### 1. 访问控制

###### 1.1 private 私有的

private 关键字是访问控制修饰符，可以应用于类、方法或字段（在类中声明的变量）。

只能在声明 private（内部）类、方法或字段的类中引用这些类、方法或字段。在类的外部或者对于子类而言，它们是不可见的。

所有类成员的默认访问范围都是 package 访问，也就是说，除非存在特定的访问控制修饰符，否则，可以从同一个包中的任何类访问类成员。

###### 1.2 protected 受保护的

protected 关键字是可以应用于类、方法或字段（在类中声明的变量）的访问控制修饰符。

可以在声明 protected 类、方法或字段的类、同一个包中的其他任何类以及任何子类（无论子类是在哪个包中声明的）中引用这些类、方法或字段。

所有类成员的默认访问范围都是 package 访问，也就是说，除非存在特定的访问控制修饰符，否则，可以从同一个包中的任何类访问类成员。

###### 1.3 public 公共的

public 关键字是可以应用于类、方法或字段（在类中声明的变量）的访问控制修饰符。

可能只会在其他任何类或包中引用 public 类、方法或字段。

所有类成员的默认访问范围都是 package 访问，也就是说，除非存在特定的访问控制修饰符，否则，可以从同一个包中的任何类访问类成员。

### 2. 类、方法和变量修饰符

###### 2.1 abstract 抽象

abstract关键字可以修改类或方法。

abstract类可以扩展（增加子类），但不能直接实例化。

abstract方法不在声明它的类中实现，但必须在某个子类中重写。

-示例-

public abstract class MyClass{}

public abstract String myMethod();

-注释-

采用abstract方法的类本来就是抽象类，并且必须声明为abstract。

abstract类不能实例化。

仅当abstract类的子类实现其超类的所有abstract方法时，才能实例化abstract类的子类。这种类称为具体类，以区别于abstract类。

如果abstract类的子类没有实现其超类的所有abstract方法，该子类也是abstract类。

abstract关键字不能应用于static、private或final方法，因为这些方法不能被重写，因此，不能在子类中实现。

final类的方法都不能是abstract，因为final类不能有子类。

###### 2.2 extends 继承

extends关键字用在class或interface声明中，用于指示所声明的类或接口是其名称后跟有extends关键字的类或接口的子类。

-示例-

public class Rectangle extends Polygon{

}

-注释-

在上例中，Rectangle类继承Polygon类的所有public和protected变量和方法。

Rectangle类可以重写Polygon类的任何非final方法。

一个类只能扩展一个其他类。

###### 2.3 final

final关键字可以应用于类，以指示不能扩展该类（不能有子类）。

final关键字可以应用于方法，以指示不能重写任何子类中的方法。

-示例-

public final class MyFinalClass{

}

public class MyClass{

public final String myFinal Method(){

<statements>

}

}

-注释-

一个类不能同时是abstract又是final。abstract意味着必须扩展类，final意味着不能扩展类。

一个方法不能同时是abstract又是final。abstract意味着必须重写方法，final意味着不能重写方法。

###### 2.4 finally

finally关键字用来定义始终在try-catch-finally语句中执行的块。

finally块通常包含清理代码，用在部分执行try块后恢复正常运行。

-示例-

try{

<可能引发异常的块>

}catch(<java.lang.Exception或子类>e){

<处理异常e的代码>

}finally{

<有异常或无异常情况下都执行的语句>

}

-注释-

开始和结束标记{和}是finally子句语法的一部分，即使该子句只包含一个语句，也不能省略这两个标记。

每个try块都必须至少有一个catch或finally子句。

如果执行try块的任何部分，不论是否出现异常，也不论try或catch块是否包含return、continue或break语句，都一定会执行finally块中的代码。

如果不出现异常，控件将跳过try块，进入finally块。

如果在执行try块期间出现异常，并且相应的catch块包含break、continue或return语句，控件将首先穿过finally块，之后再执行break、continue或return。

###### 2.5 interface 接口

interface关键字用来声明新的Java接口，接口是方法的集合。

接口是Java语言的一项强大功能。任何类都可声明它实现一个或多个接口，这意味着它实现了在这些接口中所定义的所有方法。

-示例-

public interface IPolygon{

public float getArea();

public int getNumberOfSides();

public int getCircumference();

