## 各种内部类和枚举类的使用 + 面试题

内部类不仅经常出现在各种面试题中，还会在 Java 源码中频频出现，因此只有搞明白了 Java 内部类，才能搞定面试和看懂各种 Java 源码。

### 内部类

Java 内部类，分为以下四种：

* 成员内部类
* 静态成员内部类
* 局部内部类
* 匿名内部类

下面分别来看这些内部类的使用。

#### 成员内部类

##### **定义**

在一个类中定义了另一个类，则将定义在类中的那个类称之为成员内部类。成员内部类也是最普通的内部类。

##### **使用**

成员内部类的使用示例如下：

class Outer {  
 public Outer() {  
 System.out.println("Outer Class.");  
 }  
 class Inner {  
 public void sayHi() {  
 System.out.println("Hi, Inner.");  
 }  
 }  
}

其中 Inner 类则为成员内部类。  
而成员内部类的创建和使用，请参考以下完整的示例代码：

class InnerTest {  
 public static void main(String[] args) {  
 Outer out = new Outer();  
 // 创建成员内部类  
 Outer.Inner inner = out.new Inner();  
 inner.sayHi();  
 }  
}  
class Outer {  
 public Outer() {  
 System.out.println("Outer Class.");  
 }  
 class Inner {  
 public void sayHi() {  
 System.out.println("Hi, Inner.");  
 }  
 }  
}

##### **成员内部类的创建**

语法：

Outer.Inner inner = new Outer().new Inner();

##### **内部类访问外部类**

语法：

Outer.this.xxx

代码示例：

class Outer {  
 private String name = "OuterClass";  
 public void sayHi() {  
 System.out.println("Hi, Outer.");  
 }  
 class Inner {  
 public void sayHi() {  
 // 内部类访问外部类  
 Outer.this.sayHi();  
 System.out.println(Outer.this.name);  
 System.out.println("Hi, Inner.");  
 }  
 }  
}  
class InnerTest {  
 public static void main(String[] args) {  
 Outer.Inner inner = new Outer().new Inner();  
 inner.sayHi();  
 }  
}

##### **外部类访问内部类**

语法：

new Inner().xxx

代码示例：

class Outer {  
 public void sayHi() {  
 System.out.println(new Inner().name);  
 System.out.println("Hi, Outer.");  
 }  
 private class Inner {  
 String name = "InnerClass";  
 public void sayHi() {  
 System.out.println("Hi, Inner.");  
 }  
 }  
}  
class InnerTest {  
 public static void main(String[] args) {  
 new Outer().sayHi();  
 }  
}

##### **小结**

* 成员内部类可直接访问外部类（使用：外部类.this.xxx）；
* 外部成员类要访问内部类，必须先建立成员内部类对象；
* 成员内部类可使用任意作用域修饰（public、protected、默认、private）；
* 成员内部类可访问外部类任何作用域修饰的属性和方法；
* 外部类建立成员内部类对象之后，可以访问任何作用域修饰的内部类属性和方法。

#### 静态成员内部类

##### **定义**

在一个类中定义了另一个 static 类，则将定义在类中的那个 static 类称之为静态成员内部类。

静态成员内部类也就是给内部成员类加上 static 修饰符。

##### **使用**

静态成员内部类的使用示例如下：

class OuterClass {  
 public OuterClass() {  
 System.out.println("OuterClass Init.");  
 }  
 protected static class InnerClass {  
 public void sayHi() {  
 System.out.println("Hi, InnerClass.");  
 }  
 }  
}  
class InnerClassTest {  
 public static void main(String[] args) {  
 OuterClass.InnerClass innerClass = new OuterClass.InnerClass();  
 innerClass.sayHi();  
 }  
}

与内部成员类的创建方式 new Outer().new Inner() 不同，静态成员内部类可使用 new OuterClass.InnerClass() 的方式进行创建。

**注意** ：不能从静态成员内部类中访问非静态外部类对象。

#### 局部内部类

##### **定义**

一个类定义在另一个类的局部（方法或者任意作用域），这个类就称之为局部内部类。

##### **使用**

局部内部类的使用示例如下：

class OutClass {  
 public void sayHi() {  
 class InnerClass {  
 InnerClass(String name) {  
 System.out.println("InnerClass:" + name);  
 }  
 }  
 System.out.println(new InnerClass("Three"));  
 System.out.println("Hi, OutClass");  
 }  
}  
class OutTest {  
 public static void main(String[] args) {  
 new OutClass().sayHi();  
 }  
}

