# Readme

[Readme](#readme)  
 [第二题](#第二题)  
 [题目要求：](#题目要求-1)  
 [解题思路](#解题思路-1)  
 [结果截图](#结果截图-1)  
 [第三题](#第三题)  
 [题目要求](#题目要求-2)  
 [解题思路](#解题思路-2)  
 [结果截图](#结果截图-2)

## 第二题

### 题目要求：

采用2个producer和一个Consumer把整数放入到（或者从中取出）一个环形缓冲Circle Buffer中，这块连续的存储会被反复使用。一个写指针指向数据写入的位置，如果写到Buffer的末端，则从Buffer的头端开始写。类似的，一个读指针指向读取数据的位置，如果读到Buffer的末端，也转回到头端开始读。

### 解题思路

显然本题是一个典型的生产者消费者问题，要求创建缓冲区Buffer，故采取管程法。

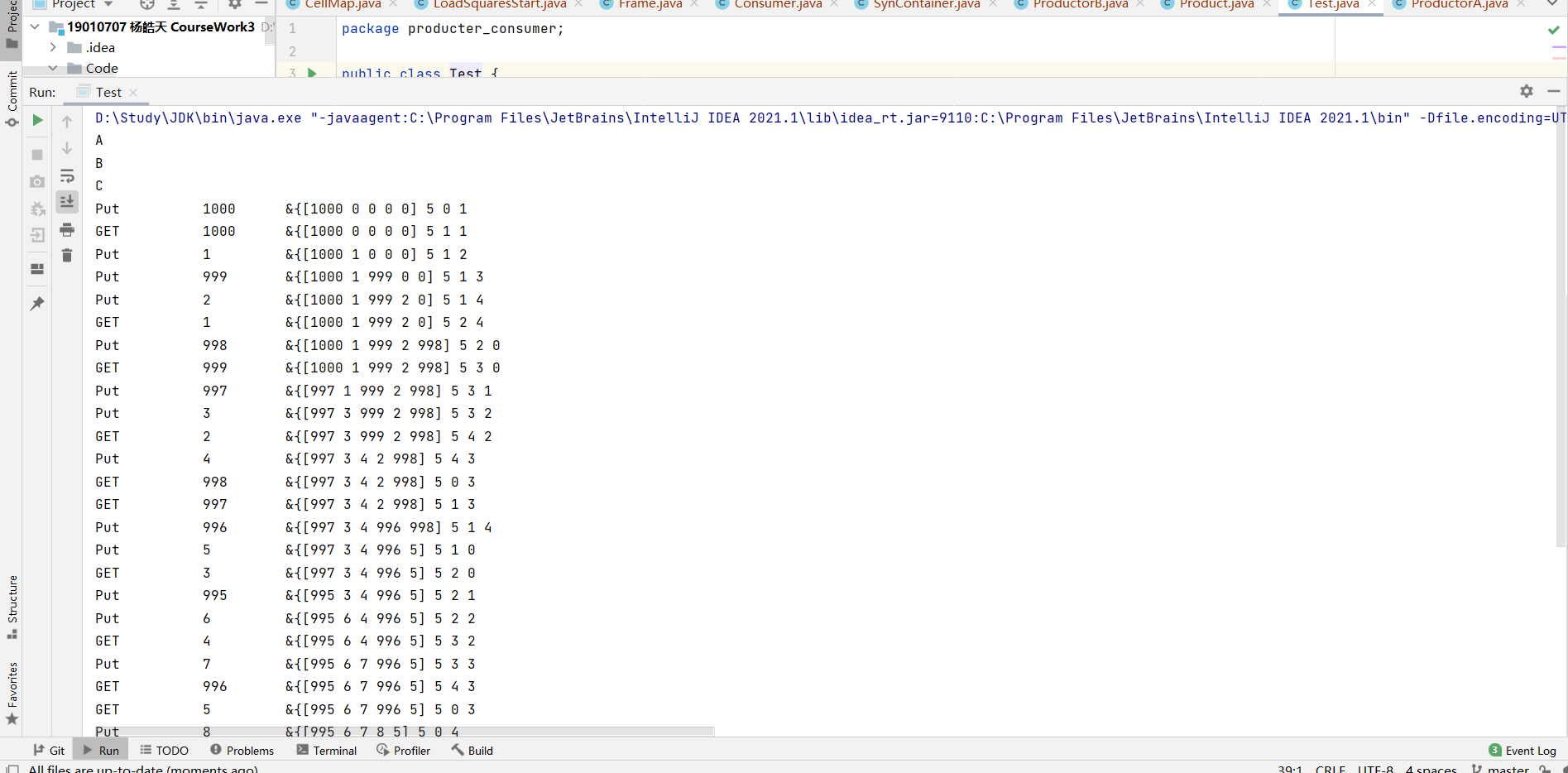
一共创建有五个类，分别是Consumer、Product、ProductorA、ProductorB、SynContainer，代表着消费者、产品、两类生产者和缓冲区。

