一. 实验题的

使用java编程实现死锁的模拟。

二. 实验目的

1. 了解如何进行java环境的配置，以及如何使用命令行进行编译运行java。
2. 复习死锁发生的4个条件。
3. 分析已有的模拟死锁代码，通过改变参数了解死锁的发生时机。

三. 实验原理

1. **死锁发生的必要条件**
    1. **互斥条件**：一个资源每次只能被一个进程使用
    2. **请求与保持条件**：一个进程因请求资源而阻塞时，对已获得的资源保持不释放
    3. **不剥夺条件**:进程已获得的资源，在未使用完之前，不能强行剥夺
    4. **循环等待条件** :若干进程之间形成一种头尾相接的循环等待资源关系
2. **Java关键字 synchronized**
    - 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。
    - 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

四. 代码实现

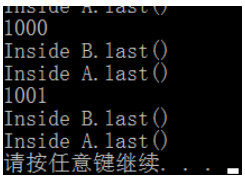
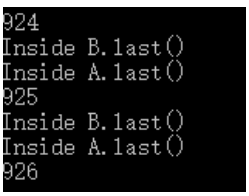
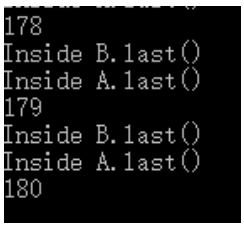
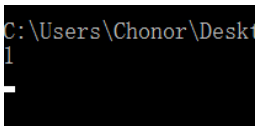
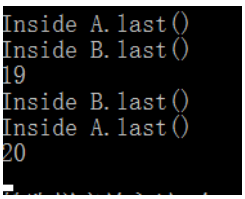
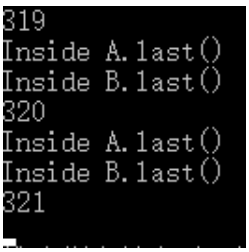
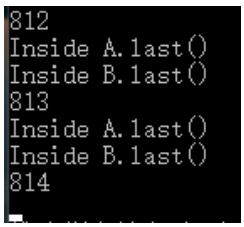
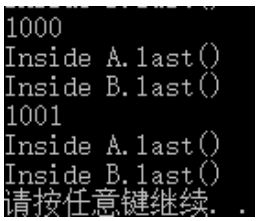
```
class A{
    synchronized void methodA(B b){
        b.last();
    }
    synchronized void last(){
        System.out.println("Inside A.last()");
    }
}
```

```
class B{
    synchronized void methodB(A a){ a.last(); }
    synchronized void last(){ System.out.println("Inside B.last()");}
}
class Deadlock implements Runnable{
    A a=new A();
    B b=new B();
    Deadlock(){
        Thread t = new Thread(this);
        int count = 14500;
        t.start();
```

