**覆盖equals时需要遵守的通用约定**：   
  覆盖equals方法看起来似乎很简单，但是如果覆盖不当会导致错误，并且后果相当严重。《Effective Java》一书中提到“最容易避免这类问题的办法就是不覆盖equals方法”，这句话貌似很搞笑，其实想想也不无道理，其实在这种情况下，类的每个实例都只与它自身相等。如果满足了以下任何一个条件，这就正是所期望的结果： 

* **类的每个实例本质上都是唯一的**。对于代表活动实体而不是值的类来说却是如此，例如Thread。Object提供的equals实现对于这些类来说正是正确的行为。
* **不关心类是否提供了“逻辑相等”的测试功能**。假如Random覆盖了equals，以检查两个Random实例是否产生相同的随机数序列，但是设计者并不认为客户需要或者期望这样的功能。在这样的情况下，从Object继承得到的equals实现已经足够了。
* **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**。大多数的Set实现都从AbstractSet继承equals实现，List实现从AbstractList继承equals实现，Map实现从AbstractMap继承equals实现。
* **类是私有的或者是包级私有的，可以确定它的equals方法永远不会被调用**。在这种情况下，无疑是应该覆盖equals方法的，以防止它被意外调用：

@Override   
public boolean equals(Object o){   
  throw new AssertionError(); //Method is never called   
}

  在覆盖equals方法的时候，你必须要遵守它的通用约定。下面是约定的内容，来自Object的规范[JavaSE6]

* **自反性**。对于任何非null的引用值x，x.equals(x)必须返回true。
* **对称性**。对于任何非null的引用值x和y，当且仅当y.equals(x)返回true时，x.equals(y)必须返回true
* **传递性**。对于任何非null的引用值x、y和z，如果x.equals(y)返回true，并且y.equals(z)也返回true，那么x.equals(z)也必须返回true。
* **一致性**。对于任何非null的引用值x和y，只要equals的比较操作在对象中所用的信息没有被修改，多次调用该x.equals(y)就会一直地返回true，或者一致地返回false。
* 对于任何非null的引用值x，x.equals(null)必须返回false。

  结合以上要求，得出了以下实现高质量equals方法的诀窍：   
**1.使用==符号检查“参数是否为这个对象的引用”**。如果是，则返回true。这只不过是一种性能优化，如果比较操作有可能很昂贵，就值得这么做。   
**2.使用instanceof操作符检查“参数是否为正确的类型”**。如果不是，则返回false。一般来说，所谓“正确的类型”是指equals方法所在的那个类。   
**3.把参数转换成正确的类型。**因为转换之前进行过instanceof测试，所以确保会成功。   
**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**。如果这些测试全部成功，则返回true;否则返回false。   
**5.当编写完成了equals方法之后，检查“对称性”、“传递性”、“一致性”。**   
**注意：** 

* 覆盖equals时总要覆盖hashCode 《Effective Java》作者说的https://www.iteye.com/images/smiles/icon_smile.gif
* 不要企图让equals方法过于只能。
* 不要将equals声明中的Object对象替换为其他的类型（因为这样我们并没有覆盖Object中的equals方法哦）

**覆盖equals时总要覆盖hashCode**   
  一个很常见的错误根源在于没有覆盖hashCode方法。在每个覆盖了equals方法的类中，也必须覆盖hashCode方法。如果不这样做的话，就会违反Object.hashCode的通用约定，从而导致该类无法结合所有基于散列的集合一起正常运作，这样的集合包括HashMap、HashSet和Hashtable。 

* 在应用程序的执行期间，只要对象的equals方法的比较操作所用到的信息没有被修改，那么对这同一个对象调用多次，hashCode方法都必须始终如一地返回同一个整数。在同一个应用程序的多次执行过程中，每次执行所返回的整数可以不一致。
* 如果两个对象根据equals()方法比较是相等的，那么调用这两个对象中任意一个对象的hashCode方法都必须产生同样的整数结果。
* 如果两个对象根据equals()方法比较是不相等的，那么调用这两个对象中任意一个对象的hashCode方法，则不一定要产生相同的整数结果。但是程序员应该知道，给不相等的对象产生截然不同的整数结果，有可能提高散列表的性能。