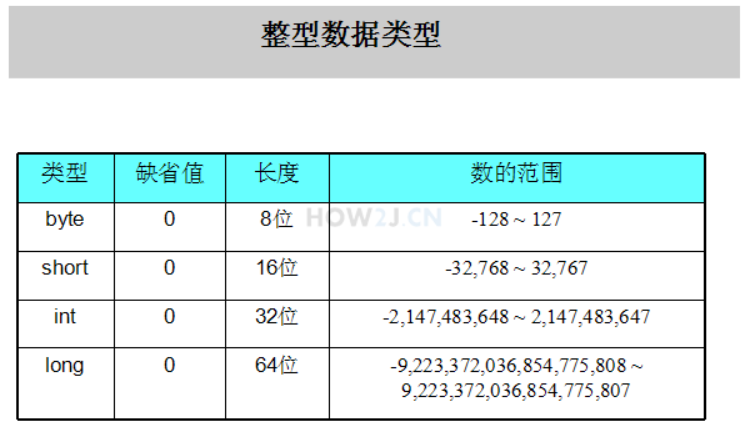
# Java复习

#### 初级JavaSE

###### 1、变量：

一个变量的类型，决定了该变量可以包含什么样的值

Java有八种基本类型，整型（4种），字符型（1种），浮点型（2种），布尔型（1种）

注：如果试图给byte类型的变量赋予超出其范围的值，则会编译报错，这时可进行类型转换。char类型只能存放一个字符，和short一样也是16位的，超过一个字符则会编译报错，值使用‘’包括起来（String类型的值使用“”包括起来）



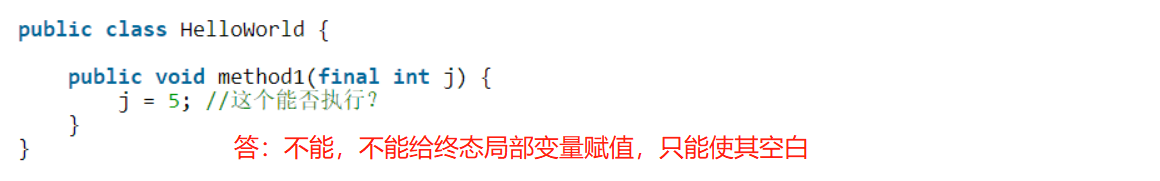
注意：默认的小数值是double类型的，所有float f2=54.112，是会直接编译报错的。可以把该数字声明为float型，在数字后面加一个字母f，如float f2=54.112f。



注意：布尔型真正存放的数据是0和1，但表示的时候不能用0和1来表示。得boolean b1=true；

###### 2、常量：

一般把常量声明成final类型，代表以后不会再修改它，该变量只有一次赋值的机会。如果需要给常量动态赋值，则可以放在static{}程序块中。而且最好放在一个单独的类中，这样其他的类就可以把它当成静态常量访问了。下面举例。



###### 3、实例化对象

因为基本类型是Java语言内置的特殊数据类型，所以基本类型变量赋值是不用new的，什么时候用到new？是在为对象分配内存空间的时候，称为实例化对象。

###### 4、类型转换：不同数据类型之间是可以按照一定的规则进行类型转换的



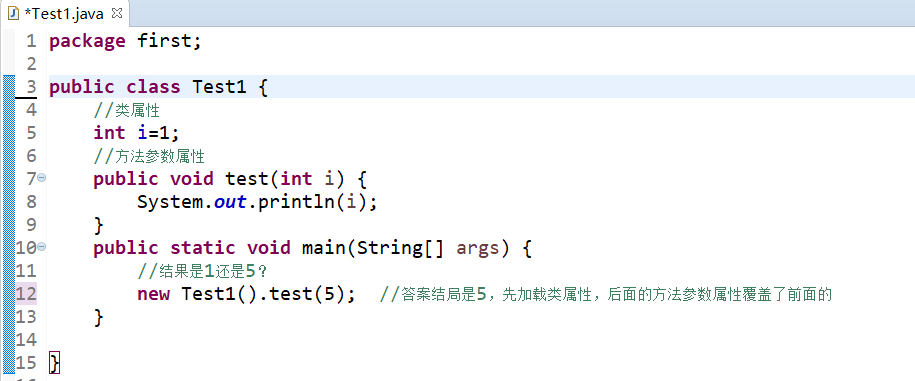
注意：强转的意思就是转是可以转，但是风险自负，毕竟高精度的数据类型转成低精度的数据类型可能会因为转换而丢失精度。

###### 5、命名规范：

通常类的首字母大写，方法的首字母小写，不要中式英文，常量名所有字母均大写，名字尽量起得让别人一看就知道这个类这个方法要干嘛，而且代码写好注释，不然自己以后回头看或者别人接手你的代码都是一种折磨。

###### **6、作用域：**

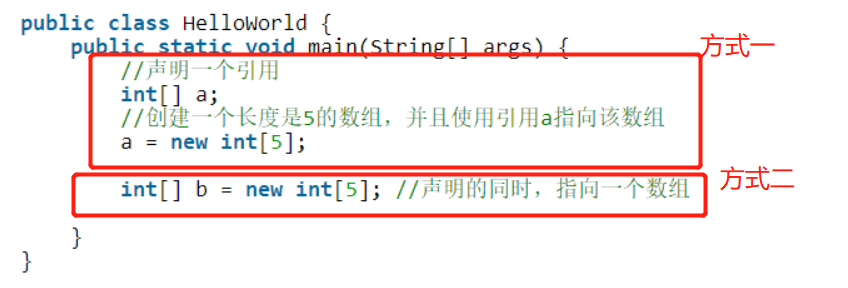
方法内部可以访问在其类里面的属性，但不能访问其类里面不同方法内部的属性，参数属性也不能访问其类里面不同方法内部的属性，如下图有个例子。



