**《Java编程思想》笔记:**

一.单根继承结构优势:

1.在单根继承中所有对象共用一个接口,所以归根结底都是相同的基本类型.

2.单根继承结构保证了所有对象都具备某些功能.

3.单根继承结构使垃圾回收器的实现变的容易的多(因为能够明确对象类型).

二.终结清理和垃圾回收:

1.如果在类中定义了finalize()方法(要尽量避免使用该方法),当垃圾器准备释放对象所用内存时,会首先调用该方法,在下一次垃圾回收动作发生时真正的回收对象占用的内存.

2.对象可能不被垃圾回收; 垃圾回收并不等于”析构”.(“析构”: c++中的概念,用于销毁对象.); 垃圾回收只于内存有关.

3.无论是”垃圾回收”还是”终结”,都不保证一定发生.如果Java虚拟机(JVM)并未面临内存耗尽的的情形,它是不会浪费时间去执行垃圾回收以恢复内存的.

4.System.gc():用于强制执行终结动作,在类中定义finalize()方法验证终结条件.

三.初始化:

1.成员初始化:

对于局部变量如果不初始化,当使用时会报错,编译不通过.但是对于类的成员(字段),基本类型都有默认值0,对象引用则为null.

2.构造器初始化:

及时变量定义散布于方法定义之间,他们仍然会在任何方法(包括构造器)之前得到初始化.

3.静态初始化:

静态初始化只有在必要时刻才会进行,也就是创建对象或者第一次访问数据时,他们才会被初始化,此后不会再次初始化.

初始化的顺序是先静态对象,然后非静态对象,先父类,后子类.

**父类静态字段初始化->父类静态块->子类静态字段初始化->子类静态块->父类非静态字段初始化->父类构造器->子类非静态字段初始化->子类构造器**

**四.对象的创建过程总结(以Dog类为例):**

**1.即使没有显示的使用static关键字,构造器也是静态方法.当首次创建Dog对象时,获取Dog类的静态方法或静态域被访问时,Java解释器必须查找类路径,以定位Dog.class文件.**

**2.然后载入Dog.class文件,有关静态初始化的所有动作都会执行.因此,静态初始化只在Class对象首次加载的时候进行一次.**

**3.当用new Dog()创建对象的时候,首次将在堆上为Dog对象分配足够的存储空间.**

**4.这块存储空间会被清零,这就自动的将Dog对象中的所有基本类型数据都设置成了默认值(对数字来说是0,对布尔型和字符型也相同),而引用则被设置成null.**

**5.执行所有出现于字段定义处的初始化动作.**

**6.执行构造器.**

五.数组初始化:

数组是相同数据类型的、用一个标识符名称封装到一起的一个对象序列活基本类型数据序列。数组是通过方括号下标操作符[]来定义和使用的.

自动的编译器错误和运行时优化都可以提高数组访问的速度.

六.可变参数列表:

要放在参数的最后;一个方法只能有一个;可以传入一个值或多个值,会自动合并成数组;可变参数列表不依赖于自动包装机制,实际上使用的是基本类型,当然可变参数列表可以与自动包装机制和谐相处.

七.final关键字:

使用final的三种情况：数据、方法、类。

1.final数据:

永不改变的编译期常量;在运行时被初始化的值;可以是基本类型和String类型;既是static又是final的域只占据一段不能改变的存储空间.

空白final:

Java允许生成"空白final",所谓空白final是指被声明为final但又未给定初始值的域.无论什么情况,编译器都确保空白final在使用前必须被初始化.必须在域的定义处或者每个构造器中用表达式对final进行赋值,这正是final域在使用前总是被初始化的原因所在.



final参数:

Java允许在参数列表中以声明的方式将参数指明为final,这意味着无法在方法中更改参数引用所指向的对象.

2.final方法:

被final修饰的方法不可以被覆盖.这样做还可以有效的"关闭"动态绑定.

3.final类:

被final修饰的类不可以被继承.

