Deadlock

PART1：代码及死锁截图

- **classA和classB:**

```
class A {
    synchronized void methodA(B b) {
        b.last();
    }

    synchronized void last() {
        System.out.println("Inside A.last()");
    }

}

class B {
    synchronized void methodB(A a) {
        a.last();
    }

    synchronized void last() {
        System.out.println("Inside B.last()");
    }

}
```

可以看到，在类A中调用了类B的方法；在类B中调用了类A的方法。 并且两个类都有关键字synchronized修饰：

1. 当它用来修饰一个方法或者一个代码块的时候，能够保证在同一时刻最多只有一个线程执行该段代码；
2. 当一个线程访问object的一个synchronized同步代码块或同步方法时，其他线程对object中所有其它synchronized同步代码块或同步方法的访问将被阻塞。

- **主类代码：**

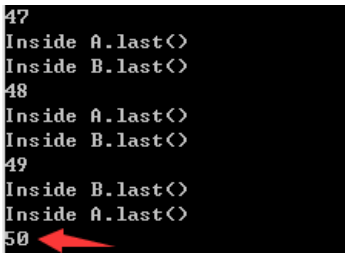
```java
//关于runnable，你们安卓开发可能已经或将用到
//就是一个线程，会在后台默默的运行
//每次调度到它执行的时候，它就运行run()中的语句
class Deadlock implements Runnable{
    A a=new A();
    B b=new B();

    //构造函数
    Deadlock(){
        Thread t = new Thread(this);
        int count = 20000;

        t.start();//线程t开始，
        while(count-->0);//等待20000
        a.methodA(b);
    }
    //runnable运行时调用的方法
    public void run(){
        b.methodB(a);
    }
    public static void main(String args[]){
        new Deadlock();
    }
}
```

- **执行文件代码：**

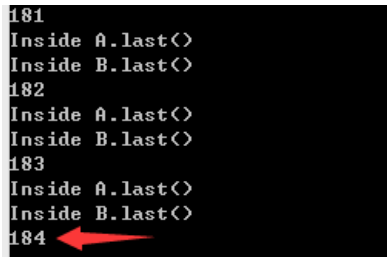
```
Deadlock.bat - 记事本
文件(F)  编辑(E)  格式(O)  查看(V)
cd /d %~dp0
@echo off
:start
set /a var+=1
echo %var%
java Deadlock
if %var% leq 1000 GOTO start
pause
```

- **经过多次的运行测试，可以发现几乎每次死锁停住的次数都不一样，下面贴出几张运行停止时的截图：**

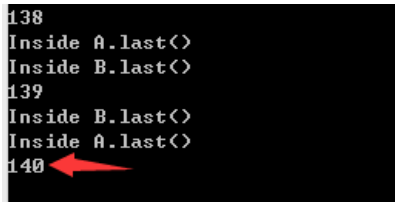
1. 死锁在第50次：



2. 死锁在第184次：



3. 死锁在第140次：



PART2：死锁的条件

死锁就是两个或者多个进程，互相请求对方占有的资源。

死锁产生的条件：

1. 互斥条件：进程要求对所分配的资源（如打印机）进行排他性控制，即在一段时间内某资源仅为一个进程所占有。此时若有其他进程请求该资源，则请求进程只能等待；
2. 不剥夺条件：进程所获得的资源在未使用完毕之前，不能被其他进程强行夺走，即只能由获得该资源的进程自己来释放（只能是主动释放)；
3. 请求和保持条件：进程已经保持了至少一个资源，但又提出了新的资源请求，而该资源已被其他进程占有，此时请求进程被阻塞，但对自己已获得的资源保持不放；
4. 循环等待条件：存在一种进程资源的循环等待链，链中每一个进程已获得的资源同时被链中下一个进程所请求。即存在一个处于等待状态的进程集合{Pl, P2, ..., pn}，其中Pi等待的资源被P(i+1)占有（i=0, 1, ..., n-1)，Pn等待的资源被P0占有。

PART3：对上述程序产生死锁的解释

当t.start()之后，以下两件事同时发生：

1. 构造函数中，当等待时间到了之后，会执行“a.methodA(b)”；
2. 当t.start()之后，线程t就被插入到调度队列里面，当调度到它的时候，就会跑run()函数里面的代码，而run函数里面有“b.methodB(a)”。

因为a.last()和b.last()都有关键字synchronized，所以在同一时刻最多只有一个线程执行其中一个进程。那么假设在某一时刻，a必须等待b执行完释放资源才能执行，而b也必须等待a执行完释放资源才能继续，两个进程都处于等待状态，那么就产生了死锁，无法向下继续执行。