储不同类型元素的方式，你可以选择 collection 类，collection 类是 Java2 考试中的新增的考点，我们将会在第十部分讨论它。你可以用数组来存储对象的句柄，你能像使用其它任意对象引用一样访问，摘录或者使用它。

声明但不分配空间

声明一个数组不需分配任何存储空间，它仅仅是代表你试图创建一个数组。跟 C/C++声明一个数组的明显区别就是空间的大小没有被特别标识。因此，下面的声明将会引起一个编译期错误。

```
int num[5];
```

一个数组的大小将在数组使用 new 关键字真正创建时被给定，例如：

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

你可以认为命令 new 的使用跟初始化一个类的实例的使用是类似的。例子中数组名 num 说明数组大小可以是任意大小的整形数据。

同时声明和创建数组

这个例子也可以使用一行语句完成：

int num[] = new int[5];
方括号也可以放在数据类型后面或者数组名后面。下面的两种都是合法的：
int[] num;
int num[];
你可以读作：
一个名字为 num 的整型数组
一个数据类型为整型名字为 num 的数组

Java 和 C/C++数组的比较

Java 数组知道它的大小，并且 Java 语言支持对意外的移动到数组末端的保护。

如果你从 Visual Basic 背景下转到 Java 开发，并且还不习惯于一直从 0 开始计数，这点是很方便的。这也可以帮你避免一些在 C/C++程序中很难发现的错误，例如移动到了数组末端并且指向了任意内存地址。

例如，下面的程序会引起一个 ArrayIndexOutOfBoundsException 异常。

```
int[] num= new int[5];
for(int i =0; i<6; i++){
        num[i]=i*2;
        }
```

访问一个 Java 数组的标准习惯用法是使用数组的 length 成员
例如：

```
int[] num= new int[5];
for(int i =0; i<num.length; i++){
        num[i]=i*2;
}
```

数组知道它的大小

假如你跳过了 C/C++的对照，Java 中的数组总是知道它们的大小，这表现在 length 字段。因此，你可以通过下面的语句动态移动数组：

```
int myarray[]=new int[10];
for(int j=0; j<myarray.length;j++){
    myarray[j]=j;
}
```

注意，数组有 length 字段，而不是 length（）方法。当你开始用一组字符串的时候，你会像 s.length()这样使用字符串的 length 方法。
数组中的 length 是域（或者说特性）而不是方法。

Java 数组和 Visual Basic 数组的对照

Java 中的数组总是从 0 开始。如果使用了 Option base 声明，Visual Basic 可能从 1 开始。

Java 中没有跟 Visual Basic 中可以使你不删除内容就改变数组大小的 redim preserve 命令等价的语句。但你可以建立一个同样大小的新数组，并且复制现有元素到里面。

一个数组声明可以有多个方括号。Java 形式上不支持多维数组，但是它可以支持数组的数组，就是我们常说的嵌套数组。

C/C++中那样的多维数组和嵌套数组的最主要区别就是，每个数组不需要有同样的长度。如果你将一个数字当作一个矩阵，矩阵不一定是矩形。按照 Java 语言规范：(http://java.sun.com/docs/books/jls/html/10.doc.html#27805)

“括号里的数指明了数组嵌套的深度”

在其他语言中，就要跟数组的维度相符。因此，你可以建立一个类似于下面的形式的二维数组：

```
int i[][];
```

第一个维度可以匹配 X，第二个维度可以匹配 Y。

声明和初始化相结合

一个数组可以通过一个语句来创建并初始化，这就代替了通过数组循环来初始化的方式。这种方法很适合小数组。下面的语句创建了一个整型数组并且赋值为 0 到 4：

```
int k[]=new int[] {0,1,2,3,4};
```

注意，你没有必要确定数组元素的数量。你可能在测验中被问到下面的语句是不是正确的问题：

```
int k=new int[5] {0,1,2,3,4} //Wrong, will not compile!
```

你可以创建数组的同时确定任何数据类型，因此，你可以创建一个类似于下面形式的字符串数组：

```
String s[]=new String[] {"Zero","One","Two","Three","Four"};
System.out.println(s[0]);
```

这句将会输出 String[0]。

数组的默认值

不同于其他语言中的变量在类级别创建和本地方法级别创建有不同的动作，Java 数组总是被设定为默认值。

无论数组是否被创建了，数组中的元素总是设为默认值。因此，整型的数组总是被置 0，布尔值总是被置 false。下面的代码编译时不会出错，并且输出 0。

```
public class ArrayInit{
    public static void main(String argv[]){
        int[] ai = new int[10];
        System.out.println(ai[0]);
    }
}
```

问题

问题 1）怎样通过一个语句改变数组大小同时保持原值不变？

1) Use the setSize method of the Array class

2) Use Util.setSize(int iNewSize)

3) use the size() operator

4) None of the above

问题 2） 你想用下面的代码查找数组最后一个元素的值，当你编译并运行它的时候，会发生什么？

```
public class MyAr{
public static void main(String argv[]){
            int[] i = new int[5];
            System.out.println(i[5]);
            }
}
```

1) Compilation and output of 0

2) Compilation and output of null

3) Compilation and runtime Exception

4) Compile time error

问题 3）作为一个好的 Java 程序员，你已忘记了曾经在 C/C++中知道的关于数组大小信息的知识。如果你想遍历一个数组并停止在最后一个元素处。你会使用下面的哪一个？

1)myarray.length();

2)myarray.length;

3)myarray.size

4)myarray.size();

问题 4）你的老板为了你写出了 HelloWorld 而很高兴地为你升职了，现在她给你分配了一个新任务，去做一个踢踏舞游戏（或者我小时候玩的曲棍球游戏）。你认为你需要一个多维数组，下面哪一个能做这个工作？

1) int i =new int[3][3];

2) int[] i =new int[3][3];

3) int[][] i =new int[3][3];

4) int i[3][3]=new int[][];

问题 5）

你希望找到一个更优雅的方式给你的数组赋值而不使用 for 循环语句，下面的哪一个能做到？

1)

```
myArray{
    [1]="One";
    [2]="Two";
    [3]="Three";
}
```

2)String s[5]=new String[] {"Zero","One","Two","Three","Four"};
3)String s[]=new String[] {"Zero","One","Two","Three","Four"};
4)String s[]=new String[]={"Zero","One","Two","Three","Four"};

问题 6）当你试着编译运行下面的代码的时候，可能会发生什么？

```
public class Ardec{
    public static void main(String argv[]){
    Ardec ad = new Ardec();
    ad.amethod();
    }
    public void amethod(){
    int ia1[]= {1,2,3};
    int[] ia2 =   {1,2,3};
    int   ia3[] = new int[] {1,2,3};
    System.out.print(ia3.length);
    }
}
```

1) Compile time error, ia3 is not created correctly
2) Compile time error, arrays do not have a length field
3) Compilation but no output
4) Compilation and output of 3

答案

答案 1）
4) None of the above
你不能改变一个数组的大小。你需要创建一个不同大小的临时数组，然后将原数组中的内容放进去。Java 支持能够改变大小的类的容器，例如 Vector 或者 collection 类的一个成员。
答案 2）
3) Compilation and runtime Exception
当你试着移动到数组的末端的时候，你会得到一个运行时错误。因为数组从 0 开始索引，并且最后一个元素是 i[4]而不是 i[5]。
答案 3）
2) myarray.length;
答案 4）
3) int[][] i=new int[3][3];
答案 5）
3)String s[]=new String[] {"Zero","One","Two","Three","Four"};
答案 6）
4) Compilation and output of 3
所有的数组的声明都是正确的。如果你觉得不太可能，可以自己编译这段代码。

目标二 定义类和变量

定义类，内部类，方法，实例变量，静态变量和自动（本地方法）变量，需要合适的选用允许的修饰词。（例如 public，final，static，abstract 诸如此类）。这些修饰词或者单独使用或者联合使用，定义了包的关系。

本目标需要注意的

我发现目标中用了“诸如此类”，这让我有些烦恼，我想你需要弄明白下面词的意思：
native
transient
synchronized
volatile

什么是类？

一个类的的定义把它很生硬描述为“方法和数据的集合”。它把面向对象编程出来之前的编程思想结合起来，这对理解该概念很有帮助。在类和面向对象程序设计前的主要概念是结构化程序设计。结构化程序设计的理念是程序员将复杂问题划分为小块的代码，一般称为函数或子程序。这符合“做一件很大很复杂的事情的好办法是把它分成一系列比较小但更容易管理的问题”的理念。

尽管结构化程序设计在管理复杂性方面很有用，但它不能容易的解决代码重用问题。程序员发现他们总是“重复发明”轮子。在试着对现实物理对象的思考中，程序设计方面的思想家找到了面向对象的理念（有时被称为 OO）。

举例来说，一个计算机厂商准备生产一种新型个人电脑，如果计算机厂商使用类似于程序设计的方式的话，就要求他建立新团队来设计新 CPU 芯片，新声卡，没准还需要另一个团队设计规划制造新的主板。事实上，这根本不可能出现。由于电脑组件接口的标准化，计算机厂商只需要联系配件供应商，并商议好他们要生产的新型号的说明书就行了。注意组件接口标准化的重要性。

比较 C++/VB 和 Java 的类

因为 Java 被设计成容易让 C++程序员学习的语言，因此两种语言在处理类上有很多相似的地方。C++和 Java 都有继承，多态和数据隐藏特性，并使用显式的修饰词。有一些不同也是因为使 Java 更容易学习和使用。

C++语言实现了多态继承，这样，一个类就可以比一个的父类（或基类）更强大。Java 只允许单继承，这样就只有一个父类。为了克服这个限制，Java 有一个被称作接口的特性。Java 语言的设计者确定接口能够提供多态继承的好处而没有坏处。所有 Java 类都是 Object 类的后代。

对象在 Visual Basic 中是语言设计之后才加入的想法。Visual Basic 有时被称作基于对象的语言而不是面向对象的语言。这就好像是语言的设计者认为类很酷，然后随着 VB4 的发布，他们决定加入一个新类型的模块，称它为类并且加上冒号，让它看起来更像 C++。VB 的类概念中失去了至关重要的元素：继承。微软在 VB5 中加入了跟 Java 的接口很相似的接口的概念。VB 类和 Java 类的最主要相似之处是引用的使用和 new 关键字。

Java 中类的角色

类是 Java 的心脏，所有的 Java 代码都在一个类里。Java 里没有自由独立代码的概念，甚至最简单的 HelloWorld 应用都是包含在类里被创建的。为了指出一个类是另一个类的派生类，我们使用 extend 关键字。如果 extend 关键字没有被使用，这个类将是基类 Object 派生的。这可以使它有一些基本的功能，比如打印自己的名字和其他一些在线程中可能需要用到的功能。

类的最简单特性

定义一个类至少需要 class 关键字，类名和一对花括号。如：

```
class classname {}
```

如果不是有特别作用的类，它在语法上是正确的（我很惊讶的发现，当我举例说明继承时，我定义了一个跟着类似的类）。

通常，一个类还会包括一个访问修饰符，放在关键字 class 前面，还会有程序体放在花括号之间。下面的是一个更好的类模版：

```
public class classname{
    //Class body goes here
}
```

创建一个简单的 HelloWorld 类

这里有一个简单的 HelloWorld 程序，它将会向控制台输出“hello world”。

```
public class HelloWorld{
public static void main(String argv[]){
        System.out.println("Hello world");
        }
}//End class definition
```

关键字 public 是一个可见的修饰符，指明了这个类对于其他类来说都是可见的。一个文件只有一个外部类可以声明为 public。内部类将会隐藏在任意位置。如果你在一个文件中定义了多于一个的 public 类，将会发生一个编译期错误。注意，Java 对每一部分都是很敏感的，包含这个类的文件名字必须是 HelloWorld.java。当然，这跟微软平台虽然保护但是却忽略文件的大小写有些差别。

关键字 class 指明了一个将被定义的类，并且类名是 HelloWorld。左花括号表明类的开

始。注意，类结束的右花括号后面没有分号。注释语句
//End class definition
使用了 C/C++中同样允许的单行类型。Java 也能够识别/**/的注释模式。

创建一个类的实例

上面描述的 HelloWorld 应用例子很浅显的告诉了你所能创建的最简单的应用，但是它漏掉了使用类时至关重要的元素，那就是关键字 new 的使用，new 指出了一个类的新实例的创建。在 HelloWorld 应用中，因为只有 System.out.println 这个唯一的 static 方法，并且不需要类使用 new 关键字创建，因此创建新实例不是必要的。static 方法只能访问 static 变量。可以稍微改进一下 HelloWorld 应用，下面举例说明一个类的新实例的创建。

```
public class HelloWorld2{
        public static void main(String argv[]){
                HelloWorld2 hw = new HelloWorld2();
                hw.amethod();
        }
        public void amethod(){
                System.out.println("Hello world");
        }
}
```

上面的代码通过这行代码创建了自己的一个新实例。
HelloWorld2 hw = new HelloWorld2();
这是使用类创建新实例的一个基本语法。注意类的名字怎样出现了两次。第一个指明了类的引用的数据类型。这需要它不能和 new 关键字所修饰真正的类的名字相同。这个类实例的名字是 hw。这仅仅是给变量选择的名字。这里有一个命名习惯，一个类的实例名以小写字母开头，而类的名字以大写字母开头。

创建方法

在上一个例子 HelloWorld2 中，一个 Java 中的方法跟 C/C++中的函数和 Visual Basic 中的子程序很相似。上例中名字为 amethod 的方法和本例中的 amethod 方法被声明为 public，这说明它可以在任何地方被访问。它有一个返回值 void，表明没有值返回。并且括号中也是空的，表明它没有参数。
同样的方法可以从下面几种方式之中选择：

```
private void amethod(String s)
private void amethod(int i, String s)
protected void amethod(int i)
```

这些例子说明了一些典型的方法签名。使用关键字 private 和 protected 说明它们将会在别处隐藏。
Java 方法和其他像 C 这样的非面向对象语言的方法的区别是 Java 方法属于类。这表明

它们通过点号指明代码属于哪个类的实例来调用。（static 方法是一个例外，但我们现在无需担心）

因此在 HelloWorld 中 amethod 通过下面的语句调用

```
HelloWorld hw = new HelloWorld();
hw.amethod();
```

在 HelloWorld 类中创建的其他实例中，方法被类的每个实例所调用。每个类的实例将能够访问它自己的变量。因此下面的代码将调用不同实例的 amethod 方法

```
HelloWorld hw = new HelloWorld();
HelloWorld hw2 = new HelloWorld();
hw.amethod();
hw2.amethod();
```

类的两个实例 hw 和 hw2 可能访问不同的变量。

自动局部变量

自动变量是方法变量。它们在方法代码开始运行时生效，并在方法结束时失效。因为它们只能在方法内可见，因此临时操作数据时比较有用。如果你希望一个值在方法被调用时保持，你需要将变量创建在类级别。

一个自动变量将“屏蔽”类级别的变量。

因此，下面的代码将打印 99 而不是 10

```
public class Shad{
public int iShad=10;
public static void main(String argv[]){
                        Shad s = new Shad();
                        s.amethod();
        }//End of main
        public void amethod(){
                        int iShad=99;
                        System.out.println(iShad);
        }//End of amethod
}
```

修饰语和封装

修饰符的可见性是 Java 封装机制的一部分。封装允许分离方法执行的接口。修饰符的可见性是 Java 封装机制至关重要的部分。封装允许分离方法执行的接口。带来的好处就是类内部的代码的细节可以被改变，同时不影响其他对象的使用。这是面向对象设计（最后不得不在某处使用这个词）的一个关键概念。

封装一般用找回或更新 private 类的变量值的方法的形式。这些方法一般是 accessor 或 mutator 方法。访问方法找回值而设置方法改变值。命名惯例是这些方法名类似于 setFOO 改变值，getFOO 得到值。注意，使用 set 和 get 来命名的方法比仅仅使程序员感到方便更重要，并且是 Javabean 系统的重要组成部分。不过我们的测试还没有涉及到 Javabean 的内容。

举一个例子，你有一个变量用来存储学生的年龄。你可能简单的用一个 public 的整型变量来存储。

```
int iAge;
```

接下来，当你的应用程序交付使用后，你可能会发现你的某些学生可能有超过 200 岁的记录，还有小于 0 岁的记录。你需要一段代码来检查错误条件。所以当你的程序改变年龄的值的时候，你用 if 语句来检查范围。

```
if(iAge > 70){
        //do something
        }
if (iAge <3){
        //do something
}
```

当你正在做这些的时候，你漏掉了一些使用过 iAge 变量的代码，所以你被召回了，因为你可能有一个 19 岁的学生，但是你的记录里却是 190 岁。

面向对象使用封装处理了这样的问题，就是创建一个访问包含年龄值的 private 域的方法，名字类似于 setAge 和 getAge。setAge 方法可能有一个整型的参数并且更新年龄的 private 值，getAge 方法没有参数但从 private 的年龄域返回值。

```
public void setAge(int iStudentAge){
        iAge = iStudentAge;
}
public int getAge(){
        return iAge;
}
```

开始，我们也许认为这么长的代码来做一小段代码就能完成的工作没有意义，但是，当这些方法能够满足你的需求时，可以帮你做更多的 iAge 域的确认工作，同时不会影响已经在使用这些信息的代码。

通过这样的代码执行处理方式，实际的程序代码行可以改变，而外面的部分（接口）保持不变。

Private（私有）

私有变量仅仅在创建它的类内部可见。这意味着它们在子类里不可见。这使变量除了当前类之外，绝缘于其他方法的修改。像是修饰语和封装里描述的，这对于将接口与接口实现分离开很有帮助。

```
class Base{
private int iEnc=10;
public void setEnc(int iEncVal){
      if(iEncVal < 1000){
          iEnc=iEncVal;
           }else
           System.out.println("Enc value must be less than 1000");
          //Or Perhaps thow an exception
         }//End if
```

```
}

public class Enc{
public static void main(String argv[]){
                        Base b = new Base();
                        b.setEnc(1001);
            }//End of main
}
```

public（共有）

public 修饰符可以应用于变量（域）或者类。它可能是你学习 Java 过程中最先接触的修饰符。想想 HelloWorld.Java 程序中被这样声明的类的代码

```
public class HelloWorld
```

这是因为 Java 虚拟机仅仅在一个声明为 public 的类中查找神奇的 main 启动方法。

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

一个 public 类有全局的作用范围，一个实例可以在程序内部或外部的任意位置创建。任何文件中只能有一个非内部类可以用 public 关键字定义。如果你用 public 关键字在一个文件中定义了超过一个非内部类，编译器将会报错。

使用 public 修饰符定义一个变量可以使它在任何位置适用。使用方法如下：

```
public int myint =10;
```

如果你希望创建一个可以在任何地方修改的变量，你可以将它声明为 public。你可以使用类似于调用方法那样的点号来访问它。

```
class Base {
     public int iNoEnc=77;
}
public class NoEnc{
public static void main(String argv[]){
                        Base b = new Base();
                        b.iNoEnc=2;
                        System.out.println(b.iNoEnc);
            }//End of main
}
```

注意，并不建议你对代码的接口和执行不加分隔的使用。如果你想改变 iNoEnc 的数据类型，你必须修改执行改变代码的每一部分。

protected（保护）

protected 有一点古怪。一个 protected 变量在类，子类和同一个包内部可见，但不是全部可见。限制就是它在包内部的可见性可能超过你的预期。在同一路径下的类都是被默认为在一个包内，因此，protected 类将会可见。这就意味着一个 protected 变量会比一个没有任何访问修饰符的变量更有可见性。

一个没有访问修饰符定义的变量称为它有默认的可见性。默认可见性是说一个变量可以

在类内部可见，而包内的其他类中均不可见，不在同一个包的子类内也不可见。

静态的（static）

虽然 static 可以起到可见性修饰符的作用，但它不是直接的可见性修饰符。static 修饰符可以应用于内部类，方法和变量。功能代码经常放在 static 方法中，例如 Math 类有完整的功能方法，如：random，sin 和 round。基本数据类型的包装类 Integer，Double 等等也有 static 方法处理包装过的基本数据类型，如返回符合字符串“2”的 int 值。

标记一个变量为 static 表明每个类只能有一个副本存在。这是与普通的情况相区别。一般情况下，一个类的每个实例都有一个整型变量的副本。在下面的非 static int 例子中，三个实例中的 int iMyVal 都有对应各自实例的不同值。

```
class MyClass{
        public int iMyVal=0;
}

public class NonStat{
public static void main(String argv[]){
        MyClass m1 = new MyClass();
        m1.iMyVal=1;
        MyClass m2 = new MyClass();
        m2.iMyVal=2;
        MyClass m3 = new MyClass();
        m3.iMyVal=99;
        //This will output 1 as each instance of the class
        //has its own copy of the value iMyVal
        System.out.println(m1.iMyVal);
        }//End of main
}
```

下面的例子说明了当你有包含 static 整型数的类的多个实例时会发生什么

```
class MyClass{
        public static    int iMyVal=0;
}
public class Stat{
public static void main(String argv[]){
            MyClass m1 = new MyClass();
            m1.iMyVal=0;
            MyClass m2 = new MyClass();
            m2.iMyVal=1;
            MyClass m3 = new MyClass();
            m2.iMyVal=99;
            //Because iMyVal is static, there is only one
            //copy of it no matter how many instances
            //of the class are created /This code    will
```

```
            //output a value of 99
            System.out.println(m1.iMyVal);
        }//End of main
}
```

你必须要忍受这样的事实，你不能在一个 static 方法内部访问一个非 static 变量。因此，下面的代码会引起一个编译时错误

```
public class St{
int i;
public static void main(String argv[]){
        i = i + 2;//Will cause compile time error
        }
}
```

一个 static 方法不能在一个子类中重写为非 static 方法

一个 static 方法不能在一个子类中重写为非 static 方法。同样，一个非 static（普通的）方法也不能在子类中重写为 static 方法。但是同样的规则对方法重载没有作用。下面的代码在它尝试重写类的方法为非 static 方法 amethod 时将会引起一个错误。

```
class Base{
        public static void amethod(){
        }
}
public class Grimley extends Base{
        public void amethod(){}//Causes a compile time error
}
```

IBM Jikes 编译器会产生下面的错误

```
Found 1 semantic error compiling "Grimley.java":

    6.          public void amethod(){}
                                <------->

*** Error: The instance method "void amethod();"
cannot override the static method "void amethod();"
declared in type "Base"
```

static 方法不能在子类中重写，但是可以被隐藏

在我的模拟测验中，我有一个问题问到 static 方法是否可以被重写，答案是不能，但是引来了大量的 email，很多人举例说明 static 方法被重写了。在子类中，重写过程包括的不仅仅是简单的替代一个方法。它还包括运行时决定哪个方法被调用取决于它的引用类型。

这里有一个例子的代码，看起来显示了一个 static 方法被重写了

```
class Base{
    public static void stamethod(){
    System.out.println("Base");
    }
}
public class ItsOver extends Base{
    public static void main(String argv[]){
```

```
    ItsOver so = new ItsOver();
    so.stamethod();
    }
    public static void stamethod(){
    System.out.println("amethod in StaOver");
    }
}
```
这段代码会被编译并且输出"amethod in StaOver"

本地的（native）

native 修饰符仅仅用来修饰方法，指明代码体不是用 Java 而是用 C 或 C++所写。native 方法经常为平台的特殊目的所写，例如访问某些 Java 虚拟机不支持的硬件。另一个原因是为了需要获得更好的性能。

一个 native 方法以一个分号结尾，而不是代码块。例如下面的代码将会调用一个可能用 C++所写的外部程序:

```
public native void fastcalc();
```

抽象（abstract）

粗略的看一下 abstract 修饰符显得很容易，但是也会漏掉它的一些隐含内容。属于主考者很喜欢问的那种狡猾的，关于那类修饰符的问题。

abstract 修饰符可以被用在类和方法上。当用在方法上时，表明方法会没有方法体（也就是没有花括号的部分），并且代码只能在子类执行时运行。但是，还有一些关于何时何处你能拥有 abstract 方法的限制和包含这类方法的类的规则。如果一个类有一个或多个 abstract 方法，或者继承了不准备运行的 abstract 方法，则它必须声明为 abstract。另外一个情况是，如果一个类实现了接口但是不准备运行接口的每个方法。但这种情况很少见。如果一个类有 abstract 方法，则它需要声明为 abstract 类不要认为一个 abstract 类不能有非 abstract 方法而感到心烦意乱。任何从 abstract 类继承而来的类都要实现基类的 abstract 方法，或者声明自身为 abstract 类。这些规则倾向于问你为什么想要创建 abstract 方法？

abstract 类对于类的设计者很有用。它使类的设计者能够创建应当被实现的方法的原型，但是真正的实现留给以后使用这个类的人。下面的例子是一个包含 abstract 方法的 abstract 类。再次注意，类必须被声明为 abstract，否则会出现编译时错误。

下面的类是 abstract 类，它会被正确编译并打印输出字符串

```
public abstract class abstr{
public static void main(String argv[]){
        System.out.println("hello in the abstract");
        }
        public abstract int amethod();
}
```

常量（final）

final 修饰符可以用在类，方法和变量上。它跟遗传关系的意思很相近，因此很容易记忆。一个 final 类可能从不被继承。另外一种想法是，一个 final 类不能作为父类。任何 final 类中的方法自动成为 final 方法。如果你不希望别的程序员“弄乱你的代码”，这是一个有效的方法。另一个好处就是效率，编译器对于一个 final 方法的工作很少。这些内容在 Core Java 的第一卷中有提及。

final 修饰符表明方法不能被重写。因此，如果你在子类中有一个同样签名的方法的话，你会得到一个编译时错误。

下面的例子说明对一个类使用 final 修饰符。这段代码将会打印字符串"amethod"

```
final class Base{
public void amethod(){
        System.out.println("amethod");
        }
}
public class Fin{
public static void main(String argv[]){
        Base b = new Base();
        b.amethod();
        }
}
```

一个 final 变量的值不能被改变，并且必须在一定的时刻赋值。这跟其他语言中的 constant 的思想比较相似。

同步的（Synchronized）

synchronized 关键字被用来保证不只有一个的线程在同一时刻访问同一个代码块。参看第七部分关于线程的内容来了解更多的关于它的运行的知识。

瞬时（Transient）

transient 修饰符是不常用的修饰符之一。它表明一个变量在序列化过程中不能被写出。

不稳定的（Volatile）

你可能对 volatile 关键字有疑问。最坏的情况就是你确认它真的是一个 Java 关键字。根据 Barry Boone 所说“它告诉编译器一个变量可能在线程异步时被改变”

接受它是 Java 语言的一部分，然后去担心别的吧。

联合使用修饰符

可见性修饰符不能被联合使用，一个变量不可能同时是 private 和 public，public 和 protected，protected 和 private。你当然可以联合使用可见性修饰符和我在下面列表中提及的修饰符。

native
transient
synchronized
volatile

这样你就可以有一个 public static native 方法了。

修饰符可以用在哪里？

问题

问题 1）当你试着编译运行下面的代码的时候，可能会发生什么？

```
abstract class Base{
        abstract public void myfunc();
        public void another(){
        System.out.println("Another method");
        }
}

public class Abs extends Base{
        public static void main(String argv[]){
        Abs a = new Abs();
        a.amethod();
        }

        public void myfunc(){
                System.out.println("My func");
                }
        public void amethod(){
        myfunc();
        }
}
```

1) The code will compile and run, printing out the words "My Func"
2) The compiler will complain that the Base class has non abstract methods
3) The code will compile but complain at run time that the Base class has non abstract methods
4) The compiler will complain that the method myfunc in the base class has no body, nobody at all to looove it

问题 2）当你试着编译运行下面的代码的时候，可能会发生什么？

```
public class MyMain{

public static void main(String argv){
        System.out.println("Hello cruel world");
        }
}
```

1) The compiler will complain that main is a reserved word and cannot be used for a class
2) The code will compile and when run will print out "Hello cruel world"
3) The code will compile but will complain at run time that no constructor is defined
4) The code will compile but will complain at run time that main is not correctly defined

问题 3）下面的哪个是 Java 修饰符？

1) public
2) private
3) friendly
4) transient

问题 4） 当你试着编译运行下面的代码的时候，可能会发生什么？

```
class Base{
        abstract public void myfunc();
        public void another(){
        System.out.println("Another method");
        }
}
public class Abs extends Base{
        public static void main(String argv[]){
        Abs a = new Abs();
        a.amethod();
        }
        public void myfunc(){
                System.out.println("My func");
                }
        public void amethod(){
        myfunc();
        }
}
```

1) The code will compile and run, printing out the words "My Func"
2) The compiler will complain that the Base class is not declared as abstract.
3) The code will compile but complain at run time that the Base class has non abstract methods
4) The compiler will complain that the method myfunc in the base class has no body, nobody at all to looove it

问题 5）你为什么可能会定义一个 native 方法呢？

1) To get to access hardware that Java does not know about
2) To define a new data type such as an unsigned integer
3) To write optimised code for performance in a language such as C/C++

4) To overcome the limitation of the private scope of a method

问题 6）当你试着编译运行下面的代码的时候，可能会发生什么？

```
class Base{
public final void amethod(){
        System.out.println("amethod");
        }
}

public class Fin extends Base{
public static void main(String argv[]){
        Base b = new Base();
        b.amethod();
        }
}
```

1) Compile time error indicating that a class with any final methods must be declared final itself

2) Compile time error indicating that you cannot inherit from a class with final methods

3) Run time error indicating that Base is not defined as final

4) Success in compilation and output of "amethod" at run time.

问题 7）当你试着编译运行下面的代码的时候，可能会发生什么？

```
public class Mod{
public static void main(String argv[]){
    }
    public static native void amethod();
}
```

1) Error at compilation: native method cannot be static

2) Error at compilation native method must return value

3) Compilation but error at run time unless you have made code containing native amethod available

4) Compilation and execution without error

问题 8）当你试着编译运行下面的代码的时候，可能会发生什么？

```
private class Base{}
public class Vis{
    transient int    iVal;
    public static void main(String elephant[]){
        }
}
```

1) Compile time error: Base cannot be private

2) Compile time error indicating that an integer cannot be transient

3) Compile time error transient not a data type

4) Compile time error malformed main method

问题 9）当你试着编译运行下面的两个放在同一个目录的文件的时候，可能会发生什么？

```
//File P1.java
package MyPackage;
```

```
class P1{
void afancymethod(){
            System.out.println("What a fancy method");
            }
}
//File P2.java
public class P2 extends P1{
      afancymethod();
}
```

1) Both compile and P2 outputs "What a fancy method" when run
2) Neither will compile
3) Both compile but P2 has an error at run time
4) P1 compiles cleanly but P2 has an error at compile time

问题 10）下面的哪一个声明是合法的？

1) public protected amethod(int i)
2) public void amethod(int i)
3) public void amethod(void)
4) void public amethod(int i)

答案

答案 1）

1) The code will compile and run, printing out the words "My Func"

一个 abstract 类可以有非 abstract 方法，但是任何扩展它的类必须实现所有的 abstract 方法。

答案 2）

4) The code will compile but will complain at run time that main is not correctly defined

main 的签名包含一个 String 参数，而不是 string 数组。

答案 3）

1) public
2) private
4) transient

虽然有些文本使用 friendly 来表示可见性，但它不是一个 Java 保留字。注意，测试很可能包含要求你从列表中识别 Java 关键字的问题。

答案 4）

2) The compiler will complain that the Base class is not declared as abstract.