##### **局部内部类特点**

* 局部内部类不能使用任何访问修饰符；
* 局部类如果在方法中，可以直接使用方法中的变量，不需要通过 OutClass.this.xxx 的方式获得。

#### 匿名内部类

##### **定义**

没有名字的内部类就叫做匿名内部类。

##### **使用**

匿名内部类的使用示例如下：

interface AnonymityOuter {  
 void hi();  
}  
class AnonymityTest {  
 public static void main(String[] args) {  
 AnonymityOuter anonymityOuter = new AnonymityOuter() {  
 @Override  
 public void hi() {  
 System.out.println("Hi, AnonymityOuter.");  
 }  
 };  
 anonymityOuter.hi();  
 }  
}

其中，new AnonymityOuter() 之后的 {…} 大括号包含的部分就为匿名内部类。

##### **匿名内部类特点**

* 匿名内部类必须继承一个父类或者实现一个接口
* 匿名内部类不能定义任何静态成员和方法
* 匿名内部类中的方法不能是抽象的

### 枚举类

枚举类是 JDK 1.5 引入的新特性，使用关键字“enum”声明。枚举功能虽小，却非常实用，大大方便了程序的开发者。

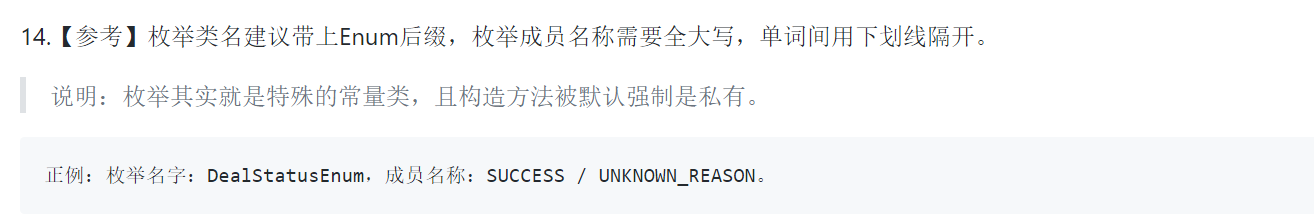
##### **枚举类的使用**

请参考以下代码：

enum ColorEnum {  
 RED,  
 BLUE,  
 YELLOW,  
 GREEN  
}  
class EnumTest {  
 public static void main(String[] args) {  
 ColorEnum color = ColorEnum.GREEN;  
 switch (color) {  
 case RED:  
 System.out.println("Red");  
 break;  
 case BLUE:  
 System.out.println("Blue");  
 break;  
 case YELLOW:  
 System.out.println("Yellow");  
 break;  
 case GREEN:  
 System.out.println("Green");  
 break;  
 default:  
 break;  
 }  
 }  
}

##### **枚举类命名规范**

《阿里巴巴 Java 开发手册》对枚举类的命名规范建议，如下图：



enter image description here

##### **扩展枚举类**

我们可以自定义一些枚举类方法，扩展枚举类的使用，请参考以下代码：

enum ColorsEnum {  
 RED("红色", 1),  
 BLUE("蓝色", 2),  
 YELLOW("黄色", 3),  
 GREEN("绿色", 4);  
 ColorsEnum(String name, int index) {  
 this.name = name;  
 this.index = index;  
 }  
 private String name;  
 private int index;  
 public String getName() {  
 return name;  
 }  
 public void setName(String name) {  
 this.name = name;  
 }  
 public int getIndex() {  
 return index;  
 }  
 public void setIndex(int index) {  
 this.index = index;  
 }  
}  
class EnumTest {  
 public static void main(String[] args) {  
 System.out.println(ColorsEnum.RED.getName());  
 System.out.println(ColorsEnum.RED.getIndex());  
 }  
}

