

**并 行 计 算**

课程实验报告

实验名称：Java并发程序设计

任课教师：徐悦甡

课程班级：15级 云计算方向

学号姓名：15130130273 石明皓

提交日期：2018年5月16日

**软件学院本科生《并行计算》**

**课程实验报告**

1. **实验名称**

第1次实验：Java并发程序设计

1. **实验日期**

2018年5月16日 软件学院实验室G346

1. **实验学生**

15130130273 石明皓

1. **实验目的**

本次实验通过展示2个简易的多线程应用场景，编写了实现其业务逻辑的Java多线程程序，学习Runnable接口、Thread类的使用方法以及多线程同步锁的应用，理解了基本Java多线程编程思想，掌握了Java基础并发程序设计的方法。

1. **实验内容**

题目一：

四个售票窗口同时出售30张电影票。

题目二：

两个人张三与李四，通过一个同一个账户，张三在柜台取钱，李四在 ATM机取钱。

1. **程序思路、结构**

题目一：

电影票的票数使用同一个静态值，不同售票窗口对象操作的均为该静态变量；

为了不出现不同柜台卖出同一张票的情况，要用到Java多线程同步锁，需要使用ReentrantLock()创建一个Lock对象；

创建售票窗口类BoxOffice实现Runnable接口，重写run()方法，在run()方法中进行售票操作；

进行售票操作时使用同步锁，即：任意一个窗口正在出售某张票时，其他窗口必须先等待该窗口卖出这张票，完成其完整售票流程。

每个窗口对象的run()方法循环执行，对某张电影票操作前lock()加锁，操作完成后unlock()解锁；

创建4个线程模拟4个售票窗口，start()启动线程。

题目二：

创建一个Bank类（用于存放账户金额，并提供柜台取钱和ATM机取钱两种方法）、一个张三类（代表在柜台取钱）、一个李四类（代表在ATM机取钱）；

Bank类中提供的两种取钱方法，均有synchronized关键字，说明方法是同步的；

两种取钱方法对同一账户进行取钱操作，故创建同一个静态值作为账户金额；

张三类的构造函数和李四类的构造函数传入的是同一个Bank类的对象，说明它们操作的是同一账户；

张三类和李四类均继承Thread类，重写了其run()方法，循环执行取钱操作；

创建2个线程模拟张三和李四的同时取款，start()启动线程。

1. **程序代码**

题目一：

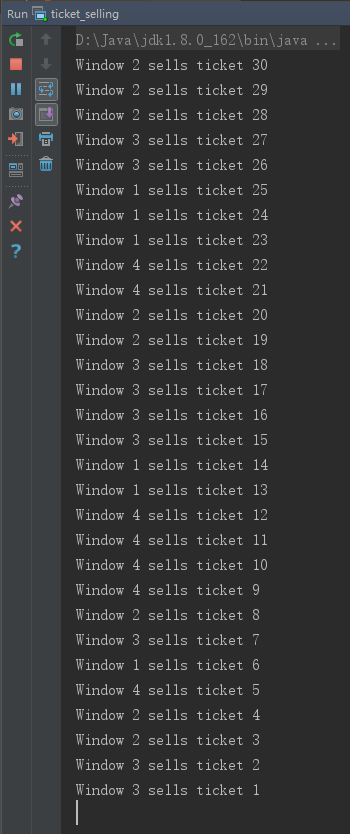
import java.util.concurrent.locks.Lock;  
import java.util.concurrent.locks.ReentrantLock;  
class BoxOffice implements Runnable {  
 private static int *ticket* = 30; //共30张电影票  
 private Lock locker = new ReentrantLock();  
 public void run() {  
 while (true) {  
 locker.lock(); //加锁  
 if (*ticket* > 0) {  
 try {  
 Thread.*sleep*(100); //每次先休眠100ms  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 System.*out*.println(Thread.*currentThread*().getName() + " sells ticket " + (*ticket*));  
 *ticket*--; //减去已卖出的票  
 }  
 locker.unlock(); //解锁  
 }  
 }  
}  
public class ticket\_selling {  
 public static void main(String[] args) {  
 BoxOffice seller = new BoxOffice();  
 Thread t1 = new Thread(seller, "Window 1");  
 Thread t2 = new Thread(seller, "Window 2");  
 Thread t3 = new Thread(seller, "Window 3");  
 Thread t4 = new Thread(seller, "Window 4");  
 t1.start();  
 t2.start();  
 t3.start();  
 t4.start();  
 }  
}

