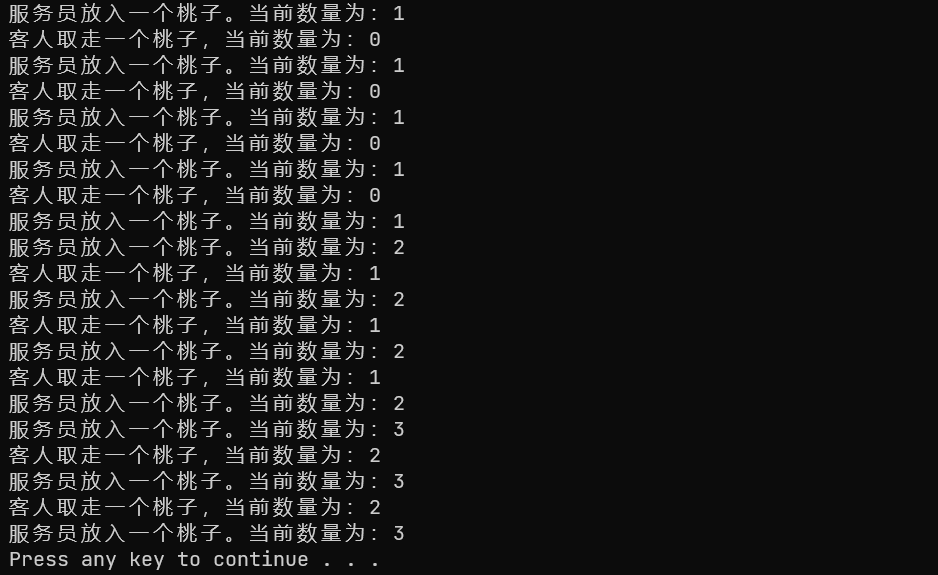
**实验十 线程的并发处理工具类**

一、 使用BlockingQueue实现线程通信

模拟服务员Servant可以往果盘放入桃子（只要果盘的数量少于5个），客人Guest每次要从果盘中取出一个桃子（只要果盘里有桃子），请用BlockingQueue的知识模拟这个场景。给出必要的注释。

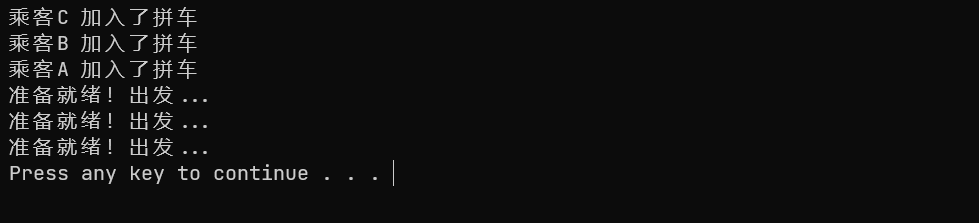
import java.util.concurrent.ArrayBlockingQueue;
  
import java.util.concurrent.BlockingQueue;
  
  
class FruitPlate {
  
 private BlockingQueue<String> plate = new ArrayBlockingQueue<>(5);
  
  
 // 服务员放桃子到果盘
  
 public void putPeach() throws InterruptedException {
  
 plate.put("Peach");
  
 System.out.println("服务员放入一个桃子。当前数量为：" + plate.size());
  
 }
  
  
 // 客人从果盘取桃子
  
 public void takePeach() throws InterruptedException {
  
 plate.take();
  
 System.out.println("客人取走一个桃子，当前数量为：" + plate.size());
  
 }
  
  
 public int size() {
  
 return plate.size();
  
 }
  
}
  
  
class Servant implements Runnable {
  
 private FruitPlate fruitPlate;
  
  
 public Servant(FruitPlate fruitPlate) {
  
 this.fruitPlate = fruitPlate;
  
 }
  
  
 @Override
  
 public void run() {
  
 try {
  
 while (true) {
  
 // 只要果盘的数量少于5个，服务员就往果盘放入桃子
  
 if (fruitPlate.size() < 5) {
  
 fruitPlate.putPeach();
  