}

-注释-

实现了接口的任何类都必须提供在该接口中的所有方法的实现。

一个类可以实现多个接口。

###### 2.6 synchronized 锁

synchronized关键字可以应用于方法或语句块，并为一次只应由一个线程执行的关键代码段提供保护。

-示例-

public class MyClass{

public synchronized static String mySyncStaticMethod(){

}

public synchronized String mySyncMethod(){

}

}

public class MyOtherClass{

Object someObj;

public String myMethod(){

<statements>

synchronized(someObj){

<statement saffectingsomeObj>

}

}

}

-注释-

synchronized关键字可防止代码的关键代码段一次被多个线程执行。

如果应用于静态方法（如上例中的MySyncStaticMethod），那么，当该方法一次由一个线程执行时，整个类将被锁定。

如果应用于实例方法（如上例中的MySyncMethod），那么，当该方法一次由一个线程访问时，该实例将被锁定。

如果应用于对象或数组，当关联的代码块一次由一个线程执行时，对象或数组将被锁定。

###### 2.7 transient 保密

transient关键字可以应用于类的成员变量，以便指出该成员变量不应在包含它的类实例已序列化时被序列化。

-示例-

public class MyClass{

private transient String password;

}

java 的transient关键字的作用是需要实现Serilizable接口，将不需要序列化的属性前添加关键字transient，序列化对象的时候，这个属性就不会序列化到指定的目的地中。(简单理解加了transient的变量只是临时变量，不会持久化，例如个人信息 银行卡密码等隐秘，只会被在当前内存中使用)  
transient使用小结  
1）一旦变量被transient修饰，变量将不再是对象持久化的一部分，该变量内容在序列化后无法获得访问。  
2）transient关键字只能修饰变量，而不能修饰方法和类。注意，本地变量是不能被transient关键字修饰的。变量如果是用户自定义类变量，则该类需要实现Serializable接口。

被transient关键字修饰的变量不再能被序列化，一个静态变量不管是否被transient修饰，均不能被序列化。

###### 2.8 volatile

volatile关键字用于表示可以被多个线程异步修改的成员变量。

解决线程中数据的可见性：变量写会立即刷新到主内存，读会读取主存中新值，跳过本地内存（共享变量的副本）

保证原子性(有些人否认)：

具体来看User user = new User的语义：  
1：分配对象的内存空间  
2：初始化对象  
3：设置user指向刚分配的内存地址

这些线程1在执行完第3步而还没来得及执行完第2步的时候，如果内存刷新到了主存，那么线程2将得到一个未初始化完成的对象。因此如果将user声明为volatile的，那么步骤2,3将不会被重排序。

单例模式的双重检验 很经典 不加volatile 会出现不完整实例问题,解决写的原子性问题。

1. 对一个volatile变量的写操作，只有所有步骤完成，才能被其它线程读取到。  
2. 多个线程对volatile变量的写操作本质上是有先后顺序的。也就是说并发写没有问题。

提供内存屏障解决有序性

误区：

很多人会认为对volatile变量的所有操作都是原子性的，比如自增i++

i++这种操作不是原子操作  
2.volatile 并不会有锁的特性

注意：volatile关键字在许多Java虚拟机中都没有实现。

-示例-

public class MyClass{

volatile int shared Value;

}

-注释-

volatile的目标用途是为了确保所有线程所看到的指定变量的值都是相同的。

###### 2.9 static 静态

static关键字可以应用于内部类（在另一个类中定义的类）、方法或字段（类的成员变量）。

-示例-

public class MyPublicClass{

public final static int MAX\_OBJECTS=100;

static int\_numObjects=0;

static class MyStaticClass{

}

static int getNumObjects(){

}

}

-注释-

通常，static关键字意味着应用它的实体在声明该实体的类的任何特定实例外部可用。

static（内部）类可以被其他类实例化和引用（即使它是顶级类）。在上面的-示例-中，另一个类中的代码可以实例化MyStaticClass类，方法是用包含它的类名来限定其名称，如MyClass.MyStaticClass。

static字段（类的成员变量）在类的所有实例中只存在一次。

可以从类的外部调用static方法，而不用首先实例化该类。这样的引用始终包括类名作为方法调用的限定符。在上面的示例中，MyClass类外部的代码以MyClass.getNumObjects()的形式调用getNumObjects()static方法。

