编号：

2020 - 2021学年第 二 学期



**实 验 报 告**

实验课程名称 软件架构与应用开发

专 业 班 级 计算机1803

学 生 姓 号 31801150

学 生 姓 名 张帅

实验指导教师 彭彬

一. 实验目的和要求

**实验要求：**

a.对照课堂讲解读懂代码结构

b.根据代码中所有的Todo要求完成整个代码工程

本实验要求每个同学单独完，本实验要求上交实验报告（word文件），每人一份，请写明白对每个Todo实现所用的方法。

二. 实验内容

1.main.java:

//*TODO 实现从配置文件 bank.properties加载counter.count配置项*

int counterCount = 1;  
Properties properties = new Properties();  
BufferedReader bufferedReader = new BufferedReader(new FileReader( System.*getProperty*("user.dir")+"\\src\\main\\java\\cn\\edu\\zucc\\pb\\bankdispacher\\bank.properties"));  
properties.load(bufferedReader);  
counterCount=Integer.*valueOf*(properties.getProperty("counter.count"));

1.使用Properties类获取配置文件。

2.使用BufferedReader类获取文件信息，其中System.getProperty("user.dir")用于获取当前文件的目录，定位到配置文件的位置。

3.使用load方法加载文件后，使用properties.getProperty("counter.count")，获取配置文件中counter.count的信息，即为开启窗口的数量。

//开始启动用户产生仿真  
Thread userGenerator = new Thread("UserGenerator"){  
 @Override  
 public void run() {  
 UserGenerator generator=new UserGenerator();  
 generator.start(100);  
 }  
};  
userGenerator.start();

1.将产生用户的仿真线程修改为如上的内容

2.启动UserGenerator线程，传入参数100表示生产至100个用户之后结束线程。

*/\*\*TODO 如果这里什么都不做，那么主线程就退出了，上面启动的所有仿真线程还活着  
 这个是不好的设计，请修改这个main方法，是否可以做到可以控制所有的仿真线程在  
 仿真运行比如100个用户后可以恰当的退出  
 \*/*try {  
 userGenerator.join();  
 *exit*=false;  
 System.*out*.println("100人获得服务，今日服务结束了，银行关门！");  
} catch (InterruptedException e) {  
 e.printStackTrace();  
}

1.使用userGenerator.join()等待生成用户线程结束

2.使用exit为标志位，用于判定银行窗口是否继续工作，将exit置位false，所有的窗口线程进结束运行。

3.打印银行关门的提示。

2.UserGenerator.java:

*/\*\*  
 \* 模拟产生n个用户  
 \** ***@param*** *nCount 产生的用户数，参数小于零一直产生  
 \*/*public void start(int nCount){  
 boolean bExit = false;  
 int index = 0;  
 while(!bExit){  
  
 //等待一点时间  
 waitSomeTime();  
  
  
 //产生一个用户  
 //*TODO 修改仿真用户到达的比例，按照Vip:Enterprise:Private=1:2:7方式产生* switch (RandomUtils.*nextInt*(1, 11)){  
 case 1:  
 UserEventQue.*getInstance*().nextUserArrive(EUserCategory.*VIP*);  
 break;  
 case 2|3:  
UserEventQue.*getInstance*().nextUserArrive(EUserCategory.*ENTERPRISE*);  
 break;  
 default:  
UserEventQue.*getInstance*().nextUserArrive(EUserCategory.*PRIVATE*);  
 break;  
 }  
 index++;  
  
 if(nCount > 0 && index >= nCount){  
 bExit = true;  
 }  
 }  
}

1.将bExit修改为false，窗口可以正常进入循环。

2. 使用andomUtils.nextInt(1, 11)随机生成一位1-10之间的数字，如果是1则视为VIP用户，如果为2或者3则视为ENTERPRISE用户，否则为PRIVATE用户，达到1：2：7的比例。

3. BankCounter.java

public void start() throws IOException {  
  
  
 //while(true)柜台一直不下班  
 while(Main.*exit*){  
 UserEvent userEvent = UserEventQue.*getInstance*().dispatchUser();  
 if(userEvent!=null){  
 //等待一点时间,模拟在办理或者等待  
 //*TODO 实现不同业务类型等待不同时间* EUserCategory type=userEvent.getCategory();  
  
  
 if(type==EUserCategory.*VIP*) {  
 waitSomeTime();  
 }  
 else if(type==EUserCategory.*PRIVATE*){  
 for(int i=0;i<2;i++){  
 waitSomeTime();  
 }  
 } else {  
 for(int i=0;i<3;i++){  
 waitSomeTime();  
 }  
 }  
  