###### **7、数组：**数组可以看成是一个固定长度的，包含了相同类型数据的容器。

①声明数组 int[] a 等同于int a[] 注意：声明数组不代表会创建数组

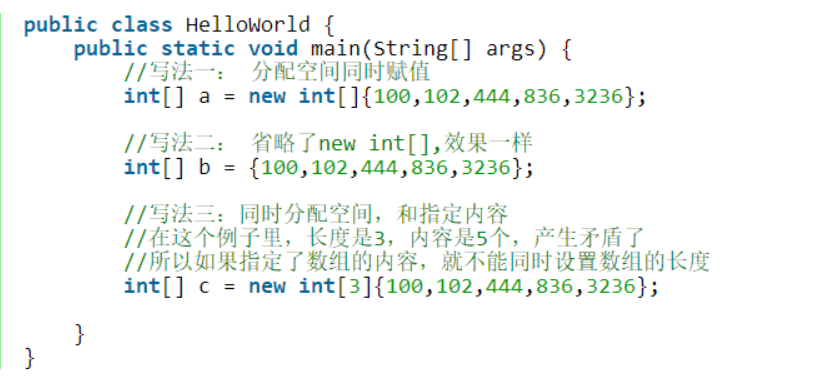
②创建数组 如下



③访问数组 a[2]

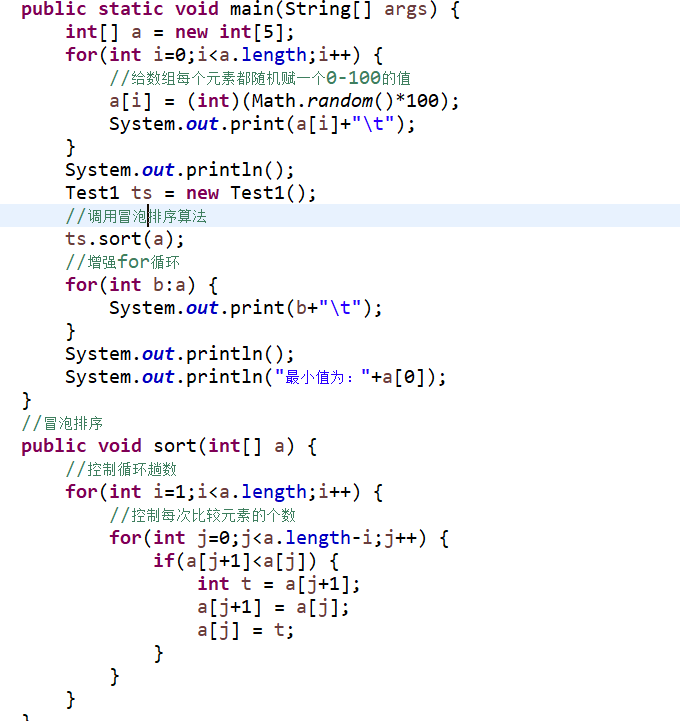
④数组长度 是不可以变的，一旦分配好空间是多少就是多少，不能增加和减少，数组索引范围是从0到length-1，超过则抛异常

⑤初始化数组写法

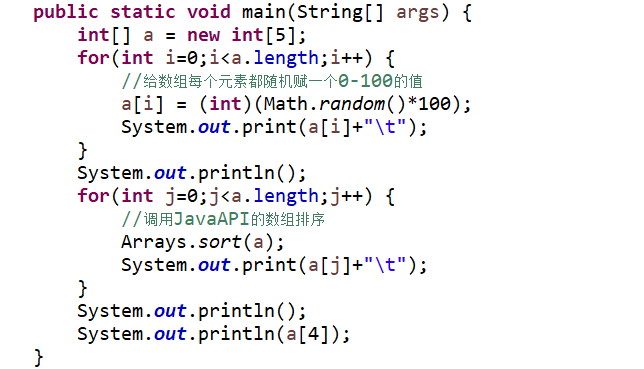


★练习：创建数组并排序找出最小值

方法一：冒泡排序解决

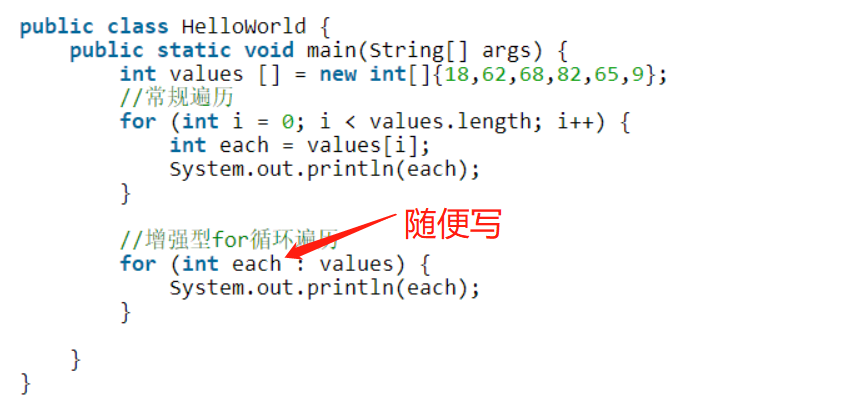


方法二：JavaAPI数组封装好的排序方法解决



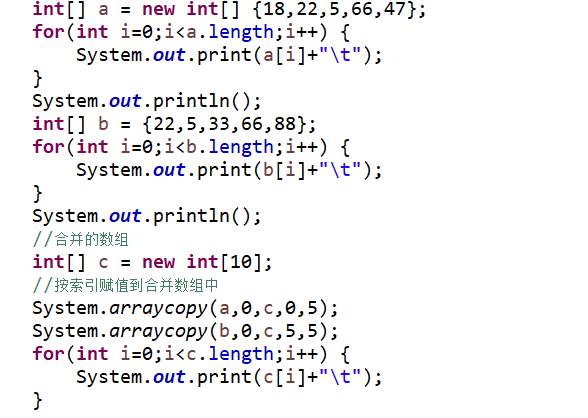
###### 8、增强for循环

注意：增强for只能用来循环取值，而不能用来修改数组里的值，要实现iterator接口的对象才能够用 for-earch循环遍历。



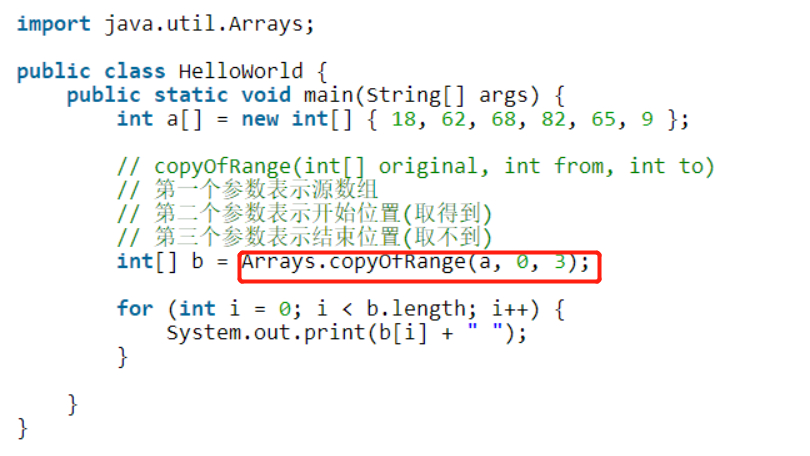
###### 9、System.arraycopy(源数组，索引，目的数组，索引，复制的长度)方法的使用