当我使用我的 JDK1.1 编译器时的真正的错误信息是：

Abs.java:1: class Base must be declared abstract.
It does not define void myfunc() from class Base.

```
class Base{

^
```

1 error

答案 5）

1) To get to access hardware that Java does not know about

3) To write optimised code for performance in a language such as C/C++

虽然创建“纯正的 Java”代码值得鼓励，但是为了允许平台的独立性，我们不能将此作为信仰，有很多时候，我们是需要 native 代码的。

答案 6）

4) Success in compilation and output of "amethod" at run time.

这段代码调用 Base 类中的 amethod 版本。如果你在 Fin 中试着执行 amethod 的重写版本，你会得到一个编译时错误。

答案 7）

4) Compilation and execution without error

因为没有调用 native 方法，因此运行时不会发生错误。

答案 8）

1) Compile time error: Base cannot be private

一个 Base 类这样的顶级类不能定义为 private。

答案 9）

4) P1 compiles cleanly but P2 has an error at compile time

虽然 P2 在 P1 的同一个路径下，但是 P1 用 package 语句声明了，所以对于 P2 不可见。

答案 10）

2) public void amethod(int i)

如果你认为选项 3 这样携带一个 void 参数是合法的，你可能需要从你的头脑中清空一些 C/C++方面的知识。

选项 4 不合法是因为方法的返回类型必须紧跟着出现在方法名之前。

目标 3，默认的构造方法

对于一个给定的类，如果有一个默认的构造方法被创建或者定义了构造方法的原型，则类也被确定了。

本目标需要注意

这是一个精致小巧的目标，通过轻松的俯瞰 Java 语言，对各方面集中研究来完成它吧。

什么是构造方法？

你需要通过明白构造方法的概念来明白本节的目标。简单来说，构造方法是一种在类实例化时自动运行的特殊类型的方法。构造器通常被用来初始化类中的值。构造器有和类同样的名字并且没有返回值。你可能会在测验中被问到这样的问题：跟类有同样名字的方法，但是有整型或者字符串型的返回值。你要多加小心并确信，任何被认为是构造方法的方法都是没有返回值的。

如果一个方法有了类同样的名字但还有返回值，它不是构造器。这里有一个例子，一个

有构造器的类，当类的实例被创建时打印字符串“Greeting from Crowle”：

```
public class Crowle{
        public static void main(String argv[]){
        Crowle c = new Crowle();
        }
        Crowle(){
        System.out.println("Greetings from Crowle");
        }
}
```

何时 Java 提供默认构造方法？

如果你没有显式定义任何构造方法，编译器会插入一个“后台”的不可见的无参数的构造方法。一般来说，这只是在理论上很重要。但是，一个重要的限制作用是，如果你没有自己创建构造方法，你就只能得到默认的无参数的构造方法了。

如果你自己创建了构造方法，Java 就不支持默认的无参数的构造方法了。

一旦你创建了自己的构造方法，你就释放了默认的无参数构造方法。如果接下来你想试着创建一个不传送任何参数的类的实例（也就是通过一个零参数构造方法调用这个类），你会得到一个错误。因此，一旦你为一个类创建了任何的构造方法，你需要创建一个无参数的构造方法。这也是像 Borland/Inprise 的 JBuilder 这样的代码产生器在你生成类的框架时会创建一个零参数构造方法的原因之一。

下面例子中的代码不会被编译。当编译器创建名字为 c 的 Base 类的实例时，它会插入一个指向无参数的构造方法的调用。由于 Base 有一个 integer 型的构造方法，无参数的构造方法此时不允许存在，一个编译期错误产生了。可以通过在 Base 类中创建一个“什么都不干”的零参数构造方法来修复这个错误。

```
//Warning: will not compile.
class Base{
Base(int i){
        System.out.println("single int constructor");
        }
}
public class Cons {
        public static void main(String argv[]){
        Base c = new Base();
        }
}
//This will compile
class Base{
Base(int i){
        System.out.println("single int constructor");
        }
        Base(){}
}
```

```
public class Cons {
        public static void main(String argv[]){
        Base c = new Base();
        }
}
```

默认构造方法的原型

这个目标要求你明白默认构造方法的原型。它当然不能有参数，并且最明显的是默认构造方法没有指定范围，但你可以定义构造方法为 public 或者 protected。

构造方法不能是 native, abstract, static, synchronized 或 final

上面这句话源于一个编译错误信息。看起来像是新版本 Java 的错误信息质量得到提高了似的。我听说 IBM 的新 Java 编译器有好的错误报告。你也许被忠告过去使用多个合适版本的 Java 编译器来检查你的代码并查找错误。

问题

问题 1） 给定下面的类定义

```
class Base{
        Base(int i){}
}
class DefCon extends Base{
DefCon(int i){
    //XX
    }
}
```

如果将标记//XX 的地方替换为下面的行，哪一行是独立合法的？

1) super();
2) this();
3) this(99);
4)super(99);

问题 2）给定下面的类

```
public class Crowle{
        public static void main(String argv[]){
        Crowle c = new Crowle();
        }
        Crowle(){
        System.out.println("Greetings from Crowle");
        }
```

```
}
```
构造方法会返回哪一种数据类型？

1) null

2) integer

3) String

4) no datatype is returned

问题 3）当你试着编译运行下面的代码的时候，可能会发生什么？

```
public class Crowle{
        public static void main(String argv[]){
        Crowle c = new Crowle();
        }
void Crowle(){
        System.out.println("Greetings from Crowle");
        }
}
```

1) Compilation and output of the string "Greetings from Crowle"

2) Compile time error, constructors may not have a return type

3) Compilation and output of string "void"

4) Compilation and no output at runtime

问题 4）当你试着编译运行下面的类的时候，可能会发生什么？

```
class Base{
        Base(int i){
        System.out.println("Base");
        }
}

class Severn extends Base{
public static void main(String argv[]){
        Severn s = new Severn();
        }
        void Severn(){
        System.out.println("Severn");
        }
}
```

1) Compilation and output of the string "Severn" at runtime

2) Compile time error

3) Compilation and no output at runtime

4) Compilation and output of the string "Base"

问题 5）下面的哪一句陈述是正确的？

1) The default constructor has a return type of void

2) The default constructor takes a parameter of void

3) The default constructor takes no parameters

4) The default constructor is not created if the class has any constructors of its own.

答案

答案 1）
4) super(99);
由于类 Base 定义了一个构造方法，编译器将不会插入默认的 0 参数的构造方法。因此，super()的调用会引起一个错误。一个 this()调用试着在当前类中调用一个不存在的 0 参数构造方法，this(99)调用会引起一个循环引用并将引起一个编译时错误。
答案 2）
4) no datatype is returned
如果定义了一个没有数据类型的构造方法，那么没有返回类型是相当明显的
答案 3）
4) Compilation and no output at runtime
方法 Crowle 因为有一个返回类型而不是构造方法。因此，类将会编译并且在运行时方法 Crowle 不会调用。
答案 4）
2) Compile time error
当类 Severn 试着在类 Base 中调用 0 参数构造方法时会产生一个错误。
答案 5）
3) The default constructor takes no parameters
4) The default constructor is not created if the class has any constructors of its own.
选项 1 相当明显，因为构造方法不会有返回类型。选项 2 不容易确定，Java 没有为方法或构造方法提供 void 类型。

目标四，重载和覆写

为任意方法定义合法的返回类型，这些方法是在本类或父类中声明过的相关方法。

本目标需要注意的

这个目标可能相当模糊，它主要是要求你理解重载和覆写的不同。为了增强你的目的性，你需要对于方法重载和方法覆写有基本的理解。请参看第六部分：

重载，覆写，运行时类型和面向对象

同一个类中的方法

我假定目标中的相关方法是指有同样名字的方法。如果一个类中的两个或者多个方法有同样的名字，就被称为方法重载。你可以在一个类中有两个同样名字的方法，但是他们必须有不同的参数类型和顺序。

通过参数的顺序和类型来区分两个重载的方法。返回类型对区分方法没有帮助。
下面的代码会引起一个编译时错误：编译器认为 amethod 试图定义同样的方法两次。这就引起了一个像下面这样的错误，
method redefined with different return type: void amethod(int)

```
was int amethod(int)
class Same{
public static void main(String argv[]){
        Over o = new Over();
        int iBase=0;
        o.amethod(iBase);
        }
        //These two cause a compile time error
        public void amethod(int iOver){
                System.out.println("Over.amethod");
        }
        public int amethod(int iOver){
                System.out.println("Over int return method");
                return 0;
        }
}
```

返回值的类型不能帮助区分两个方法。

子类中的方法

你可以在一个子类中重载一个方法，所需要的就是新方法有不同的参数顺序和类型。参数的名字或者返回类型都不作考虑。

如果你想重写一个方法，即在子类中完全取代它的功能，重写后的方法必须跟基类中被取代的原始方法有完全相同的签名。这就包括了返回值。如果你在子类中创建了一个有同样名字和签名但是有不同返回值的方法，你将会得到一个跟上例同样的错误信息：

```
method    redefined with different return type: void amethod(int)
was int amethod(int)
```

编译器认为这是错误的尝试方法重载，而不认为是方法重写。static 方法不能被重写。

如果你认为重写只是在子类中简单的替换了一个方法，你就很容易认为 static 方法也能被重写。事实上，我有很多包含人们举例指明 static 方法能被重写的代码的邮件。然而，这些并没有考虑方法重写在运行时决定哪个版本的方法被调用的细节问题。下面的代码似乎表明 static 方法是怎样被重写的。

```
class Base{
    static     void amethod(){
    System.out.println("Base.amethod");
    }
}
public class Cravengib extends Base{
    public static void main(String arg[]){
    Cravengib cg = new Cravengib();
    cg.amethod();
    }
```

```
    static void amethod(){
    System.out.println("Cravengib.amethod");
    }
}
```

如果你编译并运行这段代码，你会发现输出文本 Cravengib.amethod，这似乎很好的指明了重写。然而，对于重写，还有相对于在子类中使用一个方法简单替换另一个方法更多的东西。还有运行时决定的方法基于引用的类的类型的问题，这可以通过创建正在被实例化的类的引用类型（实例初始化语句的左半部分）来说明。

在上面的例子中，因为名字叫 amethod 的方法与类发生了关联，而不是与特定的类的实例相关联，它不在乎什么类型的类正在创建它，而仅仅在意引用的类型。因此，如果你在调用 amethod 前改变一下这一行，

```
Base cg= new Cravengib()
```

你就会发现当你运行程序时，你会得到输出：Base.amethod

cg 是一个类 Cravengib 在内存中的一个 Base 类型的实例的引用（或者指针）。如果一个 static 方法被调用了，JVM 不会检查什么类型正在指向它，它只会调用跟 Base 类相关联的方法的实例。

与上面的情况相对比：当一个方法被重写时，JVM 通过句柄检查正在指向的类的类型，并调用此类型相关的方法。可以结束这个例子了，如果你将两个版本的 amethod 方法改变为非 static，并依然创建类：

```
Base cg= new Cravengib()
```

编译并运行上述代码，你会发现 amethod 已经被重写了，并且输出 Cravengib.amethod。

问题

问题 1）给定下面的类定义

```
public class Upton{
public static void main(String argv[]){
        }
        public void amethod(int i){}
        //Here
}
```

下面哪一个在替换//Here 后是合法的？

1) public int amethod(int z){}
2) public int amethod(int i,int j){return 99;}
3) protected void amethod(long l){ }
4) private void anothermethod(){}

问题 2）给定下面的类定义

```
class Base{
        public void amethod(){
        System.out.println("Base");
        }
}
```

```
public class Hay extends Base{
public static void main(String argv[]){
        Hay h = new Hay();
        h.amethod();
        }
}
```

下面在类 Hay 中的哪一个方法将会编译并使程序打印出字符串"Hay"？

1) public int amethod(){ System.out.println("Hay");}
2) public void amethod(long l){ System.out.println("Hay");}
3) public void amethod(){ System.out.println("Hay");}
4) public void amethod(void){ System.out.println("Hay");}

问题 3）给定下面的类定义

```
public class ShrubHill{
    public void foregate(String sName){}
//Here
}
```

下面的哪一个方法可以合法的直接替换//Here？

1) public int foregate(String sName){}
2) public void foregate(StringBuffer sName){}
3) public void foreGate(String sName){}
4) private void foregate(String sType){}

答案

答案 1）

2) public int amethod(int i, int j) {return 99;}
3) protected void amethod (long l){}
4) private void anothermethod(){}

选项 1 由于两个原因不会被编译。第一个相当明显，因为它要求返回一个 integer。另一个是试着直接在类内部重新定义一个方法。把参数的名字从 i 换成 z 是无效的，并且一个方法不能在同一个类里重写。

答案 2）

3) public void amethod(){ System.out.println("Hay");}

选项 3 重写了类 Base 的方法，因此任何 0 参数调用都调用这个版本。

选项 1 将会返回一个表示你尝试重新定义一个不同返回类型的方法的错误。选项 2 将会编译对于 amethod()调用 Base 类的方法，并且输出字符串"Base"。选项 4 是为了抓住满脑子 C/C++ 的人而设计的。Java 里没有 void 方法参数这样的事。

答案 3）

2) public void foregate(StringBuffer sName){}
3) public void foreGate(String sName){}

选项 1 是试着定义一个方法两次，有一个 int 返回值并不能帮助将它与存在的 foregate 方法相区分。而像选项 4 那样改变方法的参数名，也不能与存在的方法相区分。注意，选项 2 里的 foreGate 方法有一个大写的 G。

第 2 章 流程控制和差错处理

目标一 if 和 switch 语句

用 if 和 switch 编写代码，识别这些语句的合法参数类型

If/else 语句

在 java 中 If/else 结构和你所了解的其他语言一样，switch/case 语句有一些自己的特点
if/else 的语法是

```
if(boolean condition){
        //the boolean was true so do this
        }else {
        //do something else
}
```

与在 Visual Basic 语句中不同，Java 中不存在"then"关键字
花括号在 Java 中是一个常用的复合语句的指示器，它可以使你把多行代码作为一些判断语句的一个结果来执行。这可以被看作一个程序块。else 部分常常是可选的。你可以像以下这样链接多个 if/else 语句（但是在链接了几个之后你就要考虑使用 case 结构来代替了）

```
int i=1;
    if(i==1){
        //some code
    } else if (i==2){
        //some code
    } else{
        //some code
    }
```

Java 中 if 语句的一个特性是必须带一个 boolean 类型的值。你不能像使用 C/C++习惯的那样使用任何非零的数值来表示 true，而用零来表示 false。
因此，在 Java 中以下语句将不会被编译

```
int k =-1;
        if(k){//Will not compile!
        System.out.println("do something");
}
```

因为你必须明确的使 k 的判断语句返回一个 boolean 类型的值，就像下面的例子

```
if(k == -1){
    System.out.println("do something"); //Compiles OK!
    }
```

当在 C/C++中时，你可以去掉花括号，如下

```
boolean k=true;
if(k)
```

```
System.out.println("do something");
```

这有时候被认为是不好的设计风格，因为如果你稍后要修改代码来包含更多语句，他们就会在条件语句块外，像这样

```
if(k)
        System.out.println("do something");
        System.out.println("also do this");
```

第二个输出语句将总会被执行

switch 语句

Peter van der Lindens 对于 switch 语句的评价概括起来就像他所说的“毁灭于 switch 语句”因此，这是一个你必须花费更多的精力关注的问题。switch 语句的参数必须是一个 byte，char，short 或 int 类型的变量。你也许会遇到考试题中用 float 或者 long 做 switch 语句的参数。有一个非常普遍的问题似乎就是，关于在执行 switch 语句的过程中使用 break 语句。这里有一个这类问题的例子。

```
int k=10;
switch(k){
        case 10:
                System.out.println("ten");
        case 20:
                System.out.println("twenty");
        }
```

常识判断，执行 case 语句后面的指令，然后碰到另一个 case 语句，编译器就应该结束执行 switch 语句。但是，就像程序设计者所熟知的，case 语句只在碰到 break 语句的时候才终止执行。结果，在上面例子中，ten 和 twenty 都将被输出。可以作为一个问题提出来的另一个小的特性就是使用 default 语句。

注意：default 语句不是必须在 case 语句的结尾处出现

按照惯例 default 语句是放在 case 选项的结尾处，所以通常代码写成如下形式

```
int k=10;
switch(k){
        case 10:
                System.out.println("ten");
                break;
        case 20:
                System.out.println("twenty");
                break;
        default:
                System.out.println("This is the default output");
}
```

这种方法反映大多数人的思维方式。当你尝试其他可能情况时，会执行 default 输出。但是，如果不是被要求的话，把 defalt 语句写在 switch 语句的顶部，在语法上也是正确的。

```
int k=10;
    switch(k){
```

```
        default: //Put the default at the bottom, not here
            System.out.println("This is the default output");
            break;
        case 10:
            System.out.println("ten");
            break;
        case 20:
            System.out.println("twenty");
            break;
}
```

if 和 switch 语句的合法参数

正如先前所提到的，if 语句只能用 boolean 类型参数，而 switch 语句只能用 byte，char，short 或者 int 类型作参数。

三项 ？操作符

一些程序员主张三项操作符很有用。我不这么认为。在目标中并没有特别提到它，所以如果在考试中出现的话请告诉我。

其他流程控制语句

虽然公布的目标只提到了 if/else 和 case 语句，考试中也许会涉及 do/while 和 while loop 语句。

练习

习题 1）

创建一个文件含有一个公共类叫 IfElse。创建一个方法叫 go,它接收 main 方法的字符串数组参数作为它的参数。在这个方法中创建了一个 if/else 程序块，用来查看来自数组的第一个元素，用字符串的 equals 方法来判断输出。如果为"true"则打印"ok"，如果为"false"则打印"Not ok"，如果是 true 或 false 以外的字符串则打印"Invalid command parameter"，用一个 if/else if/else 语句这样的次序进行设计。

习题 2）

修改这个 IfElse 类，使 if 语句可以检查传到 go 方法的字符串数组是否是零长度串，使用数组 length 域来检查。如果长度为零则输出"No parameter supplied"，把现有的 if/else if/else 块放在这个练习的 else 中，使程序能实现原版本的功能。

答案 1）

```
public class IfElse{
    public static void main(String argv[]){
        IfElse ie = new IfElse();
        ie.go(argv);
    }
    public void go(String[] sa){
    String s = sa[0];
    if(s.equals("true")){
        System.out.println("OK");
    }else if(s.equals("false")){
        System.out.println("Not OK");
    }else{
        System.out.println("Invalid command parameter");
    }
    }
}
```

答案 2）

```
public class IfElse{
    public static void main(String argv[]){

        IfElse ie = new IfElse();
        ie.go(argv);
    }
    public void go(String[] sa){
    if(sa.length ==0){
        System.out.println("No parameter supplied");
    }else{
        String s = sa[0];
        if(s.equals("true")){
        System.out.println("OK");
        }else if(s.equals("false")){
        System.out.println("Not OK");
        }else{
        System.out.println("Invalid command parameter");
        }
    }
    }
}
```

问题

问题 1) 编译运行下列代码时会发生什么情况？

```
public class MyIf{
boolean b;
public static void main(String argv[]){
  MyIf mi = new MyIf();
}
MyIf(){
            if(b){
                  System.out.println("The value of b was true");
                  }
                  else{
                  System.out.println("The value of b was false");
                  }
      }
}
```

1) Compile time error variable b was not initialised
2) Compile time error the parameter to the if operator must evaluate to a boolean
3) Compile time error, cannot simultaneously create and assign value for boolean value
4) Compilation and run with output of false

问题 2) 编译运行下列代码时会发生什么情况？

```
public class MyIf{
public static void main(String argv[]){
       MyIf mi = new MyIf();
      }
MyIf(){
      boolean b = false;
      if(b=false){
            System.out.println("The value of b is"+b);
            }
      }
}
```

1) Run time error, a boolean cannot be appended using the + operator
2) Compile time error the parameter to the if operator must evaluate to a boolean
3) Compile time error, cannot simultaneously create and assign value for boolean value
4) Compilation and run with no output

问题 3) 编译运行下列代码时会发生什么情况？

```
public class MySwitch{
public static void main(String argv[]){
      MySwitch ms= new MySwitch();
      ms.amethod();
      }
```

```
public void amethod(){
      char k=10;
            switch(k){
            default:
                  System.out.println("This is the default output");
                  break;
              case 10:
                  System.out.println("ten");
                  break;
              case 20:
                  System.out.println("twenty");
            break;
           }
      }
}
```

1) None of these options
2) Compile time error target of switch must be an integral type
3) Compile and run with output "This is the default output"
4) Compile and run with output "ten"

问题 4) 编译运行下列代码时会发生什么情况？

```
public class MySwitch{
public static void main(String argv[]){
      MySwitch ms= new MySwitch();
      ms.amethod();
      }
public void amethod(){
      int k=10;
            switch(k){
            default: //Put the default at the bottom, not here
                  System.out.println("This is the default output");
                  break;
              case 10:
                  System.out.println("ten");
              case 20:
                  System.out.println("twenty");
            break;
           }
      }
}
```

1) None of these options
2) Compile time error target of switch must be an integral type
3) Compile and run with output "This is the default output"
4) Compile and run with output "ten"

问题 5) 下面哪个是不能用于 switch 语句的参数？

1) byte b=1;

2) int i=1;

3) boolean b=false;

4) char c='c';

答案

答案 1）

4) Compilation and run with output of false

因为 boolean b 在类级中被创建，它不需明确初始化，而且它有默认的 boolean 值 false。if 语句判断一个 boolean 值，所以 b 符合这个要求。

答案 2）

4) Compilation and run with no output

因为 b 是 boolean 类型，if 语句不会产生错误。如果 b 是任何其他的数据类型，在你试图赋值而不是比较的时候错误就产生了。下列表达

if(b=false)

通常是一个程序员的错误。程序员大多要表现

if (b==false)

如果 b 的类型是 boolea 以外的任意类型，会导致编译期错误。if 表达式的要求是必须返回一个 boolean 类型，因为

(b=false)

返回一个 boolean 类型，所以被接受（如果无用处）。

答案 3）

4) Compile and run with output "ten"

答案 4）

1) None of these options

因为下句后缺少 break 语句

case 10;

实际输出结果会是"ten"接着是"twenty"

答案 5）

1) byte b=1;

2) int i=1;

4) char c='c';

switch 语句可以使用 byte，char 或 int 作参数。

目标二 循环，break 和 continue

用循环格式编写代码，使用带标签和不带标签的 break 和 continue 语句，声明循环计数器的值在循环执行中或循环结束时

for 语句

最常用的循环方法就是应用 for 语句。对于 for 语句在其他的编程语言中有非常相似的结构。比如 C/C++和 perl 就有 for 结构。很多程序员在循环中使用 for 结构，因为其简洁，自含，容易理解而且不容易混乱。类似 C++而与 C 语言不同，循环控制变量可以在 for 语句中定义和初始化。如下

```
public class MyLoop{
        public static void main(String argv[]){
        MyLoop ml = new MyLoop();
        ml.amethod();
        }
         public void amethod(){
                    for(int K=0;K<5l;K++){
                          System.out.println("Outer "+K);
                          for(int L=0;L<5;L++)
                          {System.out.println("Inner "+L);}
                       }
             }
}
```

内循环代码在每次外循环执行时会循环执行五次。所以输出为：

```
Outer 0;
Inner 0
Inner 1
Inner 2
Inner 3
inner 4
Outer 1;
Inner 0
Inner 2
```

for 语句和 Visual Basic 的 for/next 循环一样。你可以认为它的语法是

```
for(initialization; conditional expression;increment)
```

其中条件表达式必须是 boolean 判断，就像 if 语句的简单形式。在上例的代码中，for 语句紧跟着是花括号中的程序块。类似 if 语句，当不需要使用程序块时，你可以使用下面这样的简单形式

```
 for(int i=0;i<5;i++)
                    System.out.println(i);
```

在任何版本中你都不能用分号来结束一个 for 行，如果你这么做，for 循环就会原地打转直到条件满足，然后就会以“直线”的方式执行下面的代码。在此例中你不是必须在 for

循环中定义变量，但是如果在循环中定义变量，当跳出循环时变量也就跳出了它的作用域。按照变量作用域尽可能小的说法，这可以看作是一个优点。

for 中的块为空在语法中也是正确的，这样循环会永远进行下去

```
for(;;){
        System.out.println("forever");
    }
```

但是用 while(true)的形式可能会更加简洁

```
while(true){
        System.out.println("true");
    }
```

while 循环和 do 循环，意料之中

while 和 do 循环的运行就像你想象的一样，和在其他语言中相同。
因此，while 会依照判断执行零到多次，而 do 会执行一到多次。while 循环的语法是：

```
while(condition){
        bodyOfLoop;
}
```

像 if 语句一样，条件是一个 boolean 类型的判断。同样，你不能像 C/C++习惯的那样用零来代表 false，而用任意其他值来代表 true。
所以，你可能会像下面那样创建一个 while 循环

```
while(i<4){
        i++;
        System.out.println("Loop value is :"i);
}
```

注意，如果变量 i 为 4，或者比 4 大，当你到达 while 语句时，将没有输出。相反，do 循环总是会执行一次。所以，不管进入循环时变量 i 的值是什么，以下代码总是会得到至少一个输出。

goto 语句，科学还是迷信？

Java 的设计者决定同意写过著名文章"Goto 有害"的编程领袖 Edsger Dijkstra 的观点。因为不加选择的使用 goto 语句会导致“意大利面条似的代码”难以维护，不可使用，而且这被认为是不好的编程风格。“意大利面条似的代码”是指不容易表述逻辑开始和结束的代码。goto 语句有时会被说成“无条件跳转”，也就是可能会写这样的代码，不进行判断就从程序的一部分跳转到另外一处。这在某些情况下是有用的，即 Java 为 break 和 continue 关键字提供了有标签和无标签两个版本。

```
public class Br{
    public static void main(String argv[]){
        Br b = new Br();
        b.amethod();
```

```
    }
    public void amethod(){
        for(int i=0;i <3;i ++){
        System.out.println("i"+i+"\n");
        outer://<==Point of this example
        if(i>2){
                break outer;//<==Point of this example
        }//End of if

        for(int j=0; j <4 && i<3; j++){
        System.out.println("j"+j);
        }//End of for
    }//End of for
    }//end of Br method
}
```

然后，你需要挑出代码中哪个是要输出的字母组合。顺便说一下，"\n"是输出一个空白行。

跳转到标签

在有些条件下从内循环跳到外循环常被描述，你可以使用带标签的 break 和 continue 语句来实现它。一个标签是一个简单的非关键字，后面跟一个冒号。通过在 break 或 continue 后使用标签，你的代码可以跳转到此标签处。这是便捷的实现部分条件循环的方法。你当然可以用 if 语句，但是一个 break 语句更方便。按照 Elliotte Rusty Harold，一个著名的 Java 作者所说，“在整个 Java1.0.1 源代码中，只用了七个 continue 语句写 java 包。”这意味着在实际编程中你可能不会得到充分的练习，所以为了考试你要花费更大的精力来学好它。考试题的编写者好像热爱设计费解的网状的带有 break 和 continue 语句的循环，你可能永远不会遇到有好的设计的代码。

关键概念

break 语句完全放弃执行当前循环，continue 语句只放弃整个循环中当前本次循环
做下面的例子

```
public class LabLoop{
    public static void main(String argv[]){
    LabLoop ml = new LabLoop();
    ml.amethod();
    }
    public void amethod(){
    outer:
    for(int i=0;i<2;i++){
        for(int j=0;j<3;j++){
        if(j>1)
            //Try this with break instead of continue
```

```
                continue outer;
            System.out.println("i "+ i + " j "+j);
            }
        }//End of outer for
        System.out.println("Continuing");
        }
}
```

这个版本有以下输出

```
i 0 j 0
i 0 j 1
i 1 j 0
i 1 j 1
Continuing
```

如果你用 break 替换 continue，i 计数器会在零处停止，因为外循环会被放弃，而不会简单的进入下一个递增。

问题

问题 1) 编译运行一个方法中的下列代码时会发生什么情况？

```
  for(int i=0;i<5;){
                    System.out.println(i);
                    i++;
                    continue;
                    }
```

1) Compile time error, malformed for statement
2) Compile time error continue within for loop
3) runtime error continue statement not reached
4) compile and run with output 0 to 4

问题 2)编译运行下列代码时会发生什么情况？

```
public class LabLoop{
                    public static void main(String argv[]){
                    LabLoop ml = new LabLoop();
  ml.amethod();
                    mainmethod:
                    System.out.println("Continuing");
                    }
            public void amethod(){
                    outer:
                    for(int i=0;i<2;i++){
                            for(int j=0;j<3;j++){
```

```
                                    if(j>1)
                                        break mainmethod;
                                        System.out.println("i "+ i + " j "+j);
                                }
            }//End of outer for
    }
}
```

1)
i 0 j 0
i 0 j 1
Continuing
2)
i 0 j 0
i 0 j 1
i 1 j 0
i 1 j 1
Continuing
3)
Compile time error
4)
i 0 j 0
i 0 j 1
i 1 j 0
i 1 j 1
i 2 j 1
Continuing

问题 3)编译运行下列代码时会发生什么情况？

```
public void amethod(){
outer:
      for(int i=0;i<2;i++){
      for(int j=0;j<2;j++){
            System.out.println("i="+i + " j= "+j);
            if(i >0)
            break outer;
      }
      }
      System.out.println("Continuing with i set to ="+i);
}
```

1) Compile time error
2)
i=0 j= 0
i=0 j= 1

i=1 j= 0
3)
i=0 j= 0
i=0 j= 1
i=1 j= 0
i=2 j= 0
4)
i=0 j= 0
i=0 j= 1

问题 4)编译运行下列代码时会发生什么情况？

```
int i=0;
        while(i>0){
                System.out.println("Value of i: "+i);

        }
                do{
                        System.out.println(i);
                        } while (i <2);

                }
```

1)
Value of i: 0
followed by
0
1
2
2)
0
1
2
3)
Value of i: 0
Followed by continuous output of 0
4) Continuous output of 0

问题 5) 编译运行下列代码时会发生什么情况？

```
public class Anova{
    public static void main(String argv[]){
    Anova an = new Anova();
    an.go();
    }
    public void go(){
    int z=0;
    for(int i=0;i<10; i++,z++){
```

```
            System.out.println(z);
        }
        for(;;){
            System.out.println("go");
        }
        }
}
```

1) Compile time error, the first for statement is malformed
2) Compile time error, the second for statement is malformed
3) Output of 0 to 9 followed by a single output of "go"
4) Output of 0 to 9 followed by constant output of "go"

问题 6)下列代码的输出结果是什么？

```
public class MyFor{
        public static void main(String argv[]){
            int i;
            int j;
        outer:
            for (i=1;i <3;i++)
            inner:
        for(j=1; j<3; j++) {
            if (j==2)
            continue outer;
            System.out.println("Value for i=" + i + " Value for j=" +j);
        }
        }
}
```

1) Value for i=1 value for j=1
2) Value for i=2 value for j=1
3) Value for i=2 value for j=2
4) Value for i=3 value for j=1

答案

答案 1）

4) compile and run with output 0 to 4
这是一个很奇怪但是完全正确的语句

答案 2）

3) Compile time error
你不能武断的跳入另一个方法，在 goto 语句中会带来很多有害的结果。

答案 3）

1) Compile time error
这实际上不是关于 break 和 continue 的问题。这段代码不会被编译，因为变量对 for 循环外部来说永远是不可见的。所以最后的 System.out.println 语句会引起编译时错误。

答案 4）
1) Continuous output of 0
没有值被增加，而且如果第一次判断不为真时 while 循环将不会执行。

答案 5）
4) Output of 0 to 9 followed by constant output of "go"
第一个 for 循环结构不常用但是完全正确。

答案六
1) Value for i=1 value for j=1
2) Value for i=2 value for j=1

目标三 try/catch 和方法重写

编写代码合理使用异常和异常处理机制（try catch finally）,定义和重写方法抛出异常.

