1. 创建和销毁对象
2. 用静态工厂方法代替构造器

public static Boolean valueOf(Boolean b){

return b ? Boolean.true:Boolean.false;

}

一优势：它们有名称。

二优势：不必在每次调用它们的时候都创建一个对象。

三优势：它们可以返回原返回类型的任何子类型的对象。

四优势：所返回的对象的类可以随着每次调用而发生变化，这取决于静态工厂方法的 参数值（只要是已声明的返回类型的子类型）。

五优势：方法返回的对象所属的类，在编写包含该静态工厂方法的类时可以不存在。

一缺点：类如果不含公有的或者受保护的构造器，就不能被子类化。

二缺点：程序员很难发现它们。

第二条 遇到多个构造器参数时要考虑使用构造器

new Test(String a, String b, String c);

简而言之，重叠构造器模式可行，但是当有许多参数的时候，客户端代码会很难编写并且仍然较难以读懂。

JavaBeans模式：先调用一个无参构造器来创建对象，然后在调用setter方法来设置每个必要的参数。

JavaBeans不足之处：在构造过程中JavaBean可能处于不一致的状态。

JavaBeans模式使得把类作出不可变的可能性不复存在。

Builder模式模拟了具名的可选参数。

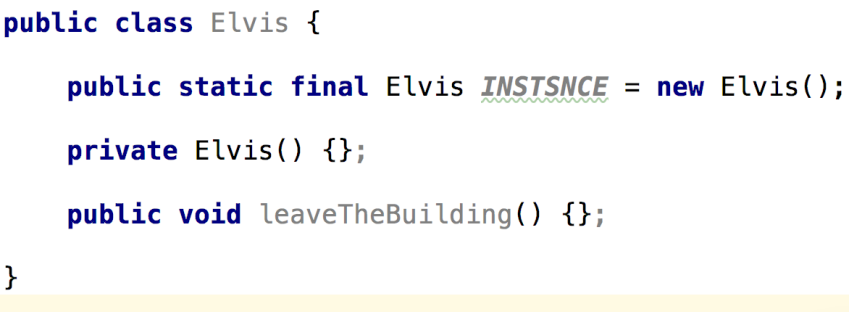
Builder模式也适用于类层次结构：子类方法声明返回超级类中声明的返回类型的子类型，这被称为协变返回类型。

如果类的构造器或者静态工厂中具有多个参数，设计这种类时，Builder模式就是一种不错的选择。

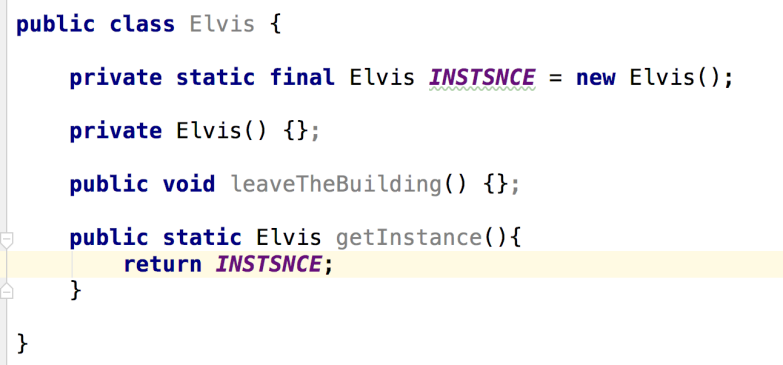
第三条 用私有构造器或者枚举类型强化Singleton属性

Singleton是指仅仅被实例化一次的类。Singleton通常被用来代表一个无状态的对象，如函数，或者那些本质上唯一的系统组件。使类成为Singleton会使它的客户端测试变得十分困难。

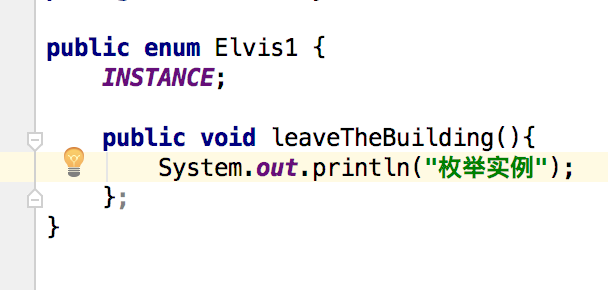
实现Singleton有两种常见的方法，这两种方法都要保持构造器为私有的，并导出公有的静态成员，以便允许客户端能够访问该类的唯一实例。