★练习：复制数组和合并数组

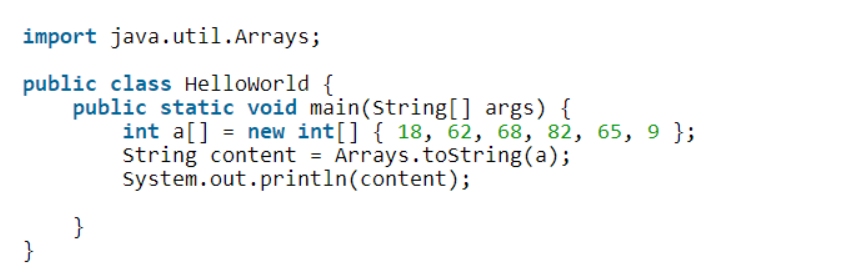


###### 10、Arrays工具类

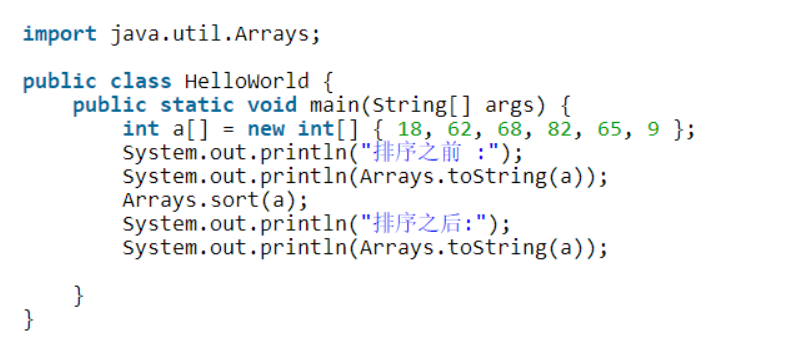
①数组复制 copyOfRange（）方法



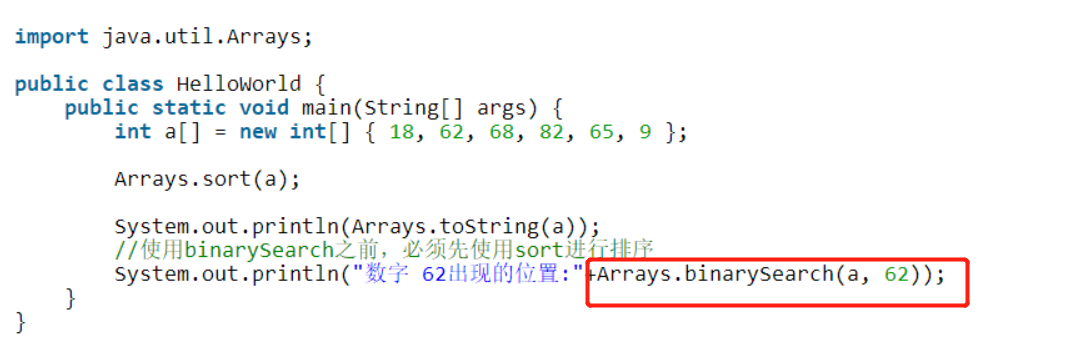
②把数组转换为字符串一次性打印出来 toString（）方法



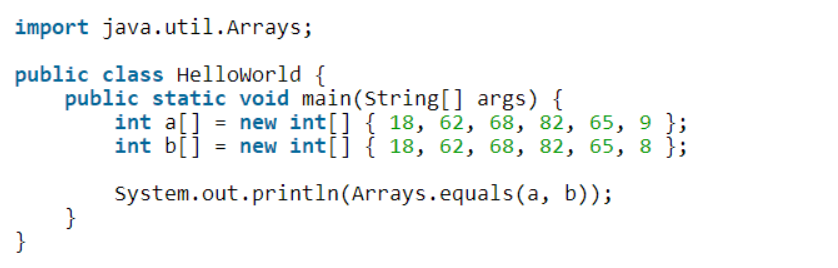
③排序 sort（）方法



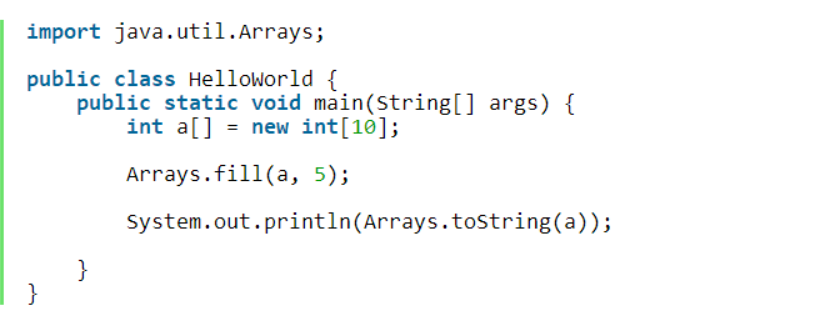
④搜索 binarySearch（）方法



⑤判断2个数组是否相同 equals（）方法



⑥使用同一个值填充数组 fill（）方法

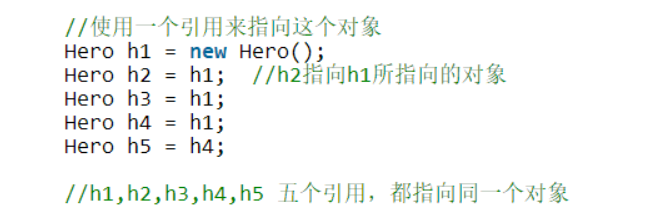


###### 11、类和对象

①如果一个变量的类型是类类型，而非基本类型，该变量则叫作引用。



注意：可多个引用，一个对象；



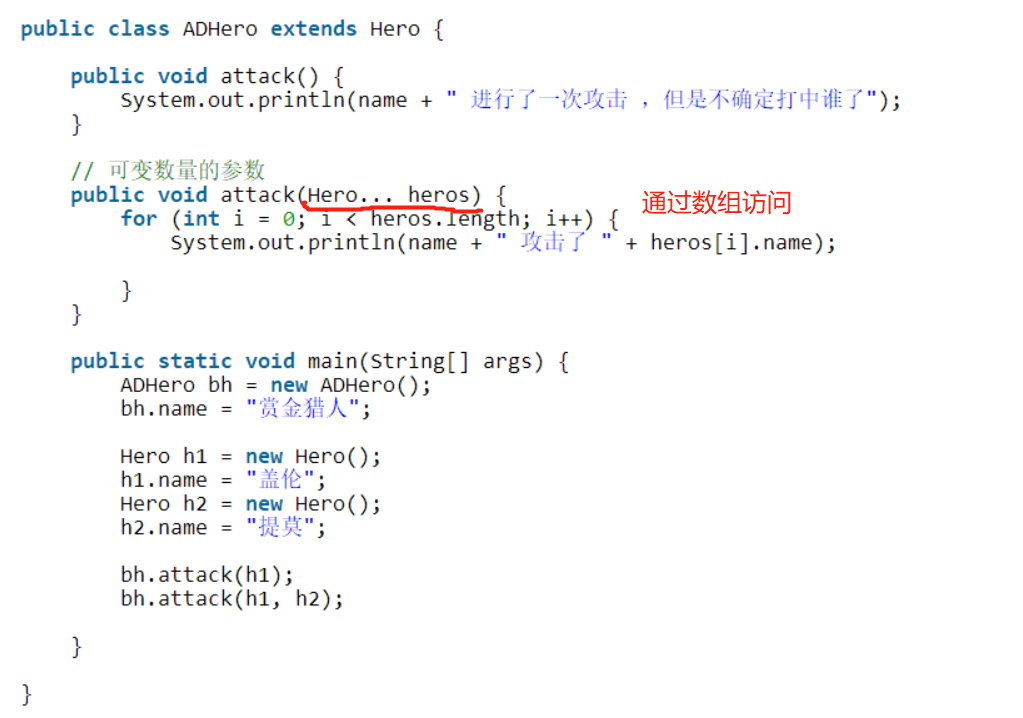
但一个引用，同一时间只能指向一个对象，即下图错误示范举例。



②继承

方法重载指的是方法名一样，但内部参数类型或者个数不一样

练习：可变数量的参数举例



③通过一个类创建一个对象，这叫做实例化。实例化是通过调用构造方法实现的。

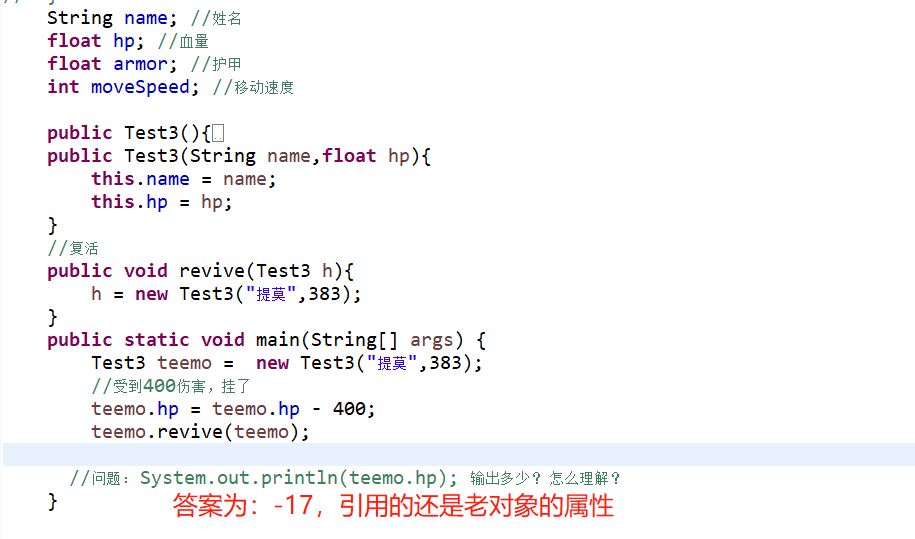
在实例化一个对象的时候必然调用到构造方法，如没写构造方法Java语言会自动生成一个无参的构造方法。