一个异常情况是当程序进入一个不是很正常的状态.异常捕获有时是指错误捕获.一个典型的异常例子是当程序试图打开一个不存在的文件时或者你试图访问一个数组中不存在的元素时.

try 和 catch 语句是构建 Java 异常处理的一部份.不论 C/C++还是 Visua Basic 都没有直接对应 Java 异常处理的结构.C++支持异常,但是是可选的,Visial Basic 支持 On Error/Goto 错误捕获，这带有早期不灵活的 BASIC 编程时代的味道。

Java 异常是 Java 语言的一个结构。例如如果你要执行 I/O 操作，你必须把它放在错误处理中。你当然可以不把它放在处理中，这毫无作用。下面是一个小代码片断，我用 Borland/Inprise JBuilder 临时停止控制台输出，等待按任意键继续

```
public class Try{
import java.io.*;
    public static void main(String argv[]){
        Try t = new Try();
        t.go();
    }//End of main
public void go(){
     try{
    InputStreamReader isr = new InputStreamReader(System.in);
    BufferedReader br = new BufferedReader(isr);
        br.readLine();
      } catch(Exception e){
     /*Not doing anything when exception occurs*/
     } //End of try
    System.out.println("Continuing");
  }//End of go
```

}

在这个例子中，错误出现时没有任何处理，但是程序员一定知道错误有可能发生。如果你移去 try 和 catch 字句，代码将完全不会被编译。编译器知道 I/O 方法会引发异常而且需要异常处理代码。

与 Visal Basic 和 C/C++比较

Visal Basic 或 C/C++允许抛出混合“快且脏”的程序，假装没有错误发生过，Java 比它们严格些。记得 DOS 的最初版本被他的创作者叫做 QDOS，因为是快且脏的 DOS，看看我们已经在这样的环境下生活了多久。当你开始把快且脏的程序放到 try/catch 块当中，也就开始了真正的错误跟踪。这不是完全的束缚和编程律条，这只是劝说你“做正确的事”。

方法重写，抛出异常

在子类中一个重写的方法可能只抛出父类中声明过的异常或者异常的子类。这只适用于方法重写而不适用于方法重载。所以如果如果一个方法有完全相同的名称和参数，它只能抛出父类中声明过的异常或者异常的子类。但是它抛出很少或者不抛出异常。所以下面的例子将不被编译

```
import java.io.*;
class Base{
public static void amethod()throws FileNotFoundException{}
}

public class ExcepDemo extends Base{
   //Will not compile, exception not in base version of method
   public static void amethod()throws IOException{}
}
```

如果在父类中有抛出 IOException 异常的方法，在子类中的方法抛出 FileNotFoundException，代码将编译通过。再次，记住只适用于方法重写，在方法重载中没有类似规定。一个在子类中重写的方法可能会抛出异常。

throw 子句

我们在代码中需要包含可能抛出异常的 try/catch 块的一个原因就是，你的代码可以开始展现出什么可能发生，而不是什么应该发生。你可以通过使用 throws 字句作为方法声明的一部分来把异常放到堆栈中。这就有效的说明“当一个错误发生时，这个方法抛出这个异常，并且这个异常必须被调用它的方法捕获”。

这有一个使用 throw 子句的例子

```
import java.io.*;
public class Throws{
```

```
    public static void main(String argv[]){
        Throws t = new Throws();
    try{
    t.amethod();
    }catch (IOException ioe){}
        }
    public void amethod() throws IOException{
        FileInputStream fis = new FileInputStream("Throws.java");
    }
}
```

问题

问题 1） 编译运行以下代码会发生什么情况？

```
import java.io.*;
class Base{
public static void amethod()throws FileNotFoundException{}
}
public class ExcepDemo extends Base{
public static void main(String argv[]){
        ExcepDemo e = new ExcepDemo();
}
public static void amethod(){}
protected ExcepDemo(){
 try{
   DataInputStream din = new DataInputStream(System.in);
   System.out.println("Pausing");
   din.readChar();
   System.out.println("Continuing");
   this.amethod();
   }catch(IOException ioe) {}
}
}
```

1) Compile time error caused by protected constructor
2) Compile time error caused by amethod not declaring Exception
3) Runtime error caused by amethod not declaring Exception
4) Compile and run with output of "Pausing" and "Continuing" after a key is hit

问题 2） 编译运行以下代码会发生什么情况？

```
import java.io.*;
class Base{
public static void amethod()throws FileNotFoundException{}
}
public class ExcepDemo extends Base{
```

```
public static void main(String argv[]){
        ExcepDemo e = new ExcepDemo();
}
public static void amethod(int i)throws IOException{}
private ExcepDemo(){
  try{
      DataInputStream din = new DataInputStream(System.in);
      System.out.println("Pausing");
      din.readChar();
      System.out.println("Continuing");
      this.amethod();
      }catch(IOException ioe) {}
   }
}
```

1) Compile error caused by private constructor
2) Compile error caused by amethod declaring Exception not in base version
3) Runtime error caused by amethod declaring Exception not in base version
4) Compile and run with output of "Pausing" and "Continuing" after a key is hit

问题 3) 编译运行以下代码会发生什么情况？

```
import java.io.*;
class Base{
public static void amethod()throws FileNotFoundException{}
}

public class ExcepDemo extends Base{
public static void main(String argv[]){
        ExcepDemo e = new ExcepDemo();
}

public static void amethod(int i)throws IOException{}
private boolean ExcepDemo(){
  try{
      DataInputStream din = new DataInputStream(System.in);
      System.out.println("Pausing");
      din.readChar();
      System.out.println("Continuing");
      this.amethod();
      return true;
      }catch(IOException ioe) {}
      finally{
      System.out.println("finally");
      }
```

```
        return false;
    }

}
```

1) Compilation and run with no output.
2) Compilation and run with output of "Pausing", "Continuing" and "finally"
3) Runtime error caused by amethod declaring Exception not in base version
4) Compile and run with output of "Pausing" and "Continuing" after a key is hit

问题 4)　以下哪个要求程序员添加外部的 try/catch 异常处理。

1)Traversing each member of an array
2) Attempting to open a file
3) Attempting to open a network socket
4) Accessing a method in other class

问题 5) 编译运行以下代码会发生什么情况？

```
import java.io.*;
class granary{
        public void canal() throws IOException{
        System.out.println("canal");
        }
}

public class mmill extends granary{
        public static void main(String argv[]){
        System.out.println("mmill");
        }
        public void canal(int i) throws Exception{
        System.out.println("mmill.canal");
        }
        public void    canal(long i) {
        System.out.print("i");
        }
}
```

1) Compile time error
2) Runtime errors
3) Compile error, mmill version of canal throws Exception not in granary version
4) Compilation and run with output of mmill

答案

问题 1) 答案

4) Compile and run with output of "Pausing" and "Continuing" after a key is hit

在子类中的重写方法不能抛出在基类中没有抛出的异常。在这个例子中的方法 amethod 没有抛出异常，所以编译不会出现问题。构造器不能是 protect 类型的。

问题 2) 答案

4) Compile and run with output of "Pausing" and "Continuing" after a key is hit
在这个版本中 amethod 被重写了，没有限制抛出或不抛出异常。

问题 3) 答案

1) Compilation and run with no output.
好的，我有点跑题了，注意构造器有一个返回值。这把它变成了一个普通方法，而且当没有实例被创建时它将不会被调用。

问题 4) 答案

2) Attempting to open a file
3) Atempting to open a network socket
通常来说，所有的 I/O 操作都需要外在的使用 try/catch 块的异常处理。JDK1.4 考试不明确的覆盖 I/O，但是也许会提到错误处理的内容。

问题 5) 答案

4) Compilation and run with output of mmill
什么样的异常可以被抛出的限制只是应用于被重写的方法，不用于被重载的方法。因为 canal 方法在 mmill 版本中被重载（也就是它带有了不同的参数类型），所以不会有编译或运行错误。

目标四 什么情况下产生异常

识别发生在代码片断指定位置的异常产生的结果。注意：异常必须是运行时异常，一个被检查的异常或者一个错误（代码可能包括 try,catch 或者 finally 子句，在任何可能的组合中）

目标注释

这个目标要求你理解可控的和不可控异常（一种你要写代码捕获，另一种不用），理解 finally 子句如何工作。

检查和非检查异常

虽然 Java 强调你把捕获异常代码插入到他们可能发生的地方像 I/O 操作等，这样比较方便，但是如果你必须把这些代码插入到程序员应该控制程序状态的地方，就不方便了。这种情况的一个例子就是遍历数组的每一个元素。Java 中一个优美的地方就是它不需要程序员的介入而明确的报告这种异常类型的方式。这种自动异常处理是由把异常分为可控和不可控异常实现的。像内存耗尽或者访问到数组末尾这种情况会自动识别，而试图打开不存在的文件就需要明确的 try/catch 异常捕获。

默认的非检查信息

非检查异常出现的一个默认结果就是一个信息会被发送到控制台。例如下面代码

```
public class GetArg{
    public static void main(String argv[]){
        System.out.println(argv[0]);
    }
```

如果编译运行代码而不输入命令行参数，你会在控制台得到一个错误信息

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException
        at GetArg.main(GetArg.java:3)
```

同时这种情况是可以用于学习目的编写的程序的，在实际程序中，用户大概不会访问控制台，也不会理解这样的信息。最好编写代码来检查可能产生非运行异常的数值。所以代码可以修改成

```
public class GetArg{
    public static void main(String argv[]){
        if(argv.length ==0){
        System.out.println("Usage: GetArg param");
        }else{
        System.out.println(argv[0]);
        }
    }
}
```

检查异常

要求程序员编写代码处理检查异常。如果没有代码处理检查异常可能会出现编译不通过。就像前面的代码带有 try/catch 块结构，但是写一个空 catch 块，不对产生的异常进行任何处理也是可以的。当然这通常不是好的设计，当你碰到要在异常时编写代码的问题时，你可能还是要在这些代码中做些有用的事。在 catch 块常做的两件事是产生出错信息和打印错误跟踪。异常系统提供一个非常方便的方法，通过 getMessage 方法产生常见的有意义的错误信息。

看看以下代码

```
import java.io.*;
public class FileOut{
    public static void main(String argv[]){
    try{
    FileReader fr = new FileReader("FileOut.txt");
    }catch(Exception e){
        System.out.println(e.getMessage());
    }
    }
```

}

如果你在一个没有 FileOut.txt 文件的目录中编译运行这段代码，你会得到一个错误信息 FileOut.txt (No such file or directory)

finally 子句

在考试中你可能会被问到，在什么情况 try/catch 子句后的 finally 方法会被执行。简单回答就是 finally 子句总是会被执行，甚至当你觉得它可能不会被执行时。所以说，try/catch/finally 语句的执行顺序是你应该注意确认它在什么情况下是怎么执行的。

关键概念

不论在 try/catch 部分是不是有返回，try/catch 块的 finally 子句总会执行。

少数情况下当有如下调用时，finally 子句不会被执行

System.exit(0);

考试往往不会在这个规则上考你。

考试更可能给你一个包括 return 语句的例子，来误导你认为代码不执行 finally 语句就返回。不要被误导，finally 子句总是会执行。

try/catch 子句在它结构正确时一定会捕获错误。所以你不能试图在捕获特殊的 IOException 之前写一个捕获一般异常的 catch，来捕获所有的 Exception.

下面代码将不会通过编译

```
try{
DataInputStream dis = new DataInputStream(System.in);
dis.read();
}catch (Exception ioe) {}
catch (IOException e) {//Compile time error cause}
finally{}
```

这段代码会在编译时发出错误信息，更特殊的 IOException 将不会被达到。

问题

问题 1) 下列那个需要建立 try/catch 块或者重新抛出异常？

1) Opening and reading through a file
2) Accessing each element of an array of int values
3) Accessing each element of an array of Objectts
4) Calling a method defined with a throws clause

问题 2) 编译运行下列代码会发生什么情况？

```
import java.io.*;
class Base{
public static void amethod()throws FileNotFoundException{}
```

```
}
public class ExcepDemo extends Base{
public static void main(String argv[]){
        ExcepDemo e = new ExcepDemo();
}
public boolean amethod(int i){
 try{
     DataInputStream din = new DataInputStream(System.in);
     System.out.println("Pausing");
     din.readChar();
     System.out.println("Continuing");
     this.amethod();
     return true;
     }catch(IOException ioe) {}
     finally{
     System.out.println("Doing finally");
     }
     return false;
  }
  ExcepDemo(){
       amethod(99);
  }
}
```

1) Compile time error amethod does not throw FileNotFoundException
2) Compile, run and output of Pausing and Continuing
3) Compile, run and output of Pausing, Continuing, Doing Finally
4) Compile time error finally clause never reached

答案 1)

打开读一个文件

4) Calling a method defined with a throws clather Resources on this topic

数组元素的类型对错误处理没有任何影响。通过定义 throws 子句在方法中的使用，可能会抛出一个异常，这个异常类型会被使用它的代码捕获或者再抛出。

答案 2)

3) Compile, run and output of Pausing, Continuing, Doing Finally

finally 子句总是会运行。

目标五、六 使用断言

编写正确使用断言的代码，并且区分适当和不适当的断言使用。识别关于断言机制的正确论述。

目标的评论

断言是随着 2002 年中期 JDK1.4 考试版本的发布而添加到 Sun Certified Java Programmers 考试目标中的。由于它们是考试的新特性，你一定会在考试碰到这类题目。断言是其他面向对象语言的特性，在它被加入到 Java 中的时候一度面临很大的压力。

断言是因 JDK1.4 的发布而添加到 Java 语言中的，所以在写著此书的时候没有太多关于这个主题的笔墨。

断言为何存在

C++中可以使用断言，但 C 或 Visual Basic（或者据我了解还有 Pascal）中没有，所以很多人都没有使用过。如果事情到了很多 C++程序都没有使用过它们的地步。断言是一个相当简单的概念，你只需要写一个始终都是 true 的语句，但是在形式上它们可以从最终的编译版本中去除，所以不会导致运行时开销。使用 JDK1.4 之前模拟断言功能的结构来书写代码是完全可以的，但是以这种方式做起来会很困难，它们在运行时会被关闭。

在 C/C++语言中，断言可以使用语言预处理器来创建，在新闻组中有大量关于 Java 语言是否应该有预处理系统的讨论。观点的分歧在于有人认为预处理宏是恶魔的东西，它带来了创造过于复杂结构的机会，有人认为它会给语言带来不可思议的力量。无论是哪种方式，Java 设计者倾向于实现预处理机制，并且在 JDK1.4 中包含了断言。

如何使用断言

何地以及如何使用断言大概需要一种类似于何地及如何使用注释的判断方式。一些程序员从来不使用注释，这种风格的程序被广泛认为是糟糕的程序。因为大约 80%的代码是由其他人而不是原作者维护的，所以注释是很重要的。断言可以被认为是对注释的扩展，因为它相当于告诉人们阅读一段始终都是 true 的代码的注释。使用断言，而不是通过注释指定一个语句始终为 true，你可以使用断言来声明它始终是 true 的。

然后，如果你运行含有断言的代码，不必像使用注释那样依赖于仔细代码，但运行代码的时候将会检查你的断言是否为 true，如果它们不是断言错误就会被抛弃。

正如名字暗示的那样，断言被用来断定某些东西应该始终都是 true 的。当程序正常运行时，断言就失效了，不会带来性能开销。当程序员在查找问题时，断言可以被激活，如果任何断言语句不为 true，断言异常就会被抛出。断言是 JDK1.4 中的关键部分，而且不需要在源文件中加入额外的 import 语句。但是，因为过去程序员已经使用了单词 assert 来创建他们自己的断言，编译过程需要一个命令行的参数来告诉编译器使用 JDK1.4 中的断言。这需要如下的形式

javac –source1.4 Myprog.java

如果你以如下形式正常运行程序

java Myprog

断言失效了，不会抛出断言异常。如果你后来需要查明一个问题，并确定所有的断言条目都是 true，你可以像下面那样激活断言来运行程序。

java –enableassertions Myprog

你应该断言什么为 **true**？

断言可以被用在任何你认为应该始终为 true 的地方。例如，一个人的年纪大于 0 应该始终是 true 的。如果一个人的年纪小于 0，你的程序或输出就会有很大的问题。另一个例子，如果你正在登记人们死亡的日期，你的程序（或你的道德）可能会有死亡日期是否可以在未来的问题，所以，你可以对未来的死亡日期进行断言。

例如，如果你正面临 case 语句或一组 if/else 语句，你可能相信代码始终会在到达最后的测试之前退出。想象一下，如果你有一个处理媒体类型的程序。你的程序希望能够处理 jpg，mpg，avi 或 gif 文件。你设置了一个根据文件类型分支的 case 语句。因为你相信文件类型将始终是这些类型之一，如果你到达了 case 语句的末尾而没有分支了，这将是一个很明显的问题，你可以在缺省选项处放置一个断言语句。

你应该在哪里使用断言？

断言不可以用来强制程序的公共接口。一个最常见的程序公共接口是它的命令行参数。因此，传统上程序员会通过查看从命令行传入的 String 数组 args 中的值来检查传递给 Java 程序的命令行。典型地，如果数组没有包含程序期望类型的值，程序将退出并打印出指明正确命令行格式的消息。断言机制的引入不会改变这些。使用断言来检查程序的命令行参数是不合适的，因为断言永远都不会被激活。

使用断言来检查传递给公共方法的参数是不合适的。因为你的公共方法可能会被别人写的程序使用，你无法确定他们是否激活了断言，因此正常运行程序会出错。但是，使用断言来检查传递给私有方法的参数是恰当的，因为这些方法通常是由能够访问源代码的人来调用的。同样的假定可以作用于受保护或同一个包中的受保护方法。如你所见，这些仅仅是指导方针，但是考试可能会询问基于这些指导方针的问题。

断言语法

断言语句由两种格式

简单的

assert somebooleantest

和

assert somebooleantest : someinformativemethod

在第一个简单版本中，断言测试某物是 true 的，如果它不是断言错误则抛弃。例如，如果你正在测试一个人的年龄是否大于 0，你可能创建如下形式的断言

assert (iAge>);

复杂的版本可能是如下形式

assert (iAge) : “age must be greater than zero”;

这个例子很简单，因为右手边的表达式只是一个简单的字符串，但是这可以是任何有返回值的函数调用，例如一个除返回值为 void 以外的方法。

课后测试题

问题 1）下面哪些论断是正确？

1） Using assetions requires importing the java.util.assert package

2） Assertions should be used to check the parameters of public methods

3） Assertions can be used as a substitute for the switch/case construct

4） An assertion that a persons date of death > date of birth is appropriate

问题 2）如果如下代码没有显式激活断言而成功的编译并运行，会发生什么呢？

```
class Language {
    public static final int java = 1;
    public static final int pascal = 2;
    public static final int csharp = 3;
}
public class Mgos {
    static int lang = 0;
    public static void main (String argv []) {
        switch (lang) {
        case Language.java:
            System.out.println (“java”);
            break;
        case Language.pascal:
            System.out.println (“pascal”);
            break;
        case Language.csharp:
            System.out.println (“csharp”);
            break;
        default:
            assert false: lang;
        }
    }
}
```

1） An unmatched parameter exception will be thrown

2） An assert exception will be thrown

3） The program will run with not output

4） Output of “csharp”

问题 3）下面哪些是值得使用断言结构的候选？

1） An input form has a field for a person’s age. If the person entering the date of birth and the date of death enters an age of death that is before the age of birth the assertion mechanism is used to cause a dialog warning box to be shown and the data will not enter the system.

2） A Text Editing program has a file save mechanism. The assert mechanism is used to check if the drive that is being save to really exists. If the drive does not exist an assertion will be thrown generating a warning to the program operator.

3） A program is being created for food preparation that involves cooking meat. Code is included so that if the value of a temperature variable reading appears to be negative an assert

exception is thrown.

4） A school attendance system is being created. In the system is code that will throw an assert exception if a child's age is calculated to be less than zero.

问题 4）如果激活 JDK1.4 的断言，编译如下代码会发生什么？

```
public class Bbridge {
    int iRunningTotal = 0;
    public static void main (String argv []) {
        Bbridge bb = new Bbrige ();
        bb.go (argv [0]);
    }
    public void go (String s) {
        int i = Integer.parseInt (s);
        setRunningTotal (i);
        assert (iRunningTotal > 0) : getRunningTotal ();
    }
    public String getRunningTotal () {
        return "Value of iRunningTotal " + iRunningTotal;
    }
    public int setRunningTotal (int i) {
        iRunningTotal += i;
        return iRunningTotal;
    }
}
```

1） Compile time error, getRunningTotal does not return a Boolean

2） Compile time error malformed assert statement

3） Compilation and no output given a command parameter of 1

4） Compilation and assert error given a command parameter of 0

问题 5）下面的论断哪些是正确的？

1） The assert system introduces no backward compatibility issues

2） The assert system should be used to enforce command line usage

3） Asserts should be used to check for conditions that should never happen

4） Asserts can be used to enforce argument constraints on private methods.

答案

答案 1）

4）An assertion that a persons date of death > date of birth is appropriate

使用断言不需要引入任何包，但是它确实要求 JDK1.4 或更高版本，并且需要对 JDK 工具使用命令行参数。断言不能用来检查方法参数的值，因为在正常（非测试）模式中断言检查会被禁用。一个人死亡的日期比出生日期大的概念始终是正确的，所以使用断言结构是恰当的。

答案 2）

3）The program will run with no output

运行程序时没有从命令行显式激活断言将不会产生断言错误。

答案 3）

3）A program is being created for food preparation that involves cooking meat. Code is included so that if the value of a temperature variable reading appears to be negative an assert exception is thrown.

4） A school attendance system is being created. In the system is code that will throw an assert exception if a child's age is calculated to be less than zero.

选项 1 和 2，以及 2 和 4 之间的重要不同点在于，选项 1 和 2 中断言机制在程序正常运行过程中是必需的。断言对于正常运行程序或标准运行时检查不是必要的。当然，对于选项 3 和 4 你可能希望包含运行时检查而不是断言，但是因为描述没有指明程序正常运行的产出依赖于这个测试，所以使用断言是恰当。

答案 4）

3）Compilation and no output given a command parameter of 1

4） Compilation and assert error given a command parameter of 0

答案 5）

3）Asserts should be used to check for conditions that should never happen

4） Asserts can be used to enforce argument constraints on private methods.

最少的对于程序设计语言运行方式的知识就能指出新特性都会引起向后兼容问题。如果程序员在 JDK1.4 之前使用了单词 assert 作为变量，你将需要在使用 JDK1.4 时传递一个命令行参数来指出这一点。使用断言来检查命令行参数是不合适的，因为断言检查永远都不会被打开。

第 3 章 垃圾收集

为什么想收集垃圾

你可能是一位经验非常丰富的 Java 程序员，但是你未必想过弄清楚垃圾收集的来龙去脉。的确垃圾收集在 Java 程序中有点奇怪。本章中垃圾收集是指释放前面分配的内存，这些内存不会再被程序继续使用。当内存已经变得没用的时候，我们把它们叫做垃圾，它们的存在还会使得其他可用内存空间变得混乱。

Java 语言设计的非常出色，其中之一就是你不用担心垃圾收集。C/C++程序员必须要手动分配和释放内存，这会导致一个问题出现就是“内存泄露”。有些版本的 Windows 程序，比如 Word 和 Excel，可能几次简单的打开和关闭应用程序就会引起某些问题出现。有时候内存泄露可能最终导致系统死机，你不得不重新启动电脑。在成千上万的 C/C++代码中，程序员很可能分配一块内存却忘记释放它。

Java 和垃圾

与 C/C++不同，Java 语言会自动释放不再使用的引用。你不用从成千上万代码中苦苦查找不会再使用的内存。你也不需要知道如何分配合适大小的空间给不同的数据类型，以确保程序的兼容性。因此，看起来你没有必要知道垃圾收集的细节知识。有一种情况例外，就是你想通过考试或者想了解垃圾收集的真实情况。

如果你编写程序过程中需要创建大量的对象和变量，这时候如果知道引用什么时候会被释放是非常重要的。**你需要知道自动垃圾收集的工作原理，你可以建议或者鼓励虚拟机进行垃圾收集，但是记住你不能强迫它作这个工作。**

finalize

Java 语言保证一个对象的 finalize 方法在对象被回收之前会调用。与其他类似垃圾回收的行为不同的是，这里是“保证”。但是 finalize 方法到底做什么呢？

乍一看，finalization 像是 C/C++语言中的析构器，在对象销毁之前清理其资源。不同的是 Java 语言不需要释放资源，因为垃圾回收器会处理内存分配。但是如果你引用了其他外部资源，比如文件信息，那么就有必要在 finalization 中释放资源了，这也是在 JDK 1.4 里面提出的参考。

当垃圾收集器判断出已经没有引用指向这个对象的时候，垃圾收集器就会调用对象的 finalize 方法。

因为垃圾收集器回收垃圾的行为是不确定的，你不知道什么时候他们会执行来收集垃圾。因此你也就没有办法知道什么时候 finalize 方法会被调用。但是，你一定想知道考试对垃圾回收这部分的要求，我们往下看。

垃圾收集的确是一个考点陷阱，因为你没有明显的方法来决定什么时候垃圾收集可用。因此你不能编写下面的代码：

```
if(EligibleForGC(Object){ //Not real code
    System.out.print("Ready for Garbage");
    }
```

正因为如此，你必须掌握下面的原则。

一旦一个对象不被其他任何对象引用的时候，它就变成可回收的对象了。你可以使用 System.gc()来建议垃圾回收器收集垃圾，但是这并不能保证执行。

在方法中声明的本地变量在方法退出的时候就无效了，这个时候方法中的本地变量就成为了可回收的，方法每次执行的时候本地变量都会被重新创建。

无法访问

当代码已经无法再访问对象的时候，这个对象就成为了可垃圾回收的。有两种情况下会出现对象无法再被访问，第一，对象的引用设置为 null；第二，指向这个对象的引用指向了其他的对象。有这样一种考试题目，在代码的某个部分把引用设置为 null，你必须找出在哪里对象成为了可垃圾回收的。这种类型的题目比较简单。但是另外一种情况就不是这么明显了，我们看看下面的代码例子。

```
class Base{
    String s;
    Base(String s){
    this.s = s;
    }
    public void setString(String s){
    this.s = s;
    }
}
public class UnReach{
    public static void main(String argv[]){
    UnReach ur = new UnReach();
    ur.go();
    }
    public void go(){
    Base b1 = new Base("One");
    b1.setString("");
    Base b2 = new Base("Two");
    b1 = b2;

    }
```

什么时候 b1 成为可垃圾回收的呢？假设你不被 b1 设置为空字符串所影响，那么你就可以判断出当 b1 指向 b2 的时候，原来的 b1 成为可垃圾回收的了。

课后测试题

问题 1）下面哪段代码可以建议虚拟机执行垃圾收集？

1) System.free();
2) System.setGarbageCollection();
3) System.out.gc();
4) System.gc();

问题 2) 在下面的代码片断中插入一行代码确保 Integer 对象被垃圾收集器回收。

```
public class Rub{
    Integer    i= new Integer(1);
    Integer    j=new Integer(2);
    Integer k=new Integer(3);
public static void main(String argv[]){
    Rub r = new Rub();
    r.amethod();
}

public void amethod(){
System.out.println(i);
```

```
        System.out.println(j);
        System.out.println(k);
        }
}
```

1) System.gc();
2) System.free();
3) Set the value of each int to null
4) None of the above

问题 3）下面那句话是正确的？

1)You cannot be certain at what point Garbage collection will occur
2) Once an object is unreachable it will be garbage collected
3) Both references and primitives are subject to garbage collection.
3) Garbage collection ensures programs will never run out of memory

问题 4）在哪里第 8 行创建的 sb 对象成为可垃圾回收的？

