**Java 修饰符**

Java语言提供了很多修饰符，主要分为以下两类：

* 访问修饰符
* 非访问修饰符

修饰符用来定义类、方法或者变量，通常放在语句的最前端。我们通过下面的例子来说明：

public class className {

// ...

}

private boolean myFlag;

static final double weeks = 9.5;

protected static final int BOXWIDTH = 42;

public static void main(String[] arguments) {

// 方法体

}

**访问控制修饰符**

Java中，可以使用访问控制符来保护对类、变量、方法和构造方法的访问。Java支持4种不同的访问权限。

默认的，也称为default，在同一包内可见，不使用任何修饰符。

私有的，以private修饰符指定，在同一类内可见。

共有的，以public修饰符指定，对所有类可见。

受保护的，以protected修饰符指定，对同一包内的类和所有子类可见。

**默认访问修饰符-不使用任何关键字**

使用默认访问修饰符声明的变量和方法，对同一个包内的类是可见的。接口里的变量都隐式声明为public static final,而接口里的方法默认情况下访问权限为public。

实例：

如下例所示，变量和方法的声明可以不使用任何修饰符。

String version = "1.5.1";

boolean processOrder() {

return true;

}

**私有访问修饰符-private**

私有访问修饰符是最严格的访问级别，所以被声明为private的方法、变量和构造方法只能被所属类访问，并且类和接口不能声明为private。

声明为私有访问类型的变量只能通过类中公共的getter方法被外部类访问。

Private访问修饰符的使用主要用来隐藏类的实现细节和保护类的数据。

下面的类使用了私有访问修饰符：

public class Logger {

private String format;

public String getFormat() {

return this.format;

}

public void setFormat(String format) {

this.format = format;

}

}

实例中，Logger类中的format变量为私有变量，所以其他类不能直接得到和设置该变量的值。为了使其他类能够操作该变量，定义了两个public方法：getFormat() （返回format的值）和setFormat(String)（设置format的值）

**公有访问修饰符-public**

被声明为public的类、方法、构造方法和接口能够被任何其他类访问。

如果几个相互访问的public类分布在不同的包中，则需要导入相应public类所在的包。由于类的继承性，类所有的公有方法和变量都能被其子类继承。

以下函数使用了公有访问控制：

public static void main(String[] arguments) {

// ...

}

Java程序的main() 方法必须设置成公有的，否则，Java解释器将不能运行该类。

**受保护的访问修饰符-protected**

被声明为protected的变量、方法和构造器能被同一个包中的任何其他类访问，也能够被不同包中的子类访问。

Protected访问修饰符不能修饰类和接口，方法和成员变量能够声明为protected，但是接口的成员变量和成员方法不能声明为protected。

子类能访问Protected修饰符声明的方法和变量，这样就能保护不相关的类使用这些方法和变量。

下面的父类使用了protected访问修饰符，子类重载了父类的openSpeaker()方法。

class AudioPlayer {

protected boolean openSpeaker(Speaker sp) {

// 实现细节

}

}

class StreamingAudioPlayer {

boolean openSpeaker(Speaker sp) {

// 实现细节

}

}

如果把openSpeaker()方法声明为private，那么除了AudioPlayer之外的类将不能访问该方法。如果把openSpeaker()声明为public，那么所有的类都能够访问该方法。如果我们只想让该方法对其所在类的子类可见，则将该方法声明为protected。

**访问控制和继承**

请注意以下方法继承的规则：

* 父类中声明为public的方法在子类中也必须为public。
* 父类中声明为protected的方法在子类中要么声明为protected，要么声明为public。不能声明为private。
* 父类中声明为private的方法，不能够被继承。

**非访问修饰符**

为了实现一些其他的功能，Java也提供了许多非访问修饰符。

static修饰符，用来创建类方法和类变量。

Final修饰符，用来修饰类、方法和变量，final修饰的类不能够被继承，修饰的方法不能被继承类重新定义，修饰的变量为常量，是不可修改的。

Abstract修饰符，用来创建抽象类和抽象方法。

Synchronized和volatile修饰符，主要用于线程的编程。

**Static修饰符**

* **静态变量：**

Static关键字用来声明独立于对象的静态变量，无论一个类实例化多少对象，它的静态变量只有一份拷贝。 静态变量也被成为类变量。局部变量能被声明为static变量。

