Java更多的库谜题76：乒乓 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/138/2021_2022_Java_E6_9B_B4_E5_A4_9A_c97_138719.htm 下面的程序全部是由同步化（synchronized）的静态方法组成的。那么它会打印出什么呢？在你每次运行这段程序的时候，它都能保证会打印出相同的内容吗？ public class PingPong{ public static synchronized void main(String[] a){ Thread t = new Thread(){ public void run(){ pong(). } }. t.run(). System.out.print("Ping"). } static synchronized void pong(){ System.out.print("Pong"). } } 在多线程程序中，通常正确的观点是程序每次运行的结果都有可能发生变化，但是上面这段程序总是打印出相同的内容。在一个同步化的静态方法执行之前，它会获取与它的Class对象相关联的一个管程（monitor）锁[JLS 8.4.3.6]。所以在上面的程序中，主线程会在创建第二个线程之前获得与PingPong.class相关联的那个锁。只要主线程占有着这个锁，第二个线程就不可能执行同步化的静态方法。具体地讲，在main方法打印了Ping并且执行结束之后，第二个线程才能执行pong方法。只有当主线程放弃那个锁的时候，第二个线程才被允许获得这个锁并且打印Pong 。根据以上的分析，我们似乎可以确信这个程序应该总是打印PingPong。但是这里有一个小问题：当你尝试着运行这个程序的时候，你会发现它总是会打印PongPing。到底发生了什么呢？ 正如它看起来的那样奇怪，这段程序并不是一个多线程程序。不是一个多线程程序？怎么可能呢？它肯定会生成第二个线程啊。喔，对的，它确实是创建了第二个线程，但是它从未启动这个线程。相反地，主线程会调用那

个新的线程实例的run方法，这个run方法会在主线程中同步地运行。由于一个线程可以重复地获得某个相同的锁 [JLS 17.1] ，所以当run方法调用pong方法的时候，主线程就被允许再次获得与PingPong.class相关联的锁。pong方法打印了Pong并且返回到了run方法，而run方法又返回到main方法。最后，main方法打印了Ping，这就解释了我们看到的输出结果是怎么来的。 要订正这个程序很简单，只需将 t.run 改写成 t.start。这么做之后，这个程序就会如你所愿的总是打印出 PingPong 了。 这个教训很简单：当你想调用一个线程的start方法时要多加小心，别弄错成调用这个线程的run方法了。遗憾的是，这个错误实在是太普遍了，而且它可能很难被发现。或许这个谜题的教训应该是针对API的设计者的：如果一个线程没有一个公共的run方法，那么程序员就不可能意外地调用到它。Thread 类之所以有一个公共的run方法，是因为它实现了 Runnable 接口，但是这种方式并不是必须的。另外一种可选的设计方案是：使用组合（composition）来替代接口继承(interface inheritance)，让每个Thread实例都封装一个Runnable。正如谜题47中所讨论的，组合通常比继承更可取。这个谜题说明了上述的原则甚至对于接口继承也是适用的。