第23条：请不要在新代码中使用原生态类型

使用泛型的优点：
1.会进行编译前检查，保证在编译时将错误检查出来
2.在获取元素时，编译器会进行隐式转换，并确保不会出现失败。

为什么还要允许原生态类型?
这是为了提供兼容性，因为在泛型出现的时候，已经存在大量没有使用泛型的Java代码

在新代码中不应该使用像List这样的原生态类型，使用参数化的类型以允许插入任意对象，如List<Object>，这还是可以的。
如果要使用泛型，但不确定或者不关心实际的参数类型，就可以使用一个问号代替。

原生态类型List与参数化类型List<Object>之间的区别?
1.前者逃避了泛型检查，后者则明确告诉编译器，它能够持有任意类型的对象。
2.List<String>是原生态类型List的一个子类型，而不是参数化类型List<Object>的子类型。

原生态类型List与无限制通配符List<?>之间的区别?
1.通配符类型是安全的，原生态则不安全。
2.可以将任何元素放进无限制通配符List<?>集合中，因此很容易破坏该集合的类型约束条件。但不能将除null之外的任何元素放到Collection<?>中，因为无法猜测你会得到哪种类型的对象。要是无法接受这些限制，就可以使用泛型方法或者有限制的通配符。

不要在新代码中使用原生态类型，但有两个小小的例外，两者都源于“泛型可以在运行时被擦除”
1.在类文字中必须使用原生态类型，例如List.class是合法的，但List<String.class>则不合法。
2.在参数化类型而非无限制通配符上使用instanceof操作是非法的。

第24条：消除非受检的警告

尽可能消除每一个非受检的警告。
如果消除了所有警告，就可以确保代码是类型安全。
如果无法消除警告，同时可以证明引起警告的代码是类型安全的，就可以在尽可能小的范围中，用@SuppressWarnings("unchecked")注解禁止该警告。要注意把禁止该警告的原因记录下来。

第25条：列表优先于数组

为什么不允许创建泛型数组?
原因：因为泛型数组不是类型安全的。数组是协变的，可以将任何一个泛型数组转化为Object[]存入任意类型的元素，可能在运行时抛出ClassCastException异常。即使虚拟机不允许创建泛型数组，但可以将一个Object[]转换成泛型数组，或者将Object[]中的元素转化成泛
型E，这两种情况都会因为不是类型安全的而出现警告。

数组与范型的不同点：
1.数组是协变的：意思是如果Sub为Super的子类型，那么数组Sub[]就是Super[]的子类型。
泛型是不可变的：对于任意两个不同的类型Type1和Type2，List<Type1>既不是List<Type2>的子类型，也不是List<Type2>的超类型。
2.数组是具体化的，数组会在运行时才知道并检查它们的元素类型约束。
泛型则是通过擦除来实现的，泛型只是在编译时强化它们的类型信息，并在运行时丢弃（或者擦除）它们的元素类型信息。

当调用可变参数方法时，如果需要创建一个数组来存放varargs参数。如果这个数组的元素类型不是可具体化的，就会得到一条警告，除了把它们禁止，并且避免在API中混合使用泛型与可变参数之外，别无它法。

数组和范型不能很好的混用，如果混合起来使用，并且得到编译时的错误和警告，就应该使用列表代替数组。

第26条：优先考虑泛型

列表优先于数组，但是实际情况中，并不可能总是或者总想在泛型中使用列表?
1.Java并不是生来就支持列表，因此有些泛型例如ArrayList，必须在数组的基础

上实现。
2.为了提升性能，其他泛型例如HashMap也在数组上实现。

每当编写需要泛型类型的数组类时，都会碰到不能创建泛型类型的数组的问题，怎么解决?
1.直接绕过创建泛型数组的禁令：创建一个Object的数组，并将它转换成泛型数组类型。
2.将elements域的类型从E[]改为Object[]，把数组中获取的元素由Object转换成E。

第27条：优先考虑泛型方法

什么是类型推断?
编译器通过检查方法参数包括输入参数和返回值的类型来计算类型参数的值。

第28条：利用有限制通配符来提升API的灵活性

参数化类型是不可变的，意味着，对于任何两个截然不同的Type1和Type2而言，List<Type1>既不是List<Type2>的子类型，也不是它的超类型。但是有时候我们需要实现更加灵活性的API，怎么办?
可以利用有限制类型的通配符来处理类似的情况。

通配符类型基本法则：
PECS表示producer-extends，consumer-super
利用场景：
1.为了获得更大限度的灵活性，要在表示生产者或者消费者的输入参数上使用通配符。
2.可以通过这条基本法则，判断输入参数是否作为E生产者，还是E消费者。

不要用通配符类型作为返回类型，除了为用户提供额外的灵活性之外，它还会强制用户在客户端代码中使用通配符类型。

对于一个方法既可以使用类型参数也可以用通配符进行申明，如何选择?
如果类型参数只在方法声明中出现一次，就可以用通配符取代它。如果是无限制的类型参数，就用无限制的通配符取代它；如果是有限制的

类型参数，就用有限制的通配符取代它。

优先使用通配符而非类型参数存在问题：
如果list的类型为List<?>，你不能把null之外的任何值放到List<?>中。
解决办法：编写一个辅助方法来捕捉通配符类型。为了捕捉类型，辅助方法必须是泛型方法。

第29条：优先考虑类型安全的异构容器

集合API说明了泛型的一般用法，限制你每个容器只能有固定数目的类型参数。有时候需要绕开这种限制，需要有更多的灵活性。
可以考虑使用类型安全的异构容器：Class对象充当参数化键的部分。

类型安全的异构容器有两种局限性：
1.客户端只要以原生态使用Class对象就能破坏容器的类型安全。
解决办法：在容器存放的方法中检验instance是否真的是type所表示的类型的实例。
2.容器不能用在不可具体化的类型中。
解决办法：对于这种局限性，没有令人满意的解决方案，可以利用限制类型参数或者有限制通配符，来限制可以表示的类型。
这种情况下如果将一个类型Class<?>的对象转换成有限制通配符类型，会产生编译时警告，可以使用类Class提供的asSubclass方法将
Class对象转换其参数表示的类的一个子类。