 System.*out*.println("用户"+userEvent.getSeq()+"离开了,离开时间："+new Date());  
 }  
  
 }

1.读取Main.java中的标志位Main.exit用于判定是否退出循环。

2. 通过EUserCategory type=userEvent.getCategory()获取读取到的用户的类型，根据不同类型获取循环调用waitSomeTime()函数1/2/3次到达等待不同时间的目的

3.打印用户离开银行的信息。

4. UserEventQue.java

//保证线程安全；*TODO: 思考，这里为什么需要保证线程安全*synchronized (UserEventQue.class){  
 event.setSeq(*SEQ*++);  
 System.*out*.println("用户"+event.getSeq()+"来了,服务类型:"+event.getCategory()+",达到时间:"+event.getArriveTime());  
 events.add(event);  
}

1.思考，此处的events和SEQ都是临界资源（公共资源），如果不使用synchronized则可能产生多个线程同时访问这些临界资源的情况，产生脏数据等错误。

2.新增了打印用户到达银行的提示

public UserEvent dispatchUser() throws IOException {  
 //使用工厂类动态加载算法实现类  
 IUserSelector selector = null;  
 try {  
 selector = UserSelectorFactory.*getSelector*();  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 } catch (IllegalAccessException e) {  
 e.printStackTrace();  
 } catch (InstantiationException e) {  
 e.printStackTrace();  
 }  
  
 if(selector != null){  
 //保证线程安全；*TODO: 思考，这里为什么需要保证线程安全* synchronized (UserEventQue.class){  
 UserEvent event = selector.select(events);  
 //*TODO 移除队列中选中的event* if(event!=null)  
 {  
 System.*out*.println("用户"+event.getSeq()+"得到了服务：服务类型："+event.getCategory().toString()+",服务时间："+new Date());  
 events.remove(event);  
 return event;  
 }  
  
 }  
 }  
  
 return null;  
}

1.思考，此处的events是临界资源（公共资源），如果不使用synchronized则可能产生多个线程同时对临界资源进行删除的情况，产生错误。

2.移除队列选中的event时， 首先判断选中的event是否为null，之后使用remove()函数移出events列表中指定的event对象。同时提示等待队列中的指定用户获得服务，离开等待队列。

5.UserSelectorFactory.java

public class UserSelectorFactory {  
 public static IUserSelector getSelector() throws IOException, ClassNotFoundException, IllegalAccessException, InstantiationException {  
 //*TODO 实现从配置文件 bank.properties加载selector配置项* String selectorName = "";  
 Properties properties = new Properties();  
 //使用InPutStream读取配置文件  
 BufferedReader bufferedReader = new BufferedReader(new FileReader( System.*getProperty*("user.dir")+"\\src\\main\\java\\cn\\edu\\zucc\\pb\\bankdispacher\\bank.properties"));  
 properties.load(bufferedReader);  
 selectorName = properties.getProperty("selector");  
  
  
 //*TODO 通过反射接口从上面的selectorName进行实现类创建* IUserSelector selector = (IUserSelector) Class.*forName*(selectorName).newInstance();  
  
 return selector;  
 }  
}

1.同读取窗口数量，使用Properties类获取配置文件并通过properties.getProperty("selector")获取配置文件中使用的算法信息。

2.使用Class.forName(selectorName)通过在配置文件中获得的selectorName，获取指定的实现类（对应不同的算法），通过反射接口进行实现类创建。

6. TimeFirstSelector.java

public class TimeFirstSelector implements IUserSelector {  
 public UserEvent select(List<UserEvent> events) {  
 //*TODO 实现到达时间优先算法的用户选择器* if(events.size()==0)  
 return null;  
  
 return events.get(0);  
 }  
}

1.首先判定events队列中是否为空，为空则返回null，窗口将不进行处理

2.如果队列中不为空，则返回队列的第一位用户，即最先达到的用户。

7. VipFirstSelector.java

public UserEvent select(List<UserEvent> events) {  
  
  
  
 int i;  
 for (i = 0; i < events.size(); i++) {  
 if (events.get(i).getCategory() == EUserCategory.*VIP*) {  
 return events.get(i);  
 }  
 }  
 if (i == events.size()&&events.size()!=0) {  
 return events.get(0);  
 }else{  
 return null;  
 }  
  