```
public class RJMould{
        StringBuffer sb;
        public static void main(String argv[]){
        RJMould rjm = new RJMould();
        rjm.kansas();
      }
public void kansas(){
     sb = new StringBuffer("Manchester");
     StringBuffer sb2 = sb;
     StringBuffer sb3 = new StringBuffer("Chester");
     sb=sb3;
     sb3=null;
     sb2=null;
       }
}
```

1) Line 11
2) Line 9
3) Line 12
4) Line 13

答案

答案 1）

4) System.gc();

答案 2）

4) None of the above 你只能建议垃圾回收器运行，但是无法决定他会在代码的哪个部分执

行。注意只有对象的实例才可能成为垃圾回收对象，原始数据类型不会。

答案 3）
1) You cannot be certain at what point Garbage collection will occur
一旦一个对象不能在被访问，那么他将成为可垃圾回收的。但是你不能确定它什么时候会被回收。垃圾回收机制只对对象有效，对原始类型无效。你应该知道垃圾收集不能确保程序不会出现内存不足的情况。但是他能保证不再被使用的内存可以成为可用的。

答案 4）
4) Line 13
第 9 行创建的 sb2 指向了第 8 行创建的对象，直到它成为不可到达的对象的时候，sb 才成为可垃圾回收的。

答案 5）
1) finalize will always run before an object is garbage collected
对象在垃圾回收之前，它的 finalize 方法会被调用。Finalize 方法不能在对象被回收后调用，因为那时候对象已经不存在了。当一个对象不能访问的时候，他就成为了可垃圾回收的，但是你无法保证它什么时候会被回收。选项 4 在 java 中是不正确的，在 C++中正确。

第 4 章 语言基础

目标一 包，引入，内部类，接口

正确识别结构化的包声明，引入子句，类声明（包含内部类在内的所有形式），接口声明，方法声明（包括类运行入口的 main 方法），变量声明和标识符。

目标的注解

这是一个奇怪的使用短语表达的目标。它似乎在要求你理解何时，如何以及为何使用引入子句和包子句，以及应该将接口子句和变量子句放在什么地方。

包

名称package意味着类的集合，有点类似于类库。使用包也有点像使用目录。如果你在一个文件中放置一个包子句，此文件只对同一个包中的其他类可见。包有助于解决命名冲突问题。因为你只能使用这么多有意义的名字作为类名，最终，你可能需要使用或创建相同名称的类。通过在类之前附加一个完整的包名，你可以多次使用相同的名字。包名的使用惯例是用组织的internet域名来创建类。因此，当创建一个叫做Question的类来表示一个虚拟的测试题时，我使用我的网站域名 www.jchq.net来创建目录结构。

WWW 部分无法唯一标识网站的任何信息，所以使用的域名将是 net.jchq。为了在我唯一的包中创建类，我创建了目录 net，并在此之下创建一个叫做 jchq 的目录。接着，在那个目录中我可以创建叫做 Question 的类，类的开头为如下包定义：

package net.jchq.*;

这将赋予你访问此包/目录中任何类的权利。可选地，你可以仅仅指定一个需要获取访问权限的类，使用如下行：

package net.jchq.Question;

引入

import 子句必须出现在任何 package 子句之后和任何代码之前。引入子句不能出现在类中，类声明之后或其他任何地方。

import 子句允许你直接使用类名，而不必使用完整的包名来限定它。一个例子就是类名 java.awt.Button 通常被简写为 Button，只要你已经将如下子句放在文件的起始位置：

import java.awt.*;

如果我随后想要创建我的 Question 类的实例，我只需要引入包或指定此类的完整包名。为了引入其他包中的类，我将需要如下行：

import net.jchq.*;

为了指定类的完整包名，我需要使用如下风格的语法。

jchq.net.Question question = new net.jchq.Question();

你可以想象，经常性地输入完全限定的包名显得不够灵活，所以引入类通常是首选的方案。

请注意，使用引入子句对性能没有影响。这类似于在 DOS（或 Unix）环境中设定一个路径声明。这只是简单的为类设定有效性或路径，并不是直接将代码引入程序中。仅仅在程序中实际地使用类才会影响性能。

你可以在包自己之前放置一段注释，但不能是其他任何内容。你可能会遇到将引入子句放于包子句前面的考试题。

//你可以在包子句之前放置一段注释

package MyPack;

```
public class MyPack {}
```

下面的代码会导致错误

```
import java.awt.*;
//错误：将引入子句放于包子句前面

//语句会导致编译时错误
package MyPack;
public class MyPack {}
```

package 子句可能会包含点号来指定包层次。因此如下代码将不会导致编译错误

```
package myprogs.MyPack;

public class MyPack {}
```

记住，如果你没有在源文件中放置包子句，这将被认为有一个相当于当前目录的缺省包。这与在“1.2 节 定义和访问控制”中提到的可见性有关。

类和内部类声明

一个文件只能包含一个外部 public 类。如果你试图创建一个包含多个 public 类的文件，编译器将会报告特定的错误。一个文件可以包含多个非公共类，但是记住这将为每个类生成单独的.class 输出文件。公共类在文件中的放置位置是没有关系的，只要在文件中仅有一个公共类。

内部类是在 JDK1.1 中提出的。这个想法是为了允许一个类在另一个类中定义，在一个方法中定义，以及创建匿名内部类。这会带来一些有趣的影响，特别是对于可见性。

这是一个简单的内部类的例子：

```
class Outer {
    class inner{}
}
```

这会导致生成如下名称的类文件

```
Outer.class
Outer$Inner.class
```

内部类的定义仅仅在现有的 Outer 类的上下文中可见。因此，如下代码会导致编译时错误

```
class Outer {
    class Inner{}
}

class Another {
public void amethod () {
    Inner I = new Inner();
    }
}
```

涉及到类 Another 的时候，类 Inner 是不存在的。它只能存在于 Outer 类实例的上下文中。因此如下代码运行良好，因为在创建 Inner 实例的时候，有一个指向外部类的 this 实例。

```
class Outer {
    public void mymethod () {
        Inner I = new Inner();
    }
    public class Inner {}
}
```

但是，如果 Outer 类的 this 实例不存在时会发生什么呢。为了弄清楚为此提供的相当古怪的语法的含义，试着将如上例子中的 new 关键字看作属于 this 实例当前的上下文。

这样，你可以改变创建实例的代码行，如下

```
Inner i = this.new Inner ();
```

这样，如果你需要从一个 static 方法或其他没有 this 对象的地方创建 Inner 的实例，你可以把 new 当作属于外部类的一个方法来使用

```
class Outer {
    public class Inner {}
}

class another {
public void amethod () {
    Outer.Inner i = new Outer ().new Inner ();
    }
}
```

尽管有了我口齿伶俐的解释，我发现这个语法不够直观，并在学完 5 分钟后就忘记了。

你很有可能会在考试中遇到这个问题，所以请给予额外的注意。

内部类的一个好处是内部类一般可以访问它的嵌套类（或外部类）的域。不像外部类，内部类可以是 private 或 static。主考者似乎有可能问一些归结为“一个内部类可以是 static 或 private”的问题。

静态内部类的方法当然可以访问其嵌套类的任何静态域，因为那些域将只会有一个实例。

声明在方法中的内部类

内部类可以在方法中创建。这是像 Borland JBuilder 那样的 GUI 生成工具在创建事件处理器时花很多功夫做的事情。

这是一个这种自动生成的代码的例子

```
buttonControl1.addMouseListener (new java.awt.event.MouseAdapter () {
    public void mouseClicked (MouseEvent e) {
        buttonControl1_mouseClicked (e);
    }
});
```

请注意第一个圆括号之后的 new 关键字。它指出在方法 addMouseListener 中一个匿名内部类正在被定义。通常地，这个类可以使用一个名字定义，这可能会使它更容易被人们读懂，但是由于在其他地方不需要对其进行处理，取名字不会有太多帮助。

如果你手动创建这些代码，很容易会被数字以及花括号和圆括号的层次弄糊涂。请注意完整的结构为何是以分号结束的，因为这实际上是一个方法调用的结束。

正如你猜的那样，一个匿名内部类不能由程序员给定构造函数。考虑一下，构造函数是一个没有返回值，并且名字与类名相同的方法。咄！我们在谈论没有名字的类。一个匿名类可以继承其他类或实现单一接口。这个特别的限制似乎不会在考试中考到。

在方法中定义的类的域可见性

定义在方法中的类只能访问嵌套方法中的域，如果他们是被定义为 final 的。这是因为定义在方法中的变量通常被认为是自治的（automatic），例如他们仅当方法执行时才存在。在创建在方法中的类中定义的域可能比嵌套方法的生命周期长。

因为 final 变量不能被修改，JVM 可以确保它的值保持恒定，甚至在外部方法运行终止之后。你很可能在考试中遇到这方面的问题，包括考察作为参数传递给方法的变量状态的问题（是的，他们也必须是 final 的）。

创建接口

接口是 Java 用来解决缺少多继承的方式。有趣的是，Visual Basic 使用关键字 interface 并以与 Java 相似的方式来使用此概念。有时候接口方法被认为面向契约编程。通过关键字“implements”来使用接口。因此，类可以被声明为

```
class Malvern implements Hill, Well {
    public
}
```

主方法

因为 java 中的所有代码必须存在于类中，必须有一个特别的或“有魔力的”的方法来引导程序开始运行。这个方法具有如下署名

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

从声明中的每一项来分析，关键字 public 意味着方法到处可见。static 部分意味着方法属于类本身，而不是属于任何特定的实例。这意味着不需要创建类的实例就可以调用它。单词 void 意味着方法没有返回值。注意单词 main 都是小写的。在圆括号中的部分指出方法接受一个 String 数组的参数。当然，单词 String 必须以大写 S 开头。参数 arg 的名字没有关系，你可以叫它 bicycle 或 trousers 或任何正确的变量名称，它都可以正确运行。但是参数命名为 arg 是一个值得坚持的惯例。因为数组的方括号可以跟在名字或类型之后，将参数声明为 String [] arg 也是可以接受的。

请注意，为了 Sun Certified Java Programmers，这是正确的署名。你可能发现其他类似的署名在现实中也可以运行，但是为了考试（以及未来兼容性的目的）你应该使用这种署名。因为这个方法是静态的，所以它被调用（或被 Java 环境有效跟踪）的时候不需要创建类的实例。同样，因为它是静态的，你不可以操作非静态的方法或数据。因为这样，main 方法经常包含极少的代码，典型地，它包含代码来创建嵌套类的实例，然后调用真正使程序完成工作的非静态方法。

由系统传递给 main 方法的 String 数组包含任何在程序开始时从命令行传入的参数。当然，有了现代图形用户界面环境，更普通的方法是通过点击图标来启动程序，这些都不会给传递参数带来变化。

考试的目标 4.2 明确地要求你理解命令行参数是如何传递给 main 方法的，以及如何访问它们。这就是说……

陈述传递给 main 方法的参数数组的下标值与命令行参数的对应关系。

课后测试题

问题 1）假设有如下代码

```
public class FinAc {
    static int l = 4;
    private int k = 2;

public static void main (String argv [] ) {
    FinAc a = new FinAc();
    a.amethod();
    }

public void amethod () {
    final int i = 99;
    int j = 6;
    class CInMet {
        public void mymethod (int q) {
                        // Here
            }// end of mymethod
        }// End of CInMet

        CInMet c = new CInMet ();
        c.mymethod (i);
}// End of amethod
}
```

如下变量中，哪些在由注释//Here 标记的行上是可见的？

1） l
2） k
3） i
4） j

问题 2）下面哪个选项可以正确编译？

1）

```
// A Comment
import java.awt.*;
class Base {}
```

2）

```
import java.awt.*;
package Spot;
class Base ();
```

3）

```
// Another comment
package myprogs.MyPack;
public class MyPack {}
```

4）

```
class Base {}
import java.awt.*;
public class Tiny {}
```

问题 3）如下论述哪些是正确的？

1） An inner class may be defined as static
2） An inner class may NOT be define as private
3） An anonymous class may have only one constructor
4） An inner class may extend another class

问题 4）从不存在当前 this 引用的代码中如何创建内部类的实例？

1） Outer.Inner i = new Outer ().new Inner ();
2） Without a this reference an inner class cannot be created
3） Outer.Inner i = Outer ().new new Inner ();
4） Outer i = Outer.new ().Inner ();

问题 5）如下哪些是开始执行 Java 程序的 main 方法的正确形式？

1） public static void main (String[] bicycle);
2） public void main (String argv[]);
3） public static int main (String args[])
4） public static void main (String args[]);

问题 6）试图编译如下代码时会发生什么？

```
abstract class Base {
    abstract public void getValue (Base b);
}
public class Robinwood extends Base {
    public static void main (String argv[]) {
    Robinwood rw = new Robinwood();
    rw.main();
    }
    public void main () {
    getValue (this);
    }
    public void getValue (Base b) {
    }
}
```

1） Compile error, only methods can be marked as abstract
2） Compile error, the name “main” is reserved for the startup method
3） Compile error, the parameter to the getValue call is of the wrong type

4） Compilation without error

问题 7）

```
// located in the East end
package spital;
abstract class Spital {
public Spital (int i) {}
}
public class Mudchute extends Spital {
    public static void main (String argv[]) {

    Mudchute ms = new Mudchute ();
    ms.go ();
    }
    public Mudchute () {
    super (10);
    }
    public void go () {
    island();
    }
    public void island () {
    System.out.println (“island”);
    }
}
```

1） Compile time error, any package declaration must appear before anything else
2） Output of 10 followed by island
3） Output of 10 followed by “spital island”
4） Compile time error

问题 8）对于定义在方法中的类，什么规则管理对于嵌套类中变量的访问？

1） The class can access any variable
2） The class can only access static variables
3） The class can only access transient variables
4） The class can only access final variables

答案

答案 1）

1） l
2） k
3） i

定义在方法中的类只能看到来自嵌套方法中的 final 域。但是它可以看到嵌套类中包括私有域在内的域。域 j 没有被定义为 final。

答案 2）

1）

```
//A Comment
import java.awt.*;
class Base {};
```

3)

```
//Another comment
package myprogs.MyPack;
public class MyPack {}
```

任何包子句必须是文件中的第一个条目（除去注释）。引入子句必须在任何包子句之后和代码之前。

答案 3）

1） An inner class may be defined as static

2） An inner class may extend another class

一个匿名类怎么能有构造函数呢？内部类可以被定义为私有的。

答案 4）

1） Outer.Inner i = new Outer ().new Inner ();

答案 5）

1） public static void main (String[] bicycle);

2） public static void main (String args[]);

选项 2 可以编译，但是不能成为程序的启动方法，因为它没有声明为 static。选项 3 不能编译，因为它被声明为返回 int 值。

答案 6）

4）Compilation without error

为一个非启动方法取名为“main”在语法上是正确的，但这是很糟糕的风格。因为类 Robinwood 继承自类 Base，所以可以将其作为参数传递给一个期望得到 Base 类型参数的方法。

答案 7）

2）Output of 10 followed by island

包声明必须出现在除注释外的任何内容之前，注释可以出现在任何地方。

答案 8）

4）The class can only access final variables

注意这一限制适用于嵌套方法中的变量，而不是嵌套类中的变量。

目标二 使用接口

识别正确实现了接口的类，这些接口既可以是 java.lang.Runnable，也可以是试题中完整指定的接口。

接口——面向契约编程

接口是总所周知的“面向契约编程”的一部分。这意味着一个程序员创造了一些东西来迫使其他程序员遵循一组条件。接口同样也被认为是 Java 用来无缺点获取一些多继承好处

的方式。C++语言具有多继承，这意味这一个类可以有多个父类。多继承与单继承的优缺点是编程理论家之间广泛争论的话题。

Runnable 接口

Runnable 接口是线程机制的一部分，线程机制将在其他的考试目标中进行明确地陈述。Runnable 接口指定实现了它的类必须定义一个具有如下署名的方法

public void run ()

使用接口的关键字是 implements，因此，如果你打算创建一个实现 Runnable 接口的类，代码类似于

```
public class MyClass implements Runnable {
    public void run () {}
}
```

当然，为了做一些有用的事，你需要在 run 方法体中添加一些代码，但是仅仅创建一个具有合适署名的 run 方法已经足够完成 Runnable 接口要求的契约了。除非你具有一个完全正确署名的方法，否则你会得到一个编译时错误。

目标三 从命令行传递值

陈述传递给 main 方法的参数数组的下标值与命令行参数的对应关系。

注意：这似乎是一个微小的主题，几乎不值得使之成为一个目标。

这个主题可以找出更具经验的 C/C++程序员，因为 argv []的第一个元素是命令行中程序名称后面的第一个字符串。因此，如果程序运行如下。

```
java myprog myparm
```

元素 argv [0]将包含“myparm”。如果你具有 C/C++背景，你可能认为它包含“java”。Java 不包含与 Visual Basic 中的 Option Base 等价的元素（译者注：VB 中可以使用 Option Base 来限定下标的缺省下界），并且所有的数组都是从元素 0 开始。

以如下程序为例

```
public class MyParm {
public static vooid main (String argv []) {
```

```
        String s1 = argv [1];
        System.out.println (s1);
        }
    }
```

为了强调 argv 是一个 String 数组，我将参数 1 传给一个 String。如果你使用如下命令运行程序

java MyParm hello there

输出结果将是 there，而不是 MyParm 或 hello。

课后测试题

问题 1）假设类 Cycle 中有如下主方法，并且有命令行

java Cycle one two

输出是什么？

```
public static void main (String bicycle []) {
    System.out.println (bicycle [0]);
}
```

1） None of these options
2） Cycle
3） one
4） two

问题 2）如何从命令行中获取传递给主方法的值？

1） Use the System.getParms () method
2） Assign an element of the argument to a string
3） Assign an element of the argument to a char array
4） None of these options

答案

答案 1）

3） one

答案 2）

2） Assign an element of the argument to a string

目标四 识别关键字

识别所有 Java 程序设计语言的关键字。注意：不会出现关于关键字与描述常量之间的深层次区别的问题。

目标的注解：你可能希望在学习较少使用的关键字的基础上解决这个目标，确保你没有延续来自其他你所知道的语言中的“坏朋友”，特别是 C/C++。考试特别强调识别关键字。此目标的第二部分提到的深层次区别是在 JDK1.4 版本的考试中加入的目标。似乎越来越多的人担心 true/false 和 null 是不是关键字。我想你可以从评论中断定你不会被询问关于 true/false 和 null 的问题。

Java 关键字

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

通过使用语言，你将会逐渐认识大部分的 Java 关键字，但是，在考试中可能会出现极少使用的例外和保留词。

一些极少使用的词（当然是对于初学者而言）的例子

volatile

transient
native
strictfp

带星号的词是保留的，现在并没有被使用。注意所有的关键字都是小写的，所以 for 是关键字而 FOR 不是。

课后测试题

问题 1）下面哪些是 Java 关键字？

1） double
2） Switch
3） then
4） instanceof

问题 2）下面哪些不是 Java 关键字？

1） volatile
2） sizeOf
3） goto
4） try

答案

答案 1）

2） double
4）instanceof

注意，switch 中的大写字母 S 意味着它不是关键字，单词 then 是 Visual Basic 中的一部分而不是 Java。

答案 2）

2） sizeOf

这是 C/C++中用来决定一个原始数据在某个特定平台上的大小的关键字。因为 Java 中原始

数据在所有平台上都具有相同大小，所以这个关键字没有被使用。

目标五 未赋值的变量

陈述当没有显式赋值时，使用任何类型的变量或数组元素的影响。

9.2.1. 变量

你可以学习使用 Java 编程而不必真正理解这个目标背后的议程，但是它确实代表着有价值的实际知识。本质上，类级别的变量总是会被赋予一个缺省值，而一个成员变量（包含在方法中）将不会被赋予任何缺省值。如果你试图访问一个未赋值的变量会发生错误。例如

```
class MyClass {
    public static void main (String argv[]) {
        int p;
        int j = 10;
        j = p;
    }
}
```

这段代码将会导致如下错误：

“error variable p might not have been assigned”

从 C/C++给你足够的自由给 p 留下一个任意值的倾向来看，这被认为是一个很受欢迎的改变。如果 p 定义在类级别，它就会被赋予其缺省值，而不会有错误产生。

```
class MyClass {
static int p;
    public static void main (String argv []) {
        int j = 10;
        j = p; System.out.println (j);
        }
}
```

整型数的缺省值是 0，所以这将会打印出 0。

数字类型的缺省值是 0，布尔型为 false，对象引用唯一的缺省值类型是 null。

数组

学习这部分目标需要理解一个简单的规则。任何基本类型的数组元素的值将**总是**被初始化为缺省值，无论数组是否被定义。无论数组定义为类级别还是方法级别，元素值都会被设定为缺省值。你可能会遇到询问一个数组的某个特定元素包含什么值的问题。除非是对象数组，否则答案都不会是 null（或者如果它们被特别的设定为 NULL）。

课后测试题

问题 1）假设有如下代码，元素 b [5]包含什么？

```
public class MyVal {
public static void main (String argv []) {
    MyVal m = new MyVal ();
    m.amethod ();
}

public void amethod () {
boolean b [] = new Boolean [5];
    }
}
```

1）1
2）null
3）" "
4）none of these options

问题 2) 假设有如下构造函数，mycon 的元素 1 包含什么？

```
MyCon () {
    int [] mycon = new int [5];
}
```

1）0
2）null
3）" "
4）None of these options

问题 3）试图编译和运行如下代码时会发生什么？

```
public class MyField {
int i = 99;
public static void main (String argv []) {
    MyField m = new MyField ();
    m.amethod ();
    }
```

```
void amethod () {
    int i;
    System.out.println (i);
}
```

1） The value 99 will be output
2） The value 0 will be output
3） Compile time error
4） Run time error

问题 4）试图编译和运行如下代码时会发生什么？

```
public class MyField {
String s;
public static void main (String argv []) {
    MyField m = new MyFeild ();
    m.amethod ();
    }
void amethod () {
    System.out.println (s);
    }
}
```

1） Compile time error s has not been initialized
2） Runtime error s has not been initialized
3） Blank output
4） Output of null

答案

答案 1）

4）none of these options

数组元素从 0 开始编号，因此数组没有元素 5。如果你试图运行

```
System.out.println (b [5])
```

你会得到一个异常。

答案 2）

1）0

这种情况下，构造函数产生的效果与其他方法没有什么不同。无论在哪里创建，一个整型数组的所有元素都将被初始化为 0。

答案 3）

3） Compile time error

你会得到一个编译时错误，指出变量 i 没有被初始化。类级别的变量 i 会转移你的注意力，因为它会被方法级别版本所覆盖。方法级别的变量不会被初始化为缺省值。

答案 4）

4） Output of null

创建在类级别的变量总是会被赋予一个缺省值。对象引用的缺省值是 null，并且使用

System.out.println 隐式调用 toString 方法时会打印出 null。

目标六 数据类型的范围和格式

陈述所有原始数据类型的范围，声明 String 文字以及使用所有允许的格式，基数和表示法来声明原始数据类型。

目标的注解

这是一个有点令人烦恼但又很容易遇到的目标。你可以书写大量 Java 代码而不需要了解原始数据类型的范围，但是要记住这些细节也不会花很多时间。对能使用所有格式的要求要小心，不要忽略了八进制格式。

整数原始数据类型的大小

当这个目标要求原始数据类型的范围时，我假定它只要求以 2 的幂次表示，而不是确实表示的数值。由于 byte 的大小很直观，基于我对 PC 的基本经验，是 8 比特，在我的记忆中只有三种整型类型需要学习，

整型数据类型的范围

Name	Size	Range
byte	8 bit	-2^{7} to $2^{7}-1$
short	16 bit	-2^{15} to $2^{15}-1$
int	32 bit	-2^{31} to $2^{31}-1$
long	64 bit	-2^{63} to $2^{63}-1$

声明整型数

有三种方式声明整型数。缺省的，如你所期望的是十进制。这里是一些选项

声明 18 为整型数

Decimal	18

Octal	022 (Zero not letter O)
Hexadecimal	0x12

如果你编译并运行这个小类，你每次都会得到 18 的输出。

```
public class Lit {
public static void main (String [] argv) {
    int i = 18;
    int j = 022; //Octal version: Two eights plus two
    int k = 0x12; //Hex version: One sixteen plus two
    System.out.println (i);
    System.out.println (j);
    System.out.println (k);
    }
}
```

Roberts 和 Heller 描述了 6 种声明整型数的方法，因为对于 Java 来说是很少见，字母 X 不是大小写敏感的，16 进制符号中的字母 A 到 F 也是这样。我觉得仅仅记住有三种方式以及字母是非大小写敏感的要更容易一点。

浮点原始数据类型的大小

浮点数是有点奇怪的“野兽”，因为在计算的时候会出现意想不到的结果。引用 Peter Van Der Linden 的话“The exact accuracy depends on the number being represented”。为补偿变量的精确度，你确实会接触到巨大得超出想象的数据。因此，最大的 double 可以存储达到 17 后面跟 307 个 0 的数。所以你甚至可以存储经济报刊上说的 Bill Gates 所拥有的价值那么大的数值（直到 Linux 得到了全世界的控制权，那么整数就能很好的完成任务了）。

浮点类型的范围

float	32 bit
double	64 bit

记住，一个具有小数部分的数据的缺省类型是 double 而不是 float。这会有点让人困惑，因为你可能会认为“浮点数”的缺省类型是 float。你可能会在考试中遇到类似于如下形式的题目。

如下能通过编译吗？

```
float i = 1.0;
```

直觉会告诉你这是可以编译成功的。不幸地是考试不是为考察你的直觉而设计的。这会导致编译时错误，因为它试图将 double 值赋给一个 float 类型。你可以这样修改代码

```
float i = 1.0F;
```

或者甚至是

```
float i = (float) 1.0;
```

使用后缀字母指定数据类型

如上一节中演示的那样，你可以通过一个后缀字母告诉 Java 一段数字文字的类型。如下那些是可用的

指定数据类型的后缀

float	F
long	L
double	D

布尔值和字符值

布尔和字符原始数据类型有点古怪。如果你有 C/C++的背景，请特别注意 boolean 并确保没有从其他语言中带来任何“坏朋友”。boolean 数不能被赋予除 true 或 false 以外的其他的值。true 或 false 的值不等价于 0，-1 或其他任何数字。

char 是 Java 中唯一的未赋值的原始数据类型，它是 16 位长的。char 类型可以用来表示一个 Unicode 字符。Unicode 是 ASCII 码的替代方案，它使用 2 个字节来存储字符而不是 ASCII 码中的 1 个字节。这提供了 65K 个字符，尽管不足以覆盖所有文本，但已经是对 255 字符的 ASCII 码的很大的改进了。国际化是一个完全属于其本身的话题，而且仅仅因为你能够在中文或越南语中表示字符，并不能表示他们可以在标准的英文风格的操作系统上正常显示。

可以通过将字符包含在单引号中来创建 char，如下

char a = ‘z’;

注意使用的是单引号′而不是双引号 " 。

这在以英文为中心的世界中运行良好，但是由于 Java 是一个全球的系统，char 可能会包含任何存在于 Unicode 系统中的字符。这可以使用在 16 进制数之前放置\u 完成，并将整个表达式放入单引号中。

因此，空格符可以被表示为

char c = ‘\u0020’

如果你给一个 char 赋予普通的数字，它将输出为文字字符。这样，如下程序将打印出字母 A（ASCII 码为 65）和空格。

```
public class MyChar {
public static void main (String argv []) {
    char i = 65;
    char c = ‘\u0020’;
    System.out.println (i);
    System.out.println (“This” + c + “Is a space”);
    }
}
```

声明字符串文字

String 类型不是原始数据类型但它是如此重要以至于在某些场合 Java 把它当作原始数据。特性之一就是可以声明字符串文字而不需要使用 new 来初始化类。
String 文字相当易懂。确定你记住了 String 文字包含在双引号中，而 char 文字使用单引号。像这样

String name = “James Bond”

更多关于 String 类的信息请看目标 9.3 和 5.2。

课后测试题

问题 1）下面哪些可以编译成功？

1） float f = 10f;
2） float f = 10.1;
3） float f = 10.1f;
4） byte b = 10b;

问题 2）下面哪些可以编译成功？

1） short myshort = 99S;
2） String name = ‘Excellent tutorial Mr Green’;
3） char c = 17c;
4） int z = 015;

问题 3）下面哪些可以编译成功？

1） boolean b = -1;
2） Boolean b2 = false;
3） int i = 019;
4） char c = 99;

答案

答案 1）

1） float f = 10f;
2） float f = 10.1f;

没有这样的 byte 表达形式。选项 2 将会导致错误，因为一个含有小数部分的数字的缺省类型是 double。

答案 2）

4）int z = 015;

不存在字母 c 和 s 这种文字指示符，一个 String 必须包含在双引号种，而不是例子中的单引号。

答案 3）

2）boolean b2 = false;

4）char c = 99;
选项 1 显然是错的，因为 boolean 只能被赋予 true 或 false。选项 3 有点狡猾，因为这是正确的声明八进制数的方式，但是在八进制中你只能使用数字 0 到 7 而不能是 9。或许这儿有点小把戏。

第 5 章 运算符和赋值

目标 1，应用运算符

确定使用任意运算符的结果，包括运算符赋值和 instanceof 来操作任意类型类的作用域或可达域以及以上的组合。

instanceof 运算符

instanceof 运算符是一个很陌生的东西，在我眼里，它更像是一个方法而不是一个运算符。你可能没用过它而写了大量的 Java 代码，但为了考试的目的，你需要弄懂它。instanceof 运算符在运行时测试一个类的类型然后返回一个布尔值。它一般用来说：这个类是一个 instanceof 的那个类吗？
如果你像下面这样的小地方用它，看起来不是很有用

```
public class InOf {
        public static void main(String argv[]){
        InOf i = new InOf();
                if(i instanceof InOf){
                    System.out.println("It's an instance of InOf");
                  }//End if
        }//End of main
}
```

你可能会认为这段代码会输出
"It's an instance of InOf"

但是当你访问了一个涉及到向下多层的对象引用的时候，情况可能会改变。你可能有一个把组件作为参数的方法，它可能真正指向一个 Button，Label 或其他任何东西。这种情况下，instanceof 运算符就可以用来测试对象的类型，执行匹配的角色，然后调用适当的方法。用下面的代码举例说明：

```
import java.awt.*;

public class InOfComp {
        public static void main(String argv[]){
        }//End of main
```

```
        public void mymethod(Component c){
        if( c instanceof Button){
                Button bc = (Button) c;
                bc.setLabel("Hello");
                }
                else
        if (c instanceof Label){
                Label lc = (Label) c;
                lc.setText("Hello");
                }
        }//End of mymethod
}
```

如果运行时的测试和角色匹配没有执行适当的方法，setLabel 和 setText 将不可用。注意，instanceof 测试反对一个类名但不反对类的对象引用。

+运算符

像你期望的一样，+运算符会把两个数相加。因此下面的代码将会输出 10

```
int p=5;
int q=5;
System.out.println(p+q);
```

+运算符在 Java 中是一个罕见的操作符重载的例子。C++程序员习惯于能够重载运算符为他们定义的任何意义。Java 程序员没有这种便利，但是由于对于字符串来讲，加号用来做串联是很有用的。因此，下面的代码将编译

```
String s = "One";
String s2 = "Two"
String s3 = "";
s3 = s+s2;
System.out.println(s3);
```

这段代码将会输出字符串 OneTwo。注意，两个连接的字符串中间没有空格。
如果你是 Visual Basic 背景的程序员，下面的语法可能不熟悉

```
s2+=s3
```

这句代码在 Java 中可以表达成更接近于 Visual Basic 形式的语句

```
s2= s2+s3
```

在某些情况下，Java 可能在后台调用 toString 方法。就像名字显示的那样，这个方法会试着转换为一个 String 表达。对一个整型来讲，这就意味着数字 10 在 toString 调用后会返回字符串“10”
这点在下面的代码中有所展现。

```
int p = 10;
String s = "Two";
```

```
String s2 = "";
s2 = s + p;