八.内部类:

1.如果想从外部类的非静态方法之外的任意位置创建某个内部类的对象,那么必须像在main()方法中那样,具体的指明这个对象的类型:OutClassName.InnerClassName.



1. 当生成一个内部类的对象时,此对象与制造它的外围对象之间就有了一种联系,所以它能访问其外围对象的所有成员,而不需要任何特殊条件,此外,内部类还拥有其外部类的所有元素的访问权。这是因为内部类对象获取到了一个指向外部类对象的引用,当访问外部类的成员时，就是用这个引用来选择外部类的成员。外部类不可以直接获取内部类的成员。





3.如果需要生成对外部类对象的引用，可以使用外部类的名字后面紧跟远点和this。这样产生的引用自动的具有正确的类型，这点在编译期就被知晓并受到检查，没有任何运行时开销。



嵌套类（静态内部类）：



4.在方法和作用域内的内部类:

4.1.一个定义在方法中的类

4.2.一个定义在作用域内的类,此作用域在方法的内部

4.3.一个实现了接口的匿名类

4.4.一个匿名类,它扩展了有非默认构造器的类

4.5.一个匿名类,它执行字段初始化

4.6.一个匿名类,它通过实例初始化实现构造(匿名类不可能有构造器)

匿名内部类与正规的继承相比有些受限，因为匿名内部类既可以扩展；类，也可以实现接口，但是不能两者兼备。而且如果是实现接口，只能实现一个接口。匿名内部类不能是抽象的，它必须要实现继承的类或者实现的接口的所有抽象方法。匿名内部类中不能存在任何的静态成员变量和静态方法。





匿名内部类和方法和作用域内的内部类实现字段初始化得到的值必须是final的.内部类并不是直接调用方法传递的参数，而是利用自身的构造器对传入的参数进行备份(还有外部类对象引用)，自己内部方法调用的实际上是自己的属性而不是外部方法传递进来的参数。简单理解就是，拷贝引用，为了避免引用值发生改变，例如被外部类的方法修改等，而导致内部类得到的值不一致，于是用final来让该引用不可改变。故如果定义了一个匿名内部类，并且希望它使用一个其外部定义的参数，那么编译器会要求该参数引用是final的。主要就是为了保持该值在内外类中的一致性。(但是如果是全局变量的话可以不用是final,该变量是通过外部类对象引用(this)得到的,非静态内部类的对象都保存了外部类的对象的引用，因而内部类中对任何外部类成员变量的修改都会真实地反应到外部类实例本身，所以不需要用final来修饰它。)





接口内部的类：



多层嵌套类：



5.闭包与回调:

闭包：是一个可调用的对象,它记录了一些信息,这些信息来自于创建它的作用域。通过定义可以看出，内部类是面向对象的闭包，因为它不仅仅包含外部类对象的信息，还自动拥有一个指向外部类对象的引用，在此作用域内，内部类有权操作所有的成员，包括private成员。

回调：通过回调，对象能够携带一些信息，这些信息允许它稍后的某个时刻调用初始对象。

**九.Collection与Map（基础）:**

1、List:

1.1.ArrayList长于随机访问元素,但是在List的中间插入和移除元素时较慢。

1.2.LinkedList（双向链表），它通过代价较低的在List中间进行插入和删除操作，提供了优化的顺序访问。LinkedList在随机访问方面相对较慢，但它的特性集较ArrayList更大。