构造方法特征：没有返回类型；方法名与类名相同；构造方法也可以重载

④this：即指代当前对象

用途：this访问属性（如this.name）；this在构造方法里调用其他构造方法（如this(name)）

⑤传参 基本类型传参 和 类类型传参



###### 12、修饰符

private的属性：是私有的，自己类才可以访问，不同类还有子类都不能继承和访问。

package，friendly，default的属性：是默认的，自己类和同包类可访问，同包子类可继承，不同包子类和不同包类不可继承和访问。

protected的属性：是受保护的，自己类和同包类可访问，同包子类和不同包子类都可继承。

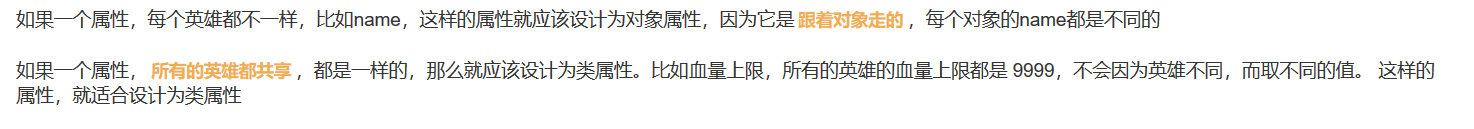
public的属性：是公有的，不同的属性都可访问和继承。



###### 13、类属性和类方法

由static修饰符修饰的属性，又叫作 静态属性；而对象属性，又叫做非静态属性。

如果一个属性声明了static静态属性，则所有对象共享这个属性。通过类.类属性 调用访问



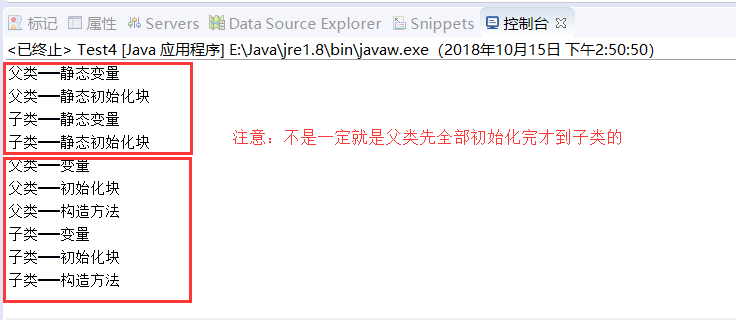
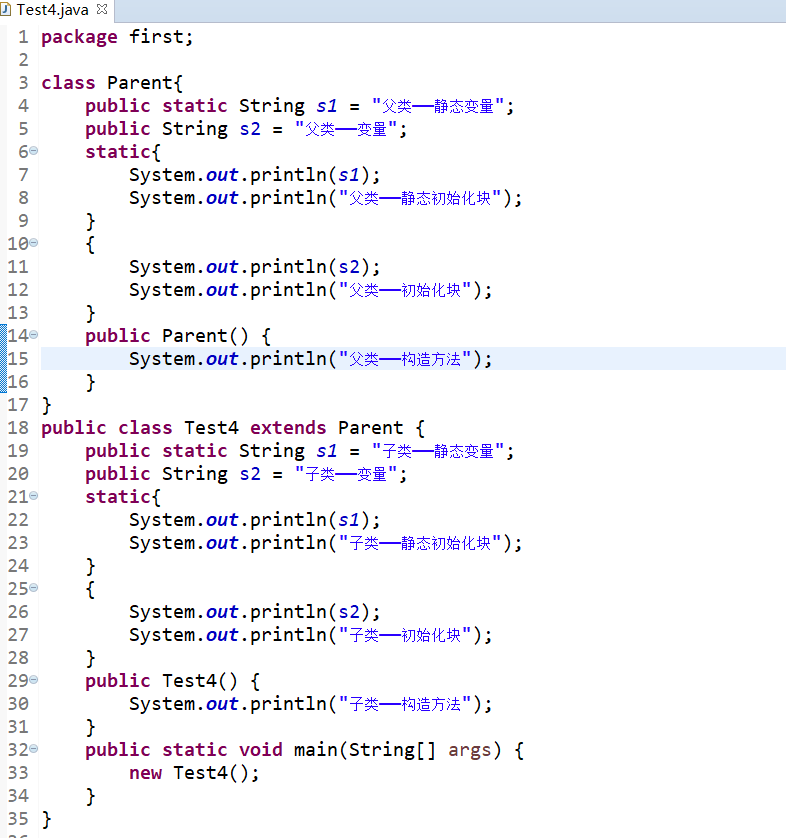
由static修饰符修饰的方法，又叫作 静态方法；而对象方法，又叫作实例方法，非静态方法，访问一个对象方法，必须建立在有一个对象前提下，通过对象来调用访问。而类方法则不需要对象的存在，通过类.类方法调用访问。（比如Math.random（）方法）