}

1.首先寻找events队列中的第一位VIP用户，如果存在即返回该用户。

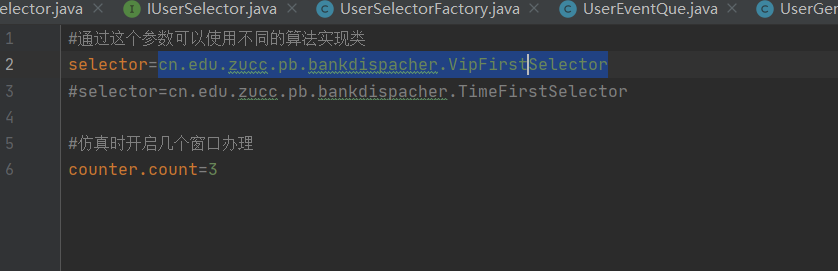
2.如果队列中不存在VIP用户，在第一位用户（最早达到的用户）获得服务。

3.如果队列为空则返回null，窗口将不做处理。

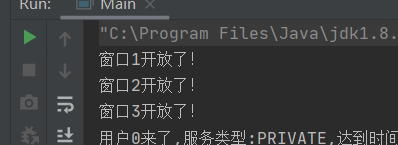
三. 实验结果

尝试一：

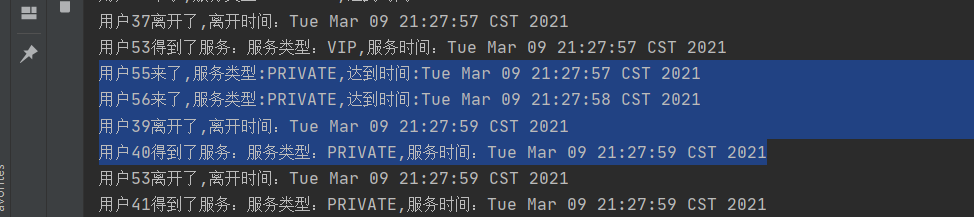
配置文件信息：

注：开放三个窗口，使用VIP优先算法  


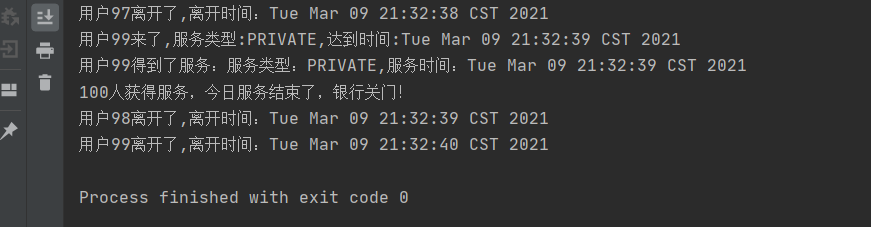
开放窗口提示为3个窗口



可见，用户53作为VIP用户，比用户40先得到的服务，可见，使用了VIP优先算法。

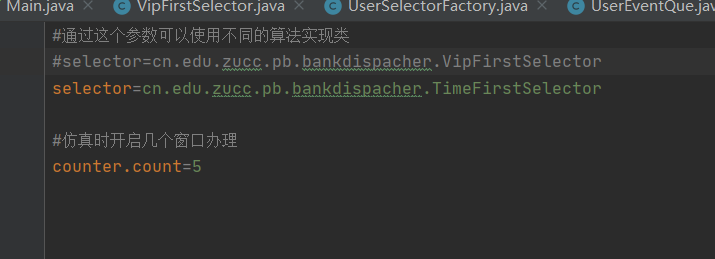


产生100位用户，程序结束输出提示，程序正常结束返回0，恰当退出。

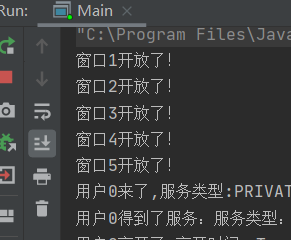


尝试二：

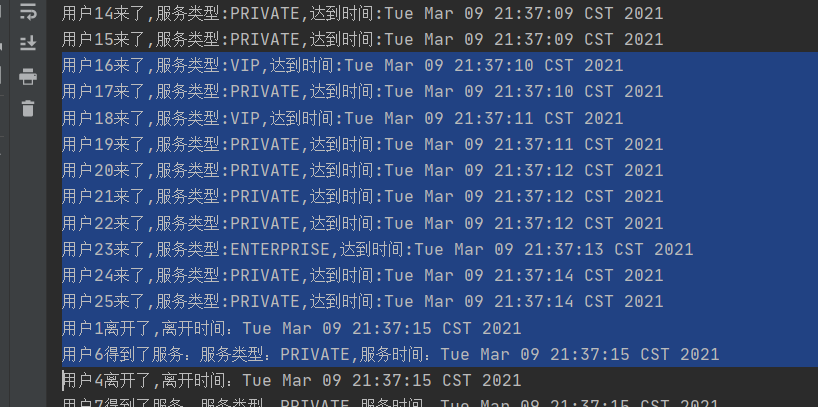
配置文件信息：

注：开放5个窗口，使用时间优先算法  


打印了五个窗口的开放提示



可见，虽然用户16是VIP用户，但是程序还是选择了先来的6号用户进行服务，使用了时间优先是算法。



程序正常退出，返回0，恰当结束

