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

方便在没有创建对象的情况下来进行调用（方法/变量）。故无this，但能通过this调用。不能调用非静态的变量和方法（3.1类的生命周期.note），但非静态可调用静态。

- **静态变量**

类初次初始化时会被初始化，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。静态变量也被称为类变量，存储在方法区。局部变量不能被声明为static变量。

- **静态方法**

声明独立于对象的静态方法。eg.main方法必须是static的，因为程序在执行main方法的时候没有创建任何对象，因此只有通过类名来访问。

- **static代码块**

只会在类加载的时候执行一次（优化程序性能）。

1. 静态内部类（static修饰类的话只能修饰内部类）： 静态内部类与非静态内部类之间存在一个最大的区别: 非静态内部类在编译

完成之后会隐含地保存着一个引用，该引用是指向创建它的外围类，但是静态内部类却没有。没有这个引用就意味着：1. 它的创建是不需要依赖外围类的创建。2. 它不能使用任何外围类的非static成员变量和方法。

2. 静态导包(用来导入类中的静态资源，1.5之后的新特性): 格式为：`import static` 这两个关键字连用可以指定导入某个类中的指定静态资源，并且不需要使用类名调用类中静态成员，可以直接使用类中静态成员变量和成员方法。

静态属性和静态方法可以被继承，但是没有被重写(overwrite)而是被隐藏.

原因：

1)．静态方法和属性是属于类的，调用的时候直接通过类名.方法名完成对，不需要继承机制及可以调用。如果子类里面定义了静态方法和属性，那么这时候父类的静态方法或属性称之为"隐藏"。如果要调用父类的静态方法和属性，直接通过父类名.方法或变量名完成，至于是否继承一说，子类是有继承静态方法和属性，但是跟实例方法和属性不太一样，存在"隐藏"的这种情况。

2)．多态之所以能够实现依赖于继承、接口和重写、重载（继承和重写最为关键）。有了继承和重写就可以实现父类的引用指向不同子类的对象。重写的功能是："重写"后子类的优先级要高于父类的优先级，但是“隐藏”是没有这个优先级之分的。

3)．静态属性、静态方法和非静态的属性都可以被继承和隐藏而不能被重写，因此不能实现多态，不能实现父类的引用可以指向不同子类的对象。非静态方法可以被继承和重写，因此可以实现多态。

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。修饰的变量不能被改变，一旦赋了初值，就不能被重新赋值。被声明为final的对象的引用不能指向不同的对象，但是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：**

可被子类继承，但不可被修改（不可重写）。类中所有的private方法都隐式地指定为final。

- **final类：**

不能被继承，final类中的所有成员方法都会被隐式地指定为final方法。eg：Math、String等。

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。抽象类可以不包含抽象方法。

- **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。
例如：public abstract sample();

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。当对象被反序列化时，被transient修饰的变量值不会被持久化和恢复。transient只能修饰变量，不能修饰类和方法。

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。