* **静态方法：**

Static关键字用来声明独立于对象的静态方法。静态方法不能使用类的非静态变量。静态方法从参数列表得到数据，然后计算这些数据。

对类变量和方法的访问可以直接使用classname.variablename和classname.methodname的方式访问。

如下例所示，static修饰符用来创建类方法和类变量。

public class InstanceCounter {

private static int numInstances = 0;

protected static int getCount() {

return numInstances;

}

private static void addInstance() {

numInstances++;

}

InstanceCounter() {

InstanceCounter.addInstance();

}

public static void main(String[] arguments) {

System.out.println("Starting with " +

InstanceCounter.getCount() + " instances");

for (int i = 0; i < 500; ++i){

new InstanceCounter();

}

System.out.println("Created " +

InstanceCounter.getCount() + " instances");

}

}

以上实例运行编辑结果如下:

Started with 0 instances

Created 500 instances

**Final修饰符**

**Final变量：**

Final变量能被显式地初始化并且只能初始化一次。被声明为final的对象的引用不能指向不同的对象。但是final对象里的数据可以被改变。也就是说final对象的引用不能改变，但是里面的值可以改变。

Final修饰符通常和static修饰符一起使用来创建类常量。

实例:

public class Test{

final int value = 10;

// 下面是声明常量的实例

public static final int BOXWIDTH = 6;

static final String TITLE = "Manager";

public void changeValue(){

value = 12; //将输出一个错误

}

}

**Final方法**

类中的Final方法可以被子类继承，但是不能被子类修改。

声明final方法的主要目的是防止该方法的内容被修改。

如下所示，使用final修饰符声明方法。

public class Test{

public final void changeName(){

// 方法体

}

}

**Final类**

Final类不能被继承，没有类能够继承final类的任何特性。

实例：

public final class Test {

// 类体

}

**Abstract修饰符**

**抽象类：**

抽象类不能用来实例化对象，声明抽象类的唯一目的是为了将来对该类进行扩充。

一个类不能同时被abstract和final修饰。如果一个类包含抽象方法，那么该类一定要声明为抽象类，否则将出现编译错误。

抽象类可以包含抽象方法和非抽象方法。

实例：

abstract class Caravan{

private double price;

private String model;

private String year;

public abstract void goFast(); //抽象方法

public abstract void changeColor();

}

**抽象方法**

抽象方法是一种没有任何实现的方法，该方法的的具体实现由子类提供。抽象方法不能被声明成final和static。

任何继承抽象类的子类必须实现父类的所有抽象方法，除非该子类也是抽象类。

如果一个类包含若干个抽象方法，那么该类必须声明为抽象类。抽象类可以不包含抽象方法。

抽象方法的声明以分号结尾，例如：public abstract sample();

实例：

public abstract class SuperClass{

abstract void m(); //抽象方法

}

class SubClass extends SuperClass{

//实现抽象方法

void m(){

.........

}

}

**Synchronized修饰符**

Synchronized关键字声明的方法同一时间只能被一个线程访问。Synchronized修饰符可以应用于四个访问修饰符。

实例：

public synchronized void showDetails(){

.......

}

**Transient修饰符**

序列化的对象包含被transient修饰的实例变量时，java虚拟机(JVM)跳过该特定的变量。

该修饰符包含在定义变量的语句中，用来预处理类和变量的数据类型。

实例：

public transient int limit = 55; // will not persist

public int b; // will persist

**volatile修饰符**

Volatile修饰的成员变量在每次被线程访问时，都强迫从共享内存中重读该成员变量的值。而且，当成员变量发生变化时，强迫线程将变化值回写到共享内存。这样在任何时刻，两个不同的线程总是看到某个成员变量的同一个值。一个volatile对象引用可能是null。

实例：

public class MyRunnable implements Runnable

{

private volatile boolean active;

public void run()

{

active = true;

while (active) // line 1

{

// 代码

}

}

public void stop()

{

active = false; // line 2

}

}

一般地，在一个线程中调用run()方法，在另一个线程中调用stop()方法。如果line 1中的active位于缓冲区的值被使用，那么当把line 2中的active设置成false时，循环也不会停止。