System.out.printlns(s2);
```

这段代码会输出 Two10

记住，只有+运算符可以对字符串进行重载，如果你对字符串使用除号和减号（/ -），你会得到一个错误。

为不同类型的原始型变量赋值

一个布尔类型不能赋值给除了布尔类型外的变量。对于 C/C++程序员来讲，记得这说明一个布尔类型不能赋值为-1 或 0，而一个 Java 布尔类型不能用零或非零来替换。
除了布尔型之外，我们学习这个目标的一般规则就是扩展转换是允许的，当然不考虑精确性的危险。由于缩小转换会产生降低精确性的结果，因此是不允许的。对于扩展，我的意思是一个例如 byte 这样的变量，占用一个 byte（8bit），可以赋值给一个 integer 这样占用多个 bit 的变量。

如果你试着将一个 integer 赋值给一个 byte，你会得到一个编译时错误

```
byte b= 10;
int i = 0;
b = i;
```

原始类型可能赋值给一个“更宽”的数据类型，一个 boolean 只能赋值给另一个 boolean。你可能预料到你不能将一个原始类型赋值给一个对象或相反的操作，这包括原始类型的包装类。因此，下面的代码是非法的

```
int j=0;

Integer k = new Integer(99);

j=k; //Illegal assignment of an object to a primitive
```

赋值对象和赋值原始类型的一个重要区别是，原始类型在编译时被检查而对象在运行时被检查。我们将会在后面提到，当一个对象在编译时没有被完全处理的话，还会有重要的含义。

你当然可以运行一个角色来阻止一个变量适应一个窄一些的数据类型。一般不建议你放任精确性降低。但是如果你真的想这样干，Java 使用 C/C++的习惯，将数据类型封装进（）中，因此，下面的代码将会编译运行。

```
public class Mc{
public static void main(String argv[]){
    byte b=0;
    int i = 5000;
    b = (byte) i;
    System.out.println(b);
    }
```

```
}
```

输出结果是

-120

可能不是很有必要这么干。

给不同类型的对象引用赋值

将一个对象引用赋值给另一个的一般规则是，你可以对继承树向上赋值但不能向下复制。你可以这样想，如果你将一个子类的实例赋值给基类，Java 知道哪个方法会在子类中。但是一个子类可能有基类没有的额外方法。你可以使用操作符强制转换。

对象引用可以从子类向基类赋值。

下面举例说明如何向上转换对象引用。

```
class Base{}

public class ObRef extends Base{
public static void main(String argv[]){
        ObRef o = new ObRef();
        Base b = new Base();
        b=o;//This will compile OK
        /*o=b;    This would cause an error indicating
                  an explicit cast is needed to cast Base
                  to ObRef */

        }
}
```

++和-运算符

你可能因为发现在一个并非微不足道的 Java 程序没有使用++或-运算符而感到巨大的压力，因而很容易认为你知道的一切只是为了测试的目的而需要知道的知识。这些运算符可以用于变量的前增或者后增。如果你不明白它们的差别，可能会因为一个相当容易的问题而导致测试丢分。举例来说，下面的代码编译运行后，你认为它会向控制台发送什么呢？

```
public class PostInc{
    static int i=1;
    public static void main(String argv[]){
    System.out.println(i++);
    }
}
```

如果你编译运行了这段代码，可以看到输出是 1 而不是 2。这是因为++放在 i 的后面，因此自增（加 1）会在这行运行后发生，对于-运算符也是一样的规则。

位移运算符

我恨整个的位移部分。它需要你的大脑充满了非直觉的能力，因此很少有程序员用它。现实中的典型应用的例子是密码系统和低级别的镜像文件。你可能写了大量的 Java 程序但却从没自己移动过一个 bit。但是学习它的大部分的原因是为了针对测试而不是为了其他想法。测试一般会有至少一个问题是关于位移运算符的。如果你具有 C++背景，你可能会被误导成认为你的所有 C++知识都可以直接转换成 Java 语言。

为了清楚的理解这点，你必须用二进制方式思考，也就是知道每个 bit 位的值
32, 16, 8, 4, 2, 1

不仅仅是欣赏二进制，你还需要从整体上把握“高度褒扬的二进制”编号系统。在这个系统描述中，第一 bit 位表示这个数是正数还是负数。尽量依靠你的直觉，你会发现当你理解二进制系统的工作原理类似于汽车里程表时，事情开始变得奇怪了。想象每个轮子有一个 1 或者 0 在上面。如果你想把下面的显示回退的时候
00000000 00000000 00000000 00000001
如果后退了一下，就会显示
11111111 11111111 11111111 11111111 11111111
这表示-1。如果再后退一下，会显示
11111111 11111111 11111111 11111111 1111110

这些例子有些过于单纯化了。直到我学习 Java 程序测试，我都只是认为二进制系统只是利用第一位来表明标记部分。但你可以看到，实际情况要复杂的多。为了帮你多理解一点关于符号的知识，我写了一个相当简单的程序，它会显示一个在命令行给定的数的 bit 模式。也可以改进为八进制形式，但同样很容易的得到大体的结果。

```
public class Shift{
    public static void main(String argv[]){
    int i = Integer.parseInt(argv[0]);
    System.out.println(Integer.toBinaryString(i));
    }
}
```

如果你是从 C/C++背景转到 Java，你可以为 Java 中的右移运算符比 C/C++稍微明确一点而得到一点安慰。在 C/C++中，右移运算符可能是取决于编译器执行的有符号数或无符号数。如果你是从 Visual Basic 背景转来，恭喜你可以在低级别的情况下编程了。

注意，本节的目标仅仅是要求你理解应用这些运算符到 int 值后的结果。这比应用运算符到 byte 或者 short 上显得容易一些，尤其是负数，更有可能得到不可预料的结果。

正数的无符号右移

我以无符号右移开始是因为它是最怪异的位移，而且需要对二进制表示法有充分的理解。而处理负数时会更加的怪异，因此我以正数开始。无符号右移操作将一个数字作为一个纯粹的 bit 模式，忽略特定的标识位。记住，一旦你开始将一个数字作为一系列 bit 时，任何 bit 级别的操作可能带给你将它作为一个普通数字时想不到的结果。

无符号右移操作包括两个操作数，第一个数字是需要位移的数，跟在运算符后面的数字

是需要位移的位置，例如下面
3 >>> 1

表示你将数字 3 的 bit 向右移动一位。

二进制补码方式的系统意味着数字开头的 bit 位表示它是正数还是负数。如果是 0，表示数字是正数，如果是 1，表示它是负数。无符号数的右移的第一位总是被 0 补上，这就意味着一个无符号数右移操作总是得到正数的结果。
如果你能想起数字 3 的二进制形式
011
并且将它右移一位
3 >> 1

你会得到
001

注意，有新值的位离开了数的末端并被有效的抛弃了。
如果你执行了两位的右移，你可能因为数字变成了 0 并且 0 覆盖了所有的 bit 位置而感到有点惊讶。如果你一直增加右移的数量，例如 6 位，10 位或 20 位，你会发现像你预料的那样结果一直是 0。可是如果你坚持到
3 >>>32

得到了令人惊讶的结果 3，为什么会这样？

在移动之前的后台，一个模 32 被作为操作数。模运算符，在 Java 中指一个数被另一个数通过%字符除，然后返回余数。同时一个数如果小于求模的数，将会直接返回原值。同时，如果一个数位移不到 32 时，模运算符不会注意到这种情况，一旦到了 32，模运算符就会起作用了。

因此，32%32 返回 0，当作没有东西剩余而使运算符作用于

3 >>> 32

的值是 3，也就是说 3 被移动了 0 位。
我开始依靠直觉没能发现这点，因此我写了下面的代码

```
 public class shift{
   static int i=2;
    public static void main(String argv[]){
            System.out.println(32 % 32);
            System.out.println( 3    >>> 32);
            }
 }
```

这段代码的输出是
0
3
一个模 32 在位移操作数执行时起作用，这影响超过 32 位的位移。

负数的无符号右移

一个负数的无符号右移通常会得到一个正数的结果。我说的通常是因为有一个例外，就是你移动的包括符号位的原始数刚好是在 32 位结束。像刚才解释的，你通常得到一个正数的原因是无符号右移把第一个符号位用 0 代替了，这表明是一个正数。
一个负数的无符号右移有时看起来很怪异。看下面的代码
System.out.println(-3 >>> 1);
你可能认为会得到这样的数
1

也就是说符号位被 0 代替，使它成为正数，然后右移一位。但这是不会发生的，真正的结果是
2147483646
有些奇怪但却是事实。

这种奇怪的结果的背后原因跟二进制数的表示方式有关。如果你将数字的表示想象成汽车里程表代表的轮子，当你从最大可能的数开始往下数到 0 会发生什么呢？然后再回到低于 0 的第一个数？所有的数位包括表示负数的符号位都会变成 1。当你执行无符号右移时，你打破了这种数字表示方式，仅仅把符号位当作另一个数。因此，即使你从例子中-3 这样的很小的负数开始，你也会得到一个很大的正数。你可能会在测试中遇到这样的问题，问你一个负数的无符号右移的结果。正确的答案可能看起来很不可靠。
一个很小的负数通过无符号右移，可能得到一个很大的正数返回值。
有符号移动运算符<<和>>
<<和>>运算符用 0 设置新位。因此，下面的例子
System.out.println(2 << 1)

这句代码将数字 2 左移一位然后将最右边一位置 0
因此，值
010
变成
100

即十进制 4。你可以认为这个操作每次将原数翻倍。因此下面的代码
System.out.println(2 << 4)
结果是 32
你可以这样认为
2*2=4（第一次位移）
2*4=8（第二次位移）
2*8=16（第三次位移）
2*16=32（第四次位移）

当你移动到数字结尾时，这种思考可能导致非常错误的结果，因此下面的代码
System.out.println(2<<30)
结果是
-2147483648

这看起来相当不符合直觉，但是你可以想到数字 2 可以移动最多的位置，现在变成了二进制整型可以表示的最大的负数。如果你再移动一位
system.out.println(2 << 31)

结果是 0，因为每一位都变成了 0，而数字 2 已经到了末端而被抛弃了。

随着有符号右移，左边的数位（新的）在移动前带着符号位（作为对照，左移的新位置被置 0）。这意味着右移将不会影响结果数的符号位。

2 >> 2;

这次将数字 2 的所有位右移两个位置，因此，值

0010

变成

0000

或者说是十进制的 0（我认为在所有进制中 0 都是这样）

这跟执行一个重复的 integer 除法等价。在这个例子中，所有的位置都置 0。

有符号右移运算符的结果数字有同样的符号位。

我创建了一个 applet，允许你尝试各种移动运算符，并且查看十进制或者 bit 模式的结果。我已经在网页中包含了这个 applet 代码，你可以看看它是如何工作的。

BitShift Applet

运算符优先

运算符优先是指哪个运算符执行的优先权的顺序。下表是运算符优先的总结

Operator Precedence

()

++expr --expr +expr -expr ~ !

* / %

+ -

<< >> >>>

< > <= >= instanceof

== !=

&

^

|

&&

||

? :

= += -= *= /= %= &= ^= |= <<= >>= >>>=

我放在同一行的运算符有同样的优先权。通常在真正编程的时候，你将会用圆括号指定你期望的执行的表达式的顺序。这意味着你可以不必真正掌握优先级的顺序并且可以让读你代码的其他程序员更清楚。但你可能在测验中遇到建立在运算符优先级上的问题，尤其是常用的运算符如+，-，*。
如果运算符优先级的概念对你没什么意义，你可以试试下面的代码会输出什么

```
public class OperPres{
    public static void main(String argv[]){
    System.out.println(2 + 2 * 2);
        System.out.println(2 + (2 * 2));
    System.out.println(8 / 4 + 4);
    System.out.println(8 /(4 +4));
    int i = 1;
    System.out.println(i++ * 2);

    }
```

第一条语句 2+2*2 意味着 2+2 的值乘 2 得到输出结果 8，还是表示 2*2 再加 2 得到结果 6 呢？关于其他计算的类似问题可能被问到。顺便说一下，这个程序的输出结果是 66612。

问题

问题 1）在下面给定的类中，哪一个能够不出错的编译？

```
interface IFace{}
class CFace implements IFace{}
class Base{}
public class ObRef extends Base{
public static void main(String argv[]){
        ObRef ob = new ObRef();
        Base b = new Base();
        Object o1 = new Object();
        IFace o2 = new CFace();
        }
}
1) o1=o2;
2) b=ob;
3) ob=b;
```

4) o1=b;

问题 2）在下面给定的包含变量的语句，哪一个能够不出错的编译？

String s = "Hello";
long l = 99;
double d = 1.11;
int i = 1;
int j = 0;

1) j= i <<s;
2) j= i<<j;
3) j=i<<d;
4)j=i<<l;

问题 3）给定下面的变量

char c = 'c';
int i = 10;
double d = 10;
long l = 1;
String s = "Hello";

哪一个能够不出错的的编译？

1) c=c+i;
2) s+=i;
3) i+=s;
4) c+=s;

问题 4）下面的语句会输出什么？

System.out.println(-1 >>>1);

1) 0
2) -1
3) 1
4) 2147483647

问题 5）下面的语句会输出什么？

System.out.println(1 <<32);

1) 1
2) -1
3) 32
4)-2147483648

问题 6）下面的哪个语句是正确的？

1) System.out.println(1+1);
2) int i= 2+'2';
3) String s= "on"+'one';
4) byte b=255;

问题 7）当你试着编译运行下面的代码时会出现什么情况？

```
Public class Pres{
      public static void main(String argv[]){
      System.out.println( 2 * 2 | 2);
     }
}
```

1) Compile time errors, operators cannot be chained together in this manner
2) Compilation and output of 4
3) Compilation and output of 6
4) Compilation and output of 2

问题 8）当你试着编译运行下面的代码时会出现什么情况？

```
public class ModShift{
      static int i = 1;
      static int j =1;
      static int k = 0;
      public static void main(String argv[]){
      i = i << 32;
      j = j >>32;
      k = i + j;
      System.out.println(k++);
      }
```

1)Compile time error
2) Compilation and output of 3
3) Compilation and output of -3
4) Compilation and output of 2

问题 9）当你试着编译运行下面的代码时会出现什么情况？

```
public class Shift{
      static int i;
      static int j;
      public static void main(String argv[]){
      i = 2;
      j = i <<31;
      i =i++;
      System.out.println(j);
      System.out.println(i);
      }
}
```

1) -2147483648 followed by 2
2) -2147483648 followed by 3
3) 0 followed by 3
4) 0 followed by 2

问题 10）下面程序的输出是什么？

```
public class Mac{
      public static void main(String argv[]){
      System.out.println( -1 >>>1    & 2);
    }
}
```

1) 2147483647
2) -1
3) 10
4) 2

答案

答案 1）
1)o1=o2;
2)b=ob;
4)o1=b;

答案 2）
2)j= i<<j;
4)j=i<<l;

答案 3）
2)s+=i;
如果你想测试各种可能出现的情况，可以试着编译下面的代码

```
public class Llandaff{
        public static void main(String argv[]){
        Llandaff h = new Llandaff();
        h.go();
        }

        public void go(){
                char c = 'c';
                int i = 10;
                double d = 10;
                long l = 1;
                String s = "Hello";
                //Start commenting these out till it all compiles
                c=c+i;
                s+=i;
                i+=s;
                c+=s;
        }
}
```

答案 4）

4) 2147483647

即使你的大脑中可能没有出现这个数字，对于无符号右移的理解也能告诉你其他答案是错误的。

答案 5）

1) 1

bit 位会随着左移运算符“滚动”，因此

System.out.println(1 <<31);

的结果将是-2147483648

答案 6）

1) System.out.println(1+1);

2) int i= 2+'2';

第三个选项是不正确的，因为单引号指明一个字符而不是字符串。

第四个选项不会被编译，因为 255 超过了一个 byte 的范围

答案 7）

3) Compilation and output of 6

运算符的优先级高于|运算符。因此，该计算等价于（2）|2，或者认为是 4|2。|运算符比较每个位置的 bit 位，如果任意数字的该位为 1，则此位的输出也是 1。4 的顺序是 100，2 的顺序是 10，因此|运算符的结果为 110，即十进制的 6

答案 8）

4) Compilation and output of 2

当你移动数字的位置为 32 位时，模 32 会在移位时执行。32%32 是说如果你用 32 除 32 时会剩多少，答案是 0。因此数字被移动 0 位。也就是返回原数。这个问题有点阴险，因为输出结果中使用了后增运算符++。这表示在当前行结束执行后数字被增加，因此 1 加 1 得到 2 被输出。

答案 9）

4) 0 followed by 2

这道题没有看起来那么难。结合 int 型数的位数及数字 2 的 bit 模式和对于有符号左移运算符的理解，得知数字 2 的唯一一个 bit 位会被抛弃，所有 bit 位置 0。

输出结果中包括 2 而不是 3 是因为后增运算符在=运算符赋值后将 i 加 1。

答案 10）

4) 2

这个问题可能让你有些发狂的举动。但是如果你有正确的背景知识的话，你可以得到答案的。如果你了解二进制表示法，你应该知道-1 是在 int 型数的每一位都是 1。根据运算符优先级，你可以知道>>>运算符在&前执行。如果你去掉&运算符，你会发现输出结果是选项 1 中给出的最大数。但是由于&执行，结果是 2，表示输出中的 bit 位当两个数字的位都为 1 时才为 1。因此输出为 2。

目标二 equals 方法

确定在任何 java.lang.String,java.lang.Boolean 和 java.lang.Object 类的对象应用布尔类型的 equals(Object)方法的结果。

如果你有 Visal Basic 的背景（像我一样），用"="号比较两部分变量的想法是很奇怪的.然而事实上,对于通常应用的字符串的引用来讲，这是相当重要的。为了测验的目的，你可能被问到关于 equals 操作符对于对象引用的引用和布尔类型的引用这样的问题.注意是关于布尔类的问题而不是关于布尔基本类型(你不能用它来调用方法).

equals 和==的不同点

equals 方法可以被认为是对两个对象值的深层比较，而==操作是浅层比较。equals 方法比较两个对象的所指对象，而不是两个指针本身（如果你承认 Java 有指针）。这种间接方式对 C++程序员可能很清楚，但是在 Visal Basic 中没有直接的比较方式。

将 equals 方法用于字符型

equals 方法返回一个布尔型基本类型值。这表明它可以被用于 if，while 或者其他循环语句。它可以被用于使用==操作符比较基本类型的情况。当比较字符串时 equals 方法和==操作的执行有一些不同结果。对于字符串恒量和它被 Java 处理的方式是很混乱的。

Java 中有两种方法创建字符串。一个方法是用 new 操作符，这是通常的字符串创建方法

```
String s = new String("Hello");
```

但是更简短的方法是

```
String s= "GoodBye";
```

通常这两种创建字符串的方法有些微不同，但是在考试中往往会问到这个不同。

两种创建字符串的方法，当用于字符串时有相同的结果，但不用 new 关键字会创建 Java 字符串池中指向同一个字符串的指针。字符串池是 Java 存储资源的一种方法。举例说明这个结果

```
String s = "Hello";
String s2 = "Hello";
if (s==s2){
    System.out.println("Equal without new operator");
    }
```

```
String t = new String("Hello");
string u = new String("Hello");
if (t==u){
    System.out.println("Equal with new operator");
    }
```

在上一个目标中你可能认为第一个输出"Equal without new operator"不会出现，因为 s 和 s2 时不同的对象，但是==运算符判断两个对象的指针，而不是他们的值。因为 Java 存储资源的方式是重用不用 new 关键字创建的字符串，所以 s 和 s2 有相同的“地址”，所以会输出以下字符串。

"Equal without new operator"

但是对于第二组字符串 t 和 u，new 操作符迫使 Java 创建不同的字符串。因为==操作符之比较两个对象的地址而不是值，t 和 u 有不同的地址，所以"Equal with new operator"将不会输出。

关键概念

equals 方法用于字符串，但是字符串被创建后，会执行字符和字符的比较。

应用字符串池的作用，以及使用==和 equals 方法的区别不是很显而易见，
尤其是如果你有 Visal Basic 的背景。理解它的最好方法是自己写个例子看它是如何工作的。试试用 new 和不用 new 方法创建字符串。

在布尔型上应用 **equals** 方法

理解在 java.lang.Boolean 上应用 equals 运算是可能的要求。Boolean 是 boolean 基本类型的封装类型。它是对象型的可以在其上应用 equals 方法。

依照 JDK 文档，equals 方法用于 Boolean 封装类型，“只是在参数非空而且布尔对象有相同的布尔值时返回真”。

例如

```
Boolean b1 = new Boolean(true);
Boolean b2 = new Boolean(true);
if(b1.equals(b2)){
    System.out.println("We are equal");
    }
```

boolean 和 Boolean 只有微小的区别，当你对 Java 的 if 运算非常熟悉，你就知道不能像 C/C++程序员一样应用隐含的方式，就像这样

```
int x =1;
if(x){
    //do something, but not in Java
    }
```

这在 Java 中不能执行，因为 if 操作的变量必须是 boolean 判断，Java 没有 C/C++中任何非空值都可被认为真的概念。但是以下 Java 代码可以通过

```
boolean b1=true;

if(b1){
    //do something in java
    }
```

虽然这是不好的编程方法，但是在语法上是正确的。因为 if 操作的变量是 boolean 类型。

在对象类型上应用 **equals** 方法

对于 Java 的基本设计，任何类的实例也是 java.lang.Object 的实例。试试 equals 判断对象类型的返回值应用 toString()方法.对于对象变量 toString 方法简单返回内存地址。所以与使用==操作的结果一样。因为 Java 不是设计成操作内存地址和指针的，所以这不是个有用的判断。

看下面的例子

```
public class MyParm{
public static void main(String argv[]){
        Object m1 = new Object();
        Object m2 = new Object();
        System.out.println(m1);
        System.out.println(m2);
        if (m1.equals(m2)){
                System.out.println("Equals");
                }else{
                System.out.println("Not Equals");
                }
        }
}
```

如果你编译运行这段代码，会得到如下输出

```
java.lang.Object@16c80b
java.lang.Object@16c80a
Not Equals
```

这些奇怪的值是内存地址，大概根本不是你想得到的。

问题

问题 1)编译运行以下代码时会发生什么情况？

```
public class MyParm{
public static void main(String argv[]){
        String s1= "One";
        String s2 = "One";
        if(s1.equals(s2)){
                System.out.println("String equals");
                }

        boolean b1 = true;
        boolean b2 = true;
        if(b1.equals(b2)){
                                System.out.println("true");
                                }
        }
}
```

1) Compile time error
2) No output
3) Only "String equals"
4) "String equals" followed by "true"

问题 2) 编译运行以下代码时会发生什么情况？

```
String s1= "One";
String s2 = new String("One");
if(s1.equals(s2)){
                System.out.println("String equals");
            }
Boolean b1 = new Boolean(true);
Boolean b2 = new Boolean(true);
if(b1==b2){
            System.out.println("Boolean Equals");
    }
```

1) Compile time error
2) "String equals" only
3) "String equals" followed by "Boolean equals"
4) "Boolean equals" only

问题 3)编译运行以下代码的结果是什么？

What will be the result of attempting to compile and run the following code?

```
Object o1 = new Object();
Object o2 = new Object();
o1=o2;
if(o1.equals(o2))
        System.out.println("Equals");
}
```

1) Compile time error
2) "Equals"
3) No output
4) Run time error

答案

答案 1)
1) Compile time error
b1.equals() 这一行会引发错误，因为 b1 是简单类型，简单类型没有任何方法。如果创建基本类型的封装类 Boolean 你就可以应用 equals 方法。

答案 2)
2) "String equals" only
用==操作符简单判断基本类型的封装类 Boolean 的一个实例的内存地址。

答案 3)
2) "Equals"
因为一个对象的实例可以赋值给另一个对象用
o1=o2;
它们现在就指向同一个内存地址，equals 方法判断将返回 true。

目标三 &、|、&&和||运算符

在一个包含运算符&、|、&&、||和值已知的变量的表达式中，指出哪个运算符被求值，表达式的值是多少？

很容易忘记哪个逻辑运算用哪个运算符和它们所做的操作，确保你可以在考试中说出它们的区别。如果你初次接触这些运算符，你可能值得花时间好好记忆你才不会对它们这些按位运算符和逻辑运算符的操作搞乱。你可能会记得“双逻辑”这种表达很奇怪。

逻辑运算符的短路效应

逻辑运算符（&&、||）在用于“短路”逻辑像 C/C++的 AND 和逻辑 OR 操作时有一点特别的结果。如果你来自 Visal Basic 背景这就有些奇怪了，因为 Visal Basic 会计算所有的操作数的值。如果你理解 AND，你就会理解 Java 的方法，如果第一个操作数为假，第二个操作数的值就没有作用了，所有的结构都为假。对于逻辑 OR 也是，如果第一个操作数为真，所有的计算结果将为真，因为只要一个操作数为真最后的结果就为真。这种依靠一边结果的压缩计算可能有一个结果。请看下面的例子

```
public class MyClass1{
                        public static void main(String argv[]){
                        int Output=10;
                        boolean b1 = false;
                        if((b1==true) && ((Output+=10)==20))
                        {
                        System.out.println("We are equal "+Output);
                        }else
                        {
                        System.out.println("Not equal! "+Output);
                        }
                }
}
```

"Not equal 10"会被输出。这说明 Output+=10 这个运算永远不会被执行，因为在第一个操作数的值为 false 时运算就停止了。如果你把 b1 的值改成 true，运算就会像你想得一样执行，输出会是"We are equal 20".

当你真的不想在有任何值为 false 时进行其它运算时，这也许有时是便捷的方法，但是当你完全不熟悉时这也许会产生意外的结果。

按位运算符

&和|运算符用于做整形的按位与和或操作.在考试中你会遇到这样的问题,给出一个十进制的数,然你用按位与和或运算计算.要执行这些操作你需要熟悉从十进制到二进制的转换,并且知道其比特形式.这有一个典型的例子

下面运算的结果是什么?
3 | 4

3 的二进制比特形式是
11

4 的二进制比特形式是

100

要执行二进制或运算,两个数的每个比特都要互相比较.如果任一个比特为 1 则结果中的比特数为 1.所以这个操作的结果的二进制形式是
111
也就是十进制的 7.

目标没有特别要求你知道按位 XOR 运算,用^符号执行。

用二进制思考

如果你感到用二进制思考不舒服（我更习惯用十进制思考），你也可能想做一些练习来掌握这个问题和二进制转换操作。如果你使用 windows 你可能发现用计算器的科学模式很有用。你可以在标准模式下选则 View 和 switch 来变成科学模式。在科学模式中，你可以转换数值来看它的十进制和二进制模式，这会显示数的二进制形式。有一个很方便的窍门，我如果在写比特转换 applet 之前知道就好了，就是怎样用整形来表示比特形式。这里有一个实现这个的小程序。

```
public class BinDec{
public static void main(String argv[]){
        System.out.println(Integer.parseInt("11",2));
        System.out.println(Integer.toString(64,2));
        }
}
```

如果你编译运行这段程序，将会得到输出
3
1000000

注意程序怎样把二进制 11 转换成十进制的对应数 3，又是怎样把十进制数 64 转换成相应的比特形式。每个方法的第二个参数是基数。所以在这个例子中会把书转换成 2 为基数的，而我们常常会用 10 为基数的数。

问题

问题 1)你尝试编译运行以下代码时会发生什么情况

```
int Output=10;
boolean b1 = false;
if((b1==true) && ((Output+=10)==20)){
    System.out.println("We are equal "+Output);
    }else
    {
```

```
        System.out.println("Not equal! "+Output);
}
```

1) Compile error, attempting to perform binary comparison on logical data type
2) Compilation and output of "We are equal 10"
3) Compilation and output of "Not equal! 20"
4) Compilation and output of "Not equal! 10"

问题 2)下面一行代码会有什么输出

```
System.out.println(010|4);
```

1) 14
2) 0
3) 6
4) 12

问题 3)下面哪项编译没有错误

1)

```
int i=10;
int j = 4;
System.out.println(i||j);
```

2)

```
int i=10;
int j = 4;
System.out.println(i|j);
```

3)

```
boolean b1=true;
boolean b2=true;
System.out.println(b1|b2);
```

4)

```
boolean b1=true;
boolean b2=true;
System.out.println(b1||b2);
```

答案

答案 1)

4) Compilation and output of "Not equal! 10"
输出是"Not equal 10". 这表明运算 Output+=10 没有被执行，因为运算在第一个操作数被算出为 true 时就停止了。如果你把 b1 的值改成 true，运算会像你想得那样进行，输出结果为 "We are equal 20";.

答案 2)
4) 12
和二进制 OR 目标相同，这个问题要求你理解开头的零表示八进制符号，第一个 1 表示数中有一个 8，没有其他数。所以十进制运算是
8|4
转换成二进制形式为
1000
0100

1100
|运算符表示两个数每一位进行运算，其中有一个为 1，相应位的结果就为 1。

答案 3)
2,3,4
选项一不会通过编译，因为它试图在整型上运行逻辑 OR 操作。逻辑或只能用于操作 boolean 类型参数。

目标四 在方法中传递对象和基本类型值

判断传递对象和基本类型参数到方法中的结果，在方法中进行赋值或者其它的修改操作。

目标中注意事项

目标可能会问你是否理解传递值到方法中时会发生什么结果。如果方法中的代码改变了变量，对外部的方法是否可见？直接引用 Peter van der Lindens 的 Java 程序员解答的一段话（在 http://www.afu.com）

//引用
所有的变量（基本类型值和对象的引用值）都是值传递。但是这不是全部情况，对象是经常通过 Java 的引用变量来操作的。所以也可以说对象是通过引用传递的（引用变量通过值传递）。这是变量不使用对象的值而是像前一个问题那样描述的使用对象引用的值的结果。

最后一行：调用者对基本类型参数（int，char 等）的拷贝在相应参数变化时不会改变
。但是，在被调用方法改变相应作为参数传递的对象（引用）字段时，调用者的对象也改变其字段。
//引用结束

如果你来自 C++背景，你可能对值传递参数和用&符号传递引用参数熟悉。在 Java 中没有这样的选择，所有的都是值传递。但是看起来并不总是这样。如果你传递的对象是对象

的引用，你不能直接对对象的引用进行操作。

所以如果你操作一个传递到方法的对象的字段，结果就好像你按引用传递（任何改变结果都会返回到调用函数）。

将对象引用作为方法变量
请看下面的例子

```
class ValHold{
        public int i = 10;
}

public class ObParm{
public static void main(String argv[]){
        ObParm o = new ObParm();
        o.amethod();
        }

        public void amethod(){
                ValHold v = new ValHold();
                v.i=10;
                System.out.println("Before another = "+ v.i);
                another(v);
                System.out.println("After another = "+ v.i);
        }//End of amethod

        public void another(ValHold v){
                v.i = 20;
                System.out.println("In another = "+ v.i);
        }//End of another
}
```

程序的输出结果是