二，都会返回同一个对象引用，所以，永远不会创建其他的Elvis实例



单元素的枚举类型经常成为实现Singleton的最佳方法。



1. ：通过私有构造器强化不可实例化的能力

企图通过将类做成抽象类来强制该类不可被实例化是行不通的。该类可以被子类化，并且该子类也可以被实例化。由于只有当类不包含显示的构造器时，编译器才会生产缺省的构造器，因此只要让这个类包含一个私有构造器，它就不能被实例化。

public class Test {

//为非实例性禁用默认构造函数

private Test(){

//避免不小心在类的内部调用构造器。

throw new AssertionError();

}

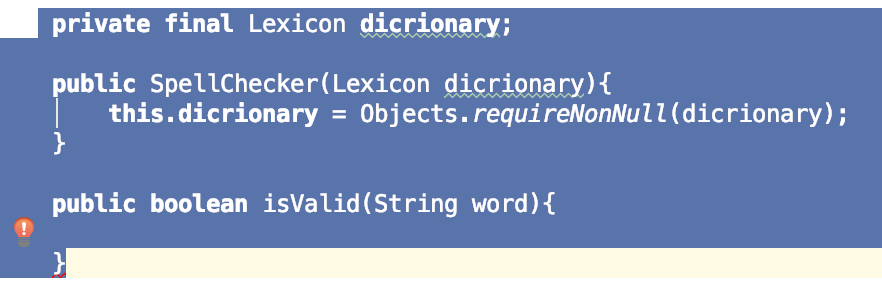
}

副作用：它使得一个类不能被子类化，所有的构造器都必须显式的或隐式的调用超类构造器，在这种情况下，子类就没有可访问的超类构造器可调用了。

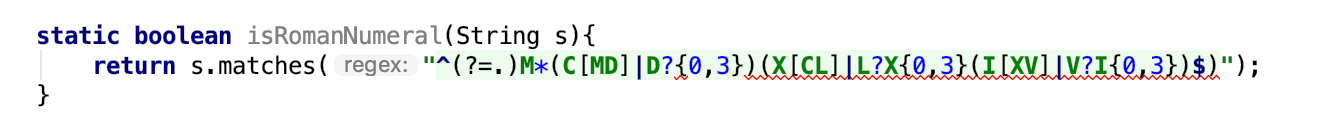
1. ：优先考虑依赖注入来引用资源

静态工具类和Singleton类不适合于需要引用底层资源的类。

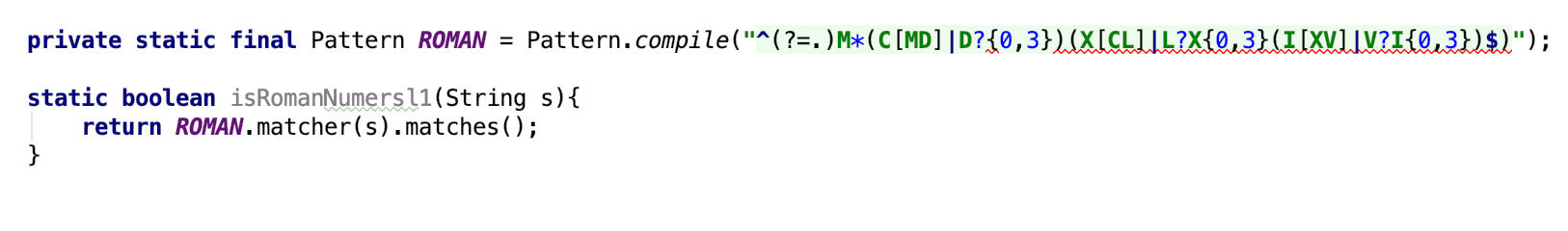
当创建一个新的实例时，就将该资源传到构造器中，这是依赖注入的一种形式：词典就是拼写检查器的一个依赖，在创建拼写检查器时就将词典注入其中。



第六条：避免创建不必要的对象



虽然String.matches方法最易于查看一个字符串是否与正则表达式相匹配，但并不适合在注重性能的情形中使用。



适配器是指这样一个对象：它把功能委托给一个后备对象，从而为后备对象提供一个可以替代的接口。

自动装箱使得基本类型和装箱类型之间的差别变得模糊起来，但是并没有完全消除。

要优先使用基本类型而不是装箱基本类型，要当心无意识的自动装箱。

1. ：消除过期的对象引用

清空对象引用应该是一种例外，而不是一种规范行为。

Stack类自己管理内存。存储池包含了elements数组的元素，数组活动区域中的元素是已分配的，而数组其余部分的元素则是自由的。

只要类自己管理内存，程序员就应该警惕内存泄漏问题。

内存泄漏的另一个常见来源是缓存：（1）可以用WeakHashMap代表缓存，当缓存中的项过期之后，它们就会自动被删除。

内存泄漏的第三个常见来源是监听器和其他回调。