在Object类中存在方法：

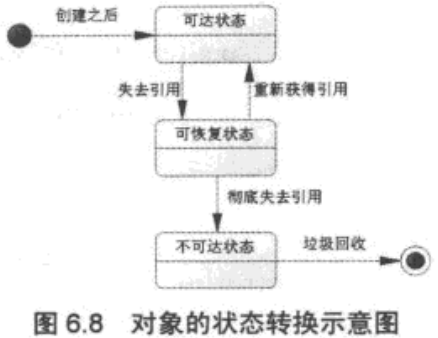


当创建对象、数组等引用类型实体时，系统都会在堆内存中为之分配一块内存区，对象就保存在这块内存区中，当这块内存区不再被任何应用变量引用时，这块内存就变成垃圾，等待垃圾回收机制进行回收，垃圾回收机制具有如下特征：

1. 垃圾回收机制只负责回收堆内存中的对象，不会回收任何物理资源（例如数据库连接、网络IO等资源）
2. 程序无法精确控制垃圾回收的运行，垃圾回收会在合适的时候进行，当回想永久性地失去应用后，系统就会在合适的时候回收它所占的内存
3. 在垃圾回收机制回收任何对象之前，总会先调用它的finalize方法，该方法可能使该对象重新复活（让一个引用变量重新引用该对象），从而导致垃圾回收机制取消回收

**对象在内存中的状态**

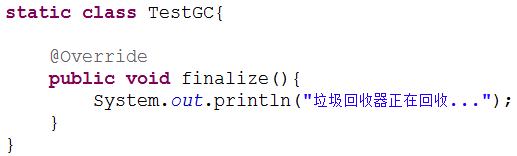
1. 可达状态：当一个对象呗创建后，若有一个以上的引用变量应用它，则这个对象在程序中处理可达状态
2. 可恢复状态：如果程序中某个对象不再有任何引用变量引用它，则进入可恢复状态。在这种状态下，系统的垃圾回收机制准备回收该对象所占用的内存，在回收该对象之前，系统会调用所有可恢复对象的finalize方法进行资源清理，如果系统在调用该方法时候重新让一个引用变量引用该对象，则这个对象会再次变为可达状态，否则该对象进入不可达状态
3. 不可达状态：当对象与所有引用变量的关系都被切断，且系统已经调用所有对象的饿finalize方法后依然没有使该对象变为可达状态，那么这个对象将永久性失去应用，，最后变为不可达状态。只有当一个对象处理不可达状态时，系统才会真正回收该对象所占用的资源。

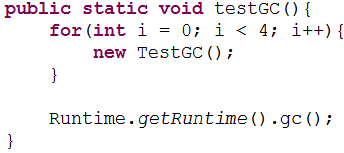


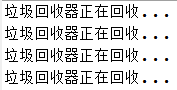
**强制垃圾回收**

程序无法精确控制java垃圾回收的时机，但我们依然可以强制系统进行垃圾回收---这种强制只是通知系统进行垃圾回收，但系统是否进行垃圾回收依然不确定。大部分时候，程序强制系统垃圾回收后总会有一些效果，强制系统垃圾回收有如下两个方法：

1. 调用System类的gc静态方法：System.gc()
2. 调用Runtime对象的gc方法：Runtime.getRuntime().gc()







注意：如果不调用Runtime.getRuntime().gc()则不会输出文本（已验证）

**关于finalize方法：**

1. 永远不要主要调用某个对象的finalize方法，该方法应交给垃圾回收器调用
2. Finalize方法何时被调用，是否被调用具有不确定性，不要把该方法当成一定会被执行的方法
3. 当JVM执行可恢复对象的finalize方法时，可能使该对象活系统中其他对象重新变为可达状态
4. 当JVM执行finalize方法时出现异常时，垃圾回收器不会报告异常，程序继续执行

除此之外，System和Runtime类中都提供了一个runFinalization防范，可以强制垃圾回收机制调用系统中可恢复对象的finalize方法。

**对象的软、弱、虚引用**