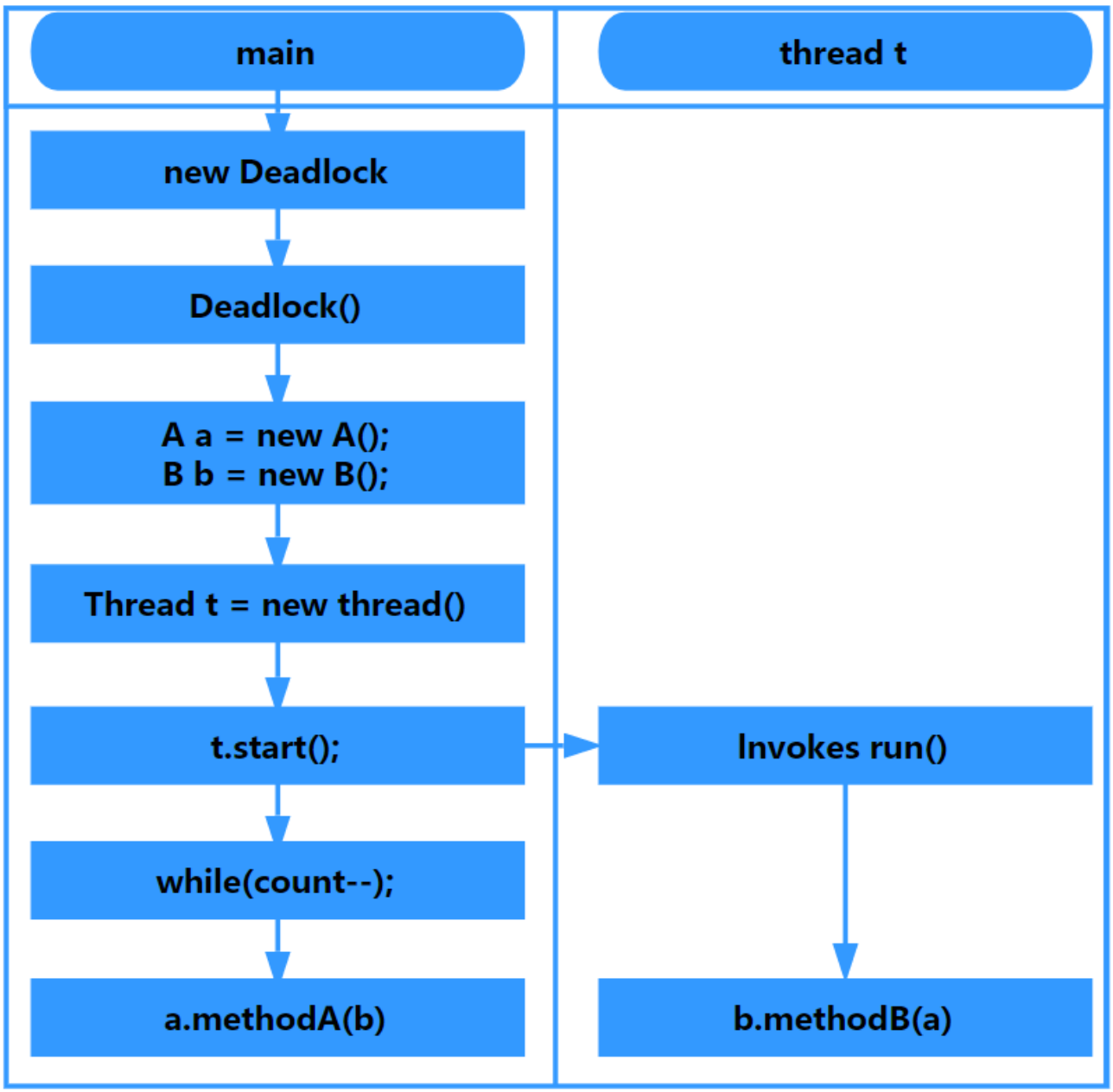
```
            while(count-->0);
            a.methodA(b);
        }
        public void run(){
            b.methodB(a);
        }
        public static void main(String args[]){
            new Deadlock();
        }
    }
```

五. 结果分析

改变不同Count的测试结果如下：

死锁发生时间	Count=1000 1000次内不发生死锁	Count=8000 1000次内基本不发生死锁	Count=10000 100+后发生死锁	Count=14500 大概率个位数内发生死锁
				
死锁发生时间	**Count=15000 20以内大概率发生死锁**	**Count=20000 300次左右死锁**	**Count=30000 1000次内较少发生死锁**	**Count=40000 1000次以内不发生死锁**
				