```
Before another = 10
In another = 20
After another = 20
```

看变量 i 是怎么被修改的。如果 Java 总是值传递（也就是对变量的拷贝），它是怎么被修改的呢？过程是这样的方法收到了句柄的拷贝或者对象的引用，但是这个引用的作用类似于指向真实的指针。对这个字段的改变会反映到它所指的值。这有些像是在 C/C++中指针的自动间接应用的的作用。

基本类型作为方法参数

当你对方法传递基本类型参数，是直接传递值。方法得到它的拷贝，任何修改都不会在

外部方法得到反映。请看以下例子

```
public class Parm{
public static void main(String argv[]){
                    Parm p = new Parm();
                    p.amethod();
          }//End of main

          public void amethod(){
                    int i=10;
                    System.out.println("Before another i= " +i);
                    another(i);
                    System.out.println("After another i= " + i);
          }//End of amethod

          public void another(int i){
                    i+=10;
                    System.out.println("In another i= " + i);
          }//End of another

}
```

程序的输出结果如下

```
Before another i= 10
In another i= 20
After another i= 10
```

习题 1）

以下所给代码的输出是什么？

```
class ValHold{
          public int i = 10;
}

public class ObParm{
public static void main(String argv[]){
          ObParm o = new ObParm();
          o.amethod();
          }
          public void amethod(){
                    int i = 99;
                    ValHold v = new ValHold();
                    v.i=30;
                    another(v,i);
                    System.out.println(v.i);
          }//End of amethod
```

```
        public void another(ValHold v, int i){
                i=0;
                v.i = 20;
                ValHold vh = new ValHold();
                v =    vh;
                System.out.println(v.i+ " "+i);
        }//End of another
}
```

1) 10,0, 30
2) 20,0,30
3) 20,99,30
4) 10,0,20

答案

答案 1）
4) 10,0,20

第 6 章 重载，重写，运行时类型和 OO

目标一 封装和 OO 设计

陈述封装在面向对象设计中的好处，并编写实现紧密封装的类的代码，陈述“is a”和“has a”的关系。

“Is a”和“has a”关系

这是一个很基础的 OO 问题，你很可能在考试中碰到一个题目。本质上，它是为了考察你是否理解何时在谈论对象所属的类结构以及何时是在谈论一个类拥有的方法或域。

因此，猫是动物的一种（IS A），猫有尾巴（HAS A）。当然，区别可能会模糊不清。如果你是一名动物学家并且知道动物种类群的正确名字，你可能会说猫是（IS A）longlatinwordforanimalgroupwithtails（一个很长的表示有尾巴的动物群的拉丁单词）。

但是出于考试的目的，这不在考虑范围之内。

考试题目趋向于这种类型：根据一段对于潜在层次结构的描述，你会得到诸如什么应该是域，什么应该是新的子类的问题。这些问题乍一看比较复杂，但是如果你仔细阅读的话都十分明显。

封装

Java1.1 的目标中没有特别提到封装，虽然你会被急切的要求学习 Java 而不没有机会接触概念。封装包含将类的接口从实现中分离出来。这意味着你无法“偶然地”破坏某个域的值，你必须使用方法来修改值。

通常，要实现这一点，需要创建私有变量（域），它们只能通过方法来更新和提取。这些方法的标准命名规范是

setFieldName
getFieldName

例如，你要改变形状的颜色，你会创建如下形式的方法对

```
public void setColor (Color c) {
    cBack = c;
    }

public Color getColor () {
    return cBack;
    }
```

控制变量访问的主要关键字为

public
private
protected

不要受到误导而认为访问控制系统与安全有关。它不是为防止程序员攻击变量而设计的，而是为了帮助避免不期望的修改。

使用上面 Color 例子的标准方法是将 cBack 域设为私有的。一个私有域只在当前类内部可见。这意味着程序员不能偶然地在另一个类中写代码来修改它的值。这有助于减少 bug 的引入。

接口与实现的分离使得在一个类中修改代码而不破坏其他代码变得更简单。

对于类的设计者这使他们能够修改类而不必破坏使用它的程序。类的设计者可以为域修改的“安全检查”插入额外的检查流程。我曾经致力于保险项目，此项目中的客户的年龄值可能小于 0。如果这个值被保存在简单的域中，比如整数，就没有明显的地方可以存放检查流程。如果年龄只可以通过 set 和 get 方法访问，就可以通过这种不破坏现存代码的方式来对插入进行 0 或负数年龄检查。当然，随着开发的进行，会发现更多需要检查的情况。

对于类的最终用户，这意味着他们不需要理解内部工作，呈现在他们面前的是一个清晰的处理数据的接口。最终用户可以相信更新类代码不会破坏他们现有的代码。

运行时类型

因为多态机制允许在运行时选择执行方法的版本，有时候将要运行的方法并不明显的。以如下代码为例。

```
class Base {
int i = 99;
public void amethod () {
        System.out.println (“Base.amethod ()”);
        }
}

public class RType extends Base {
int i = -1;
        public static void main (String argv []) {
        Base b = new RType (); //<= Note the type
        System.out.println (b.i);
        b.amethod ();
        }

        public void amethod () {
                System.out.println (“RType.amethod ()”);
        }
}
```

注意，b 引用的类型是 Base，但是实际的类型是类 RType。对 amethod 的调用将启动 RType 中的版本，但是 b.i 输出的调用将引用 Base 类中的域 i。

课后测试题

问题 1）假设你被给予如下设计

“一个人有姓名，年龄，地址和性别。你将要设计一个类来表示一类叫做病人的人。这种人可以被给予诊断，有配偶并且可能活着”。假设表示人的类已经创建了，当你设计病人类时如下哪些应该被包含在内？

1） registration date
2） age

3） sex
4） diagnosis

问题 2）当你试图编译并运行如下代码时会发生什么？

```
class Base {
int i = 99;
public void amethod () {
        System.out.println (“Base.amethod ()”);
        }
        Base () {
                amethod ();
        }
}

public class RType extends Base {
int i = -1;
        public static void main (String argv []) {
        Base b = new RType ();
        System.out.println (b.i);
        b.amethod ();
        }
        public void amethod () {
                System.out.println (“RType.amethod ()”);
        }
}
```

1）

```
RType.amethod
-1
RType.amethod
```

2）

```
RType.amethod
99
RType.amethod
```

3）

```
99

RType.amethod
```

4）

Compile time error

问题 3）你的首席软件设计者向你展示了她正要创建的新电脑部件系统的草图。在层次结构的顶端是一个叫做 Computer 的类，在此之下是两个子类。一个叫做 LinuxPC，另一个叫做 WindowsPC。两者之间最大的不同点是一个运行 Linux 操作系统，另一个运行 Windows 系统（当然另一个不同在于一个需要不停的重启，另一个则能够可靠的运行）。在 WindowsPC 之下是两个子类，一个叫做 Server，另一个叫做 Workstation。你如何评价你的设计者的工作？

1） Give the go ahead for further design using the current scheme
2） Ask for a re-design of the hierarchy with changing the Operation System to a field rather than Class type
3） Ask for the option of WindowsPC to be removed as it will soon be absolete
4） Change the hierarchy to remove the need for the superfluous Computer Class.

问题 4）假设有如下类

```
class Base {
    int Age = 33;
}
```

关于对 Age 域的访问，你会如何修改来改进这个类？

1） Define the variable Age as private
2） Define the variable Age as protected
3） Define the variable Age as private and create a get method that returns it and a set method that updates it
4） Define the variable Age as protected and create a set method that returns it and a get method that updates it

问题 5）下面哪些是封装的好处？

1） All variables can be manipulated as Objects instead of primitives
2） by making all variables protected they are protected from accidental corruption
3） The implementation of a class can be changed without breaking code that uses it
4） Making all methods protected prevents accidental corruption of data

问题 6）指出三个面向对象编程的主要特点？

1） encapsulation, dynamic binding, polymorphism
2） polymorphism, overloading, overriding
3） encapsulation, inheritance, dynamic binding

4） encapsulation, inheritance, polymorphism

问题 7）你如何在类中实现封装？

1） make all variables protected and only allow access via methods
2） make all variables private and only allow access via methods
3） ensure all variables are represented by wrapper classes
4） ensure all variables are accessed through methods in an ancestor class

答案

答案 1）

1） registration date
2） diagnosis

对于病人来说，注册日期是一个合理的添加域，并且设计明确地指出病人应该有诊断报告。由于病人是人的一种，它应该有域 age 和 sex（假设它们没有被声明为私有的）。

答案 2）

2）

RType.amethod
99
RType.amethod

如果这个答案看起来靠不住，试着编译并运行代码。原因是这段代码创建了一个 RType 类的实例但是把它赋予一个指向 Base 类的引用。在这种情况下，涉及的任何域，比如 i，都会指向 Base 类中的值，但是方法的调用将会指向实际类中的方法而不是引用句柄中的方法。

答案 3）

2） Ask for a re-design of the hierarchy with changing the Operating System to a field rather than Class type

答案 4）

3） Define the variable Age as private and create a get method that returns it and a set method that updates it

答案 5）

3） The implementation of a class can be changed without breaking code that uses it

答案 6）

4） encapsulation, inheritance, polymorphism

我曾经在一次工作面试上遇到这个问题。我得到了那份工作。不能保证你一定会在考试中遇到类似的问题，但是知道的话会很有用。

答案 7）

2）make all variables private and only allow access via methods

目标二 重写和重载

编写调用重写或重载的方法以及父类的或重载过的构造函数；并且描述调用这些方法的效果。

目标的评论

术语重载（overloaded）和重写（overridden）太相近了以至于会造成混淆。我记忆的方式是想象某物被践踏（overridden）字面上的意思是被沉重的交通工具压过并且不再是其原来的样子。某物负载过重（overloaded）仍然在移动，但是负担过重的功能将使其花费巨大的努力。这只是一个区别两者的小窍门，跟 Java 中实际的操作没有任何关系。

重载方法

重载是 Java 中实现面向对象，多态机制等概念的方式之一。多态性（Polymorphism）是由多个单词组成的词语，Ply 意为“很多”，“morphism”暗示着含义。因此，重载允许同一个方法名称具有多种意思或用途。方法重载是编译器的技巧，依赖于不同的参数，它允许你使用相同的名称来完成不同的动作。这样做的好处是 Java 可以在运行时决定调用的方法而不是在编译时决定。

因而，设想一下你正在为模拟 Java 认证考试设计系统接口。答案可能作为整数，布尔数或文本字符串得到。你可以为每一个参数类型创建一个方法，并给予相应的名字，比如

markanswerboolean (Boolean answer) {

```
    }
markanswerint (int answer) {
    }
markanswerString (String answer ) {
    }
```

这样可以正常运行，但这也意味着类的未来用户需要知道更多不必要的方法名。使用一个单一的方法名会更实用，编译器可以根据参数类型和数目来决定调用的实际代码。

进行方法重载不需要记住任何关键字，你只要创建多个具有不同数目或类型的参数的同名方法就可以了。参数的名称并不重要，但是数目和类型必须不同。如下是一个 markanswer 方法重载的例子

```
void markanwwer (String answer) {
    }
void markanswer (int answer) {
    }
```

如下不是重载的实例，它会导致编译时错误，指出这是重复的方法声明。

```
void markanswer (String answer) {
    }
void markanswer (String title) {
    }
```

返回值类型并不是实现重载署名的要素。

因此，改变如上代码使其放回 int 值仍然会导致编译时错误，但是这一次指出方法不能用不同的返回值类型进行重新定义。

重写方法

重写方法意味着它的所有功能被完全取代了。重写是在子类中对一个定义在父类中的方法进行修改。为了重写方法，要在子类中定义一个与父类中具有完全相同署名的方法。这样做会覆盖父类中的方法，并且此方法的功能再也不能被直接访问了。

Java 提供了一个重写的例子，就是每个类都从最高父类 Object 中继承的 equals 方法。继承的 equals 版本仅仅在内存中比较类引用的实例。这通常不是我们想要的，特别是对于 String。对于 String，你通常希望通过逐个字符的比较来确定两个字符串是否相同。为了做到这一点，String 中的 equals 版本进行了重写，并能执行逐个字符的比较。

调用基类的构造函数

构造函数是一种在每次创建类的实例时自动运行的特殊方法。Java 能够识别构造函数，因为它们具有与类本身相同的名字，并不需要返回值。与其他方法一样，构造函数可以接受参数，并且根据如何初始化类，你可以传递不同的参数。如此，以 AWT 包中的 Button 类为例，通过重载提供了两个构造函数的版本。一个是

```
Button ()
Button (String label)
```

因此，你可以创建一个没有标签的按钮，并在稍后设定，或者使用普通的版本在创建的时候就设定标签。

但是，构造函数是不能被继承的，所以如果你想从父类中获得一些有用的构造函数，缺省是不可用的。因此，如下代码将不能编译通过

```
class Base {

public Base () {}
public Base (int i) {}
}

public class MyOver extends Base {
public static void main (String argv []) {
    MyOver m = new MyOver (10); // Will Not compile
    }
}
```

要从父类中得到构造函数，你需要使用神奇的关键字 super。这个关键字可以被当作一个方法来使用，并且传递适当的参数使之与你要求的父类中的构造函数相吻合。在以下修改了上述代码的例子中，关键字 super 被用来调用基类中接受 integer 参数的构造函数版本，这段代码编译时不会报错。

```
class Base {
public Base () {}
public Base (int i) {}
}

public class MyOver extends Base {
public static void main (String arg []) {
        MyOver m = new MyOver (10);
        }
    MyOver (int i) {
        super (i);
```

```
    }
}
```

使用 this ()调用构造函数

与使用 super ()调用基类中构造函数的方式相同，你可以使用 this 调用当前类中的其他构造函数。这样，在前面的例子中你可以像下面那样定义另一个构造函数

```
MyOver (String s, int i) {
    this (i);
    }
```

如你猜测的，这将会调用当前类中那个只接受一个整数参数的构造函数。如果你在构造函数中使用 super ()或 this ()，必须第一个调用它。由于只有一个能被第一个调用，你不能在构造函数中既使用 super ()又使用 this ()。

因此，如下代码会导致编译时错误。

```
MyOver (String s, int i) {
    this (i);
    super (); // Causes a compile time error

}
```

基于构造函数不能被继承的知识，很明显重写是不切合实际的。如果你有一个叫做 Base 的基类，你创建了一个继承它的子类，对于要重写构造函数的子类，它的名字必须跟父类相同。这会导致编译时错误。这是一个没有层次意义的例子。

```
class Base {}
class Base extends Base {} //Compile time error!
```

构造函数和类层次

构造函数总是从层次结构的顶端开始称作向下。在考试中，你很可能会遇到一些题目涉及到在类层次中多次调用 this 和 super，你必须指出输出什么内容。当你遇到复杂的层次结构时请格外小心，这可能跟构造函数没有关系，可能由于构造函数同时调用了 this 和 super，而导致编译时错误。

有如下例子

```
class Mammal {
```

```
        Mammal () {
            System.out.println (“Creating Mammal”);
        }
    }

    public class Human extends Mammal {
    public static void main (String argv []) {
        Human h = new Human ();
        }
        Human () {
        System.out.println (“Create Humn”);
        }
    }
```

当运行代码时，由于隐式调用了基类中的无参构造函数，首先会输出字符串“Create Mammal”。

课后测试题

问题 1）假设有如下类定义，如下哪些方法可以合法放置在“//Here”的注释之后？

```
    public class Rid {
    public void amethod (int i, String s) {}
    // Here
    }
```

1） public void amethod (String s, int i) {}
2） public int amethod (int i, String s) {}
3） public void amethod (int i, String mystring) {}
4） public void Amethod (int i, String s) {}

问题 2）假设有如下类定义，哪些代码可以被合法放置在注释“//Here”之后？

```
    class Base {
        public Base (int i) {}
    }

    public class MyOver extends Base {
    public static void main (String arg []) {
            MyOver m = new MyOver (10);
            }

        MyOver (int i) {
            super (i);
```

```
        }

        MyOver (String s, int i) {
            this (i);
            //Here
        }
    }
```

1） MyOver m = new MyOver ():
2） super ();
3） this (“Hello”, 10);
4） Base b = new Base (10);

问题 3）假设有如下类定义

```
class Mammal {
    Mammal () {
        System.out.println (“Mamml”);
    }
}

class Dog extends Mammal {
    Dog () {
        System.out.println (“Dog”);
    }
}

public class Collie extends Dog {
public static void main (String argv []) {
    Collie c = new Collie ();
}

    Collie () {
        this (“Good Dog”);
        System.out.println (“Collie”);
    }

    Collie (String s) {
    System.out.println (s);
    }
}
```

将会输出什么？

1） Compile time error
2） Mammal, Dog, Good Dog, Collie
3） Good Dog, Collie, Dog, Mammal
4） Good Dog, Collie

问题 4）下面哪些论述是正确的？

1） Constructors are not inherited
2） Constructors can be overridden
3） A parental constructor can be invoked using this
4） Any method may contain a call to *this* or *super*

问题 5）试图编译并运行下面代码会发生什么？

```
class Base {
        public void amethod (int i, String s) {
        System.out.println (“Base amethod”);
        }
        Base () {
        System.out.println (“Base Constructor”);
        }
}

public class Child extends Base {
int i;
String Parm = “Hello”;
public static void main (String argv []) {
    Child c = new Child ():
    c.amethod ():
}

void amethod (int i, String Parm) {
        super.amethod (i, Parm);
        }
    public void amethod () {}
}
```

1） Compile time error
2） Error caused by illegal syntax super.amethod (i, Parm)
3） Output of “Base Constructor”
4） Error caused by incorrect parameter names in call to super.amethod

问题 6）试图编译并运行如下代码时将发生什么？

```
class Mammal {

Mammal () {
        System.out.println (“Four”);
        }

public void ears () {
        System.out.println (“Two”);
        }
}

class Dog extends Mammal {
    Dog () {
    super.ears ();
    System.out.println (“Three”);
    }
}

public class Scottie extends Dog {
public static void main (String argv []) {
    System.out.println (“One”);
    Scottie h = new Scottie ();
    }
}
```

1） One, Three, Two, Four
2） One, Four, Three, Two
3） One, Four, Two, Three
4） Compile time error

答案

答案 1）

1） public void amethod (String s, int i) {}
4）public void Amethod (int i, String s) {}

Amethod 中的大写字母 A 意味着这是不同的方法。

答案 2）

4）Base b = new Base (10);

任何 this 或 super 的调用都必须是构造函数中的第一行。由于方法已经调用了 this，不能有别的调用插入了。

答案 3）

2） Mammal, Dog, Good Dog, Collie

答案 4）

1） Constructors are not inherited

父类的构造函数应该使用 super 调用，而不是 this。

答案 5）

1） Compile time error

这会导致一个错误，意思是说“你不能重写方法使其访问权限更靠近私有”。基类的 amethod 版本被明确的标注为 public，但是在子类中没有标识符。好了，所以这不是在考察你的构造函数重载的知识，但是他们也没在考试中告诉你主题。若这段代码没有省略关键字 public，将会输出“Base constructor”，选项 3。

答案 6）

3）One, Four, Two, Three

类是从层次的根部往下创建的。因此，首先输出 One，因为它在 Scottie h 初始化之前创建。然后，JVM 移动到层次的基类，运行“祖父类”Mammal 的构造函数。这会输出“Four”。然后，运行 Dog 的构造函数。Dog 的构造函数调用 Mammal 中的 ears 方法，因此输出“Two”。最后，Dog 的构造函数完成，输出“Three”。

目标三 创建类实例

编写创建任何具体类实例的代码，包括正常的高层次类，内部类，静态内部类和匿名内部类。

目标的注释

这份材料的一些内容在别的地方提到过，特别是目标 4.1 中。

实例化类

具体类是指能够被实例化为对象引用（也简称为对象）的类。因此，抽象类是不能被实例化的，所以不能创建对象引用。记住，包含任何抽象方法的类本身也是抽象的，并且不能被实例化。

实例化类的关键是使用关键字 new。典型地，如下所示

Button b = new Button ();

这个语法意为变量 b 是 Button 类型的，并且包含指向 Button 实例的引用。但是，尽管引用的类型经常与被实例化的类的类型是一样的，但这不是必要的。因此，如下代码也是合法的

Object b = new Button ();

这个语法指出 b 引用的类型是 Object 而不是 Button。

声明和实例化不是必须出现在同一行上。可以这样创建一个类的实例。

```
Button b;
b = new Button ();
```

内部类是随着 JDK1.1 的发布而加入的。它们允许一个类在另一个类中定义。

内部类

内部类是随着 JDK1.1 的发布而引入的。它们允许类被定义在其他类中，有时候被称作嵌套类。它们被广阔的使用在新的 1.1 事件处理模型中。你肯定会在考试中遇到嵌套类范围的问题。

这是一个简单的例子

```
class Nest {
    class NetIn {}
}
```

这段代码编译后的输出是两个 class 文件。如你所想的，第一个是

Nest.class

另一个是

Nest$NestIn.class

这说明了嵌套类通常只是个命名规范，而不是一种新的类文件。内部类允许你逻辑性地组织类。它们在你希望访问变量时也有广泛的好处。

嵌套高层类

嵌套高层类是一个包容高层类的静态成员。

这样，修改之前的简单例子

```
class Nest {
    static class NestIn {}
}
```

这种类型的嵌套经常用来简单的组合相关的类。因为类是静态的，它不需要在外部类实例存在的情况下才能实例化内部类。

成员类

我认为成员类是“普通内部类”。成员类类似于类的其他成员，你必须在创建内部类的实例之前首先实例化外部类。由于需要结合外部类的实例，Sun 引入了新的语法允许在创建内部类的同时创建外部类的实例。这形成如下形式

Outer.Inner i = new Outer ().new Inner ();

为了弄清楚为此提供的新语法的意思，设法认为在上面例子中使用的关键字 new 属于 this 当前存在的实例中，因此，你可以将创建实例的代码修改为

Inner i = this.new Inner ();

因为成员类无法脱离外部类的实例存在，它可以访问外部类中的变量。

创建在方法中的类

这种类更正确的叫法应该是局部类，但是把它们当作创建在方法中，有助于让你知道最有可能在什么地方遇到它们。

局部类只在它的代码块或方法中可见。在局部类定义中的代码只能使用包容块中的 final 局部变量或方法的参数。你很有可能在考试中遇到这样的题目。

匿名类

你对于匿名内部类的第一反应可能是“你为什么要这么做，而且如果它没有名字，你怎么能引用它呢？”

要回答这些问题，请考虑下面的情形。你可能会遇到不停的为类实例捏造自我描述的名字的情况。这样，对于事件处理，两件需要了解的重要事情是等待处理的事件和处理器附属的模块的名字。为事件处理器实例取名字不会有多大价值。

至于如果类没有名字，如何引用该类的问题，你是做不到，而如果你需要通过名字来引用它，就不应该创建匿名类。缺乏名字有另一个副作用，就是你不能为它设定任何构造函数。

这是一个创建匿名内部类的例子

```
class Nest {
public static void main (String argv []) {
      Nest n = new Nest ();
            n.mymethod (new anon () {});
      }
      public void mymethod (anon i) {}
}
class anon {}
```

请注意匿名内部类是如何在 mymethod 的调用的圆括号中同时声明和定义的。

课后测试题

问题 1）下面哪些论述是正确的？

1）A class defined within a method can only access static methods of the enclosing method
2）A class defined within a method can only access final variables of the enclosing method
3）A class defined with a method cannot access any of the fields within the enclosing method
4）A class defined within a method can access any fields accessible by the enclosing method

问题 2）下面哪些论述是正确的？

1） An anonymous class cannot have any constructors
2） An anonymous class can only be created within the body of a method
3） An anonymous class can only access static fields of the enclosing class
4） The class type of an anonymous class can be retrieved using the getName method

问题 3）下面哪些论述是正确的？

1） Inner classes cannot be marked private
2） An instance of a top level nested class can be created without an instance of its enclosing class
3） A file containing an outer and an inner class will only produce one .class output file
4） To create an instance of an member class an instance of its enclosing class is required.

答案

答案 1）

2）A class defined within a method can only access final variables of the enclosing method

这种类可以访问传递给包容方法的参数

答案 2）

1） An anonymous class cannot have any constructors

答案 3）

2） An instance of a top level nested class can be created without an instance of its enclosing class
4）To create an instance of a member class an instance of its enclosing class is required.

内部类会被放在它自己的.class 输出文件中，使用格式

Outer$Inner.class.

高层次嵌套类是一个静态类，因而不需要包容类的实例。成员类是普通的非静态类，因而需要有一个包容类的实例。

第 7 章　线程

目标一 实例化和启动线程

通过使用 java.lang.Thread 和 java.lang.Runnable 在代码中定义，实例化，和启动新线程。

什么是线程？

线程是表面上看似和主程序并行运行的轻量级进程。与进程不同的是它与程序的其他部分共享存储空间和数据。在这里线程的英文单词 thread 实际上是“thread of execution” 的缩写，you might like to imagine a rope from which you have frayed the end and taken one thread. 它依然是主线程的一部分，但它可以独立出来，自己完成操作。这里请注意，启动一个多线程的程序和仅仅启动一个程序的多个同一程序是有区别的，因为一个多线程的程序将会对统一程序内的数据进行读取和存储。

一个可以显示多线程用处的例子就是打印，当你按下打印按钮的时候，你肯定不希望主程序直到打印完成才开始响应。最棒的就是你可以让打印进程“在后台”悄悄的运行，同时你可以使用主程序的其他部分。

而且当打印线程出现故障的时候主程序可以对此做出响应，一个讲解多线程最佳的通用例子就是创建一个每当你按下按钮的时候弹出一个弹球的图形用户界面程序。因为现在处理器速度快的原因，导致表面上看每个线程似乎是独享 CPU，这是由于处理器在各个线程之间的切换速度很快，控制弹球跳动的代码更像是在处理器上运行唯一一个程序。不像大部分程序那样，线程并不是嵌入到 Java 语言的最核心部分，它的大部分，依然是继承自最原始的类——Object,旧一点的语言如 C/C++并没有为线程设定标准。

当你在为 Java 程序员认证考试学习的时候，你必须要对当一个程序启动一个新线程有一定认识，这个时候程序不再是在单一的路径上执行。因为当一个线程 A 先于线程 B 执行，并不意味着线程 A 一定会比线程 B 先结束，当然线程 B 也不一定是在线程 A 结束后才开始运行。因此你有可能会遇到这样的问题，如“最有可能输出以下哪段代码？”，最准确的输出结果决定于底层的操作系统和同一时间正在一起运行的其它程序。

正是因为一个多线程程序在你的机器操作系统的组合上产生一个特定的输出，因此它不能保证在其他不同的系统上也能有同样的输出结果。出考题的人会凭空假设程序在一个更加通用的平台上运行的（如 Windows），考题可以考察底层操作系统对 Java 线程的影响。

不要只把注意力放在考试的关于线程的考点上，因为它仅仅是考察你对一小部分线程知识是否掌握的很牢固。有很多线程相关的概念考试并没有覆盖到，如果你仅仅是为考试做打算，那么你可以对线程组，线程池，线程优先级等概念不做了解。当然，对这些概念的了解对更深层次领会 Java 语言编程是有好处的，如果你只想把精力集中在应付考试的考点上，

那么你只需要对该指南上列出考点进行学习就行了。

两种创建线程的方式

在这两种方法中，使用 Runnable 似乎更常见一些，但是出于考试的原因，你必须对这两种方法都了解。下面这种方法就是让对象在创建的时候，用实现 Runnable 接口的方法来实现创建线程。

```
class MyClass implements Runnable{
  public void run(){//Blank Body}
}
```

创建两个线程并运行它们。

```
MyClass mc = new MyClass();
MyClass mc2 = new MyClass();
Thread t = new Thread(mc);
Thread t2 = new Thread(mc2);
t.start();
t2.start();
```

这里需要注意的是，线程 t 并不肯定是比线程 t2 先结束运行，当然由于 run()方法里面并没有任何代码，因此线程 t 很有可能比线程 t2 先结束运行。即使你让该段代码在你的机器上运行上千次，无法改变它最终的运行结果，当然无法保证在其他的系统上运行结果也是一样的，或者当你更改系统的环境配置时，也有可能发生变化。

注意到在用实现 Runnble 接口的方法来创建线程时，必须要求创建一个 Thread 对象的实例，并且必须要在创建的时候，把实现该 Runnable 接口的对象作为构造方法的参数传递进去。

任何一个类在它实现一个接口的时候，它必须同时要创建该接口中已经定义的方法。该方法不一定非要有任何意义，比如，方法里面的内容可以为空，但是它必须在这个类的内部出现。因此在上面那个例子里出现了空的 run()方法，不包含 run()将会导致程序在编译期报错。

当你需要在一个类里面创建一个有某种用途的线程的时候，你需要在我上面的

//Blank Body

部分写一些东西进去。

另外一种创建线程的方法就是直接使类继承自 Thread。这样做非常简单，但同时你也无法再继承其他的对象，因为 Java 只支持单继承。因此当你创建一个 Button 对象的时候你无法使用这种方法来添加线程的功能，因为它已经是继承了 AWT Button 这个类的，这使你不得不在继承方面动一点脑筋。不过一些反对的声音认为这种创建线程的方法更符合面向对象的思想。但不管怎么说，你必须为了 Java 考试对这种方法有一定了解。

实例化和启动一个线程

尽管在线程中运行的方法是 run()，你并不需要调用这个方法来启动一个线程，而是调用 start() 方法来启动一个线程。这点很关键，因为它极有可能在考试中出现。这点很有可能让你栽跟头，因为这和大多数往常你所遇见的 Java 编程的惯例不一样。通常如果你会把一段代码放到一个方法里面去，当你需要执行它的时候，你只需要调用该方法。如果你直接调用 run()方法，当然这也不会造成什么错误，只是它会象一个普通的方法那样运行，而不是作为线程的一部分来执行。

Runnable 接口并不包括 start()方法，同样也不包含其它一些线程中有用的方法（如 sleep(),suspend()等等），你只需将你已经实现了 Runnable 接口的对象作为一个构造方法参数传递给一个已经实例化的 Thread 对象。

当你需要一个已经实现了 Runnable 接口的对象执行多线程任务的时候，你需要使用以下的代码。

```
MyClass mc = new MyClass();

Thread t = new Thread(mc);

t.start();
```

尽管是 run 方法在运行，但一个线程的启动是调用 start 方法。

再一次强调你不是调用 run()方法，而是 start()方法来启动一个线程，尽管在 run()方法里面的代码才是线程执行的时候所运行的。

如果你的对象是继承 Thread，你可以简单的调用 start()方法来启动它。缺点就是 Thread 的子类因为单继承的原因再无法继承其它功能的类。

练习题

习题 1）当你试图编译运行下列代码的时候会发生什么？