###### 14、属性中的初始化顺序

对象属性的初始化有三种：①声明该属性时直接初始化；②初始化块；③构造方法中初始化

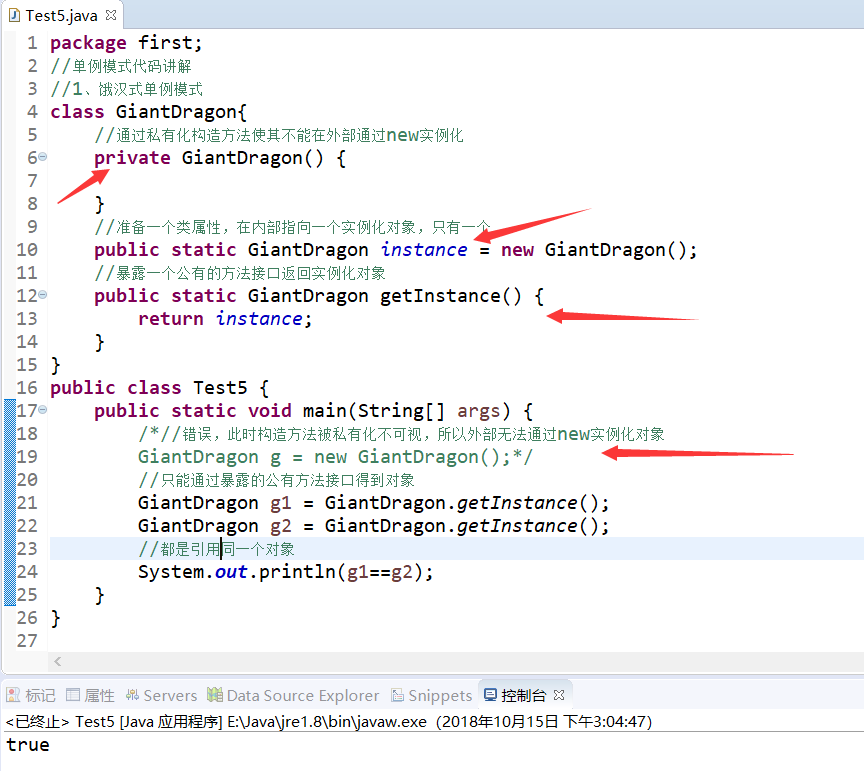
类属性的初始化有两种：①声明该属性时直接初始化；②静态初始化块

初始化顺序：（静态变量、变量）->（静态初始化块、初始化块）->构造方法

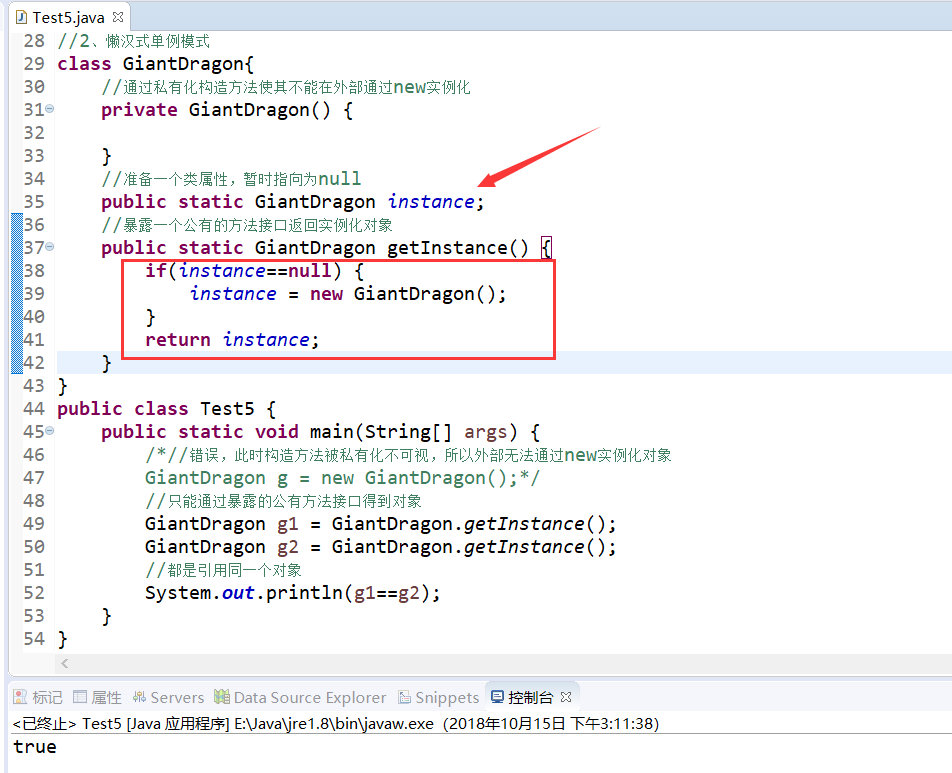


###### 15、单例模式

①饿汉式单例模式 无论何时都会创建一个实例



②懒汉式单例模式 只有在调用getInstance的时候才会创建实例



综上所诉的单例模式，什么时候使用饿汉式或者懒汉式呢？具体看业务需求，如果业务上允许有充分的启动时间和初始化时间，就用饿汉式，否则就用懒汉式。饿汉式是立即加载的方式，无论是否会用到这个对象，都会加载；如果在构造方法里写了耗性能的代码，比如数据库连接代码，那么在启动的时候就会有些卡顿。懒汉式是延迟加载的方式，只有使用了才会加载，并且有线程安全的考量，在启动时明显会比饿汉式快一些，但第一次调用的时候就会比较慢了。

