```
package com.multithread.wait;
public class MyThreadPrinter2 implements Runnable {

    private String name;
    private Object prev;
    private Object self;

    private MyThreadPrinter2(String name, Object prev, Object self) {
        this.name = name;
        this.prev = prev;
        this.self = self;
    }

    @Override
    public void run() {
        int count = 10;
        while (count > 0) {
            synchronized (prev) {
                synchronized (self) {
                    System.out.print(name);
                    count--;

                    self.notify();
                }
                try {
                    prev.wait();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }

        }
    }

    public static void main(String[] args) throws Exception {
```

```
        Object a = new Object();
        Object b = new Object();
        Object c = new Object();
        MyThreadPrinter2 pa = new MyThreadPrinter2("A", c, a);
        MyThreadPrinter2 pb = new MyThreadPrinter2("B", a, b);
        MyThreadPrinter2 pc = new MyThreadPrinter2("C", b, c);


        new Thread(pa).start();
        Thread.sleep(100);  //确保按顺序A、B、C执行
        new Thread(pb).start();
        Thread.sleep(100);
        new Thread(pc).start();
        Thread.sleep(100);
        }
}
```

输出结果：

ABCABCABCABCABCABCABCABCABCABC

先来解释一下其整体思路，从大的方向上来讲，该问题为三线程间的同步唤醒操作，主要的目的就是ThreadA->ThreadB->ThreadC->ThreadA循环执行三个线程。为了控制线程执行的顺序，那么就必须要确定唤醒、等待的顺序，所以每一个线程必须同时持有两个对象锁，才能继续执行。一个对象锁是prev，就是前一个线程所持有的对象锁。还有一个就是自身对象锁。主要的思想就是，为了控制执行的顺序，必须要先持有prev锁，也就前一个线程要释放自身对象锁，再去申请自身对象锁，两者兼备时打印，之后首先调用self.notify()释放自身对象锁，唤醒下一个等待线程，再调用prev.wait()释放prev对象锁，终止当前线程，等待循环结束后再次被唤醒。运行上述代码，可以发现三个线程循环打印ABC，共10次。程序运行的主要过程就是A线程最先运行，持有C,A对象锁，后释放A,C锁，唤醒B。线程B等待A锁，再申请B锁，后打印B，再释放B，A锁，唤醒C，线程C等待B锁，再申请C锁，后打印C，再释放C,B锁，唤醒A。看起来似乎没什么问题，但如果你仔细想一下，就会发现有问题，就是初始条件，三个线程按照A,B,C的顺序来启动，按照前面的思考，A唤醒B，B唤醒C，C再唤醒A。但是这种假设依赖于JVM中线程调度、执行的顺序。