```
public class Runt implements Runnable{
```

```
public static void main(String argv[]){
        Runt r = new Runt();
        Thread t = new Thread(r);
        t.start();
        }

        public void start(){
        for(int i=0;i<100;i++)
                System.out.println(i);
        }
}
```

1) Compilation and output of count from 0 to 99
2) Compilation and no output
3) Compile time error: class Runt is an abstract class. It can't

习题 2）下列哪一项表述是正确的？

Which of the following statements are true?

1) Directly sub classing Thread gives you access to more functionality of the Java threading capability than using the Runnable interface
2) Using the Runnable interface means you do not have to create an instance of the Thread class and can call run directly
3) Both using the Runnable interface and subclassing of Thread require calling start to begin execution of a Thread
4) The Runnable interface requires only one method to be implemented, this is called run

习题 3）当你试图编译运行下列代码的时候会发生什么？

```
public class Runt extends Thread{
public static void main(String argv[]){
        Runt r = new Runt();
        r.run();
        }
        public void run(){
        for(int i=0;i<100;i++)
                System.out.println(i);
        }
}
```

1) Compilation and output of count from 0 to 99
2) Compilation and no output
3) Compile time error: class Runt is an abstract class. It can't be instantiated.
4) Compile time error, method start has not been defined

习题 4）下列哪一项表述是正确的？

1)To implement threading in a program you must import the class java.io.Thread

2) The code that actually runs when you start a thread is placed in the *run* method

3) Threads may share data between one another

4) To start a Thread executing you call the *start* method and not the *run* method

习题 5） 下列哪一项是让线程开始运行的正确代码？

1)

```
public class TStart extends Thread{
    public static void main(String argv[]){
      TStart ts = new TStart();
      ts.start();
      }
      public void run(){
      System.out.println("Thread starting");
      }
}
```

2)

```
public class TStart extends Runnable{
      public static void main(String argv[]){
      TStart ts = new TStart();
      ts.start();
      }
      public void run(){
      System.out.println("Thread starting");
      }
}
```

3)

```
public class TStart extends Thread{
      public static void main(String argv[]){
      TStart ts = new TStart();
      ts.start();
      }
      public void start(){
      System.out.println("Thread starting");
      }
}
```

4)

```
public class TStart extends Thread{
      public static void main(String argv[]){
      TStart ts = new TStart();
      ts.run();
      }
      public void run(){
      System.out.println("Thread starting");
      }
}
```

练习题答案

答案 1）

3） Compile time error: class Runt is an abstract class.它不能被实例化.
这个类实现了 Runnable 接口，但是没有定义 run()方法。

答案 2）

3)不管是继承 Thread 对象还是实现 Runnable 接口，都要使用 start()方法来让该线程开始运行。
4) 实现 Runnable 接口只需要定义一个 run()的方法。

答案 3）

1) 编译输出从 0-99。

尽管如此,注意到这段代码并没有让线程运行，run()方法不应该这样被调用。

答案 4）

2)当你让线程跑起来的时候运行的实际上是 run()方法里面的代码。
3) 线程之间可以彼此共享数据信息。
4) 当你需要一个线程开始运行的时候调用的是 start()方法而不是 run()方法。

你不需要导入额外的类，因为线程是 Java 语言的一部分。

答案 5）

1) 仅选项 1 是一个有效的方式开始一个新的线程执行。 选项 2 的代码继承 Runnable 接口但没意义,因为 Runnable 是接口不是类，接口使用 implements 关键字。 选项 3 的代码直接地调用起动方法。 如果您运行这个代码您将发现文本输出，但由于直接调用方法，并不是因为一个新的线程在运行。 选项 4 也一样，直接地调用运行线程仅是另一个方法，并且象其他的一样执行。

目标二 何时线程会被阻止运行

对什么情况下线程会被阻止运行有一定认识。

关于该目标的解释

“线程会被阻止运行”的表述看上去很笼统，它的意思是该线程已经被人为的暂停？还是这个线程已经被彻底销毁？其实“线程会被阻止运行”的意思是线程被阻塞了。

可能造成线程阻塞的原因

线程阻塞的原因可能是

1） 线程已经被设置了一定长度的睡眠时间。

2） 调用了 suspend()方法，它将一直保持阻塞直到 resume()方法被调用。

3） 该线程因为被调用了 wait()方法被暂停了，当收到 notify 或者 notifyAll 消息的时候该线程会重新被激活。

出于对付考试的原因，sleep()，notify 和 notifyAll()是这些造成线程组塞的原因非常需要掌握的。

sleep()方法是一个静态的可以暂停线程一定毫秒时间长度的方法。还有一个版本可以支持设定睡眠的时间单位为十亿分之一秒的版本。我认为没有多少人会在有如此精确的机器或者实现 Java 的平台上进行工作。下面是一个展示线程如何进入睡眠状态的例子，注意这个 sleep()方法是如何抛出 InterruptedException 异常的。

```
public class TSleep extends Thread{
public static void main(String argv[]){
        TSleep t = new TSleep();
        t.start();
        }

        public void run(){
            try{
                while(true){
                        this.sleep(1000);
                        System.out.println("looping while");
                       }
               }catch(InterruptedException ie){}
        }
}
```

当 Java2 版本发布的时候，Thread 类里面的 stop(),suspend()和 resume()方法已经被认为是过时的了（不提倡继续再使用，并且在编译期会报出警告提示）。同时 JDK 文档认为

//Quote
这种方式因为它固有的造成死锁可能的原因也不再提倡使用了。当目标线程正在所锁定保护系统的临界资源的监视器时候因为被暂停而保持阻塞状态，其他线程将不能再读写该临界资源直到该目标线程解除死锁状态。如果一个线程解除该目标线程的组塞，而同时又试图在调用 resume()之前保持该监视器的锁定状态，那么将会造成一个死锁。这样的死锁具有代表性，就像”frozen”进程一样。需要更多的信息请参考为什么 Thread.stop, Thread.suspend 和 Thread.resume 不提倡再被继续使用的原因。
//End Quote
线程的通过 wait/notify 的协议来进行阻塞操作将在下一个目标中进行表述。

使用 Thread 包中的 yield 方法

由于 Java 线程对平台依赖的本质，你不能保证一个线程会把对 CPU 资源的使用权移交给另一个线程。某些操作系统的线程调度规则会自动给不同的线程分配 CPU 的占有时间。而另一些操作系统则仅仅是让线程独享处理器资源。因为上述原因，Java 的 Thread 包里面构造了一个静态的名叫 yield()的方法可以让当前正在运行状态的线程让出正在占用的 CPU 周期。该进程则返回“准备运行”状态，这样线程规划系统可以有机会让其他线程进来调用 CPU 资源运行。如果没有其他的线程在“准备运行”运行状态，则刚刚让出 CPU 资源的线程马上重新恢复到运行状态。

限制/抢占

每一个线程都有一个设定好的 CPU 占用周期来运行。一旦它用完了设定好的一个 CPU 占用周期时间，那么它将停止占用 CPU 资源以让其他正在等待中的线程获得机会运行。当一个线程进入之前设定好的 CPU 占用时间那么它的一个新的运行周期就又开始了。这种机制的好处就在于你可以让所有的线程都跑起来而花费最少的时间。

没有时间 限制/共享

优先级系统将会决定哪个线程将会运行。一个相对来说最高优先级的线程将会获得时间来运行。一段运行在该系统中的程序必须使自己能够自动地让每个线程让出 CPU 资源的占用，让所有线程共享 CPU 资源。

Java 线程的优先级

Java 考试并不认为你需要对系统如何设置线程的优先级。尽管知道这些机制是非常有的。同时这样的局限性让你意识到 Thread 包中的 yield()方法的重要性是非常有用的。你可以通过 Thread 包中 Thread.setPriority 来设置线程的优先级，你可以通过 getPriority 来获得线程的优先级，一个新建线程的默认优先级是 Thread.NORM_PRIORITY。

Quiz

练习题

习题 1）当你试图编译运行下列代码的时候会发生什么？

```
public class TGo implements Runnable{
public static void main(String argv[]){
        TGo tg = new TGo();
        Thread t = new Thread(tg);
        t.start();
        }
        public void run(){
                while(true){
                        Thread.currentThread().sleep(1000);
                        System.out.println("looping while");
                        }
        }
}
```

1) Compilation and no output
2) Compilation and repeated output of "looping while"
3) Compilation and single output of "looping while"
4) Compile time error

习题 2）下面哪种方式是推荐的让线程阻塞的方式？

1) sleep()
2) wait/notify
3) suspend

4) pause

习题 3）下列哪一项表述是正确的？

1) The sleep method takes parameters of the Thread and the number of seconds it should sleep
2) The sleep method takes a single parameter that indicates the number of seconds it should sleep
3) The sleep method takes a single parameter that indicates the number of milliseconds it should sleep
4) The sleep method is a static member of the Thread class

习题 4）下列哪一项表述是正确的？

Which of the following statements are true?

1) A higher priority Thread will prevent a lower priorty Thread from getting any access to the CPU.

2) The yield method only allows any higher priority priority thread to execute.

3) The Thread class has a static method called yield

4) Calling yield with an integer parameter causes it to yield for a specific time

练习题答案

答案 1）

Compile time error

sleep()方法将会抛出 InterruptedException 异常。除非让这个代码段放到 try/catch 块里面去，否则这段代码将无法编译。

答案 2）

1) sleep()
2) wait/notify

Java2 版本里面 suspend()方法已不推荐再继续使用。

答案 3）

3) sleep()方法只需要一个表示线程睡眠时间长度的参数。
4) sleep()方法是 Thread 类里面的一个静态方法。

答案 4）

线程类有一个静态方法 yield,调用它可以允许任何等待的线程按照底层操作系统的计划安排执行.没有带一个整数型参数的 yield 方法.是否高优先级的线程比低优先级能获得更多的 CPU 时间与平台有关,并不确定。

目标三 何时线程会被阻止运行

编写代码的时候使用同步的 wait ,notify 和 notifyAll 方法，以防止并行读取问题的发生，同时保证各个线程之间的正常通信。当执行同步的 wait,notify 和 notifyAll 方法的时候对线程和线程之间以及线程和对象锁之间的内部交互进行定义。

为什么你需要 wait/notify 法则？

一个更容易理解的方式，比如你想象一下数据库中的一条整型的变量数据，如果你没有一些锁定数据的措施的话，你将会面临数据污染的危险。这样一个用户可以将这条数据取出来，经过一定运算后再放回去。期间如果其他的用户也将该数据取出来进行运算后返回，那么第一个用户运算后返回的数据将会失效。就像数据库在任何事先不可知的情况下要处理更新一样，所以一个多线程程序必须要有应付这种可能性的机制。为了考试，你十分有必要对本目标的内容进行研究，一些十分有经验的 Java 程序员对 wait/notify 法则也并不是十分了解，这是一个普遍现象，强烈建议读者写一些简单的程序来熟悉这个法则，并对后面的模仿考试的练习题进行针对性的练习。

一个银行/帐户 的例子

下面的代码讲解了同步的线程之间对同一个数据进行操作。它一个名叫 bank 的类，它主要用来驱动多个运行着 Business 类中的数据处理方法的线程。Bussiness 线程实际上就是对 Account 里面的金额进行加减操作。下面代码的思想展示了多线程是如何“踩到对方的脚”并造成数据污染的，但是是有代码可以阻止这类事情发生的。为了“修复”已经存在的这个数据污染我调用了 sleep()方法，你可以认为是等同于暂停，当 bank 中有代码写入操作数据库的时候。如果没有调用这个 sleep()方法，数据污染发生的可能性就依然存在。你不得不运行很多次程序，这样才能让这个毛病显现出来。

```
public class Account{
    private int iBalance;
    public void add(int i){
    iBalance = iBalance + i;
    System.out.println("adding " +i +" Balance = "+ iBalance);
    }
    public    void withdraw(int i){
    if((iBalance - i) >0 ){
        try{
```

```
            Thread.sleep(60);
            }catch(InterruptedException ie){}
            iBalance = iBalance - i;
        }else{
            System.out.println("Cannot withdraw, funds would be < 0");
        }
        if(iBalance    < 0){
            System.out.println("Woops, funds below 0");
            System.exit(0);
        }

        System.out.println("withdrawing " + i+ " Balance = " +iBalance);
        }
        public int getBalance(){
        return iBalance;
        }
}
```

关键字 synchronized

关键字 synchronized 可以用在标记一段声明或者锁定一段代码，保证在同一时间只有一个线程能够运行它的一个实例。进入这段代码将会受到负责监视它的监视器的保护。这个过程是由一个锁定系统实现的。你也可以看到用监控,或者使用互斥来形容(互不相关)。一个锁分配给一个对象以保证同一时间只能有一个线程的进入，因此当一个线程试图进入的时候必须试图获得这个锁的许可。其它的线程将无法进入这段代码，知道第一个进入的线程完成然后释放这个锁。请注意的是这里的锁是基于对象而不是基于方法的。

关键字 synchronized 放在方法的名字之前，如：

```
synchronized void amethod() { /* method body */}
```

关键字 synchronized 也可放在代码段的括号之前，如：

```
synchronized (ObjectReference) { /* Block body */ }
```

注释的部分是指对象或者类的里面某段需要监视器需要锁定的部分。大部分情况下我们使用的是前者，而不是后者。

当一个被关键字 synchronized 标记的代码开始执行以后，拥有它的对象将保持锁定状态，它将不能被调用直到锁定状态被解除。

```
synchronized void first();
synchronized void second();
```

有更好的办法比在一个代码块前加上关键字synchronized能获得串行化的好处,它必须用于联接管理可串行化代码锁的代码 。

wait/notify

除了锁可以获得和释放以外，每个对象都会暂停或者进入等待当其它的线程获得这个锁的时候。这使得线程之间沟通情况随时运行。由于 Java 语言的单继承性，每一个子类都是继承自最原始的 Object 对象,从它那获得这个线程级的通信能力。

***wait* 和 *notify* 应该放在 *Synchronized* 关键字标记的代码中以保证当前的代码在监视器的监控之中。**

在一个标记为 synchronized 的代码中调用 wait()方法，会造成运行这段代码的这个线程交出锁的权限并进入睡眠状态。这种情况通常是为了其他的线程来接管这个锁以进行下一步操作。如果没有让线程唤醒并重新进入运行状态的 notify()或者 notifyAll()方法的话，那么 wait()方法也变得毫无意义。一个典型的使用 wait()/notify()法则来让线程之间进行通信的例子，看上去它似乎陷入了死循环。

```
//producing code
while(true){
try{
  wait();
  }catch (InterruptedException e) {}
}

//some producing action goes here
notifyAll();
```

如果真的是这样，那这段代码真的是垃圾。当你第一眼看到这段代码的时候你会感觉它会一直这样运行下去。其实 wait()会告诉它交出锁让其它线程运行，直到你调用了 notify 或者 notifyAll 方法。

线程调度不是独立的，不能依靠虚拟机让它用同一种方式运行。

不象 Java 的大部分特性，线程在不同的平台上会有不同的表现。这两点分别是线程的优先级和线程的调度。线程调度的两种途径是：抢占和时间片

在一个可以进行抢占的系统上，程序可以通过抢占来获得 CPU 独享周期。在一个实行时间片分配的系统上，每个线程都会获得一个 CPU 独享周期，然后进入准备运行状态。这样可以确保不会让一个线程一直独享 CPU。缺点在于你无法预测这个线程会运行多长时间才会结束，也无法预测什么时候这个线程会再运行，因此通常建议你使用 notify 或者 notifyAll 方法。尽管 Java 把线程的优先级按 1-10 从低到高来分配。一些平台能够识别这个优先级的属性，但其它的却不能。

notify 方法会唤醒一个线程让它进入重新要求获得某对象的监控权限。你不能确定哪个线程被唤醒了。如果你只是有一个线程被唤醒了当然不会存在这种问题。如果你有很多线程等待唤醒，那么等待最长时间的那个将被唤醒。尽管如此，你依然不能确定，线程的优先级对最后结果也有影响。因此一般推荐你使用 notifyAll 而不是 notify，不要对线程的优先级和调度进行任何假设。你可能要让你的代码在尽量多的平台上运行来测试一下，当然，并不总是这样的。

练习题

问题 1）下列哪一个关键字表示该线程放弃了该对象的锁？

1) release
2) wait
3) continue
4) notifyAll

问题 2）下列哪一是关于关键字 synchronized 的表述是最合适的？

1) Allows more than one Thread to access a method simultaneously
2) Allows more than one Thread to obtain the Object lock on a reference
3) Gives the notify/notifyAll keywords exclusive access to the monitor
4) Means only one thread at a time can access a method or block of code

问题 3）当你试图编译运行下列代码的时候会发生什么？

```
public class WaNot{
int i=0;
```

```
public static void main(String argv[]){
        WaNot w = new WaNot();
        w.amethod();
        }

        public void amethod(){
        while(true){
                try{
                        wait();
                    }catch (InterruptedException e) {}
                i++;
            }//End of while

        }//End of amethod
}//End of class
```

1)Compile time error, no matching notify within the method
2)Compile and run but an infinite looping of the while method
3)Compilation and run
4)Runtime Exception "IllegalMonitorStatException"

问题 4）你如何使用 wait/notify 法则指定某个线程被唤醒？

1) Pass the object reference as a parameter to the notify method
2) Pass the method name as a parameter to the notify method
3) Use the notifyAll method and pass the object reference as a parameter
4) None of the above

问题 5）下列哪项表述是正确的？

1) Java uses a time-slicing scheduling system for determining which Thread will execute
2) Java uses a pre-emptive, co-operative system for determining which Thread will execute
3) Java scheduling is platform dependent and may vary from one implementation to another
4) You can set the priority of a Thread in code

练习题答案

答案 1）

Wait

答案 2）

以为着在同一时间内只有一个线程对该代码段或方法进行操作。

答案 3）

Runtime Exception “IllegalMonitorStateException”

wait/notify 法则只能在被标记为 synchronized 的代码段里面使用，在该题这种情况下调用代码会抛出异常。

答案 4）

4) None of the above.

wait/notify 法则没有为哪个线程将被激活提供方法。

答案 5）

3) Java 的调度平台不是独立的，它最终的实现结果是不确定的。
4) 你可以在代码中为代码设定优先级。

第 8 章 Java 的 lang 包

目标一 Math 类中的方法

在开发过程中运用 java.lang.Math 中的如下方法：abs ， ceil ， floor ， max ， min ， random ， round ， sin， cos， tan， sqrt。

本节需要注意的问题

Math 类是不可被继承的，它里面的方法都是静态的。这或许是好事，因为它降低了了混乱情况发生的可能性。在这一块你几乎肯定会遇到问题，如果仅仅是因为你忽视它们而造成错误的发生，那将非常遗憾。

abs

因为我薄弱的数学基础，一开始我几乎对如何使用 abs()这个方法的用法一无所知，直到为了通过 Java 程序员考试，我才开始认真学习弄懂它。它的作用是对一个数值进行取绝对值操作。因此下面的那段代码打印出来的数字是 99。如果进行操作的数是一个非负数，它会原样返回。

```
System.out.println(Math.abs(-99));
```

ceil

这个方法返回的是比被操作数大的最小 double 值。比如下面这个例子：

ceil(1.1)

它将返回 2.0

如果你换成

ceil(-1.1)

它将返回 -1.0；

floor

参考一下 JDK 的说明文档，该方法返回的是：

返回最大的（最接近正无穷大）double 值，该值小于或等于参数，并且等于某个整数。

如果觉得这表达的不够清楚，那么我们可以看一下，下面那一小段代码和它的输出情况：

```
public class MyMat{
public static void main(String[] argv){
        System.out.println(Math.floor(-99.1));
        System.out.println(Math.floor(-99));
        System.out.println(Math.floor(99));
        System.out.println(Math.floor(-.01));
        System.out.println(Math.floor(0.1));
        }
}
```

它的输出是：

```
-100.0
-99.0
99.0
-1.0
0.0
```

max 和 min

注意一下这两个方法需要两个参数。你可能会有疑问，如果仅仅传递给它们一个参数会发生错误。你可以把这两个方法看成是：

“这两个数里面哪一个是最大的？”

以下的代码演示了这两个方法是如何工作的：

```
class MaxMin{
public static void main(String argv[]){
        System.out.println(Math.max(-1,-10));
        System.out.println(Math.max(1,2));
        System.out.println(Math.min(1,1));
```

```
        System.out.println(Math.min(-1,-10));
        System.out.println(Math.min(1,2));
        }
}
```

下面是输出的结果：

```
-1
2
1
-10
1
```

random

该方法返回的是一个 0.0 到 1.0 之间的随机数。

不像一些随机数系统，Java 似乎并不支持提供种子数来增加随机性。这个方法可以用下面的方法来生成 0 到 100 之间的随机数。

从考试的角度来说，这个“返回一个 0.0 到 1.0 之间的随机数”的知识点是很重要的。因此下面的几个数字是可能输出结果：

0.9151633320773057
0.25135231957619386
0.10070205341831895

经常会遇到需要程序生成 0 到 10 之间或者 0 到 100 之间的随机数的情况。下面这行代码就是演示如何生成 0 到 100 之间的随机数：

```
System.out.println(Math.round(Math.random()*100));
```

round

返回最接近参数的一个整型数。如果小数部分大于 0.5 则返回下一个相对最小整数，如果小数部分小于等于 0.5 则返回上一个相对最大整数。如下例所示：

2.0 <=x < 2.5. then Math.round(x)==2.0
2.5 <=x < 3.0 the Math.round(x)==3.0

以下是一些例子和它们的输出：

```
System.out.println(Math.round(1.01));
System.out.println(Math.round(-2.1));
System.out.println(Math.round(20));
```

```
1
```

-2
20

sin cos tan

这三个方便快捷的方法都只需要一个 double 型的参数，它们的功能和其他语言里面的方法功能是一样的。在我 12 年的编程工作中，我还从未使用过它们。可能需要记忆的仅仅是参数类型是 double 型的。

sqrt

返回该参数的 double 型平方根。

总结

max 和 min 方法需要两个参数。
random 方法返回的数值在 0 到 1 之间。
abs 返回的是绝对值。
round 返回最接近参数的整型数，但保留符号位。

练习题

习题 1） 下列哪个选项将会编译正确？
1) System.out.println(Math.max(x));
2) System.out.println(Math.random(10,3));
3) System.out.println(Math.round(20));
4) System.out.println(Math.sqrt(10));

习题 2）下列哪个选项将会输出 1 到 10 之间的随机数？
1) System.out.println(Math.round(Math.random()* 10));
2) System.out.println(Math.round(Math.random() % 10));
3) System.out.println(Math.random() *10);
4) None of the above

习题 3）写面一行代码将会输出什么？
System.out.println(Math.floor(-2.1));

1) -2
2) 2.0

3) -3
4) -3.0

习题 4）写面一行代码将会输出什么？

System.out.println(Math.abs(-2.1));

1) -2.0
2) -2.1
3) 2.1
4) 1.0

习题 5）写面一行代码将会输出什么？

System.out.println(Math.ceil(-2.1));

1) -2.0
2) -2.1
3) 2.1
3) 1.0

习题 6）当你试图编译下列代码时将会发生什么？

```
class MyCalc extends Math{
public int random(){
        double iTemp;
        iTemp=super();
        return super.round(iTemp);
        }
}

public class MyRand{
public static void main(String argv[]){
        MyCalc m = new MyCals();
        System.out.println(m.random());
        }
}
```

1) Compile time error
2) Run time error
3) Output of a random number between 0 and 1
4) Output of a random number between 1 and 10

练习题答案

答案 1）

3) System.out.println(Math.round(20));
4) System.out.println(Math.sqrt(10));

选项 1 错误是因为 max 方法只需要一个参数，而选项 2 错误是因为 random 方法只不需要参数。

答案 2）
4) None of the above
最接近正确答案的是选项 1，但是请别忘记一个细节就是，random 方法返回的数据中包括 0，而题目问的是 1 到 10

答案 3）
4) -3.0

答案 4）
3) 2.1

答案 5）
1) -2.0

答案 6）
1) Compile time error
Math 类是不可被继承的。这段代码有一些低级的错误。你只能在构造方法里面使用 super，而它却是在 random 方法里面使用。

目标二 Strings 的不变性

描述 string 对象不变性的重要性。

String 类的不变性理论说明， string 对象一旦被创建，它就决不能被改变。Java 编程的一些经历意味着似乎并不如此。
如下面的代码所示：

```
public class ImString{
public static void main(String argv[]){
        String s1 = new String("Hello");
        String s2 = new String("There");
        System.out.println(s1);
        s1=s2;
        System.out.println(s1);
        }
}
```

如果 Strings 不能被改变，那么 s1 应该仍然打印出 Hello,但是你如果运行这个程序段，你会发现第二次输出的字符串是“There”，这是为什么呢？
不变性实际上指的是字符串指针所指向的内容。在例子中，将 s2 赋给 s1，字符串池中

“Hello”字符串不再被指向，s1 现在和 s2 指向同一个字符串。事实上“Hello”字符串没有被修改，理论上，你不能再获取到它了。

这个目标要求你认清 strings 的不变性，如果你想要改变字符串的内容的话，主要的方法就是采用 StringBuffer 类。

因为在后台实例化时，字符串连接会产生一个新的字符串，所以当你的操作大量的字符串时，比如从读取一个大的文本文件时，性能就很重要了。通常字符串不变性并不影响每天的编程，但是在考试中它经常被考到。记住不论怎么考，字符串一旦被创建，它本身就不会改变，即使指向它的指针指到别的字符串了。如果允许同一字符串再生，这就涉及到字符串在字符串池中的创建方式了。5.2 节在讲解在使用 strings 时＝与 equal 的作用时，将这个内容作为一个部分涉及到了。虽然 Java2 和 Java1.1 都没有特别到这个内容，但是我认为一些问题的回答需要 StrngBuffer 的内容。

练习题

习题 1）

已经创建了两个包含姓名的字符串，即：

String fname="John";

String lname="String"

你如果在同一个代码块中，改变这些字符串的值？

1)

fname="Fred";

lname="Jones";

2)

String fname=new String("Fred");

String lname=new String("Jones");

3)

StringBuffer fname=new StringBuffer(fname);

StringBuffer lname=new StringBuffer(lname);

4) 以上都不正确

习题 2）

假如你写了一个程序用于读取 8MB 的文本文件。一行一行的读到一个 String 对象中，但是你发现执行性能不好。最可能的解释是？

1) Java I/O 是围绕最慢的设备而设计的，它本身就很慢
2) String 类不适合 I/O 操作，字符数组将更合适
3) 因为 String 的不变性，每一次读要创建一个新的 String 对象，改为 StringBuffer 可能会提高性能
4) 以上都不正确

练习题答案

答案 1）
4）以上都不正确
一旦创建了一个 String 对象，它就只能读不能改变

答案 2）
3）因为 String 的不变性，每一次读要创建一个新的 String 对象，改为 StringBuffer 可能会提高性能
我希望你们都不会像 C 程序员那样采用一个字符数据？

目标三 包装类

本目标主要讨论包装类的重要性，包括因为特定的需求选择最合适的包装类。讲述当一个包装类的的实例代码片段运行回产生什么结果。DoubleValue, floatValue, longValue,parseXxx,getXxx,toString,toHexString 等等。

本节需要注意的问题

该目标的知识点明确在 JDK1.4 版本的考试中有明确规定，如果你看过以前的旧模拟题，你肯定不会看见里面包含本目标中的知识。因为在实际开发过程中你经常会用到本目标中的内容，所以学习起来会很容易。要特别仔细的学习这些知识点，你将会在真题库中看到它们的影子。

什么是包装类

Java 中的基本类型的包装类提供了大量非常有用的公用方法。比如你需要往一个 vetor 里面存储一列整型数据，而 Vetor 里面存储的对象类型必须是 Object 而不是基本类型数据，当你从 Vetor 中将这些对象再取出来的时候，你得要用相应基本类型的包装类中的 toxxValue 公用方法来将对象强制转换成相应的基本类型的数据。下面的代码讲述了这种技巧：

```
import java.util.*;
public class VecNum{
    public static void main(String argv[]){
    Vector v = new Vector();
    v.add(new Integer(1));
    v.add(new Integer(2));
    for(int i=0; i < v.size();i ++){
        Integer iw =(Integer) v.get(i);
        System.out.println(iw.intValue());
    }
   }
}
```

包装类提供了该对象与整形数据之间相互转换的公用方法，因此当你有一个 String 型的数据当你需要将它转换成它所代表的整型数据的时候，你可以使用包装类来完成这一系列操作。

包装类中提供的公用方法是静态的，所以你不需要实例化一个包装类的对象再对它里面的方法进行调用。当你对一个包装类赋值以后，你将不能再改边它。**如果你在考试中遇到诸如 Integer.setInt(int i)的表述，不用多想，这种方法是不存在的，它是错误的。**

公用方法

一个最有用的包装类的公用方法是一些诸如 parseXX 的方法，它的作用是把一个 String 型的数据转换成一个它所对应的基本类型数据。XX 代表包装类所能包括的数据类型。它包括 parseInt, parseLong, parseShort, parseCharacter, parseBoolean。如果你在一个 WEB 页面里面有一个字段代表一个数据类型，

如果你有一个里面包含一个表格字段的WEB页面,返回的一个String型数据可以转化成一个数值。因此该字段可能包含"101"或者"elephant"。你可以试着用包装类将这些String型的数据转化成基本数据类型.如果它不能被适当的转化(比如其中包含"elephant"),一个NumberFormatException将会被抛出。

这里有一个例子,讲解了如何将一个可能可以转化成整型的String数据转化成整型数据,当转化不能进行的时候打印出错误信息。

包装类可以构造该包装类所包装的数据类型,以及可以转化为该类型的String型。所以Integer类型的包装类可以保存任何的整形数据,但是如果你试图将一个浮点数传递给它的时候一个错误将会发生。记住,包装类不是基本数据类型,它的实例的操作方式和其他对象是一样的。你可能对考试中出现的一些代码很疑惑,它们使用数学操作符对包装类的实例进行操作。你显然可以使用“＋”来对包装类的实例进行操作，它会在后台调用toString方法。不过小心当你看到“－”，　“％”和　“＊”的时候。

```
public class String2Int{
    public static void main(String argv[]){
    try{
        int i= Integer.parseInt(argv[0]);
        System.out.println("Coverted to int val = " + i);
        }catch(NumberFormatException nfe){                    System.out.println("Could not
covert to int");
        }
    }
}
```

toHexString

toHexString方法以十六进制的无符号整数形式返回一个整数参数的字符串表示形式。它有一个孪生的兄弟方法，它能够以二进制（基数 2）无符号整数形式返回一个整数参数的字符串表示形式。对这两个方法的用法理解需要对二进制数和十六进制数的概念有一定了解。你需要知道和对象相关的比特偏移的概念。下面的代码将会输出10接着100的串。

```
public class NumberFormats{
```

```
    public static void main(String argv[]){
    System.out.println(Integer.toBinaryString(4));
    System.out.println(Integer.toHexString(16));

    }
}
```

练习题

习题 1)　下列哪项表述是正确的？

1) The Integer class has a String and an int constructor
2) The Integer has a floatValue() method
3) The wrapper classes are contained in the java.lang.Math package
4) The Double class has constructors for type double and float

习题 2)　当你试图编译运行下列代码的时候会发生什么？

