**《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.

**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.(JVM第一阶段加载)**

**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.(JVM第二阶段验证、解析、准备)**

**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.(JVM第三阶段初始化)**

**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。