 }
  
 // 模拟服务员的工作时间间隔
  
 Thread.sleep(1000);
  
 }
  
 } catch (InterruptedException e) {
  
 e.printStackTrace();
  
 }
  
 }
  
}
  
  
class Guest implements Runnable {
  
 private FruitPlate fruitPlate;
  
  
 public Guest(FruitPlate fruitPlate) {
  
 this.fruitPlate = fruitPlate;
  
 }
  
  
 @Override
  
 public void run() {
  
 try {
  
 while (true) {
  
 // 只要果盘里有桃子，客人就从果盘中取出一个桃子
  
 if (fruitPlate.size() > 0) {
  
 fruitPlate.takePeach();
  
 }
  
 // 模拟客人的用餐时间间隔
  
 Thread.sleep(1500);
  
 }
  
 } catch (InterruptedException e) {
  
 e.printStackTrace();
  
 }
  
 }
  
}
  
  
public class Main {
  
 public static void main(String[] args) {
  
 FruitPlate fruitPlate = new FruitPlate();
  
  
 // 创建服务员和客人线程
  
 Thread servantThread = new Thread(new Servant(fruitPlate));
  
 Thread guestThread = new Thread(new Guest(fruitPlate));
  
  
 // 启动服务员和客人线程
  
 servantThread.start();
  
 guestThread.start();
  
 }
  
}



阻塞队列只能保证put和take方法的同步，当使用多个Guest线程获取桃子时，无法保证System.out.println的顺序输出，可以考虑按时间对行为进行记录，最后统一输出。仅用作功能展示目的，故采用的是单线程。

二、 请用多线程计数器CountDownLatch模拟顺风车拼单的过程，拼单满三人即可发车。

import java.util.concurrent.CountDownLatch;
  
  
class RideShare {
  
 private static final int PASSENGER\_COUNT = 3;
  
 private CountDownLatch countDownLatch;
  
  
 public RideShare() {
  
 this.countDownLatch = new CountDownLatch(PASSENGER\_COUNT);
  
 }
  
  
 // 乘客加入拼单
  
 public void joinRide(String passengerName) throws InterruptedException {
  
 System.out.println(passengerName + " 加入了拼车");
  
 countDownLatch.countDown(); // 减少计数
  
 countDownLatch.await(); // 等待计数归零
  
 System.out.println("准备就绪！出发...");
  
 }
  
}
  
  
class Passenger implements Runnable {
  
 private String name;
  
 private RideShare rideShare;
  
  
 public Passenger(String name, RideShare rideShare) {
  
 this.name = name;
  
 this.rideShare = rideShare;
  
 }
  
  
 @Override
  
 public void run() {
  
 try {
  
 rideShare.joinRide(name);
  
 } catch (InterruptedException e) {
  
 e.printStackTrace();
  
 }
  
 }
  
}
  
  
public class Main {
  
 public static void main(String[] args) {
  
 RideShare rideShare = new RideShare();
  
  
 // 创建乘客线程
  
 Thread passenger1 = new Thread(new Passenger("乘客A", rideShare));
  
 Thread passenger2 = new Thread(new Passenger("乘客B", rideShare));
  
 Thread passenger3 = new Thread(new Passenger("乘客C", rideShare));
  
  
 // 启动乘客线程
  
 passenger1.start();
  
 passenger2.start();
  
 passenger3.start();
  
 }
  
}



三、 请用信号量Semaphore模拟大量学生前往丁香园食堂窗口打饭菜的情形。请给出必要的注释。

import java.util.concurrent.Semaphore;
  
  
class Canteen {
  
 private static final int MAX\_STUDENTS = 5; // 食堂窗口最大容纳学生数
  
 private Semaphore semaphore;
  
  
 public Canteen() {
  
 this.semaphore = new Semaphore(MAX\_STUDENTS);
  
 }
  
  
 // 学生打饭
  
 public void getFood(String studentName) throws InterruptedException {
  
 semaphore.acquire(); // 学生尝试获取许可证，如果许可证不足，会阻塞在这里
  
 System.out.println(studentName + " 正在取餐...");
  