执行以上代码返回的结果：

红色  
1

### 相关面试题

#### 1.Java 中的内部类有哪些？

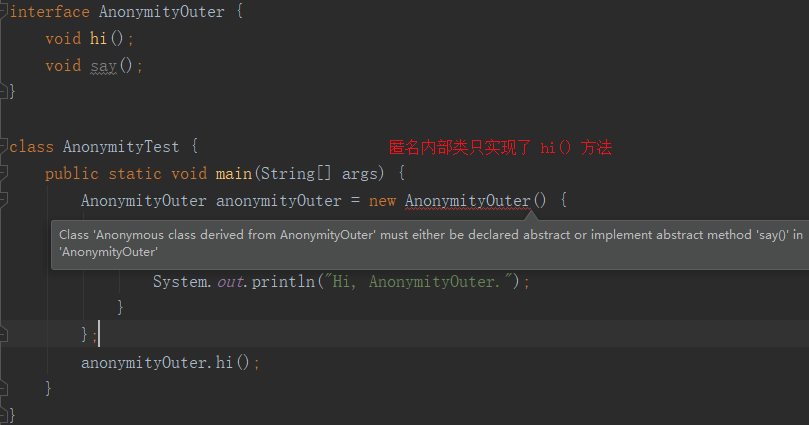
答：内部类包含以下 4 种：

* 静态内部类：static class StaticInnerClass{}；
* 成员内部类：private class InstanceInnerClass{}；
* 局部内部类：定义在方法或者表达式内部；
* 匿名内部类：(new Thread(){}).start()。

#### 2.以下关于匿名内部类说法错误的是？

A：匿名内部类必须继承一个父类或者实现一个接口  
B：匿名内部类中的方法不能是抽象的  
C：匿名内部类可以实现接口的部分抽象方法  
D：匿名内部类不能定义任何静态成员和方法

答：C  
题目解析：匿名内部类规定必须实现接口的所有抽象方法，否则程序会报错，如下图所示。

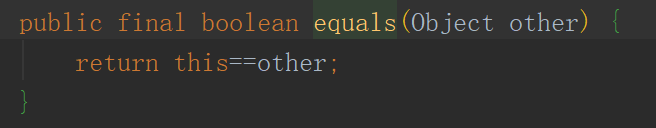


enter image description here

#### 3.以下枚举类比较“==”和“equals”结果一致吗？为什么？

class EnumTest {  
 public static void main(String[] args) {  
 ColorEnum redColor = ColorEnum.RED;  
 ColorEnum redColor2 = ColorEnum.RED;  
 System.out.println(redColor == redColor2);  
 System.out.println(redColor.equals(redColor2));  
 }  
}  
enum ColorEnum {  
 RED,  
 BLUE  
}

答：结果一致，都是 true。  
题目分析：因为枚举类重写了 equals 方法，equals 方法里直接使用的 == 比较的，而枚举类不能通过 new 进行创建，使用 ColorEnum.RED 得到的对象，其实使用的是对象的引用地址，所以 == 比较的结果一定是 true。equals 被重写的源码如下图：



enter image description here

#### 4.使用静态内部类的好处有哪些？

答：使用静态内部类的好处如下：  
作用域不会扩散到包外；

* 可以通过“外部类.内部类”的方式直接访问；
* 内部类可以访问外部类中的所有静态属性和方法。

#### 5.以下代码执行的结果是？

class OuterClass {  
 String name = "OuterClass";  
 protected static class InnerClass {  
 String name = "InnerClass";  
 public void sayHi() {  
 System.out.println(OuterClass.this.name);  
 }  
 }  
}  
class InnerClassTest {  
 public static void main(String[] args) {  
 OuterClass.InnerClass innerClass = new OuterClass.InnerClass();  
 innerClass.sayHi();  
 }  
}

答：程序报错。  
题目解析：在静态成员内部类中不能直接访问非静态外部类，因此程序会报错。

#### 6.成员内部类和局部内部类有什么区别？

答：内部成员类和局部内部类的区别如下。

* 内部成员类可以使用任意访问修饰符，局部内部类不能使用任何访问修饰符；
* 局部内部类是声明在外部类的方法或其他作用域范围内的，内部类是直接声明在外部类之中的，与方法和属性平级。

#### 7.为什么要使用内部类？内部类的使用场景有哪些？

答：使用内部类的好处有以下两个。

* 可以作为多继承的一种实现方式，最早内部类的实现就是平衡 Java 语言中没有多继承的一种方式；
* 方便将存在一定逻辑关系的类组织在一起，又可以对外界隐藏。  
  内部类可以作为多继承的一种实现方式进行使用，因为每个内部类都能独立的继承一个类或接口，所以整个类就可以实现多继承。

