# 实验 6 多线程

## 实验目的

1. 掌握 Java 多线程的概念和实现
2. 掌握、使用继承 Thread 产生线程
3. 掌握 Runnable 接口产生线程
4. 掌握 wait-notify 机制

## 实验步骤

**步骤 1**：创建工程，工程名以学号姓名方式命名，“2020010505-薛飞宇”。
**步骤 2**：为每道实习题目创建对应的包，如“work1”、“work2”……；
**步骤 3**：按照要求创建源代码，进行编写，注意编码格式，如缩进、命名规范等；
**步骤 4**：规范书写实习报告；
**步骤 5**：实现与测试。附上代码或以附件的形式提交，同时贴上必要的代码运行截图；
**步骤 6**：及时总结心得体会与备忘。

## 实验内容

### 实验题 1 多线程输出字符串（继承 Thread 类）

创建主线程(main 所在的线程)和两个独立的子线程，主线程打印 20 次“A”，其中一个子线程打印 20 次“B”，另一个子线程打印 20 次“C”，输出结果类似于：

ACBCCBACBACABCBA…

本题异常简单，思路如下：首先根据要求，使用继承创建线程类 printB，这个线程的任务就是打印 20 次 B，每次打印前先休眠 100 毫秒。于是，在这个线程的 run()方法中完成这个任务即可。用同样的思路创建线程类 printC。最后，在 main()方法中，也就是主线程中完成一个方法 printA()并调用，然后分别新建这个两个线程类并调用 start()方法即可。本题的 UML 类图如下。

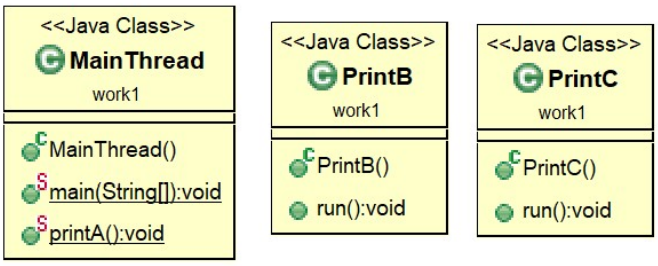


图 1.1 实验 1UML 类图

然后是本题的测试结果。

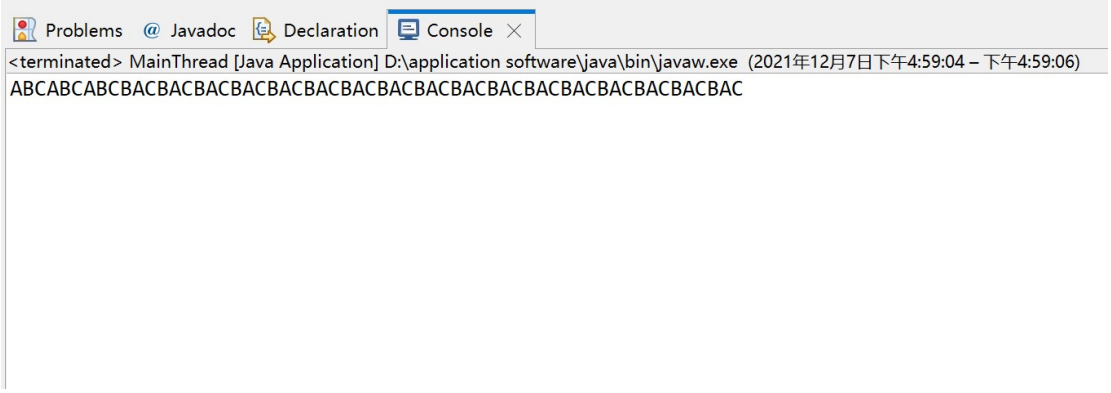


图 1.2 实验 1 结果

第一次做这道题，我还犯了一个错误，就是把 Thread 类中的 run 方法写成了 Run，导致了一些错误。

```
public class PrintB extends Thread{

    public void Run() {
        for(int i=0;i<20;i++) {
            System.out.print("B");
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
```