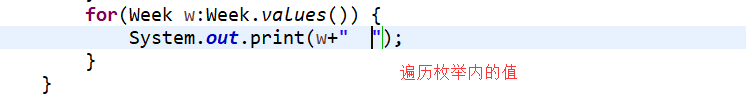
重点：单例模式面试题

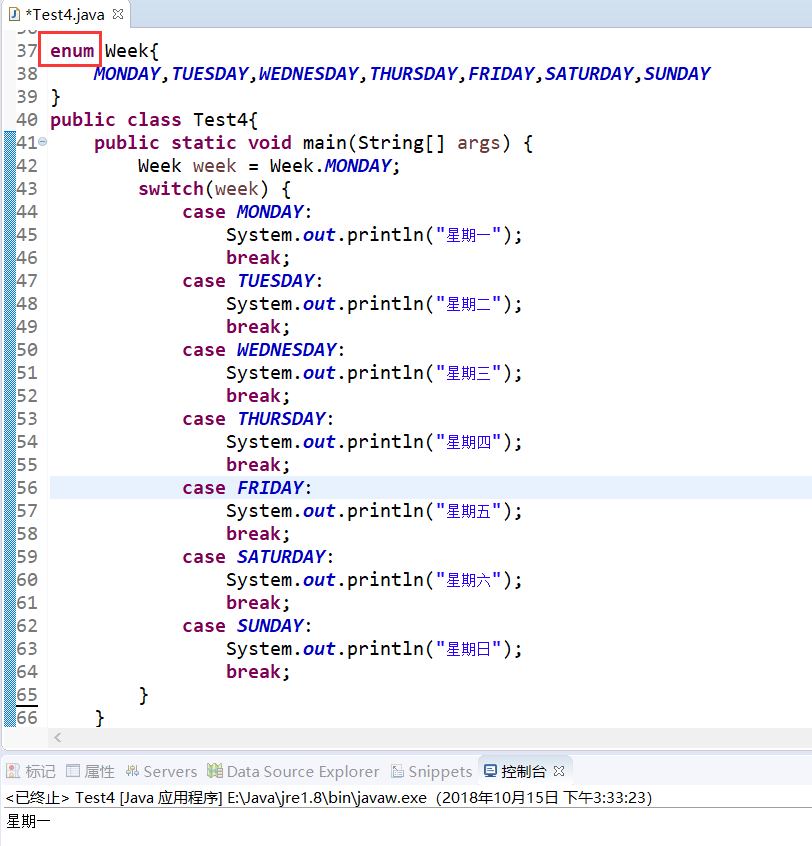
什么是单例模式？

①私有化构造方法；②准备一个静态属性来指向实例；③public static的静态方法getInstance（）返回第二步的静态属性。

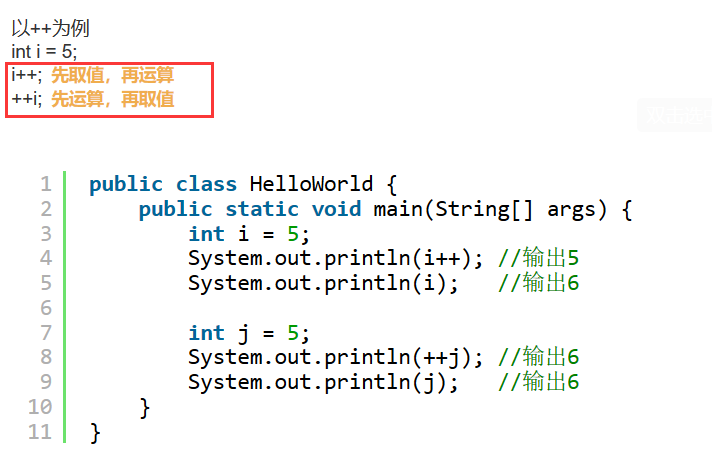
###### 16、枚举类型

其实也就是预先定义好的常量，所以枚举中的元素都采用大写命名。常常结合与switch语句，使用枚举可以很好的限制条件只在枚举范围内选择。下面用代码举例

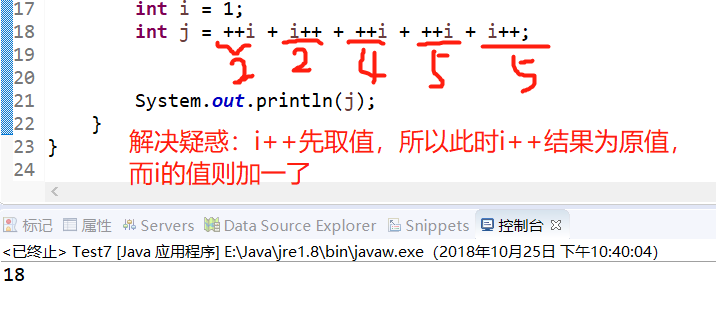




###### 17、自增自减运算



实例深入理解i++和++i的区别，如下代码：



###### 18、多线程

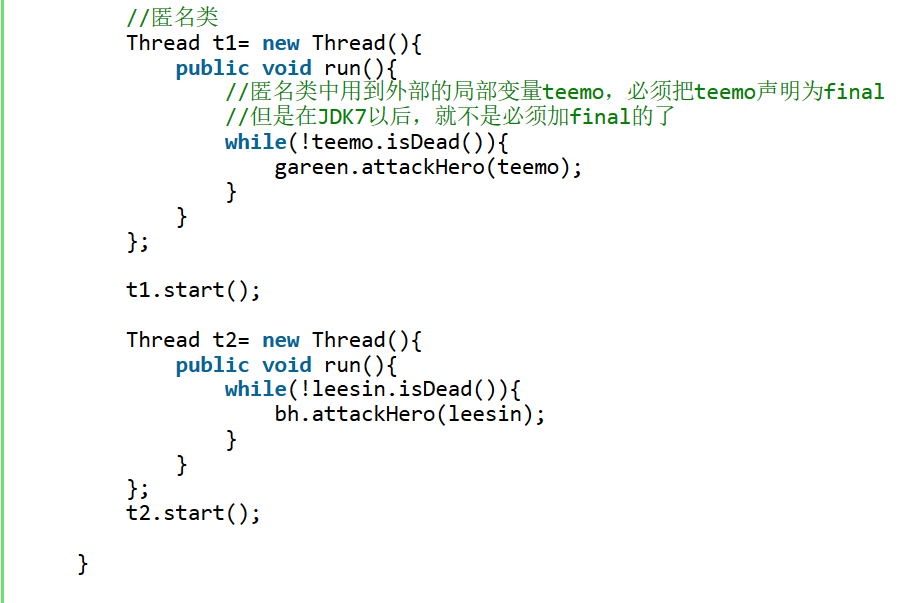
进程（Process）和线程（Thread）的区别：一个进程相当于是一个程序，而线程是在一个进程内部做的一个事情，多线程就是在内部做的多个事情。

使用三种方式创建线程：①通过继承Thread类；②通过实现Runnable接口；③通过匿名类方式

①public class TestThread extends Thread{......，里面重写run（）方法，}，外部实例化类后再通过start（）方法启动线程。

②public class implements Runnable{......，里面重写run（）方法}，外部实例化类后再通过start（）方法启动线程。