题目二：

class Bank {  
 static int *money* = 10000;  
 public synchronized void counter\_withdraw(int money1) {  
 Bank.*money* = Bank.*money* - money1;  
 System.*out*.println("ZhangSan gets $" + money1 + " from counter, and remains $" + Bank.*money* + " in the account");  
 }  
 public synchronized void atm\_withdraw(int money2) {  
 Bank.*money* = Bank.*money* - money2;  
 System.*out*.println("LiSi gets $" + money2 + " from ATM, and remains $" + Bank.*money* + " in the account");  
 }  
}  
class ZhangSan extends Thread {  
 Bank bank;  
 public ZhangSan(Bank same\_account) {  
 this.bank = same\_account;  
 }  
 public void run() {  
 while (Bank.*money* >= 100) {  
 bank.counter\_withdraw(100);  
 try {  
 *sleep*(50);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}  
class LiSi extends Thread {  
 Bank bank;  
 public LiSi(Bank same\_account) {  
 this.bank = same\_account;  
 }  
 public void run() {  
 while (Bank.*money* >= 300) {  
 bank.atm\_withdraw(300);  
 try {  
 *sleep*(100);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}  
public class withdraw\_money {  
 public static void main(String[] args) {  
 Bank account = new Bank();  
 ZhangSan p1 = new ZhangSan(account);  
 LiSi p2 = new LiSi(account);  
 p1.start();  
 p2.start();  
 }  
}

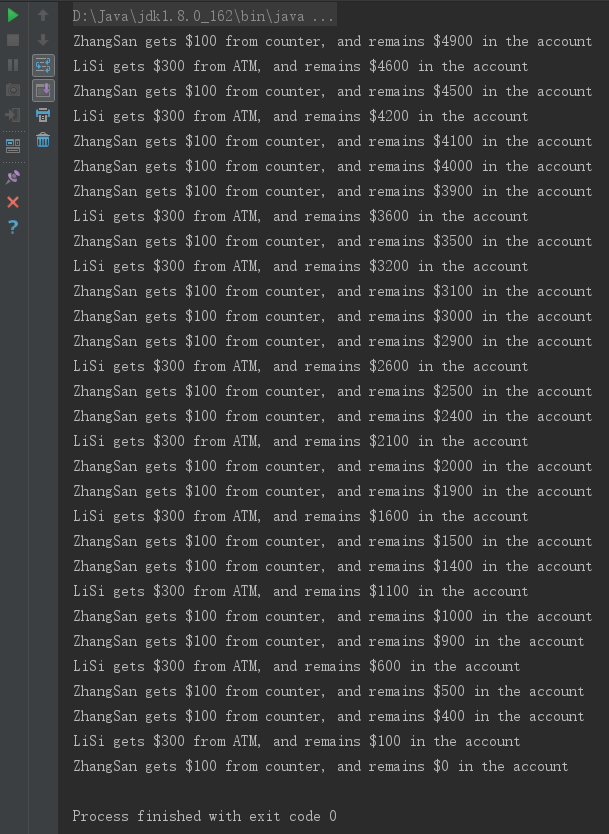
1. **实验结果**

题目一：



从程序运行结果可知，4个售票窗口线程同时卖票，不同窗口卖出不同的票，没有出现卖出同一张票的情况。

题目二：



从程序运行结果可知，张三和李四同时调用Bank类的不同方法对同一账户进行取款，由于取款过程是同步化的，因此没有出现账户金额的一致性问题。

1. **总结建议**

经过本次实验，我对并发程序设计的原理有了基本的认识；对并发程序设计时需要注意的同步问题、一致性问题有了进一步了解；同时更加熟悉Java语言在多线程编程方面的编程模式，进而提高了个人的Java编程能力。