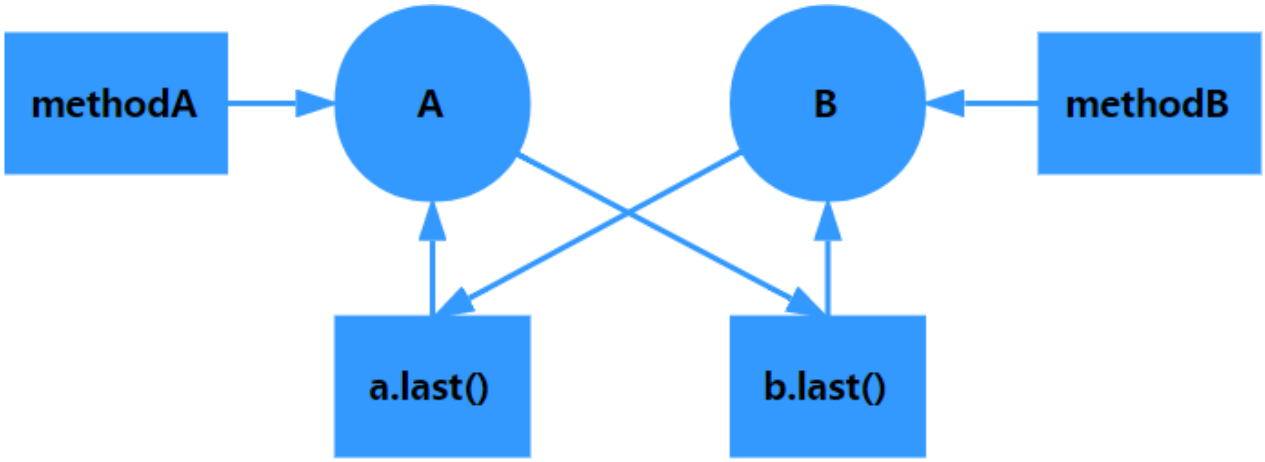
代码运行流程如下：



死锁原因分析：

1. 互斥条件：在两个class A和B中我们看到其两个函数都被synchronized修饰，此时只有唯一个一线程能够访问一个类中的这两个函数。所以此处构成了互斥条件。
2. 请求与保持条件：代码中如果请求不到资源，那么自身资源不会释放
3. 不剥夺条件：代码中不会杀死线程释放资源。
4. 循环等待：这个条件代码中的类相互访问造成。

代码中实际的死锁情况如图：



因为class中的method和last都是用synchronized修饰所以可以认为他们一体的资源

线程A(线程main)调用class A中的methodA，线程B(线程t)调用class B中的methodB，此时这两个函数分别要去访问b.last()和a.last()。此时就构成了死锁，因为A没有获得b.last()所以没法释放classA也就是a.last(),此时B也因为没有获得a.last()所以无法释放classB也就是b.last()。此时就造成了死锁。这段代码想要造成死锁的关键就在于如何让main线程访问classA的同时t线程访问classB，之后才能造成上图发生死锁情况。

我们修改一下代码如下图：

```
Deadlock(){
    Thread t = new Thread(this);
    int count = 1500;

    t.start();
    while(count-->0){
        System.out.println(count);
    }
    a.methodA(b);
}

public void run(){
    System.out.println("t strat");
    b.methodB(a);
}
```

```
选择C:\
1397
1396
1395
1394
1393
1392
1391
1390
1389
1388
1387
1386
1385
1384
1383
1382
1381
1380
1379
1378
1377
1376
1375
1374
1373
t strat
1372
1371
1370
1369
```

我们可以发现实际中t开始运行的时间和t.start()之间还是有一定的时间差的。

从上面的的改变count测试中，可以看出如果现在延时过小，那么就会是classA 先运行并调用了classB中的输出，如果延时过大那么此时classB将先运行调出classA中的输出，所以我们可以根据其输出顺序来调整到合理的延时。

那么代码中调整Count的大小就是在调整时间差，造成他们的同时访问的情况，至于.bat文件是为了多次运行，如果真正把握好时间差其实一次就能死锁，但是为什么需要.bat，就是因为whlie(count--)和t开始运行的时间之间并不是固定的，他们受到当前cpu负载之类的影响，所以需要多次运行实现让其刚好符合，也就是要寻找while(count--)和t开始运行的时间的平衡点。

六. 实验感想

这次实验主要就就是复习之前OS中学到死锁条件，只不过是用了java来进行模拟，之前电脑上也就有java环境，所以实验过程中没有什么困难，就是之前没有学过java，所以分析java代码比较麻烦点，还要去看看java一些代码的意思。

如果觉得有限么诡异的地方请看pdf这里已经尽量少修改css，但是不同编辑器不同没办法做到。

> 以下都是吐槽：
>
> markdown用来写文本加代码是挺好用的，但是对于日常的段落的是很不友好，段首缩进要么使用html要么全角空格，Tab空格无视(这是html的问题也不能怪他)。但是对于图片排版非常不友好，完全不能再文字中嵌入图片，除非我强行使用全html（当然谁也看不到效果除非我输出到html或者pdf），当然我见过的所谓各种推崇的语言吧，就没有几个能把图片排版做好的（说好的都是吹，都是简单的居中）。其次是markdown对于分页几乎没有任何处理能力，完全只能靠手动，在word上我插入代码（有高亮）还会自动分开，markdown直接无脑下一页。