③通过匿名类方式，如下截图



总结三种创建线程的方式：启动线程都是start（）方法，而不是run（），而通过继承Thread类的方式有局限性，只能继承一个类，而通过实现Runnable接口的方式则可以实现多个接口，而匿名类的好处是可以很方便的访问外部的局部变量。

常用的线程方法：

①Thread.sleep(1000); //当前线程休眠1秒，其他线程不受影响

②t1.join() //线程t1加入到当前进程来，得等到线程t1执行结束才接下去执行其他操作。

谈谈主线程：所有进程中，都会有一个主线程，比如main（）方法开始执行，就有一个主线程存在。

③t1.setPriority(Thread.MAX\_PRIORITY); //当线程处于竞争关系时，有优先级高的线程有更大的几率可能获得CPU资源

④Thread.yield(); //当前线程暂停做出让步，使用其他进程更有机会占用CPU资源

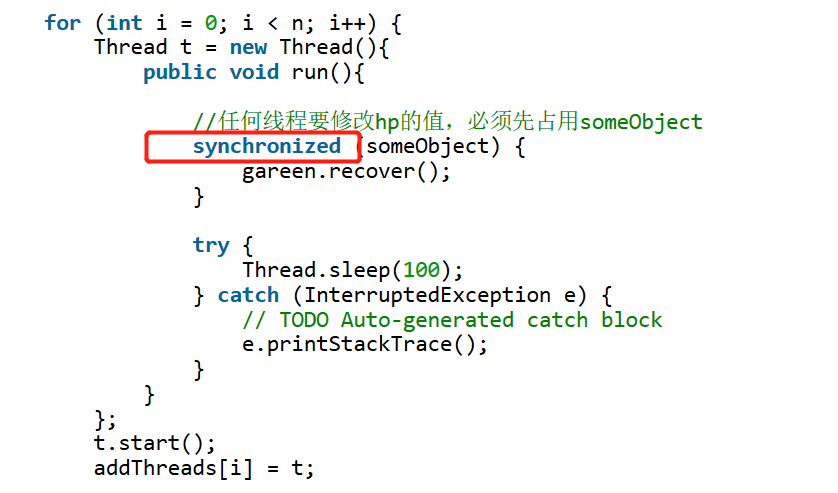
⑤t1.setDaemon(true); //设置为守护线程

守护线程就相当于公司中的后勤支持部门，当一个进程中，所有线程都只剩下守护线程的话就结束当前进程，一般守护线程用来做日志、性能统计工作。

同步

多线程中可能出现多个线程修改一个数据，从而导致脏数据的产生，于是便需要线程同步了，多线程编程中这种操作是很难被发现的，或许上百次操作才有一次程序发生错误，所以这也是最难考虑避免的。

Java解决：通过synchronized关键字来使当前线程独占对象，从而是其他线程无法同时来访问该对象，该对象称为同步对象，等到synchronized快自然结束或者异常抛出时才释放同步对象。可以如下操作：



