第8条：覆盖equals时请遵守通用约定

避免覆盖equals的情况：
类的实例本身就是唯一的，例如一些代表活动实体而不是值的类。
没有必要提供逻辑相等的功能。
超类已经覆盖了equals，从超类继承过来的行为对于子类也是合适的。
类是私有的或是包级私有的，可以确认它的equals方法永远不会被调用。
实例受控的值类可以确保“每个值至多只存在一个对象”的类

需要覆盖equals的情况：
如果对象需要提供自己独特的逻辑相等功能，并且超类没有实现期望的功能。

equals方法实现的等价关系：
自反性、对称性、传递性、一致性

实现高质量的equals方法：
1.使用==操作符检查“参数是否为这个对象的引用”。
2.使用instanceof操作符检查“参数是否为正确的类型”。
3.把参数转换成正确的类型。
4.对于该类中的每个“关键”域，检查参数中的域是否与该对象中对应的域相匹配
5.检查equals方法是否满足对称性、传递性、一致性。
6.另外需要注意：覆盖equals时总是要覆盖hashCode。
不要企图让equals方法过度去寻求各种等价关系。
不要将equals声明中的Object对象替换为其他的类型。

第9条：覆盖equals时总要覆盖hashCode

原因：
如果违反Object.hashCode的这条通用约定，会导致该类对象，不能正常使用基于散列存储结构的集合来进行存储。

Object.hashCode通用约定：
相等的对象必须具有相等的散列码。
不相等的对象最好具有不同的散列码，可能用来提高散列表的性能。

实现接近理想的散列函数：
1.定义非零的局部变量int类型的result。
2.对于对象中每个关键的域f,计算出对应int类型的散列码c。
3.将步骤2中的每个散列码c合并到result中，如：result = 31 * result + c。
4.验证相等的实例是否具有相同的散列码。

如果一个类是不可变的，并且计算散列码的开销比较大，可以考虑把散列码缓存在对象内部。

第10条：始终要覆盖toString

在实际应用中，toString方法应该返回对象中包含的所有值得关注的信息。

第11条：谨慎地覆盖clone

Cloneable接口表明这样的对象允许克隆，如果一个类实现了Cloneable，Object的clone方法就返回该对象的逐域拷贝，否则就会抛出CloneNotSupportedException异常。

克隆方法需要注意的地方：
1.如果每个域包含一个基本类型的值，或者包含一个指向不可变对象的引用，那么被返回的对象则可能正是你所需要的对象。
2.如果对象中包含的域引用了可变的对象，必须确保修改原始的实例不会破坏被克隆对象中的约束条件。
3.clone架构与引用可变对象的final域的正常用法是不相容的。
4.如果clone调用了一个被覆盖的方法，那么该方法所在的子类有机会修正它在克隆对象中的状态之前，该方法就会被先执行，这样很有可能导致克隆对象和原始对象之间的不一致。
5.线程安全的类实现Cloneable接口，clone方法必须得到很好的同步。

如何提供良好的clone方法：
所有实现了Cloneable接口的类都应该用一个公有的方法覆盖clone。此公有的方法首先调用super.clone()，然后修正任何需要修正的域。

另外，实现对象拷贝的好办法是提供一个拷贝构造器或拷贝工厂。

第11条：考虑实现Comparable接口

为什么？

如果你正在编写一个值类，它具有非常明显的内在排序关系，就应该坚决考虑实现这个接口。一旦实现Comparable接口，它就可以跟许多范型算法以及依赖于该接口的集合实现进行协作。

compareTo约定：

自反性、对称性、传递性

建议：compareTo方法施加的等同性测试，在通常情况下应该返回与equals方法相同的结果