图 1.3 错误的 Run()方法

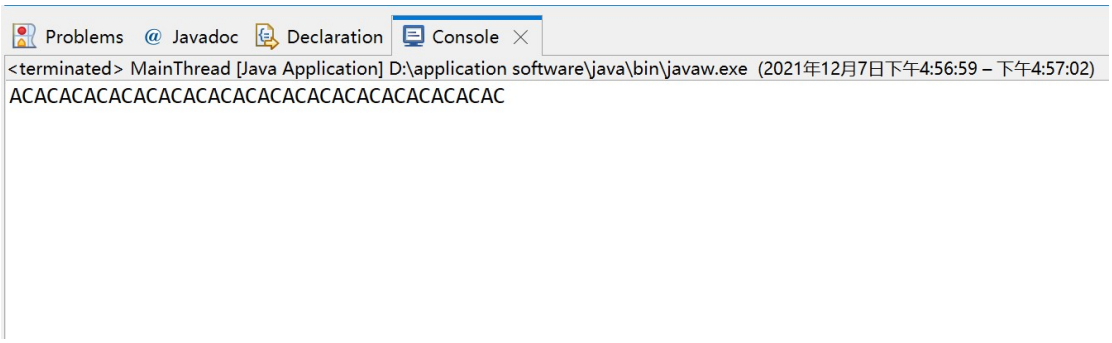


图 1.4 没有输出 B

**实验题 2 多个线程共同执行目标对象的任务（实现 Runnable）**

有一批零件（Part）的生产任务，总量是 800 个，生产一个零件耗时 10ms。有 4 个车间（Workshop）共同完成这一批任务，分别统计每个车间各自生产了多少个零件并输出。

本实验依然十分简单。首先，用 WorkShop 实现 Runnable 接口，注意在 Thread 中 target 对象是同一个，所以 all 是给三个线程共享的资源，但是如果用 thread 继承的话，all 就不是对象的共享资源，解决方案是 all 设置为静态成员变量。本题中 4 个车间共享一个 target 资源，所以新建 4 个线程对象，将 target 对象作为参数传入即可实现资源的共享。同样地，在 target 的 run()方法中，统计并输出各个线程执行了多少次。

上述方法解决了资源共享的问题，然后还要解决进程互斥访问共享资源的问题，只要将访问共享资源的方法加上关键字 synchronized 即可，这样的话普通的方法就成为了同步方法，也就实现了进程互斥。