而一个类若是其方法都是使用synchronized修饰的，这种类就叫做线程安全类，同一时间，只有一个线程能够进入该类的一个实例去修改数据，从而保证数据的安全避免了脏数据。

比如StringBuffer类和StringBuilder的区别：就是StringBuffer是线程安全的，而StringBuilder不是线程安全的类。

###### PS：重点知识、线程方面的基础面试题解

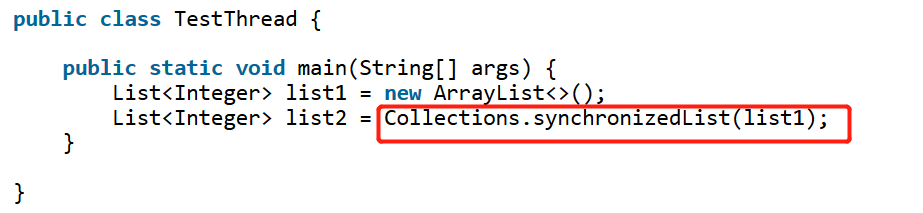
①HashMap和Hashtable的区别和联系：它们都实现了Map接口，都是以键值对的形式来保存数据，区别在于HashMap可以存放null值，而Hashtable不能存放null值，而且HashMap不是线程安全的类，Hashtable是线程安全的类。

②StringBuffer和StringBuilder的区别：当进行大量字符串拼接的操作时，如果是单线程的话就用StringBuilder会快一些，如果是多线程的话就用StringBuffer保证数据的安全性。

③ArrayList和Vector的区别和联系：通过查看源码可知两者是一模一样的，区别主要在于ArrayList不是线程安全类，而Vector是线程安全类。

④把非线程安全的集合转换为线程安全的

通过Collections.synchronizedList可以把例如ArrayList这种非线程安全类的集合转换为线程安全的。如下图：



死锁：多个线程各自想访问彼此的资源而互相等待对方释放掉资源，于是出现你等我，我等你的现象，故称为死锁。

交互：

this.wait(); //表示让占有this的线程等待，并临时释放占有。

this.notify(); //表示通知等待在this的线程可以苏醒过来了

这里注意一下：wait（）和notify（）方法，并不是Thread线程的方法，而是Object的方法。确切的说是同步对象的方法，而所有的Object都可以是同步对象，所以才说是Object的方法。wait（）是让占有着同步对象的线程临时释放当前的占用，并等待，前提是调用wait（）一定要放在synchronized块中，否则会出错。notify（）是通知一个等待在这个同步对象上的线程可以苏醒过来有机会占用对象了，而notifyAll（）则是通过多个等待在这个同步对象上的线程可以苏醒过来有机会占用对象了。



线程池：每一个线程的启动和结束都很消耗时间和占用资源，大量的启动和结束动作都会导致系统的性能变卡，响应变慢，故此引入线程池概念。

Java自带一个线程池ThreadPoolExecutor在包java.util.concurrent下

Lock对象

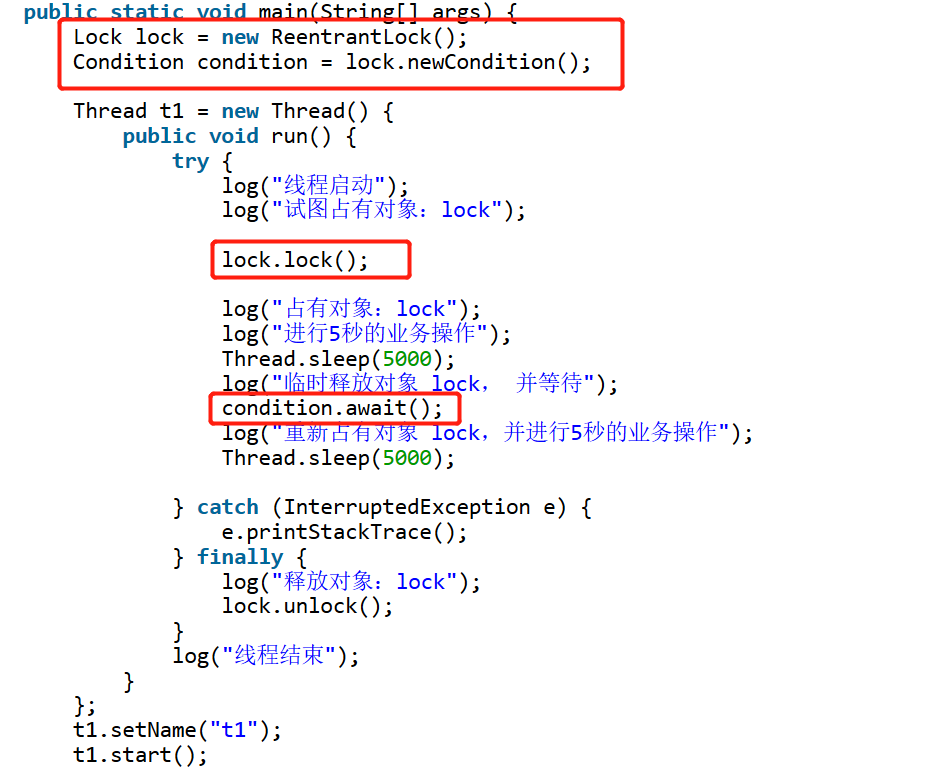
与synchronized一样，它也能达到同步的效果。Lock是一个接口，需要

Lock lock = new ReentrantLock();创建lock对象，Lock接口与synchronized关键字不同的是synchronized是占用结束后自动释放同步对象，而lock是得手动调用unlock()方法才能释放的，而且synchronized的占用是一根筋的，不占用到不罢休会一直试图占用下去，而lock是可以通过调用trylock()方法选择一个时间范围来占用对象的，如果时间到了没占用到就放弃了，此时在后面unlock释放锁的时候就需要判断是否lock占用到同步对象，否则若无占用到就释放锁是会抛出异常的。

使用lock对象实现线程交互

首先通过lock对象得到一个Condition对象，然后分别调用Condition对象的await()等待，singal()唤醒，signalAll()唤醒全部的线程方法。如下图：

总结：Lock是一个接口，而synchronized则是一个关键字，Lock可以选择性的获取锁，而synchronized则会一根筋的试图占用下去，借用lock可以避免死锁发生，而如果lock没有手动释放掉锁，同样也会造成死锁现象的。



#### 中级JavaEE