2、迭代器：

Java的Iterator只能单向移动，这个Iterator能用来：

1）、使用方法iterator（）方法要求容器返回一个Iterator；

2）、使用next（）获得序列中的下一个元素；

3）、使用hasNext（）检查序列中是否还有元素；

4）、使用remove（）将迭代器新近返回的元素删除。

ListIterator是一个更加强大的Iterator的子类型，他能用于各种List类的访问，可以双向移动。

3、Stack（栈）：

先进后出的容器，最后压入的元素第一个弹出栈。

4、Set：

元素不允许重复；无序。

HashSet提供最快的查询速度;TreeSet保持元素处于排序状态;LinkedHashSet以插入顺序保持元素。

5、Map：

键值对Entry数组。

6、Queue：

队列是先进先出容器。

7、foreach与迭代器：

foreach可以用于数组、任何Collection对象、任何实现Iterable的类、可以产生Iterable对象的方法。



**十.Java异常机制:**

**Throwable**这个Java类被用来表示任何可以作为异常被抛出的类。分为两种类型：Error用来表示编译时和系统错误；Exception是可以被抛出的基本类型。

异常还可分为检查异常和未检查异常。

检查异常：在编译时被强制检查的异常。



未检出异常：在运行时抛出的异常。都继承自RuntimeException类。



如果有finally子句，无论是否抛出异常，finally子句中的代码都会被执行。当要把除内存之外的资源恢复到它们的初始状态时，就要用到finally子句。这种需要清理的资源包括：已经打开的文件或网络连接，在屏幕上画的图形，甚至可以是外部世界的某个开关。

**十一.String:**

1、String对象不可变。String类中的每一个看起来会修改String值的方法，实际上都是创建了一个全新的String对象以及包含修改后的字符串内容，最初的String对象丝毫未动。如果原先的内容没有发生改变，则会返回指向原对象的引用。

String对象具有只读特性，指向它的任何引用都不可能改变它的值。

String对象使用"+"操作符时,编译器会自动进行优化，创建StringBuilder对象，然后调用append（）方法，最后调用toString（）方法返回结果。

2、格式化：





**十二.类型信息:**

1、运行时类型信息使得可以在程序运行时发现和使用类型信息。

如果某个对象出现在字符串表达式中（涉及“+”和字符串对象的表达式），toString（）方法就会被自动调用，以生成表示该对象的String。

在Java中，所有的类型转换都是在运行时进行正确性检查的。

2、Class对象：

2.1、类是程序的一部分，每个类都有一个Class对象，每当编写或者编译了一个新类，就会产出一个Class对象，为了产生这个类的对象，运行这个程序的Java虚拟机将使用被称为“类加载器”的子系统。类加载器子系统实际上可以包含一条类加载器链，但是只有一个原生类加载器，它是JVM实现的一部分。

所有的类都是在对其第一次使用时动态加载到JVM中的。当程序创建第一个对类的静态成员的引用时，就会加载这个类。这个证明构造器也是累的静态方法，即使在构造器之前并没有使用static关键字，因此，使用new操作符创建类的新对象也会被当做对类的静态成员的引用。

Java程序在它开始运行之前并非被完全加载，其各个部分是在必需时才加载的（static初始化伴随着类加载进行）。类加载器首先检查这个类的Class对象是否已经加载，如果尚未加载，默认的类加载器就会根据类名查找.class文件，在这个类的字节码被加载时，他们会接受验证，以确保没有被破坏，并且不包含不良Java代码。一旦某个类的Class对象被载入内存，它就被用来创建这个类的所有对象。

2.2、Class.forName（）方法用于获取Class对象的引用；无论何时，如果想在运行时使用类型信息，就必须首先获得对恰当的Class对象的引用。

2.3、使用“.class”来创建对Class对象的引用时，不会自动的初始化该Class对象。也就说只有等到使用该引用时才会进行初始化动作。

2.4、Class引用总是指向某个Class对象，他可以制造类的实例，并包含可作用于这些实例的所有方法代码。它还包含该类的静态成员，因此，Class引用表示的就是它所指向的对象的确切类型，而该对象便是Class类的一个对象。



2.5、instanceof:

返回一个布尔值,用于判断对象是不是某个特定类型的实例。

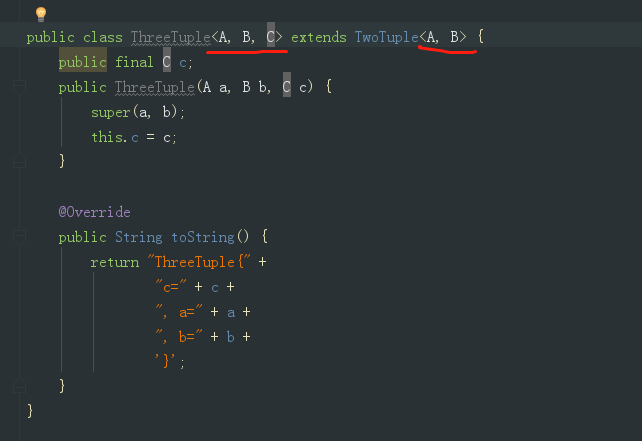
instanceof与==的区别:

instanceof保持了类型的概念,它指的是"你是这个类吗,或者你是这个类的派生类吗",而如果用==则比较的是实际的Class对象,就没有考虑继承。

**十三.泛型:**

1、泛型：参数化类型。

2、元组：将一组对象直接打包存储于其中的一个单一对象。这个容器对象允许读取其中对象，但是不允许向其中存放新的对象。元组可以具有任意长度；元组中的对象可以是任意不同的类型。



3、泛型方法：

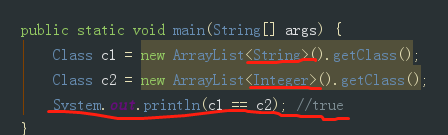
泛型方法使得方法能够独立于类而产生变化，对于一个static方法而言，无法访问泛型类的类型参数，如果一个static方法需要使用泛型能力，就必须使其成为泛型方法。

当使用泛型类时，必须在创建对象的时候指定类型参数的值，而使用泛型方法的时候，通常不必指明参数类型，因为编译器会为我们找出具体的类型，这称为类型参数推断。

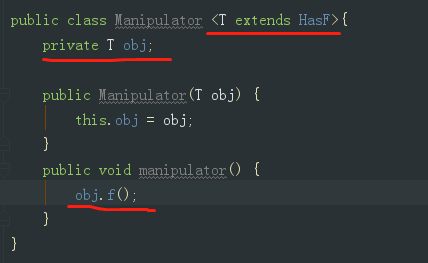
4、泛型擦除：

在泛型代码内部，无法获得任何有关泛型参数类型的信息。

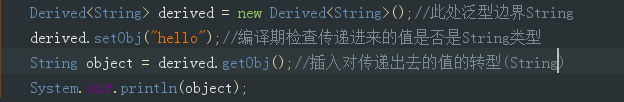
在使用泛型时，任何具体的类型信息都在编译期被擦除了，因此List<String>和List<Integer>在运行时事实上是相同的类型,这两种形式被擦除成它们的原生类型,即List。

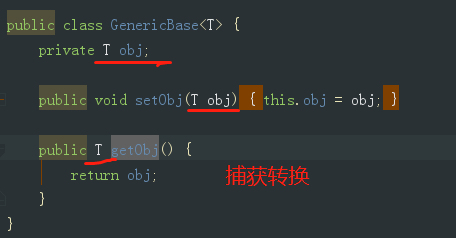


泛型类型参数将擦除到它的第一个边界，例如<T extends HasF>，编译器会把类型参数替换为它的擦除，T擦除到了HasF，就好像在类的声明中用HasF替换了T一样。



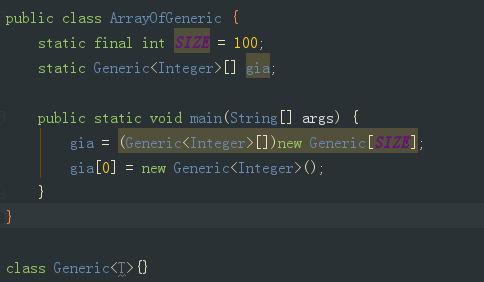
在泛型中的所有动作都发生在边界处--对传递进来的值进行额外的编译器检查,并插入对传递出去的值的转型。“边界就是发生动作的地方”。



