JAVA基础知识总结

1. 面向对象的三大特征：封装、继承、多态

封装：将类的某些信息隐藏在类内部，不允许外部程序直接访问，而是通过该类提供的方法来实现对隐藏信息的操作和访问。

优点：

只能通过规定的方法访问数据。

　隐藏类的实例细节，方便修改和实现

This关键字：

1、this关键字代表当前对象

　　this.属性 操作当前对象的属性

　　this.方法 调用当前对象的方法。

　2.封装对象的属性的时候，经常会使用this关键字。

　3.当getter和setter函数参数名和成员函数名重合的时候，可以使用this区别。如：

Public void setScreen(float screen){

this.screen = screen;

}

Java 中的内部类

　内部类（ Inner Class ）就是定义在另外一个类里面的类。与之对应，包含内部类的类被称为外部类。

　那么问题来了：那为什么要将一个类定义在另一个类里面呢？清清爽爽的独立的一个类多好啊！！

　答：内部类的主要作用如下：

　　1. 内部类提供了更好的封装，可以把内部类隐藏在外部类之内，不允许同一个包中的其他类访问该类。

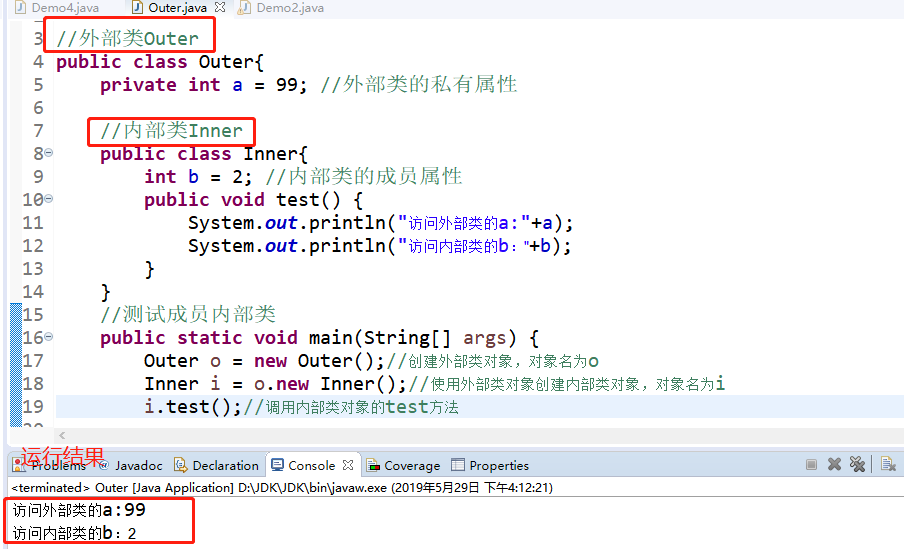
　　2. 内部类的方法可以直接访问外部类的所有数据，包括私有的数据。

　　3. 内部类所实现的功能使用外部类同样可以实现，只是有时使用内部类更方便。

　　内部类可分为以下几种：

成员内部类

内部类中最常见的就是成员内部类，也称为普通内部类。我们来看如下代码：



　从上面的代码中我们可以看到，成员内部类的使用方法：

　　1、 Inner 类定义在 Outer 类的内部，相当于 Outer 类的一个成员变量的位置，Inner 类可以使用任意访问控制符，如 public 、 protected 、 private 等

　　2、 Inner 类中定义的 test() 方法可以直接访问 Outer 类中的数据，而不受访问控制符的影响，如直接访问 Outer 类中的私有属性a

　　3、 定义了成员内部类后，必须使用外部类对象来创建内部类对象，而不能直接去 new 一个内部类对象，即：内部类 对象名 = 外部类对象.new 内部类( );