#### 8.以下代码执行的结果是？

class Outer {  
 public int num = 1;  
 class Inner {  
 public int num = 2;  
 public void show() {  
 int num = 3;  
 System.out.println(num);  
 System.out.println(this.num);  
 System.out.println(Outer.this.num);  
 }  
 }  
}  
class InnerTest {  
 public static void main(String[] args) {  
 new Outer().new Inner().show();  
 }  
}

答：输出内容如下。

3  
2  
1

#### 9.枚举有哪些应用场景？

答：枚举类的主要应用场景如下：  
**① 枚举类可作为高级的常量类**  
示例代码如下：

public enum Color {  
 RED("#FF0000", "255,0,0"),  
 GREEN("#00FFFF", "0,255,255"),  
 YELLOW("#FFFF00", "255,255,0");  
 String hex, rgb;  
 Color(String hex, String rgb) {  
 this.hex = hex;  
 this.rgb = rgb;  
 }  
}

**② 枚举类可方便的用于 switch 判断**  
示例代码如下：

switch(color)  
{  
case RED:  
 System.out.println("红灯停");  
 break;  
case GREEN:  
 System.out.println("绿灯行");  
 break;  
case YELLOW:  
 System.out.println("看情况");  
 break;  
default:  
 System.out.println("灯坏了");  
}

#### 10.枚举类在 JVM 中是如何实现的？

答：枚举类在 JVM（Java 虚拟机） 中其实是通过普通的 static final 形式实现的。  
题目解析：我们使用 javap 命令来分析枚举类最终编译的结果，查看编译后的结果，就找到了枚举类在 JVM 中的具体实现了。  
首先定义一个枚举类，代码如下：

enum DBEnum {  
 ORACLE,  
 DB2,  
 MYSQL,  
 SQLSERVER  
}

再使用命令 javac DBEnum.java 编译 .class 文件，然后再使用命令 javap DBEnum.class，我们看到最终执行的结果如下：

Compiled from "EnumTest.java"  
final class DBEnum extends java.lang.Enum<DBEnum> {  
public static final DBEnum ORACLE;  
public static final DBEnum DB2;  
public static final DBEnum MYSQL;  
public static final DBEnum SQLSERVER;  
public static DBEnum[] values();  
public static DBEnum valueOf(java.lang.String);  
static {};  
}

由此可以断定，枚举类在 JVM 中的实现也是通过普通的 static final 实现的。

#### 11.枚举类可以被继承吗？

答：不能被继承，因为枚举类编译后的实际代码是 final class 的形式，类被 final 修饰了自然不能被继承。

#### 12.枚举类是否是线程安全的？

答：枚举类是线程安全的，因为枚举类被编译后是 final class 的形式存在的，所以枚举类是线程安全的。

#### 13.枚举是否可以被序列化？

答：枚举是可以被序列化的，Oracle 官方对此给出了说明，内容如下：

Enum constants are serialized differently than ordinary serializable or externalizable objects. The serialized form of an enum constant consists solely of its name; field values of the constant are not transmitted. To serialize an enum constant, ObjectOutputStream writes the string returned by the constant’s name method. Like other serializable or externalizable objects, enum constants can function as the targets of back references appearing subsequently in the serialization stream. The process by which enum constants are serialized cannot be customized; any class-specific writeObject and writeReplace methods defined by enum types are ignored during serialization. Similarly, any serialPersistentFields or serialVersionUID field declarations are also ignored–all enum types have a fixed serialVersionUID of 0L

原文地址：<https://docs.oracle.com/javase/8/docs/api/java/io/ObjectOutputStream.html>  
大致的意思是说：枚举的序列化和其他普通类的序列化不同，枚举序列化的时候，只是将枚举对象的 name 属性输出到结果中，反序列化的时候则是通过 java.lang.Enum 的 valueOf 方法根据名字查找枚举对象。

### 总结

通过本文我们系统地学习了 Java 的各种内部类：静态内部类、成员内部类、局部内部类、匿名内部类，知道了它们特点和区别，并学习了枚举类了使用，知道了枚举类在编译之后，其实还是普通的最终类（final class）。

[点击此处下载本文源码](https://github.com/vipstone/java-%20interview/tree/master/interview-code/src/main/java/com/interview)

## 更多资源下载交流请加微信：Morstrong,加入永久会员,网盘更新更快捷！

# 本资源由微信公众号：光明顶一号，提供支持