设计两个线程交替打印100以内的奇数和偶数

使用原子类控制数字

> 缺点：while（flag）的方式还是会不停地刷CPU，优化为wait notify方式，再优化为信号量的方式

H4

H5

```
public class PrintNumber1 {

    // 声明打印的数字值
    public static AtomicInteger number = new AtomicInteger(0);

    // 控制打印的临界节点
    public static AtomicInteger count = new AtomicInteger(0);

    public static volatile boolean flag = true;

    public static void main(String[] args) {
        new Thread(() -> {
            while (number.get() < 101){
                if (flag){

 System.out.println(Thread.currentThread().getName()+"--"+ number.getAndIncrement());
                    flag = !flag;
                }

 System.out.println(Thread.currentThread().getName()+ "抢到了锁"+count.getAndIncrement());
            }
        }, "threadA").start();
        new Thread(() -> {
            while (number.get() < 101){
                if (!flag){

 System.out.println(Thread.currentThread().getName()+"--"+ number.getAndIncrement());
                    flag = !flag;
                }

 System.out.println(Thread.currentThread().getName()+ "抢到了锁" + count.getAndIncrement());
            }
        },"threadB").start();
    }
}
```

线程之间的协作：通知机制

H5

> API解读：
>
> lock.notify()：唤醒某一个等待该lock的线程，当时当前线程不会立即释放锁，而是会在执行完synchronized代码块之后释放锁。
>
> lock.notifyAll()：唤醒所有等待在该lock上的线程，同上，当前线程不会立即释放锁，在执行完synchronized代码块之后释放锁。
>
> lock.wait()：调用该方法后当前线程立即释放锁，并且该线程必须由其他线程唤醒才能继续执行。
>
> 思路：
>
> 设置一个锁，两个线程抢占该锁获取执行机会，任何一个线程执行完唤醒其他线程，再次进入某线程若不满足条件直接当前线程睡眠历史释放锁，等待其他线程唤醒。

```
1  public class NotifyPrintNumber2 {
2      private static Object lock = new Object();
3      private static int i = 1;
4
5      public static void main(String[] args) {
6          Thread oddNumber = new Thread(() -> {
7              while (i < 100) {
8                  // 因为这里没有指定的通知机制，所以下次可能还是本线程拿到锁，再次执行，导致多次无效循环
9                  synchronized (lock) {
10                     if (i % 2 == 1) {
11
 System.out.println(Thread.currentThread().getName() + "---" + i++);
12                         // 唤醒等待在lock对象上其他线程，但是该线程不会马上释放锁，而是要在执行完synchronized代码块之后才释放锁
13                         // 这里需要注意：lock.notify()执行完后由于本线程并未休眠所以接下来该线程有可能继续获得调度权，此时不满足if条件，执行else中代码，本线程进入等待
14                         lock.notify();
15                         // 为什么不再这里直接调用wait()让本线程休眠反而让该线程下次继续有机会获得调度权徒增时间开销？目的：保证在i == 100时让该线程自动结束，因为有的题目会要求打印完毕线程自动停止
16                     } else {
17                         try {
18                             // 执行wait方法后，该线程会立即释放锁
19                             lock.wait();
20                         } catch (InterruptedException e) {
```

```
                            e.printStackTrace();
                        }
                    }
                }
            }
        }, "奇数");
        Thread even = new Thread(() -> {
            while (i < 100) {
                synchronized (lock) {
                    if (i % 2 == 0) {

 System.out.println(Thread.currentThread().getName()
+ "---" + i++);
                        lock.notify();
                    } else {
                        try {
                            lock.wait();
                        } catch (InterruptedException
e) {
                            e.printStackTrace();
                        }
                    }
                }
            }
        }, "偶数");

        oddNumber.start();
        even.start();
    }
}
```

拓展：设计三个线程交替打印字母a、b、c

> 可以借鉴上面实现思路，思路和上面完全一致，不过这里要注意，要唤醒的线程不止一个，需要调用notifyAll()方法，否则满足执行条件的线程一直处在睡眠中拿不到调度权致使线程死锁。

H5

```
public class NotifyPrintNumberAbc {
    private static Object lock = new Object();
    private static int i = 0;

    public static void main(String[] args) {
        Thread aThread = new Thread(() -> {
            while (i < 100) {
                // 因为这里没有指定的通知机制，所以下次可能
还是本线程拿到锁，再次执行，导致多次无效循环
                synchronized (lock) {
                    if (i % 3 == 0) {
```

```
                        i++;

 System.out.println(Thread.currentThread().getName()
+ "---" + "---a");
                        // 唤醒所有线程
                        lock.notifyAll();
                    } else {
                        try {
                            lock.wait();
                        } catch (InterruptedException
e) {
                            e.printStackTrace();
                        }
                    }

                }
            }
        }, "aThread");
        Thread bThread = new Thread(() -> {
            while (i < 100) {
                synchronized (lock) {
                    if (i % 3 == 1) {
                        i++;

 System.out.println(Thread.currentThread().getName()
+ "---" + "---b");
                        lock.notifyAll();
                    } else {
                        try {
                            lock.wait();
                        } catch (InterruptedException
e) {
                            e.printStackTrace();
                        }
                    }

                }
            }
        }, "bThread");
        Thread cThread = new Thread(() -> {
            while (i < 100) {
                synchronized (lock) {
                    if (i % 3 == 2) {
                        i++;

 System.out.println(Thread.currentThread().getName()
+ "---" + "---c");
```

```
                        lock.notifyAll();
                    } else {
                        try {
                            lock.wait();
                        } catch (InterruptedException
e) {
                            e.printStackTrace();
                        }
                    }

                }
            }
        }, "cThread");

        aThread.start();
        bThread.start();
        cThread.start();
    }
}

```

线程通知机制程序设计套路

H5

```
    // 线程1
    synchronized (lock) {
        if (conditionA) {
            doSomething();
            // 唤醒其他线程，或者notifyAll()唤醒其他所有等
待线程
            lock.notify();
         } else {
            try {
                // 第二次进入 不满足条件自动wait释放锁
                lock.wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    // 线程2
    synchronized (lock) {
        if (!conditionA) {
            doSomething();
            lock.notify();
         } else {
            try {
                lock.wait();
```

```
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }


```