　　4、 编译上面的程序后，会发现产生了两个 .class 文件

IMG_256

　　其中，第二个是外部类的 .class 文件，第一个是内部类的 .class 文件，即成员内部类的 .class 文件总是这样：外部类名$内部类名.class

　　另外，友情提示哦：

　　1、 外部类是不能直接使用内部类的成员和方法滴。如：

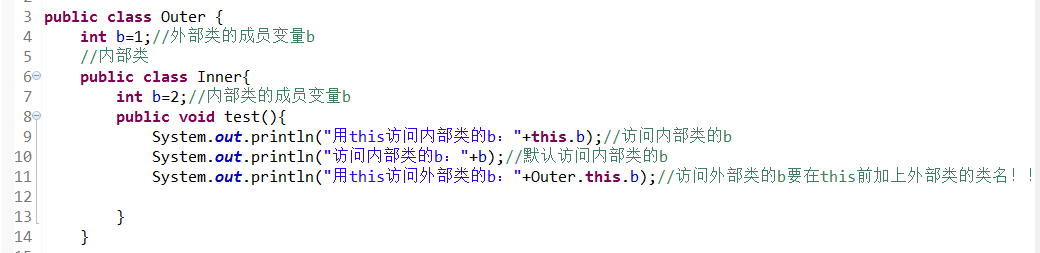


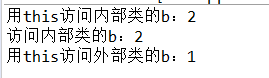
　　那么外部类如何使用内部类的成员和方法呢？？

　　答：可先创建内部类的对象，然后通过内部类的对象来访问其成员变量和方法。

　　Java 编译器在创建内部类对象时，隐式的把其外部类对象的引用也传了进去并一直保存着。这样就使得内部类对象始终可以访问其外部类对象，同时这也是为什么在外部 类作用范围之外向要创建内部类对象必须先创建其外部类对象的原因。

　　2、 如果外部类和内部类具有相同的成员变量或方法，内部类默认访问自己的成员变量或方法，如果要访问外部类的成员变量，可以使用 this 关键字。如：



　　运行结果：

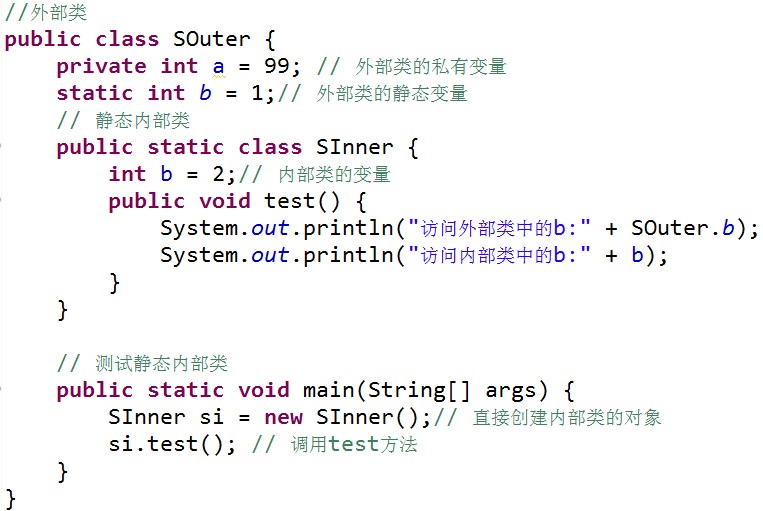
静态内部类

静态内部类是 static 修饰的内部类，这种内部类的特点是：

　　1、 静态内部类不能直接访问外部类的非静态成员，但可以通过 new 外部类().成员 的方式访问。

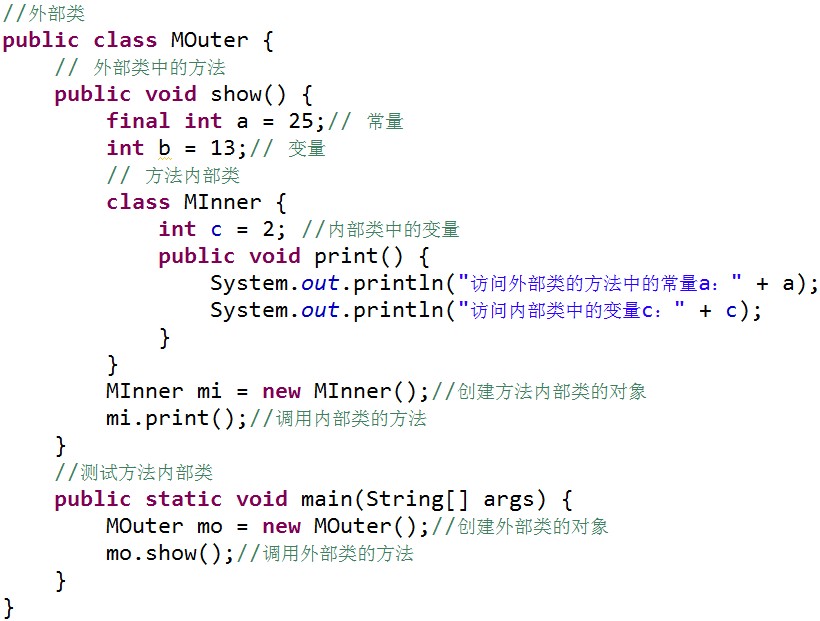
　　2、 如果外部类的静态成员与内部类的成员名称相同，可通过“类名.静态成员”访问外部类的静态成员；如果外部类的静态成员与内部类的成员名称不相同，则可通过“成员名”直接调用外部类的静态成员。