模式：

publicfinalstatic<type>varName=<value>;

通常用于声明可以在类的外部使用的类常量。在引用这样的类常量时需要用类名加以限定。在上面的-示例-中，另一个类可以用MyClass.MAX\_OBJECTS形式来引用MAX\_OBJECTS常量。

### 3.基本类型

###### 3.1 boolean

 boolean 是 Java 原始类型。boolean 变量的值可以是 true 或 false。

 boolean 变量只能以 true 或 false 作为值。boolean 不能与数字类型相互转换。

 包含 boolean 操作数的表达式只能包含 boolean 操作数。

 Boolean 类是 boolean 原始类型的包装对象类。

###### 3.2 byte

 byte 是 Java 原始类型。byte 可存储在 [-128, 127] 范围以内的整数值。

 Byte 类是 byte 原始类型的包装对象类。它定义代表此类型的值的范围的 MIN\_VALUE 和 MAX\_VALUE 常量。

###### 3.3 char

char 是 Java 原始类型。char 变量可以存储一个 Unicode 字符。

可以使用下列 char 常量：\b - 空格, \f - 换页, \n - 换行, \r - 回车, \t - 水平制表符, \' - 单引号, \" - 双引号, \\ - 反斜杠, \xxx - 采用 xxx 编码的 Latin-1 字符。\x 和 \xx 均为合法形式，但可能引起混淆。 \uxxxx - 采用十六进制编码 xxxx 的 Unicode 字符。

Character 类包含一些可用来处理 char 变量的 static 方法，这些方法包括 isDigit()、isLetter()、isWhitespace() 和 toUpperCase()。

char 值没有符号。

###### 3.4 short

short 是 Java 原始类型。short 变量可以存储 16 位带符号的整数。

Short 类是 short 原始类型的包装对象类。它定义代表此类型的值的范围的 MIN\_VALUE 和 MAX\_VALUE 常量。

Java 中的所有整数值都是 32 位的 int 值，除非值后面有 l 或 L（如 235L），这表示该值应解释为 long。

###### 3.5 int

int 是 Java 原始类型。int 变量可以存储 32 位的整数值。

Integer 类是 int 原始类型的包装对象类。它定义代表此类型的值的范围的 MIN\_VALUE 和 MAX\_VALUE 常量。

###### 3.6 long

long 是 Java 原始类型。long 变量可以存储 64 位的带符号整数。

Long 类是 long 原始类型的包装对象类。它定义代表此类型的值的范围的 MIN\_VALUE 和 MAX\_VALUE 常量。

Java 中的所有整数值都是 32 位的 int 值，除非值后面有 l 或 L（如 235L），这表示该值应解释为 long。

###### 3.7 float

float 是 Java 原始类型。float 变量可以存储单精度浮点值。

使用此关键字时应遵循下列规则：

Java 中的浮点文字始终默认为双精度。

要指定单精度文字值，应在数值后加上 f 或 F，如 0.01f。

由于浮点数据类型是实际数值的近似值，因此，一般不要对浮点数值进行是否相等的比较。

Java 浮点数值可代表无穷大和 NaN（非数值）。Float 包装对象类用来定义常量 MIN\_VALUE、MAX\_VALUE、NEGATIVE\_INFINITY、POSITIVE\_INFINITY 和 NaN。

###### 3.8 double

double 是 Java 原始类型。double 变量可以存储双精度浮点值。

由于浮点数据类型是实际数值的近似值，因此，一般不要对浮点数值进行是否相等的比较。

Java 浮点数值可代表无穷大和 NaN（非数值）。Double 包装对象类用来定义常量 MIN\_VALUE、MAX\_VALUE、NEGATIVE\_INFINITY、POSITIVE\_INFINITY 和 NaN。