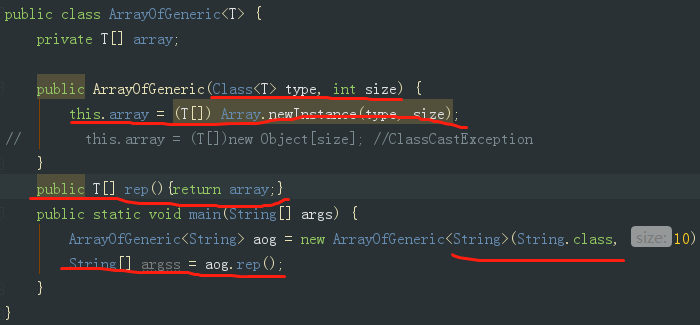


5、泛型数组：

数组将跟踪它们的实际类型，而这个类型是在数组被创建时确定的，即使gia已经被转型为Generic<Integer>[]，但是这个信息只存在于编译期，用于类型检查。在运行时，它仍旧是Object数组。成功创建泛型数组的唯一方式就是创建一个被擦除类型的新数组，然后对其转型。

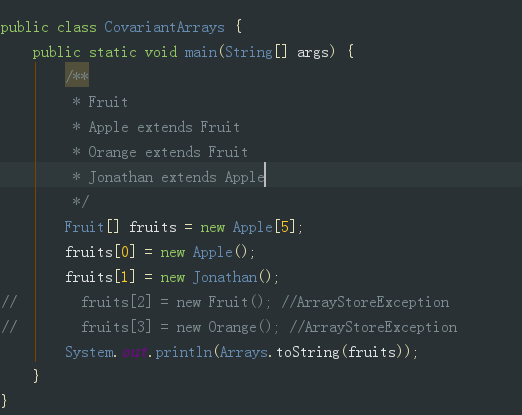


类型标记Class[T]被传递到构造器中,以便从擦除中恢复,使得我们可以创建需要的实际类型的数组，一旦我们获得了实际类型，就可以返回它，并获得想要的结果，该数组的运行时类型是确切类型T[]。



6、通配符：

6.1、数组是协变的，泛型以及泛型集合不是协变的。



6.2、? Super T和? Extend T的区别:

**PECS**

请记住PECS原则：生产者（Producer）使用extends，消费者（Consumer）使用super。

· 生产者使用extends

如果你需要一个列表提供T类型的元素（即你想从列表中读取T类型的元素），你需要把这个列表声明成<? extends T>，比如List<? extends Integer>，因此你不能往该列表中添加任何元素。

· 消费者使用super

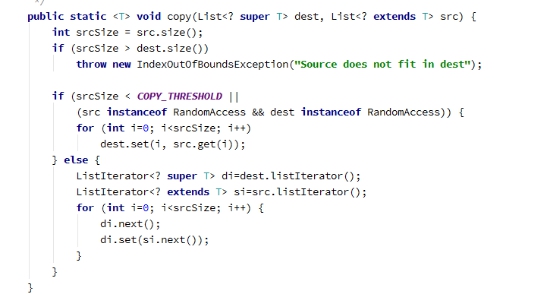
如果需要一个列表使用T类型的元素（即你想把T类型的元素加入到列表中），你需要把这个列表声明成<? super T>，比如List<? super Integer>，因此你不能保证从中读取到的元素的类型。

· 即是生产者，也是消费者

如果一个列表即要生产，又要消费，你不能使用泛型通配符声明列表，比如List<Integer>。

? Extend T描述了通配符的上界,即?为T本身或子类

? Super T描述了通配符的下界,即?为T本身或父类



7、问题：

7.1、任何基本类型都不能作为类型参数。

7.2、一个类不能实现同一个泛型接口的两种变体，由于擦除的原因，这两个变体会成为相同的接口。

7.3、使用带有泛型类型参数的转型或instanceof不会有任何效果。

7.4、由于擦除的原因，带有泛型的方法参数如果产生唯一的参数列表时，重载方法将产生相同的类型签名，必须提供有明显区别的方法名。