下面是测试结果。

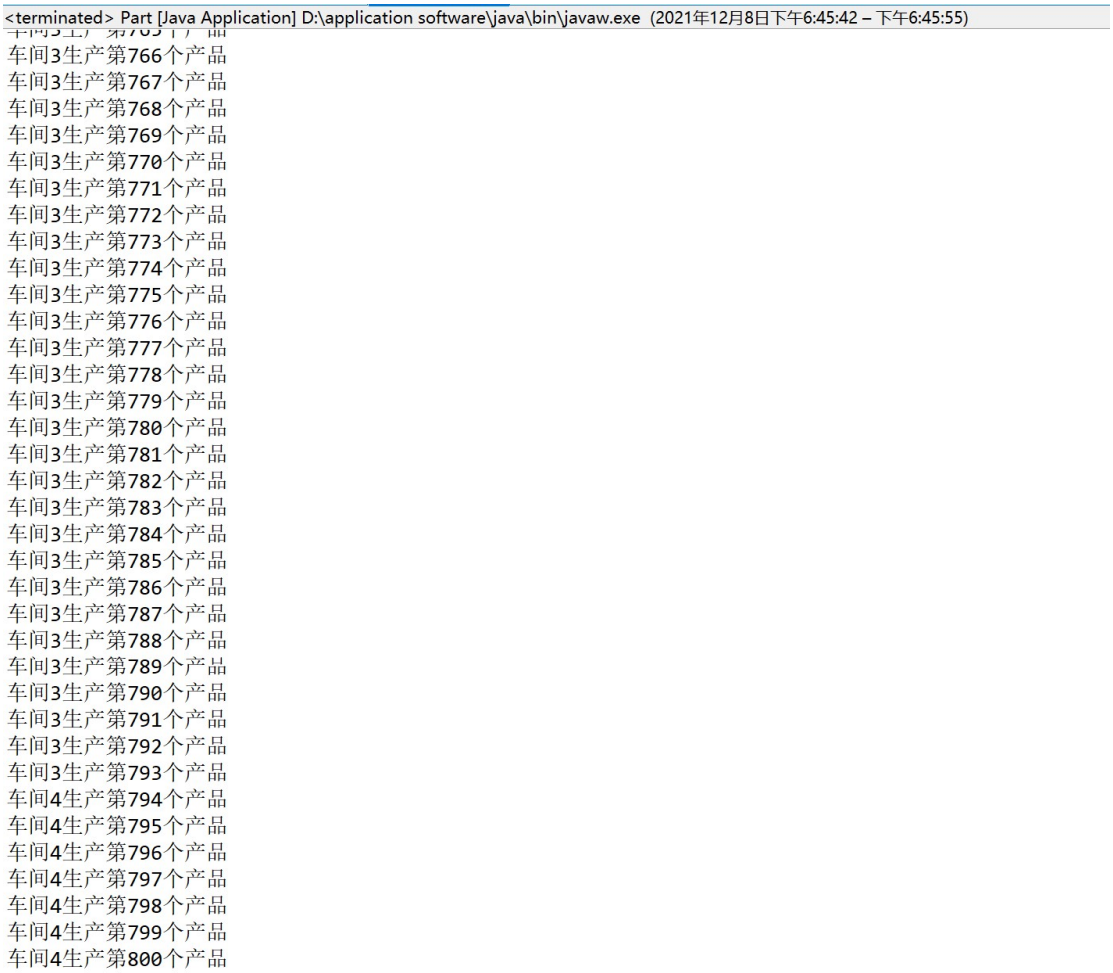


图 2.1 实验 2 结果

```
<terminated> Part [Java Application] D:\application software\java\bin\javaw.exe  (2021年12月10日下午8:52:56 – 下午8:53:09)
车间3生产第795个产品
车间3生产第796个产品
车间3生产第797个产品
车间3生产第798个产品
车间3生产第799个产品
车间3生产第800个产品
统计：车间3生产了256个产品
统计：车间4生产了222个产品
统计：车间1生产了243个产品
统计：车间2生产了82个产品
```

图 2.2 实验 2 结果

下面是本题的 UML 类图。

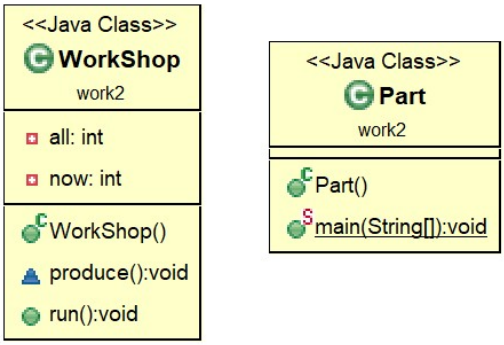


图 2.5 实验 2UML 类图

**实验题 3 生产者-消费者问题（线程间通信）**

有一种商品 Product 类，一个货架类（GoodsShelf）:拥有一个容量为 size（可根据需要设定大小）的 Product 数组。生产者（Producer）往货架上放置商品，消费者（Consumer）从货架上消费商品。二者是并发执行的。（货架空的时候只能生产，消费线程等待；货架满的时候只能消费，生产线程等待。）使用多线程并发机制和线程间通信，模拟商品编号从 1 到 100 的生产和消费过程。

实验 2 解决了不同的线程对共享资源的互斥访问，但是没有解决线程间通信问题。线程间通信需要使用 wait-notify 机制。也就是操作系统中常见的 PV 操作。在单生产者-单消费者问题中，货架上如果没有商品了，消费者就要等待，通知生产者生产，生产者生产后就要通知消费者消费；货架上如果商品满了，生产者就要等待，通知消费者消费，消费者消费后就要通知生产者生产。本题的模型就是如此。