　　3、 创建静态内部类的对象时，不需要外部类的对象，可以直接创建 内部类 对象名= new 内部类();



　　运行结果 ： [IMG_257](http://img.mukewang.com/539e94a500014f0101930052.jpg)

方法内部类

方法内部类就是内部类定义在外部类的方法中，方法内部类只在该方法的内部可见，即只在该方法内可以使用。　

　　一定要注意哦：由于方法内部类不能在外部类的方法以外的地方使用，因此方法内部类不能使用访问控制符和 static 修饰符。

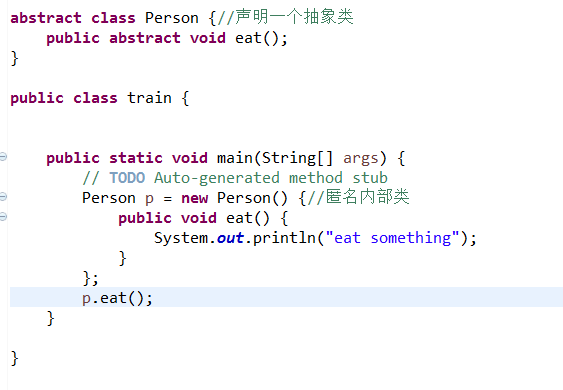
匿名内部类

匿名类是不能有名称的类，所以没办法引用他们。必须在创建时，作为new语句的一部分来声明他们。

　　但使用匿名内部类还有个前提条件：必须继承一个父类或实现一个接口。  
　　这就要采用另一种形式 的new语句，如下所示：

　　 new <类或接口> <类的主体>  
        这种形式的new语句声明一个 新的匿名类，他对一个给定的类进行扩展，或实现一个给定的接口。他还创建那个类的一个新实例，并把他作为语句的结果而返回。要扩展的类和要实现的接口是 new语句的操作数，后跟匿名类的主体。   
        注意匿名类的声明是在编译时进行的，实例化在运行时进行。这意味着 for循环中的一个new语句会创建相同匿名类的几个实例，而不是创建几个不同匿名类的一个实例。  
        从技术上说，匿名类可被视为非静态的内 部类，所以他们具备和方法内部声明的非静态内部类相同的权限和限制。  
        假如要执行的任务需要一个对象，但却不值得创建全新的对象（原因可能 是所需的类过于简单，或是由于他只在一个方法内部使用），匿名类就显得很有用。匿名类尤其适合在Swing应用程式中快速创建事件处理程式。以下是一个匿名内部类的实例：