 // 模拟学生打饭的时间
  
 Thread.sleep(2000);
  
 System.out.println(studentName + " 取餐结束！");
  
 semaphore.release(); // 学生完成打饭，释放许可证
  
 }
  
}
  
  
class Student implements Runnable {
  
 private String name;
  
 private Canteen canteen;
  
  
 public Student(String name, Canteen canteen) {
  
 this.name = name;
  
 this.canteen = canteen;
  
 }
  
  
 @Override
  
 public void run() {
  
 try {
  
 canteen.getFood(name);
  
 } catch (InterruptedException e) {
  
 e.printStackTrace();
  
 }
  
 }
  
}
  
  
public class Main {
  
 public static void main(String[] args) {
  
 Canteen canteen = new Canteen();
  
  
 // 创建大量学生线程 A-Z
  
 for (int i = 65; i <= 90; i++) {
  
 Thread studentThread = new Thread(new Student("学生 " + (char) i, canteen));
  
 studentThread.start();
  
 }
  
 }
  
}



四、 **秒钟、分钟和时针的通信\***\*(**拓展题**)\*\*

秒针走到60，分针就会增加1；分针走到60,时针就会增加1。请编写一个模拟钟表的秒针、分针和时针走动的程序，采用24小时计时法，当时针走到24，会从0开始重新计时。

编程提示：用Clock类表示钟表，有秒针、分针和时针的属性。三条线程分别调用Clock类的addSecond、addMinute和addHour的同步方法。这三个方法根据秒针、分针和时针的取值互相通信，确保当秒针变成60，分针增加1，而秒针恢复0。当分针变成60，时针增加1，而分针恢复0。秒针每走动一次需要休眠1000毫秒。

import java.util.Calendar;
  
  
class Clock {
  
 private int seconds;
  
 private int minutes;
  
 private int hours;
  
  
 public Clock() {
  
 // 获取当前系统时间
  
 Calendar calendar = Calendar.getInstance();
  
 this.seconds = calendar.get(Calendar.SECOND);
  
 this.minutes = calendar.get(Calendar.MINUTE);
  
 this.hours = calendar.get(Calendar.HOUR\_OF\_DAY);
  
 }
  
  
 // 增加秒针
  
 public synchronized void addSecond() throws InterruptedException {
  
 while (seconds >= 60) {
  
 wait();
  
 }
  
  
 Thread.sleep(1000);
  
  
 seconds++;
  
 if (seconds == 60) {
  
 seconds = 0;
  
 addMinute(); // 秒针走到60，增加分钟
  
 }
  
  
 notifyAll();
  
 }
  
  
 // 增加分针
  
 public synchronized void addMinute() throws InterruptedException {
  
 while (minutes >= 60) {
  
 wait();
  
 }
  
  
 minutes++;
  
 if (minutes == 60) {
  
 minutes = 0;
  
 addHour(); // 分针走到60，增加小时
  
 }
  
  
 notifyAll();
  
 }
  
  
 // 增加时针
  
 public synchronized void addHour() throws InterruptedException {
  
 while (hours >= 24) {
  
 wait();
  
 }
  
  
 hours++;
  
 if (hours == 24) {
  
 hours = 0; // 时针走到24，重置为0
  
 }
  
  
 notifyAll();
  
 }
  
  
 // 获取当前时间的字符串表示
  
 public String getTime() {
  
 return String.format("%02d:%02d:%02d", hours, minutes, seconds);
  
 }
  
}
  
  
public class Main {
  
 public static void main(String[] args) {
  
 Clock clock = new Clock();
  
  
 Thread secondThread = new Thread(() -> {
  
 try {
  
 while (true) {
  
 clock.addSecond();
  
 System.out.println(clock.getTime());
  
 }
  
 } catch (InterruptedException e) {
  
 e.printStackTrace();
  
 }
  
 });
  
  
 secondThread.start();
  
 }
  
}