下面看实验结果。

```
<terminated> TestProducerAndConsumer [Java Application] D:\application software\java\bin\javaw.exe  (2021年12月10日下午7:52:23 – 下午7:52:44)
货架空，请等待生产者生产
生产了89个产品
消费了第89个产品
货架空，请等待生产者生产
消费了第90个产品
货架空，请等待生产者生产
生产了90个产品
生产了91个产品
消费了第91个产品
货架空，请等待生产者生产
消费了第92个产品
生产了92个产品
货架空，请等待生产者生产
消费了第93个产品
生产了93个产品
货架空，请等待生产者生产
消费了第94个产品
货架空，请等待生产者生产
生产了94个产品
消费了第95个产品
货架空，请等待生产者生产
生产了95个产品
消费了第96个产品
货架空，请等待生产者生产
生产了96个产品
生产了97个产品
消费了第97个产品
货架空，请等待生产者生产
消费了第98个产品
货架空，请等待生产者生产
生产了98个产品
生产了99个产品
消费了第99个产品
货架空，请等待生产者生产
生产了100个产品
消费了第100个产品
```

图 3.1 实验 3 测试结果

```
<terminated> TestProducerAndConsumer [Java Application] D:\application software\java\bin\javaw.exe  (2021年12月10日下午7:56:39 – 下午7:56:59)
生产了1个产品
生产了2个产品
生产了3个产品
生产了4个产品
生产了5个产品
生产了6个产品
生产了7个产品
生产了8个产品
生产了9个产品
生产了10个产品
货架满，请等待消费者取走
生产了11个产品
消费了第10个产品
货架满，请等待消费者取走
消费了第11个产品
生产了12个产品
货架满，请等待消费者取走
消费了第12个产品
生产了13个产品
货架满，请等待消费者取走
消费了第13个产品
生产了14个产品
货架满，请等待消费者取走
生产了15个产品
消费了第14个产品
货架满，请等待消费者取走
生产了16个产品
消费了第15个产品
货架满，请等待消费者取走
消费了第16个产品
生产了17个产品
货架满，请等待消费者取走
消费了第17个产品
生产了18个产品
货架满，请等待消费者取走
消费了第18个产品
```

图 3.2 实验 3 测试结果

```
<terminated> TestProducerAndConsumer [Java Application] D:\application software\java\bin\javaw.exe (2021年12月10日下午7:56:39 – 下午7:56:59)
消费了第91个产品
生产了92个产品
货架满，请等待消费者取走
消费了第92个产品
生产了93个产品
货架满，请等待消费者取走
消费了第93个产品
生产了94个产品
货架满，请等待消费者取走
生产了95个产品
消费了第94个产品
货架满，请等待消费者取走
消费了第95个产品
生产了96个产品
货架满，请等待消费者取走
消费了第96个产品
生产了97个产品
货架满，请等待消费者取走
消费了第97个产品
生产了98个产品
货架满，请等待消费者取走
消费了第98个产品
生产了99个产品
货架满，请等待消费者取走
消费了第99个产品
生产了100个产品
消费了第100个产品
消费了第9个产品
消费了第8个产品
消费了第7个产品
消费了第6个产品
消费了第5个产品
消费了第4个产品
消费了第3个产品
消费了第2个产品
消费了第1个产品
```