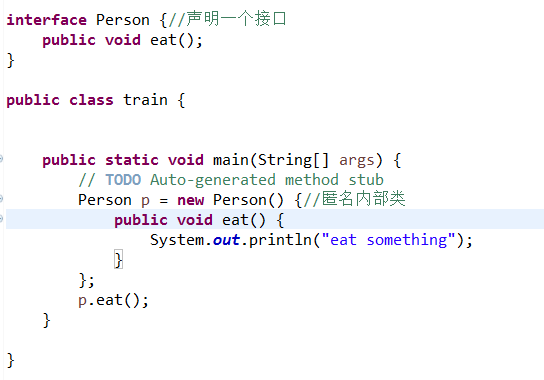
　　1、匿名内部类的基本实现：



　　运行结果：IMG_257

　　可以看到，我们直接将抽象类Person中的方法在大括号中实现了，这样便可以省略一个类的书写，并且，匿名内部类还能用于接口上。

 　2、在接口上使用匿名内部类：



　　运行结果：IMG_259

　　由上面的例子可以看出，只要一个类是抽象的或是一个接口，那么其子类中的方法都可以使用匿名内部类来实现。

　　在使用匿名内部类的过程中，我们需要注意如下几点：

      1、使用匿名内部类时，我们必须是继承一个类或者实现一个接口，但是两者不可兼得，同时也只能继承一个类或者实现一个接口。

      2、匿名内部类中是不能定义构造函数的。

      3、匿名内部类中不能存在任何的静态成员变量和静态方法。

      4、匿名内部类为局部内部类（即方法内部类），所以局部内部类的所有限制同样对匿名内部类生效。

      5、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。

　　匿名内部类重点难点：

1. 如果是在一个方法的匿名内部类，可以利用这个方法传进你想要的参数，不过记住，这些参数必须被声明为 final 。

　使用的形参为何要为final？？

　我们给匿名内部类传递参数的时候，若该形参在内部类中需要被使用，那么该形参必须要为final。也就是说：当所在的方法的形参需要被内部类里面使用时，该形参必须为final。

     首先我们知道在内部类编译成功后，它会产生一个class文件，该class文件与外部类并不是同一class文件，仅仅只保留对外部类的引用。当外部类传入的参数需要被内部类调用时，从java程序的角度来看是直接被调用：

[IMG_260](https://www.cnblogs.com/hysum/p/javascript:void(0);)

public class OuterClass {

public void display(final String name,String age){

class InnerClass{

void display(){

System.out.println(name);

}

}

}

}

[IMG_261](https://www.cnblogs.com/hysum/p/javascript:void(0);)

　　从上面代码中看好像name参数应该是被内部类直接调用？其实不然，在java编译之后实际的操作如下：

[IMG_262](https://www.cnblogs.com/hysum/p/javascript:void(0);)

public class OuterClass$InnerClass {

public InnerClass(String name,String age){

this.InnerClass$name = name;

this.InnerClass$age = age;

}

public void display(){

System.out.println(this.InnerClass$name + "----" + this.InnerClass$age );

}

}

[IMG_263](https://www.cnblogs.com/hysum/p/javascript:void(0);)

　 所以从上面代码来看，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，自己内部方法调用的实际上时自己的属性而不是外部方法传递进来的参数。

　 直到这里还没有解释为什么是final。在内部类中的属性和外部方法的参数两者从外表上看是同一个东西，但实际上却不是，所以他们两者是可以任意变化的，也就是说在内部类中我对属性的改变并不会影响到外部的形参，而然这从程序员的角度来看这是不可行的，毕竟站在程序的角度来看这两个根本就是同一个，如果内部类该变了，而外部方法的形参却没有改变这是难以理解和不可接受的，所以为了保持参数的一致性，就规定使用final来避免形参的不改变。

    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。

2. 匿名内部类中使用初始化代码块

　　 我们一般都是利用构造器来完成某个实例的初始化工作的，但是匿名内部类是没有构造器的！那怎么来初始化匿名内部类呢？使用构造代码块！利用构造代码块能够达到为匿名内部类创建一个构造器的效果。

[IMG_264](https://www.cnblogs.com/hysum/p/javascript:void(0);)

public class OutClass {

public InnerClass getInnerClass(final int age,final String name){

return new InnerClass() {

int age\_ ;

String name\_;

//构造代码块完成初始化工作 {

if(0 < age && age < 200){

age\_ = age;

name\_ = name;

}

}

public String getName() {

return name\_;

}

public int getAge() {

return age\_;

}

};

}

public static void main(String[] args) {

OutClass out = new OutClass();

InnerClass inner\_1 = out.getInnerClass(201, "chenssy");

System.out.println(inner\_1.getName());

InnerClass inner\_2 = out.getInnerClass(23, "chenssy");

System.out.println(inner\_2.getName());

}

}

1. 五大设计原则：单一职责原则、开放封闭原则、里氏替换原则、依赖倒置原则、接口分离原则