Final关键字可用于修饰类、变量和方法，final关键字有点类似于c#的sealed关键字，用于表示它修饰的类、方法和变量不可改变。

Final修饰变量时，表示该变量一旦获得了初始值就不可被改变，final既可以修饰成员变量也可以修饰局部变量、形参。因为final变量获得初始值之后不能被重新赋值，因此final修饰成员变量和修饰局部变量时有一定的不同。

**Final成员变量**

对于final修饰的成员变量而言，一旦有了初始值，就不能被重新赋值，如果既没有在定义成员变量时指定初始值，也没有在初始化块、构造器中为成员变量指定初始值，那么这些成员变量的值将一直是系统默认分配的0、’\u0000’、false或null，这些成员变量也就完全失去了存在的意义，因此java语法规定，final修饰的成员变量必须由程序猿显式地指定初始值。

归纳起来，final修饰的类Field、实例Field能指定初始值的地方如下：

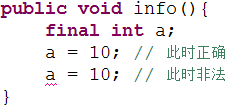
1. 类Field：必须在静态初始化块中或声明该Field时指定初始值
2. 实例Field：必须在非静态初始化块，声明该Field或构造器中指定初始值

注意：与普通成员变量不同的是，final成员变量（包括实例Field和类Field）必须由程序员显式初始化，系统不会对final成员进行隐式初始化

**Final局部变量**

系统不会对局部变量进行初始化，局部变量必须由程序员显式初始化，因此使用final修饰局部变量时，既可以在定义时指定默认值，也可以不指定默认值。

如果final修饰的局部变量在定义时没有指定默认值，则可以在后面代码中对该final变量赋初始值，但只能一次，不能重复赋值。



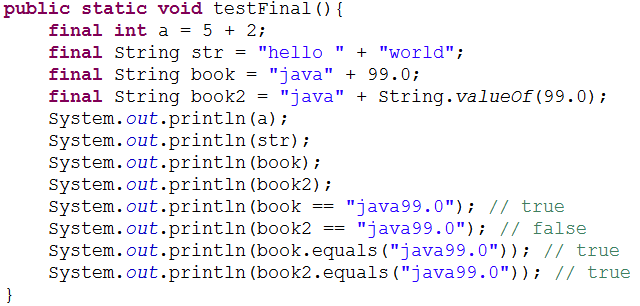
**Final修饰基本类型变量和引用类型变量的区别**

当使用final修饰基本类型变量时，不能对基本类型变量重新赋值，因此基本类型变量不能被改变，但对于引用类型变量而言，它保存的仅仅是一个 引用，final只保证这个引用类型变量所引用的地址不会改变，即一直引用同一个对象，但这个对象完全可以发生改变。

**可执行“宏替换”的final变量**

1. 使用final修饰符修饰
2. 在定义该final变量时指定了初始值
3. 该初始值可以在编译时就被确定下来

注意：final修饰符的一个重要用途就是定义“宏变量”，当定义final变量时就为该变量指定初始值，而且该初始值可以在编译时就确定下来，那么这个final变量本质上就是一个“宏变量”，编译器会把程序中所有用到该变量的地方直接替换成该变量的值。



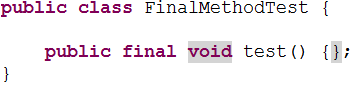
在上面的代码中a、str、book都变为“宏变量”，但是book2却不是，那是因为该变量的值需要调用String类的方法，因此编译器无法在编译时确定book2的值，因为不会当成“宏变量”处理。程序最后还判断了book与book2是否与“java99.0”相等，由于book是一个“宏变量”，它将被直接替换成“java99.0”，因此book相等，但book2和该字符串不相等。

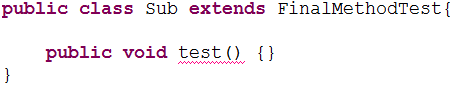
注意：java会使用常量池来管理曾经用过的字符串直接量，例如，执行String a=“java”语句，系统的字符串池中就会缓存一个字符串“java”，如果程序再次执行String b=“java”，系统将会让b直接指向字符串池中的“java”字符串，因此a==b将会返回true

**Final方法**

Final修饰的方法不可被重写，如果处于某些原因，不希望子类重写父类的某个方法，则可以使用final修饰该方法

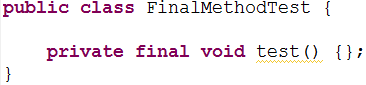
Java提供的Object类里就有一个final方法：getClass()





在上面的代码中，子类试图重写父类的test方法，会引发编译错误。

对于一个private方法，因为它仅在当前类中可见，其子类无法访问该方法，所以子类无法重写该方法--如果子类中定义了一个与父类private方法有相同方法名、相同形参列表、相同返回值类型的方法，也不是方法重写，只是重新定义了一个新方法，因此，即使使用final修饰一个private访问权限的方法，依然可以在其子类中定义与该方法具有相同方法名、相同形参列表、相同返回值类型的方法。修改为下面的形式则没问题（已测试）：



**Final类**

Final修饰的类不可以有子类，例如java.lang.Math类就是一个final类