图 3.3 实验 3 测试结果

下面是本实验的 UML 类图。

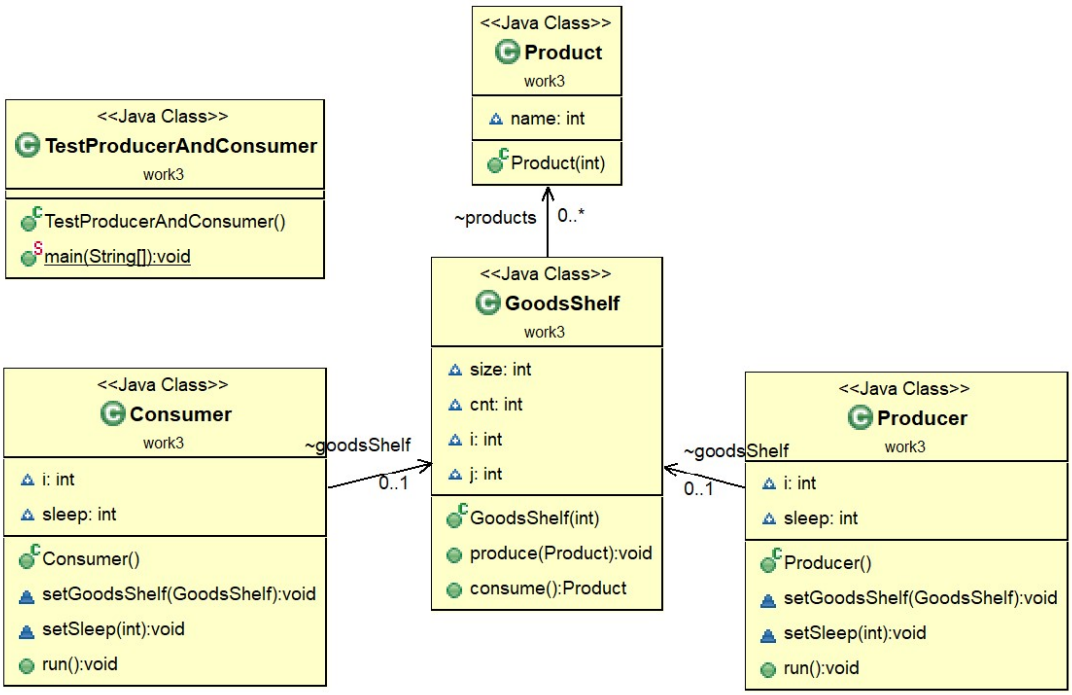


图 3.4 实验 3UML 类图

然后要解决多生产者-多消费者问题了。这个问题可以分成两个部分。第一个部分是生产者和消费者之间的同步和通信问题。第二个部分是生产者之间，消费者之间的互斥关系，也就是——生产者 1 在访问货架时，生产者 2 不能访问货架，消费者亦是如此。我尝试写了一个程序，但是没有成功。