首先应当对缓冲区进行定义。从逻辑上来看，缓冲区对象是被直接作用的对象，而生产者和消费者都对缓冲区对象在构造器中传入，是对缓冲区对象进行输入输出的操作方。生产过程中，缓冲区对象应对缓冲区接收的产品的总量与缓冲区内部当前总量进行计数（产品内容专门用一个product类进行封装）并通过缓冲区当前总量Count判断当前缓冲区内是否已满，如果已满则通过等待唤醒机制通知消费者进行消费。若未满，则在生产过程中对wri\_index进行计数，并创建对应函数进行生产者和消费者的返回操作。消费操作与之类似。

生产者类继承Runnable接口，在main函数中线程化，并在run方法中构建for循环，在循环中调用传入的缓冲区对象的push方法（生产），向缓冲区内传入数据，两个生产者分别从1传到1000和从1000传到1，并构建一个从0到500的随机大小变量，用于在单次循环过程中确定当前线程的睡眠时长，用于实现两个生产者和一个消费者轮流随机对缓冲区进行输入输出的现象。生产者类和消费者类都具有getStr和bitsCount两个类来确保对对应内容的格式化输出（bitsCount确保字符串中不同字符宽度的数字在显示中对齐输出）。

消费者类与生产者类类似。为了规避竞态条件，为生产者和消费者类中的内容加锁。

### 结果截图



## 第三题

### 题目要求

基于第二次作业的中值过滤算法分四个线程将整个图像利用多线程进行过滤。

### 解题思路

关于具体的中值过滤方法已在第二次作业的Readme中进行了详细阐述，故此处不做累述，仅说明将过滤算法线程化的过程。

首先基于效率和图片整体性的考虑，第一不应该对图片进行多次读取，而应当将图片读取后用Pixel类进行包装后再进行处理。第二应当待多线程将所有图片处理完成后再重新输出为PNG格式的图像文件，而不能在多线程中输出。

故多线程的内容主要是原算法中对具体读出的图像矩阵进行处理的内容。

即新增一个FilterSquares类，将当前线程要处理的图像位置（start，end）进行输入，并传入承载图像像素矩阵的Pixel对象。将原中值算法函数medianFilter函数作为继承了Runnable接口的FilterSquares的run方法，由于图像像素矩阵由多个线程共享，且各个线程无需对其他线程处理的内容进行修改，所以无需为函数或代码块加锁。

在对图像处理时，创建四个线程分别用FilterSquares进行初始化并运行。由于必须在线程处理完以后才可以利用remakePNG方法输出图像，故可以在remankePNG方法前加一个

while(thread1.isAlive() || thread2.isAlive() || thread3.isAlive() ||thread4.isAlive()){ }

的空循环，等待线程全部运行完之后才进行图像输出。

注意，由于中值过滤算法依靠3x3像素矩阵进行麻点过滤，所以边缘位置会缺乏处理，应当将四个线程的边缘位置做重叠处理，才可以保证不会有未处理间隙出现。

### 结果截图