```
public class WrapMat{
    public static void main(String argv[]){
    Integer iw = new Integer(2);
    Integer iw2 = new Integer(2);
    System.out.println(iw * iw2);
    System.out.println(iw.floatValue());
   }
}
```

1)Compile time error
2) Compilation and output of 4 followed by 2.0
3) Compilation and output of 4 followed by 2
4) Compile time error, the Integer class has no floatValue method

习题 3)　当你试图编译运行下列代码的时候会发生什么？

```
public class TwoEms {
    public static void main(String argv[]){
        Object[] oa = new Object[3];
        oa[0] = new Integer(1);
        int i = oa[0];
        System.out.print(i);

    }

}
```

1) Compile time error an array cannot contain object references
2) Compile time error elements in an array cannot be anonymous
3) Compilation and output of 1

4) Compile time error Integer cannot be assigned to int
5) Compilation and output of the memory address of the Integer instance

习题 4)　当你试图编译运行下列代码的时候会发生什么？

```
public class TwoPack {
    public static void main(String argv[]){
        Integer iw = new Integer("2");
        Integer iw2 = new Integer("2");
        String sOut = iw + iw2;
        System.out.println(sOut);

    }

}
```

1) Compile time error, the + operator cannot be applied to Integer
2) Compilation and output of 22
3) Compilation and output of 4
4) Compile time error, Integer has no String constructor

习题 5)　下列哪段代码是正确的？

1) System.out.println(Integer.toBinaryString(4));
2) System.out.println(Integer.toOctalString(4));
3) System.out.println(Integer.add(2,2));
4) Float[] ar = new Float[] { new Float(1.0), new Float(2.1)};

练习题答案

答案　1 ）

1) Integer类有一个整型和String型的构造方法
2) Integer类有一个floatValue()方法
4) Double类有一个float型和double型的构造方法

答案　2 ）

1)Compile time error
包装类的实例不能像基本数据类型那样进行操作，注意Integer确实有一个 floatValue方法

答案　3 ）

4) Compile time error Integer cannot be assigned to int
这段代码可以通过 Integer类的intValue 方法正常进行。 它是一个包装类对象不能赋给基本数据

类型变量。

答案　4）

1) Compile time error, the + operator cannot be applied to Integer

包装类的实例不能像基本数据类型那样进行操作，它们是对象的实例，你必须将起进行转换成基本数据类型来进行数学操作。

答案　5）

1) System.out.println(Integer.toBinaryString(4));

2) System.out.println(Integer.toOctalString(4));

4) Float[] ar = new Float[] { new Float(1.0), new Float(2.1)};

包装类的实例不能像基本数据类型那样进行操作，因此选项 3 中的add方法不存在。如果年纪需要那样操作，你需要将其转化成基本数据类型。

第 9 章　Java 的 Util 包

目标一　Collection 类/接口

为了满足特定的开发需求，选择合适的 Collection 类/接口。

本节需要注意的问题

虽然没有特别的提到，但是对本小节的集合类的知识点是Java2版本考试的新考点之一，考试中关于新出现的集合类的题目非常简单，只需要应试者知道在哪如何使用这些类，而不需要应试者完全清楚底层细节的方法和字段。

旧的 Collection　类/接口

Java2 通过一些新增加的类/接口加强了集合类的用途，早一些的 Java 版本的 Collection 类包括：

vector

hashtable

array

BitSet

在这些类当中，只有 array 包含在 1.1 版本的认证考试的考点，从 Java1.1 开始，就是对所有开发情况中经常需要用到的排序功能提供了支持是导致 Java 越来越臃肿的一个原因。

新的 Collection 类/接口

集合类的最底层是 Collection 接口，它提供了一系列所有集合类开发中常用到的方法。在开发中，或许你从未在你创建的类中实现 Collection 接口，那是因为 Java 提供了一系列 Collection 接口的子类/接口。

Java2 的 API 包含了以下几个新的 Collection 接口

Sets

Maps

所有实现 Collection 接口的类存储对象为元素而不是原始数据类型，这种机制有个缺点就是创建对象对性能的影响，而且元素在使用之前必须从 Object 类型强制转换成合适的类型，这也同时意味着集合类不要求元素是同一类型的，因为一个 Object 对象可以是任何东西。

Set

Set 是一个不可包含重复元素的集合类接口，这恰好和关系数据库中返回某一条记录的 set 概念相符合。Set 接口的奥妙就在于它的 add 方法。

add(Object o)

任何一个传递给 add 方法的对象必须实现 equals 方法，这样保证与已存数据进行对比。如果已经存在该数据，那么调用 add 方法不会对该 set 起任何影响并且返回 false。这种试图添加一个元素返回 false 的思想更像是 C/C++中使用的机制而不是 Java，在这种情况下，大多数 Java 的其他类似添加方法选择的是抛出异常。

List

list 是一个有序的可以有重复元素的集合类接口，该接口中一些重要的方法如下:

add

remove

clear

JDK 的帮助文档给出了使用 List 处理一个实际 GUI list 进行控制一列包含名为 Planets 的列表的例子。

Map

Map 是一个接口，实现它的类不能包含重复的 key,这一点和 hashtable 很相似。

为什么我们使用集合类而不使用数组？

相比较而言，集合类相对于数组一个最大的优点就是它可以自增长，你不需要在创建它的时候为它分配大小空间，缺点就是集合类只能存储 Object 对象，而不能存储原始数据类型，因此不可避免的影响了一定的性能。数组不能直接支持排序，但是这点可以通过使用静态的集合类方法来克服。以下是一个例子。

```
import java.util.*;

public class Sort{
    public static void main(String argv[]){
    Sort s = new Sort();
    }

Sort(){
    String s[] = new String[4];
    s[0]="z";
    s[1]="b";
    s[2]="c";
    s[3]="a";
    Arrays.sort(s);
    for(int i=0;i< s.length;i++)
    System.out.println(s[i]);
    }
}
```

使用 Vector

下面的例子解释了怎样将不同类的对象添加到一个Vector里面.这与数组不同,不要求每个元素必须同类型.代码会将每个对象输出到标准输出设备,它隐性调用了每个对象的toString()方法到了Java2 Vector类成为创建一个可变大小数据结构的主要方法.可以用remove()方法从Vector类移出元素

```
        import java.awt.*;
        import java.util.*;
        public class Vec{
        public static void main(String argv[]){
        Vec v = new Vec();
        v.amethod();
          }//End of main

        public void amethod(){
         Vector mv = new Vector();
```

```
            //Note how a vector can store objects
            //of different types
            mv.addElement("Hello");
            mv.addElement(Color.red);
            mv.addElement(new Integer(99));
            //This would cause an error
            //As a vector will not store primitives
            //mv.addElement(99)
            //Walk through each element of the vector
            for(int i=0; i< mv.size(); i++){
            System.out.println(mv.elementAt(i));
                  }
    }//End of amethod
}
```

在 Java2 之前 Vector 类是创建可重新分配大小的数据结构的主要手段。可以使用 remove()方法将元素从 Vector 中剔除掉。

使用 Hashtables

Hashtables 有点像 Visual Basic 中使用键来索引相应的键值的概念。除了用数值来对应元素以外，它的效果很像 Vector。哈希表的名字部分通过引用数学概念中的数字索引概念来进行解决。一个 hashtable 比 Vector 优越的地方就在于快速的查找。

BitSet

正如 BitSet 所暗示的，它存储的是一个序列比特。不要被它名字部分的 set 部分误导，它不同于数学中或者数据库领域的 set,并且它和 Java2 中提供 Sets 没有任何关联。它更适合于被看作一个存储比特的容器。一个 BitSet 适合更有效率的存储一序列代表正/否值的比特值。其他可供选择的某些集合类在存储布尔值方面不及它有效率。

参考一下 Bruce Eckel 的《Thinking in Java》：

如果仅仅是从存储的角度来看它是很有效率；如果你期待更有效率的访问，它比一些原生类型的数组要稍微慢一些。

BitSet 是一个在开发中从未需要使用的比较生僻的类，我认为它在密码领域或者图片处理的开发过重使用起来比较方便。下面让我看看你是否能够应付来自 Java2 考试的习题。

练习题

习题 1）

下面哪些是集合类？

1) Collection
2) Iterator
3) HashSet
4) Vector

习题 2）

关于 Collection interface 下面哪些是正确的？

1) The Vector class has been modified to implement Collection
2) The Collection interface offers individual methods and Bulk methods such as addAll
3) The Collection interface is backwardly compatible and all methods are available within the JDK 1.1 classes
4) The collection classes make it unnecessary to use arrays

习题 3）

下面哪些是正确的？
1) The Set interface is designed to ensure that implementing classes have unique members
2) Classes that implement the List interface may not contain duplicate elements
3) The Set interface is designed for storing records returned from a database query
4) The Map Interface is not part of the Collection Framework

习题 4）

下面哪些是正确的？

1) The elements of a Collection class can be ordered by using the sort method of the Collection interface
2) You can create an ordered Collection by instantiating a class that implements the List interface
3) The Collection interface sort method takes parameters of A or D to change the sort order, Ascending/Descending

4) The elements of a Collection class can be ordered by using the order method of the Collection interface

---

习题 5）

你希望存储少量数据并能快速访问. 你并不需要排序这些数据， uniqueness is not an issue and the data will remain fairly static 那种数据结构最适合这种需求?

1) TreeSet
2) HashMap
3) LinkedList
4) an array

习题 6）

下面哪些是 Collection 类?

1) ListBag
2) HashMap
3) Vector
4) SetList

习题 7）

怎样从 Vector 中移出元素?

1) delete method
2) cancel method
3) clear method
4) remove method

练习题答案

答案 1）

3) HashSet
4) Vector

另外两个是接口不是类

答案 2）

1) Vector 类已经被修改用类实现 Collection
2) 集合类方法提供了单个的方法和 addAll 等的批量方法。
集合类是随着 JDK1.2 新推出的. 除了旧的集合类如 Vetor,Bitset, 如果你在旧的平台上运行包含了新集合类的代码，将会抛出异常。

答案 3）

1) Set 接口是为了确保正在执行的类有特定的成员。

实现 List 接口的对象中可以包含重复的元素.尽管一个实现 Set 接口的类的元素存储的可能是用来数据库查询结果, 但它不是为了那个目的专门设计的。

答案 4）

2) 你可以通过实力化一个实现 List 接口来创建一个有序的集合类。

答案 5）

4) 一个数组

像这些简单的需求用数组是最合适的了。

答案 6）

2) HashMap
3) Vector

JDK1.2（Java2）中 Vector 这个类被“加装到”集合框架中来了。

习题 7）

4) remove 方法

目标二 实现 **hashCode**

正确与错误 hashCode 实现方法的区别

本节需要注意的问题

此宗旨是与 JDK1.4 的发布新推出的. Sun 的网站上显示这个宗旨时,用小写字母 c 拼写 hashcode,但从 Object 对象继承的方法却用了大写的 C 拼作 hashCode.这是一个已经引入到此宗旨的奇怪的话题,处理大量的非常严肃的 java 编程,却不必麻烦你实现 hashcode.真正的数据库例子不会让你困惑于此问题,但是这是你应该理解的宗旨.

它来自 **Object** 对象

这个 hashcode 方法继承自所有类对象的父类,所以任何对象的实例都可以调用 hashcode 方法,此 hashcode 方法的签名是:

public int hashCode()

所以,你可能会遇到 hashcode 的一些伪签名,比如硕返回非 int 值或者带有非空参数.尽管如此,我怀疑问题会比这个稍微理论化一些.

返回 int 值是基于 hash 的集合类的特殊应用,例如

HashTable, HashSet, HashSet

基于 hash 的集合的本质是存储键、值.你用键来查找值。所以，举个例子，你可以用一个 HashMap 来存储职工的 id 作为键，职工名字作为值。

通常，一个 hashcode 值会是对象的内存地址。你可以很容易地用一些琐碎的代码来示范这个：

```
public class ShowHash{
    public static void main(String argv[]){
        ShowHash sh = new ShowHash();
        System.out.println(sh.hashCode());
    }
}
```

当我编译、运行这段代码，就会输出 7474923，这个就是运行程序时这个类内存地址的表示。这就说明了一个 hashcode 的一个特性：在运行不同程序时它会得到不同的值。如果你考虑一个对象的内存地址，就不能确定一个程序的不同运行所得到的值。

这里有一段来自 JDK1.4 的引用，它包含了一个 hashcode 值的要求

“不管什么时候，当一个 java 应用程序执行期间，多次援引同一个对象，hashCode 方法必须一致地返回同样的 integer 类型，对象上的无信息应用的 equals 比较就被修改了。这个整数在一个应用程序的不同执行可以不必保持一致性。”

既然它说，在同样的程序运行中 hashCode 的返回值必须一致，改变了对象上的无信息应用的 equals 比较，这个就告诉我们 equals 和 hashCode 方法之间的关系了。

equals 和 hashCode

由于每个对象都继承自一个叫 Object 的最终父对象，所以它们都可以访问 equals 方法。但是，当默认情况下，它只是简单地比较对象的内存地址。在用 String 类的时候，它的弊端就戏剧性地暴露出来了。如果 String 类不实现 equals 方法自己的版本，在比较两个字符串的时候就会比较它们的内存地址，而不是字符串序列。这显然不是你想要的，基于此，String 类实现了自己的 equals 方法，可以比较两个字符串。

这里有 API 文档的另一个重点。

如果两个对象用对象的 equals 方法比较是相等的，那么它们调用 hashCode 方法必须生成同样的整数值。

此原则可以用下面的代码来解释

```
public class CompStrings{
    public static void main(String argv[]){
    String s1 = new String("Hello");
    String s2 = new String("Hello");
    System.out.println(s1.hashCode());
    System.out.println(s2.hashCode());
    Integer i1 = new Integer(10);
    Integer i2 = new Integer(10);
    System.out.println(i1.hashCode());
    System.out.println(i2.hashCode());
    }

}
```

每次程序的运行，这段代码都可以输出 s1 和 s2，i1 和 i2 的同样 hashCode 值。理论上，在不同情况下会输出不同值。

当两个对象不等时

就象上面所写，用 equals 方法判断两个不同的对象一定会返回不同 hashCode 值，这是一个看似合理的推断。实际上不是，就像 API 文档所说的那样。

如果两个依据 java.lang.Object 的 equals 方法不相等的两个对象，分别调用 hashCode 方法一定会生成不同的整数值，这显然是不确定的。但是程序员应该了解这些。

同时，你也可以查询原始的 API 文档来理解 hashCode 方法的要求。

练习题

习题 1 ）下面所述哪些是正确的？

1)一个对象的 hashCode 方法会返回任何原始的整数类型

2)依据 equals 方法，两个相等的对象调用 hashCode 方法会生成同样的结果。

3)一个对象的 hashcode 方法在一个应用程序的不同执行，一定会返回同样的值。

4) Object 类的 hashcode 方法签名是 public int hashCode()

习题 2)

定义：

```
public class ValuePair implements Comparable{
    private int    iLookUp;


    public ValuePair(int iLookUp, String sValue){
        this.iLookUp=iLookUp;
    }


    public void setLookUp(int iLookUp){
        this.iLookUp = iLookUp;
    }
    public int getLookUp(){
        return iLookUp;
    }
```

```
    public boolean equals(Object o){
       if( o instanceof ValuePair){
    ValuePair vp = (ValuePair) o;
          if(iLookUp == vp.getLookup()){
        return true;
    }
    return false;
}



    public int compareTo(Object o) {
        ValuePair vp = (ValuePair) o;
        Integer iwLookUp= new Integer(vp.getLookUp());
        if(iwLookUp.intValue() < iLookUp){
            return -1;
        }

        if(iwLookUp.intValue() > iLookUp){
            return +1;
        }
        return 0;
    }

}
```

下面那个是有效的 hashCode 方法

1)

```
public int hashCode() {
    return (int) System.currentTimeMillis();
}
```

2)

```
public char hashCode(){
    reutrn (char) iLookUp;
}
```

3)

```
public int hashCode(){
    return iLookUp;
```

```
}
```

4)

```
public int hashCode(){
    return iLookup * 100;
}
```

习题 3)

给出下面的代码

```
public class Boxes{
    String sValue;

    Boxes(String sValue){
    this.sValue=sValue;
    }

    public String getValue(){
    return sValue;
    }

    public boolean equals(Object o){
    String s = (String) o;
    if (sValue.equals(s) ){
        return true;
    }else{
        return false;
    }



    }

    public int hashCode(){
    return sValue.hashCode();

    }
}
```

哪些是正确的

1) 正确地执行 hashCode 方法
2)此类不会编译，因为 String 没有 hashCode 方法

3)不正确地执行 hashCode 方法
4)类不会编译，因为 compareTo 方法不会执行

习题 4）

判断对错

如果正确地创建了一个对象，那么调用它的 hashCode 方法将返回同样的值

1) True
2) False

练习题答案

答案 1)

2)依据 equals 方法，两个相等的对象调用 hashCode 方法会生成同样的结果。

4) Object 类的 hashcode 方法签名是 public int hashCode()

答案 2)

3)

```
public int hashCode(){
    return iLookUp;
}
```

4)

```
public int hashCode(){
    return iLookup * 100;
}
```

hashCode 方法必须返回整数值就排除了返回一个 char 值的选项 2，选项 1 返回了毫秒形式的 time 类型，由于程序的单次运行一定会得到不同的值，所以就破坏了 hashCode 的一个特殊要求。正确选项 3 和 4 可能不是 hashCode 方法的好版本，但是它们一致地得到相等值并返回正确的数据类型

答案 3)

1)正确地执行 hashCode 方法

String 类有 hashCode 方法的自己实现。如果没有，它会继承 Object 对象的 hashCode 方法，此方法简单地返回对象实例的内存地址。

答案 4）

2) False

小心任何带 always 词的问题。对象类的 hashCode 方法默认返回对象的内存地址。Java 工作的一些知识告诉我们，不同执行不一定会得到同样的内存地址。一个 hashCode 方法在一个程序的同样运行下一定会返回同样的值，但在不同运行下不一定会。如果你测试一个对象实例的 hashCode，你可能发现在多程序运行期间，好像返回同样的内存地址，但是这并不是确定的。

附录

第二届 Java Cup 全国大学生信息技术大奖赛

简要说明

比赛时间： 2006 年 9 月 1 日 – 2006 年 12 月 31 日

参赛对象： 所有中国大陆及香港、澳门和台湾地区的全日制大专院校学生（包括硕士研究生和博士研究生）

参赛形式： 由学生自行组队参加，每队的总人数不超过 4 人，可跨校组队。每队自行邀请一名指导教师。

比赛内容： 使用 NetBeans 集成开发环境开发一个 NetBeans Plugin。

比赛流程： 第一阶段（2006 年 9 月 1 日 － 2006 年 11 月 30 日）通过网上评选和专家打分挑选二十支队伍进入第二阶段的比赛。第二阶段（2006 年 12 月 1 日 － 2006 年 12 月 31 日）通过专家打分评选出 6 支获奖队伍。

奖励办法： 一等奖一名：指导教师获得 Ultra 20 工作站一台，参赛队伍获得 Ultra 20 工作站一台。

二等奖两名：指导教师获得 Ultra 20 工作站一台，参赛队伍获得高档 Java 手机一部。

三等奖三名：指导教师获得 Ultra 20 工作站一台，参赛队伍获得中档 Java 手机一部。

所有进入第二阶段比赛的指导教师和参赛队员均可获得如下纪念品：

- 带有 Java 标志的 T-Shirt 衫一件

- Java Duke 一只
- 最新版本的 Solaris 10 操作系统安装光盘（DVD）一张
- 最新版本的 Solaris 10 参考书一本
- 最新版本的 Sun 开发工具大全安装光盘（DVD）一张
- 最新版本的 Sun 开发工具参考书一本

详细信息： http://gceclub.sun.com.cn/java_cup.html

2006 年大学生 SCJP 认证考试优惠活动

简要说明

活动时间： 2006 年 8 月 1 日 – 2006 年 12 月 31 日

参加对象： 所有中国大陆（不包括香港、澳门和台湾地区）的全日制大专院校全职学生（包括硕士研究生和博士研究生）

优惠价格： 满足如上条件的学生，均可以人民币 450 元的价格参加标准的 SCJP 认证考试。考试代号如下

310-035：SUN certified programmer for the Java 2 Platform 1.4
310-055: SUN certified programmer for the Java 2 platform, SE 5.0

参加办法：

1. 在校全职学生携带本人有效的身份证件和学生证件联系就近的 Thomson Prometric 授权的考试中心报名。您可以通过 http://www.prometric.com.cn/查询离您最近的考试中心。
2. 缴纳人民币 450 元考试费，预约考试时间。请预留 7 个日历日。例如：考生想 8 月 10 日参加考试，则需要在 8 月 3 日提交考试报名表。
3. 考生收到考试中心的通知参加考试。
4. 考生向考试中心索取考试发票。

注意事项：

1. 活动时间从即日起至 2006 年 12 月 31 日截止，不会延长。因此所有考生必须及时完成考试。
2. 活动对象为在校的全职学生。
3. SUN SAI 院校计划不受影响。
4. 本活动针对仅 310-025 考试，SUN 的其它国际认证考试科目不在此活动范围内。
5. 参加考生的学生在报名时，需要同时提供身份证件和学生证件。

详细信息： http://gceclub.sun.com.cn/student_scjp.html

2006 年大学生 SCJP 认证考试优惠活动

授权考试中心

CN4	北京	北京	中国计算机软件与技术服务总公司培训中心	010	51527255,51527257	651527257	海淀区学院南路 55 号中软大厦 2#楼 5 层培训中心	100081
CN58A	北京	北京	北京昆仑瑞通科技发展有限公司	010	51905758-108	51905766	北京市海淀区知春路 113 号银网中心 A 座 606 室	100086
CNFH	北京	北京	北京翼砺索思科技有限公司	010	51662885	65888169	北京市朝阳区朝阳门外大街 22 号泛利大厦 503 室	100020
CNQ4	北京	北京	中铁信息咨询有限公司	10	51836803/51836815/	51836947	北京宣武区马连道南路 1 号 2 层	100055
CNBU	上海	上海	上海交大南洋计算机教育中心	021	62824800,62835848	62835847	广元西路 55 号，浩然高科大厦 1906	200030
CNC	上海	上海	上海慧科利普计算机有限责任公司	021	5633-3154, 56331110，56331845(都是总机) 前台钱小姐	36033365	广中路 698 号欣武大厦四楼	200072
CNFU	上海	上海	上海文华专修学院	021	5633-3154, 56331110，56331845(都是总机)	62187386	上海市卢湾区巨鹿路 417 号欣广大厦 11 楼	200030

					前台钱小姐			
CN3R	广东省	广州	广州数网园网络有限公司	020	87114795		广州五山华南理工大学北区科技园 2 号楼 407 室	510640
CN97	广东省	广州	拓维信息有限公司	020	61246500	61246501	广州市龙口东路 340-350 号天诚广场裙楼 501.502 室	510630
CNBQ	四川省	成都	成都祥网飞络技术有限公司（科力国际特认证教育中心）	028	86749489/86756958	86749319	成都市鼓楼南街 117 号世贸中心 A 座 12 层	610015
CNK	四川省	成都	成都电子科大计算机学院培训中心	028	83200-998,83200-668	8320-0544	成都一环路东一段 159 号电子信息产业大厦一层	610054
CNN9	四川省	成都	成都博讯信息产业发展有限公司	028	028-84444828	84455589	成都市一环路东 5 段 102 号	6100061
CN3V	辽宁省	大连	大连东软教育服务有限公司	0411	84835337; 84835330	84835330	大连市软件园凌北路 18 号	116023
CNN1	辽宁省	大连	大连华信计算机技术有限公司	0411	84755115、84755117	84755996	中国大连七贤岭高新技术园区高新街 6 号，华信软件大厦	116023
CNPT	天津	天津	天津智知堂培训中心	22	23657188	23657188	天津市南开区复康路 25 号 天津市教育科学研究院 7 楼	300191

CN0D	新疆	乌鲁木齐	新疆财政厅计算中心新息网技信术训中心	0991	2832 034	2823136	乌鲁木齐人民路 25 号财政局三楼计算机培训 中心	830002
CN4K	新疆	乌鲁木齐	新疆联合信息发展有限公司	0991	2835900-8181,23011 11-8115	2835900*3001	乌市解放北路 1 号附 5 号通宝大厦 3 楼	830002
CNNS	广东省	佛山	南海东软信息技术职业学院	0757	6684505	6684612	广东省佛山市南海区南海软件科技园内	528225
CNT1	河北省	保定	河北软件职业技术学院	312	2060737	2038091	河北省保定市天威中路 929 号	071000
CN9G	甘肃省	兰州	甘肃工业大学	0931	2757939	2755806	甘肃兰州七里河兰工坪 85 号甘肃工业大学	730050
CNPG	内蒙古	包头	内蒙古科技大学教务处考试中心	472	5951609	5951609	内蒙古包头市昆区阿尔丁大街 7 号（内蒙古科技大学教务处）	014010
CNBP	江苏省	南京	南京网信息技术有限公司（原江苏万和）	025	83242999	83247360	南京市中山北路 26 号新晨国际大厦 24 层	210008
CNNE	广西省	南宁	南宁视网络信息有限公司	0771	13977159269	5314790	广西南宁市七星路 129 号	530022
CNB	江	南	南昌大	0791	8304409-1333	8337825	南昌市北京东路 339 号	330029

V	西省	昌	计算技术有学机工程公司限					
CNFE	福建省	厦门	厦门微算思计技术公机有限司	592	2236691、2236681	2236681	厦门市文屏路 14 号文屏大厦 3 楼 C1	361004
CNF18	安徽省	合肥	合肥文达学校	0551	3632099 8582010(考场)	3632244	安徽合肥市黄山路 373 号	230022
CN4P	内蒙古	呼和浩特	内蒙古科基技正有限公责司（原内蒙古教盈育有限责咨询责任公司）	0471	2267864/4912615	2267864	办公地址:内蒙古呼和浩特市新城南街 1 号禾 太电脑广场 4 层 邮寄地址:呼和浩特内蒙古大学文体馆 102 室	10021
CNN6	黑龙江省	哈尔滨	哈尔滨大工业大学东亚电子仪器开发有限责任公司	451	86214945/86415228/	86415228-214	黑龙江省哈尔滨市南岗区邮政街 434 号 201 室	150001
CN58F	黑龙江省	大庆	大庆高级培训中心	459	5952943/5971637/5963657	5963657	黑龙江省大庆市让胡路区龙十路 30 号,大庆 高培中心国际认证中心	163458
CN0B	山西省	太原	金飞峰育才培训中心	0351	8089998	7017208	太原市学府东街 80 号，天马大厦三层（山西大学北校门口）	030006
CNND	浙江省	宁波	宁波京粤电脑培训学	0574	87310480/87172376	87300786	宁波解放南路 67-7 号三楼	315010

			校（银河网络教育中心）					
CNBF	江苏省	扬州	扬州亿科技网有限公司	514	0514-7878388	7878388-88	江苏省扬州市扬子江中路 260 号	225000
CN98	江苏省	昆山	昆山市中科高薪企业	512	57386171	57313157	昆山市中华园衡山路 8 号（科高人才技术培训中心市南分部）	215300
CN9W	云南省	昆明	爱因森软件技术培训基地	871	6482390	5366651-816	云南省昆明市翠湖北路 100 号物资招待所 2 楼	650021
CNFR	云南省	昆明	云南爱迪科技有限公司	871	5158617	5158615	鼓楼路 164 号云南省科协青少年科技中心	650000
CN45A	浙江省	杭州	杭州联合信息技术有限公司	0571	28919803	28919805	杭州市文一路 115 号电子科技大学实训楼 7 楼	310012
CN4X	湖北省	武汉	武汉爱信息科技有限公司	027	87522501;87522530-803;59713102	87522501	珞瑜路 272 号关山高新大厦 502 房	430074
CN91U	辽宁省	沈阳	沈阳吉大计算机培训学校	024	83892386; 23917300	23994117	沈阳市和平区文萃路 6 甲 24 号-2-4 云顶 大厦四层,沈阳市吉大计算机培训学校	110004
CNNG	辽宁省	沈阳	沈阳银河科技有限公司	024	62200316,62200339	86230210	辽宁省沈阳市皇姑区崇山中路 69 号名仕之都 A 座二单元 12 楼 2 号	110031
CN19	山东省	济南	山东省科海天发展有限公	0531	88925400/88925499/88925480/89907661	88922241	山东济南市山大路 228 号 齐鲁软件大厦 C302 室	250014

			司					
CN44	海南省	海口	海龙有南电限威脑公司	898	68550270-75 转 16	68550277	海口市国贸大道 45 号银通大厦 12 层北区	570125
CN7	广东省	深圳	黎脑有明网限电络公司	0755	26550000(HQ) 83650333(APTC)	83650222	深圳市高新区南区黎明网络大厦	518057
CN8K	广东省	珠海	珠金子海步有市电限公司	0756	2110920	2134131	珠海市香洲区凤凰南路 1113 号怡华商业中 心 8 楼 807 室	519000
CNF20	河北省	石家庄	河电化北一中机体试地	0311	83992884,83991267	83992884	石家庄友谊南大街 46 号科学院 4 号楼二楼	050051
CN43	福建省	福州	福先软州务连件有领邦服限公司	591	87843320	87834279	福州鼓屏路 136 号银联宾馆 7 楼	350003
CN45	江苏省	苏州	苏区软州限欧件新索有公司	0512	6826 9928	6826 6745	苏州阊胥路 124 号	215004
CNJ	陕西省	西安	西大科安一计国高算术级中培心交金际机技训	029	82668596	82663773	咸宁西路 28 号西安交通大学计算机系金科培 训中心	710049
CNPW	贵州省	贵阳	贵州大学	851	8022382	6804481	Nanya road 67#,Yun yan District,Guizhou university	5500025
CNPC	河南	郑州	郑维州空六间	371	65356861	65356877	郑州市经三路财富广场 B 座 9 楼	450000

	省		电子有限公司					
CN0S	四川省	重庆	重庆中典学业实公司	023	65400721,65304429,65386539;65000539,65359837,65304429	65359837	沙坪坝区沙正街 125-2 号 科技贸易市场C幢底楼 112 室	400030
CN9H	宁夏	银川	宁夏大学	0951	2061281/2061434/2082953	2082963	宁夏银川市文萃路	750021
CN92	吉林省	长春	长春方息联信产业责有限任公司	431	2061111/ 5155777/ 5155888	5155999	吉林省长春市卫星路 8722 号中天大厦 409 室	130012
CNB6	湖南省	长沙	长沙开脑源软电件发有限公司	731	2234440,2232607,4441623,	2565352	长沙蔡锷中路 10 号恒隆国际大厦 907	410005
CNPQ	山东省	青岛	青岛私翰立晟培训IT学校	0532	85784462	85784462	青岛市南区闽江路 172 号软件大厦 612 室	266071