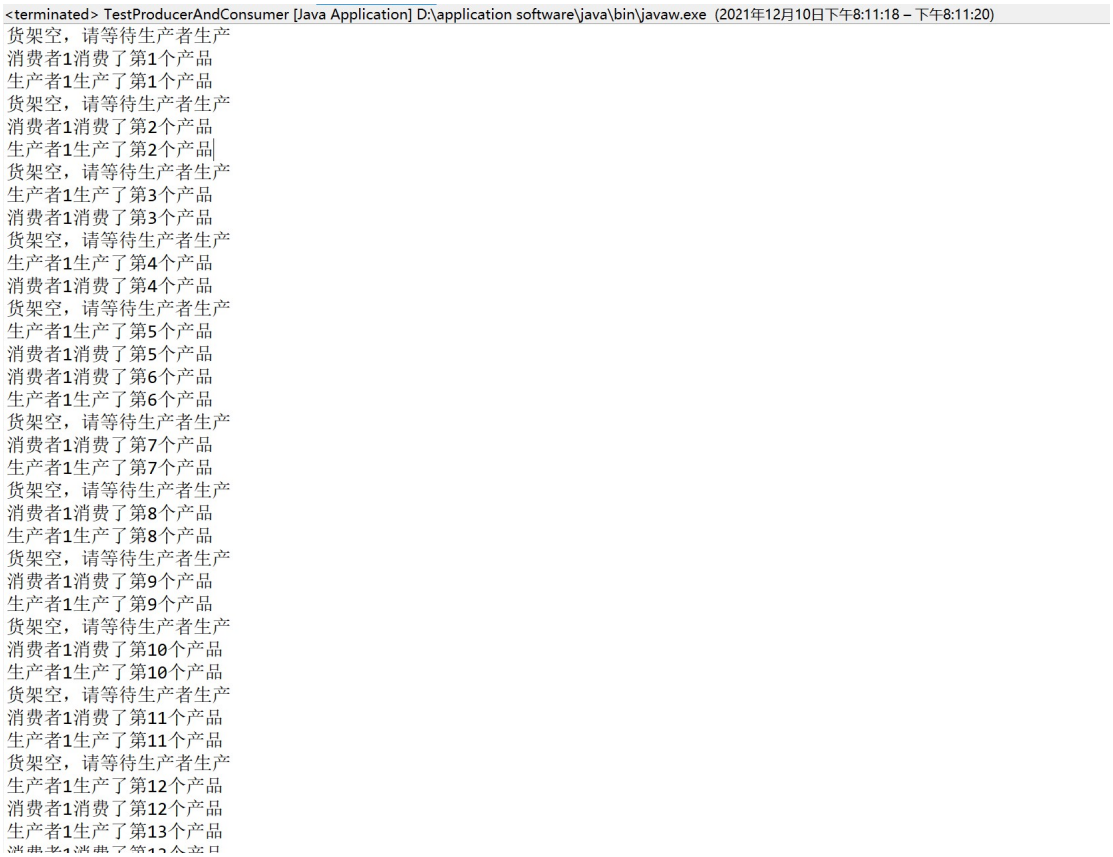


图 3.5 多生产者-多消费者问题

后来，经过我缜密的分析。我解决了这道题。

进程间通信的问题，在上面已经阐述，这里只讲进程互斥的问题。

这道题中一共有两个临界资源，一个是缓冲区计数器，就是记录当前货架上商品的数量，在代码中这个资源就是 cnt。当生产者或者消费者中的一个再访问这个资源的时候，任何别的对象都不可以访问这个资源。所以要对缓冲区计数器上锁。

第二个临界资源是生产者生产数量计数器，也就是记录生产者当前正在生产第几个商品，在代码中这个资源就是迭代变量 i。当一个生产者在生产商品的时候，访问这个迭代变量，这时候，别的生产者就不可以访问这个资源，所以也要对这个变量上锁。

