Question Description

A class named as Process-synchronization, it have a member
private int money=0;
Task:

1. Constructing ten threads, the odd ones add the money one per times, whereas the even one reduce one per times
2. Using $P()$ and $V()$ to implement synchronization to assure the money is not negative and more than 10.
3. Each operation print the Thread-ID.
4. (additional mark) Sequentially run.

Submit requirement:

1. PDF is expected.
2. Must attach the console result of running.

Solution-1

If each thread run once

Useage: `thread.join()`

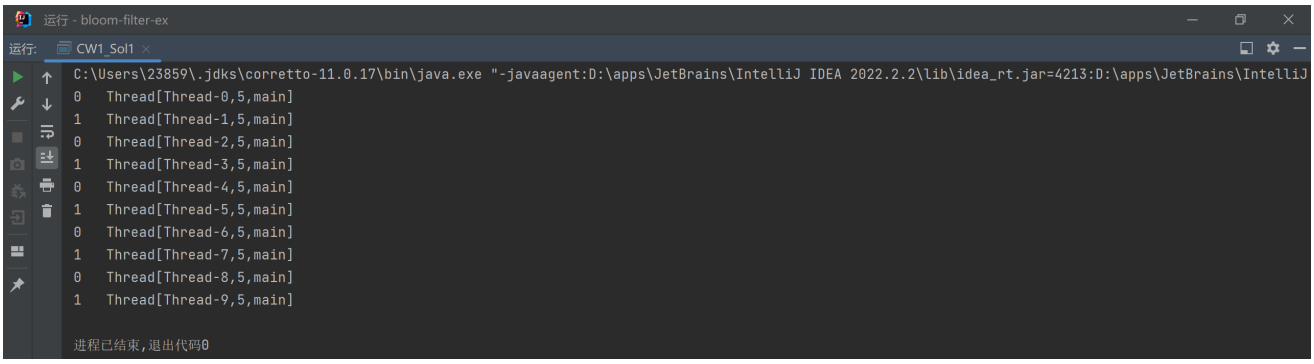
Code

```
package com.example.uc;

public class CW1_Sol1 {
    static int money = 0;
    public static int key;

    public static void main(String[] args) throws InterruptedException {
        // 每个线程执行100次
        for (int i = 0; i < 1; i++) {
            Thread pre = new Thread(new MyRunnable(null, -1));
            pre.start();
            for(int j=1;j<10;j++){
                Thread thread;
                if((j % 2) == 0){
                    thread = new Thread(new MyRunnable(pre, -1));
```

```
                }else{
                    thread = new Thread(new MyRunnable(pre, 1));
                }
                thread.start();
                pre = thread;
            }
            Thread.sleep(50);
        }
    }
    static class MyRunnable implements Runnable{
        private final Thread preThread;
        private final int ch;
        public MyRunnable(Thread thread, int ch){
            preThread = thread;
            this.ch =ch;
        }
        @Override
        public void run() {
            // TODO
            if(preThread != null){
                try{
                    P();
                    V();
                }
                catch (Exception e){
                    e.printStackTrace();
                }
            }else{
                V();
            }
        }

        /**
         * segment push         */        private void V() {
            money += ch;
            if(money<0 || money>10){money-=ch;}
            System.out.print(money);
            System.out.print("\t");
            System.out.println(Thread.currentThread());
        }

        /**
         * Segment wait         * @throws InterruptedException
```

```
        */        private void P() throws InterruptedException {
            preThread.join(); // 前面的执行玩才能执行后面的函数
        }
    }
}
```

console result



Solution-2

If Each thread run many times.