同理，消费者的迭代变量也是如此。

下面是测试结果。

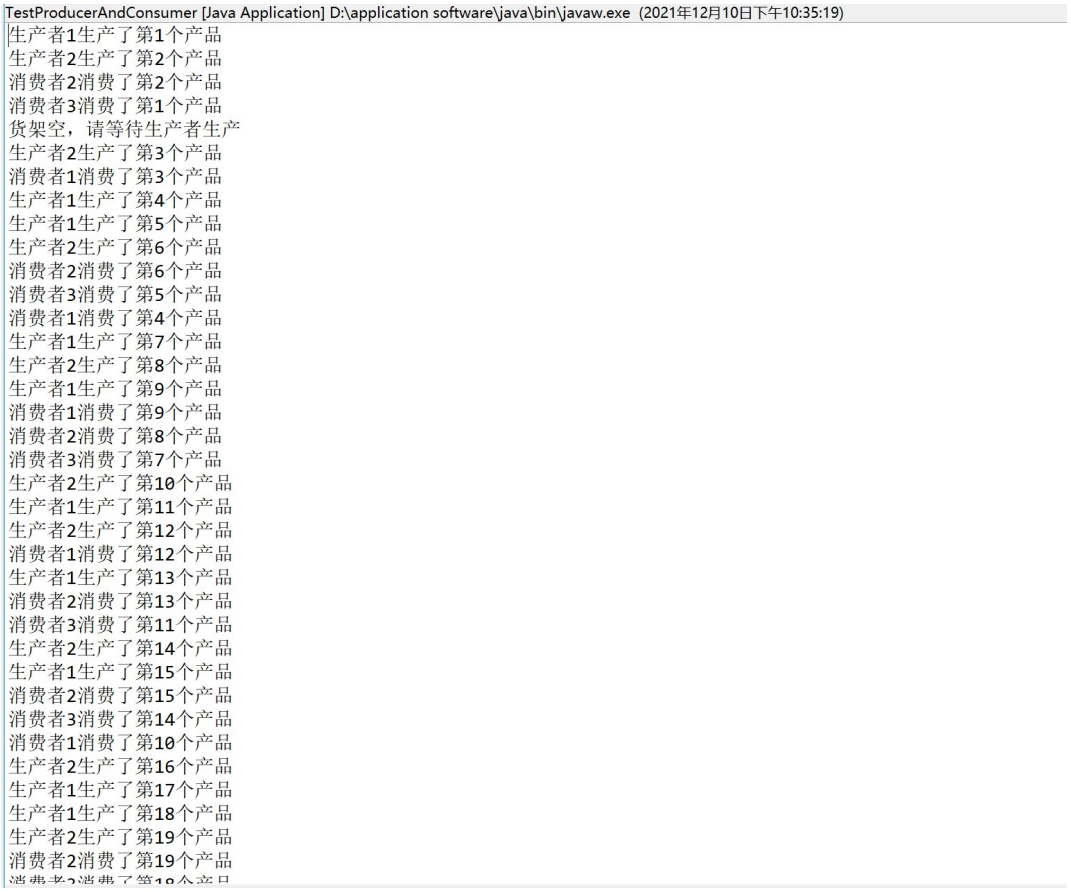


图 3.6 多生产者-多消费者问题

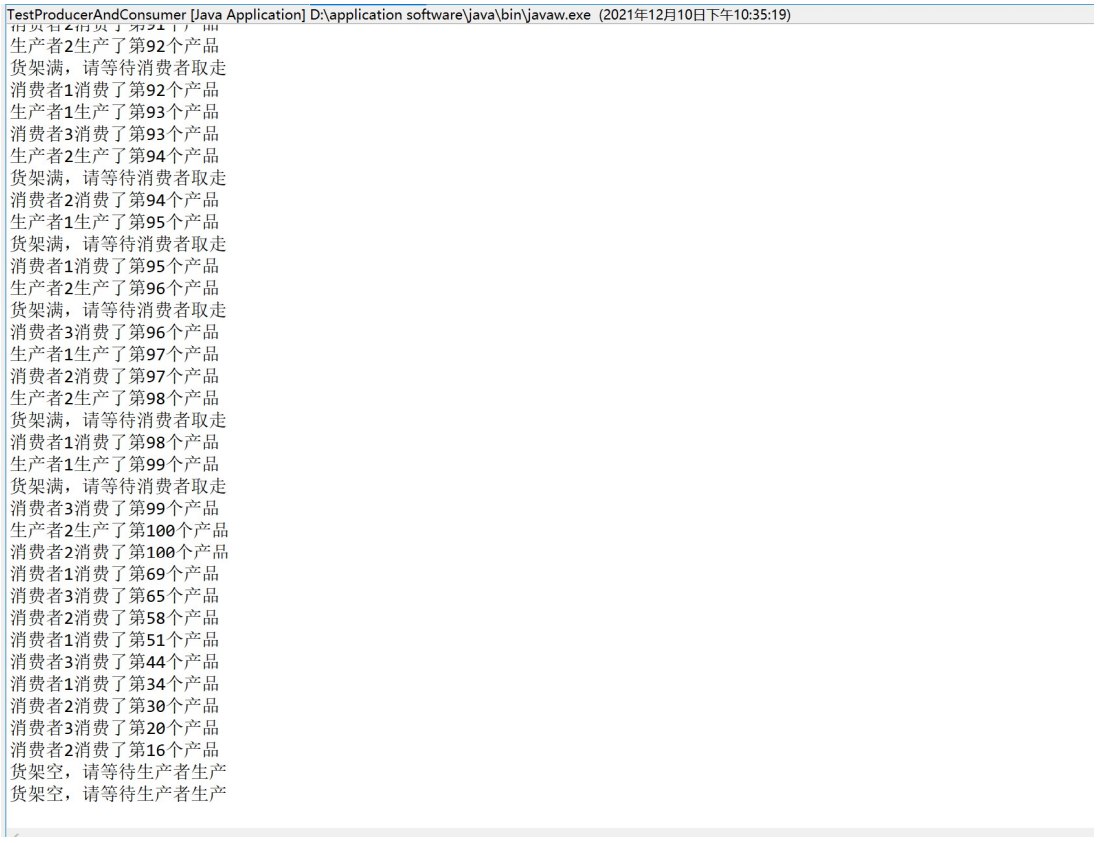


图 3.7 多生产者-多消费者问题

**本次实验尚未解决的疑难：**

本实验所有问题都已解决。

**心得体会：**

本次实验，我掌握 Java 多线程的概念和实现，掌握、使用继承 Thread 产生线程、掌握 Runnable 接口创建线程，掌握 wait-notify 机制，用 Java 语言模拟实现了车站售票问题、单生产者-单消费者、多生产者-多消费者问题，深入理解了线程同步、线程通信、线程互斥的概念。但是线程的世界还很大，在实际的开发环境中会有更多更难的线程同步和线程互斥问题，等有时间了，一定要多加练习。