Useage: `LockSupport.park()` and `LockSupport.unpark(Thread thread)`

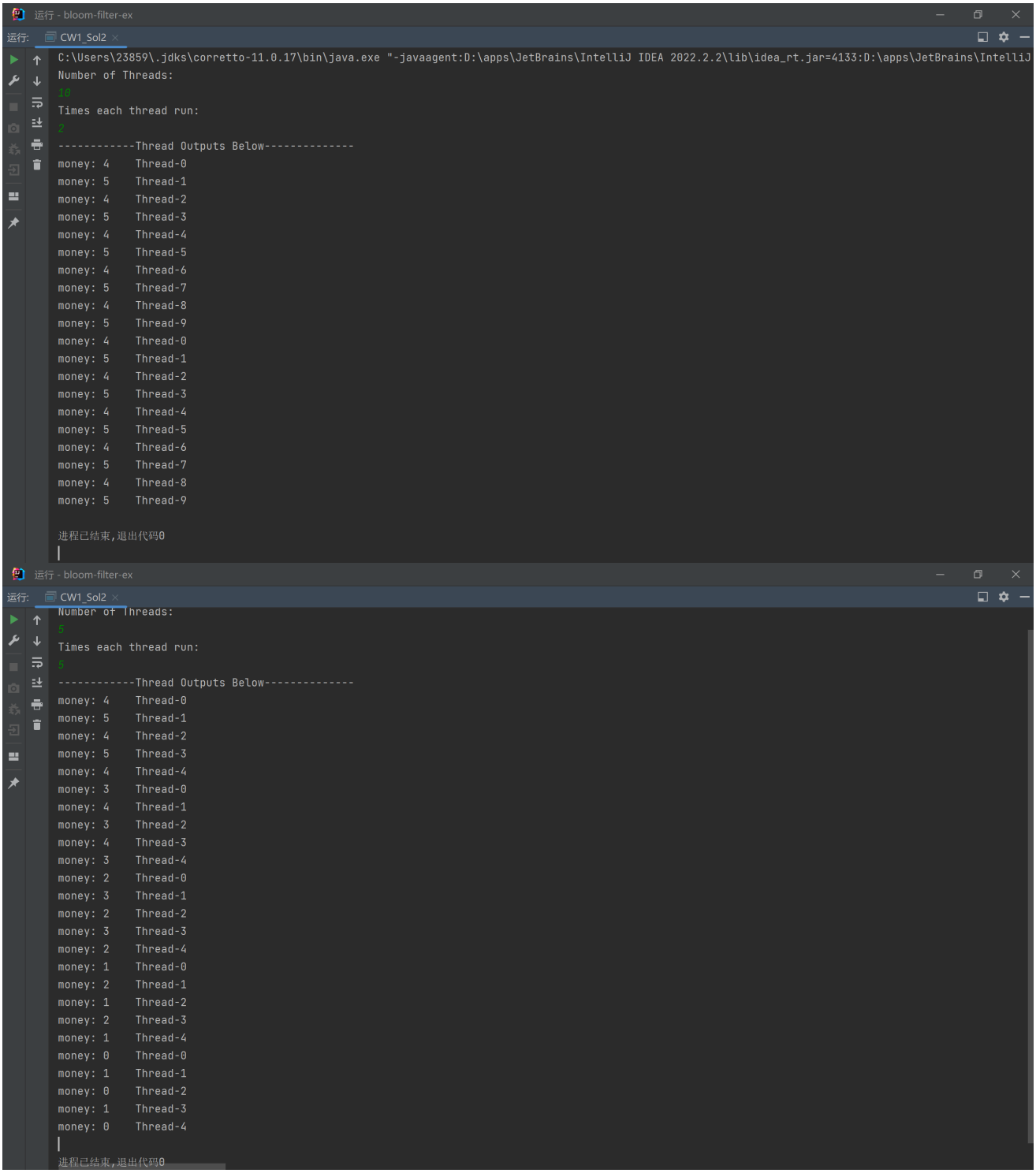
Code

```
package com.github.hyperv0id.bf;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import java.util.concurrent.locks.LockSupport;

public class CW1_Sol2 {
    static int money = 5;
    static int times;
    static List<Thread> threadPool;
    public static void main(String[] args) throws InterruptedException {
        System.out.println("Number of Threads: ");
        Scanner scanner = new Scanner(System.in);
        // 线程数
        int t_cnt = scanner.nextInt();
```

```
        System.out.println("Times each thread run: ");
        // 每个线程执行次数
        times = scanner.nextInt();
        threadPool = new ArrayList<>();

        // 每个线程执行100次
        for (int i = 0; i < t_cnt; i++) {
            int cn = i%2==0?-1:1;
            threadPool.add(new Thread(new MyRunnable(i, cn)));
        }
        for (Thread thread: threadPool) {
            thread.start();
        }
        // 解锁第一个线程
        LockSupport.unpark(threadPool.get(0));
        System.out.println("------------Thread Outputs Below------------
--");
    }
    static class MyRunnable implements Runnable{
        private final int ch;
        private final int id;
        public MyRunnable(int id, int ch){
            this.id = id;
            this.ch =ch;
        }
        @Override
        public void run() {
            for (int i = 0; i < times; i++) {
                // 阻塞自己
                P();
                // 执行，解锁下一个线程
                V();
            }
        }

        /**
         * segment push
         */
        private void V() {
            // 自己执行
            money += ch;
            if(money<0 || money>10){
                money -= ch;
```

```
                System.out.println(Thread.currentThread() + " Pass");
            }
            System.out.println("money: " + money + "\t" + "Thread-" +
id);
            // 解锁下一个线程
            LockSupport.unpark(threadPool.get((id+1) %
threadPool.size()));
        }

        /**
         * Segment wait
         */
        private void P() {
            // 阻塞自己
            LockSupport.park();
        }
    }
}
```

console result


运行 - bloom-filter-ex
运行:
CW1_Sol2
C:\Users\23859\.jdks\corretto-11.0.17\bin\java.exe "-javaagent:D:\apps\JetBrains\IntelliJ IDEA 2022.2.2\lib\idea_rt.jar=4133:D:\apps\JetBrains\IntelliJ
Number of Threads:
10
Times each thread run:
2
-----------Thread Outputs Below--------------
money: 4 Thread-0
money: 5 Thread-1
money: 4 Thread-2
money: 5 Thread-3
money: 4 Thread-4
money: 5 Thread-5
money: 4 Thread-6
money: 5 Thread-7
money: 4 Thread-8
money: 5 Thread-9
money: 4 Thread-0
money: 5 Thread-1
money: 4 Thread-2
money: 5 Thread-3
money: 4 Thread-4
money: 5 Thread-5
money: 4 Thread-6
money: 5 Thread-7
money: 4 Thread-8
money: 5 Thread-9
进程已结束,退出代码0
运行 - bloom-filter-ex
运行:
CW1_Sol2
Number of Threads:
5
Times each thread run:
5
-----------Thread Outputs Below--------------
money: 4 Thread-0
money: 5 Thread-1
money: 4 Thread-2
money: 5 Thread-3
money: 4 Thread-4
money: 3 Thread-0
money: 4 Thread-1
money: 3 Thread-2
money: 4 Thread-3
money: 3 Thread-4
money: 2 Thread-0
money: 3 Thread-1
money: 2 Thread-2
money: 3 Thread-3
money: 2 Thread-4
money: 1 Thread-0
money: 2 Thread-1
money: 1 Thread-2
money: 2 Thread-3
money: 1 Thread-4
money: 0 Thread-0
money: 1 Thread-1
money: 0 Thread-2
money: 1 Thread-3
money: 0 Thread-4
进程已结束,退出代码0