# 实验三多线程通信协同

## 1.1实验目的

1) 熟悉Java多线程相关概念；

2) 掌握直接继承Thread类和引用Runnable接口两种方式实现多线程；

3) 掌握线程通信的概念和流程；

4）学会运用线程通信技术来解决应用问题。

## 1.2实验内容

**生产者-消费者问题**

生产者线程将生产好的产品存放到仓库，消费者线程从仓库取走产品投放市场。请模拟仓库，有2个存储单元（可用数组实现）；请模拟消费者线程和生产者线程，能依次往仓库中存储和取出1-6个整数。通过线程通信技术，使得线程能够协同工作。

## 1.3实验步骤

1) 新建工程，并创建一个类Storage类，拥有一个可存储2个整数的数组；创建get方法可循环地从数组中取出数据；创建set方法可循环地从数组中存入数据；

2）创建Productor线程，线程可访问Storage类的对象，能依次往仓库中存储1-6个整数。每存储一个数字，sleep100毫秒。存满则重新回到数组开头处，以实现循环存储。如果存储位置不为空，则用wait()方法等待消费者线程唤醒；每存完数据就notify()唤醒消费者线程。

4）创建Comsumer线程，线程可访问Storage类的对象，能依次往仓库中取出1-6个整数。每取出一个数字，sleep100毫秒。取完则重新回到数组开头处，以实现循环取数字。如果取的位置不为空，则用wait()方法等待生产者线程唤醒；每去完数据就notify()唤醒生产者线程。

3) 创建测试类测试上述功能

## 1.4实验报告

记录1.3 实验步骤中完成实验的过程以及源代码（重要的源代码应该添加代码注释），并回答实验步骤中提出的问题。

实验代码：

**Storage类**

*package* mcc;  
  
  
*import* java*.*util*.*List;  
  
*/\*\*  
 \** ***@author*** *ljjtpcn  
 \*/  
  
public class* Storage {  
 */\*\*  
 \* 生产池  
 \*/  
 private* List<Integer> queue;  
 */\*\*  
 \* 一次只能放ｎ数据  
 \*/  
 private* Integer n;  
 */\*\*  
 \* 当前产了多少数据了  
 \*/  
 private* Integer current = 0;  
 */\*\*  
 \* 生产者只能最多产maxMoney个, 默认为１０个  
 \*/  
 private* Integer maxMoney = 10;  
  
 *public Storage*(List<Integer> queue, Integer n) {  
 *this.*queue = queue;  
 *this.*n = n;  
 }  
  
 *public Storage*(List<Integer> queue, Integer n, Integer maxMoney) {  
 *this.*queue = queue;  
 *this.*n = n;  
 *this.*maxMoney = maxMoney;  
 }  
  
 *public void* getNumber() {  
 *while* (*this.*getCurrent() < *this.*getMaxMoney()) {  
 *synchronized* (*this.*getQueue()) {  
 *while* (*this.*getQueue()*.*isEmpty()) {  
 System*.*out*.*println(Thread*.currentThread*()*.*getName() + "空了...等待生产");  
 *try* {  
 *this.*getQueue()*.*wait();  
 } *catch* (InterruptedException e) {  
 e*.*printStackTrace();  
 }  
 }  
 *for* (*int* i = 0; i < *this.*getQueue()*.*size(); i++) {  
 System*.*out*.*println(Thread*.currentThread*()*.*getName() + "取走了 " + *this.*getQueue()*.*get(i));  
 *try* {  
 Thread*.sleep*(100);  
 } *catch* (InterruptedException e) {  
 e*.*printStackTrace();  
 }  
 }  
 *this.*getQueue()*.*clear();  
 *this.*getQueue()*.*notify();  
 }  
 }  
 System*.*out*.*println("生产者没钱了。。。罢工结束");  
 }  
  
 *public void* setNumber() {  
 *for* (*int* i = 1; i <= *this.*getMaxMoney(); i++) {  
 *synchronized* (*this.*getQueue()) {  
 *while* (*this.*getQueue()*.*size() == 2) {  
 System*.*out*.*println(Thread*.currentThread*()*.*getName() + "已满...");  
 *try* {  
 *this.*getQueue()*.*wait();  
 } *catch* (InterruptedException e) {  
 e*.*printStackTrace();  
 }  
 }  
 *this.*getQueue()*.*add(i);  
 *this.*setCurrent(*this.*getCurrent() + 1);  
 System*.*out*.*println(Thread*.currentThread*()*.*getName() + "生产了　" + i);  
 *try* {  
 Thread*.sleep*(100);  
 } *catch* (InterruptedException e) {  
 e*.*printStackTrace();  
 }  
 *this.*getQueue()*.*notify();  
 }  
 }  
 }  
  
 *public* List<Integer> getQueue() {  
 *return* queue;  
 }  
  
  
 *public* Integer getCurrent() {  
 *return* current;  
 }  
  
 *public void* setCurrent(Integer current) {  
 *this.*current = current;  
 }  
  
  
 *public* Integer getMaxMoney() {  
 *return* maxMoney;  
 }  
  
  
}

**生产者类**

*package* mcc;  
  
*/\*\*  
 \** ***@author*** *ljjtpcn  
 \*/  
public class* Productor *implements* Runnable {  
  
 *public* Storage storage;  
  
 *public Productor*(Storage storage) {  
 *this.*storage = storage;  
 }  
  
 @Override  
 *public void* run() {  
  
 storage*.*setNumber();  
 }  
}

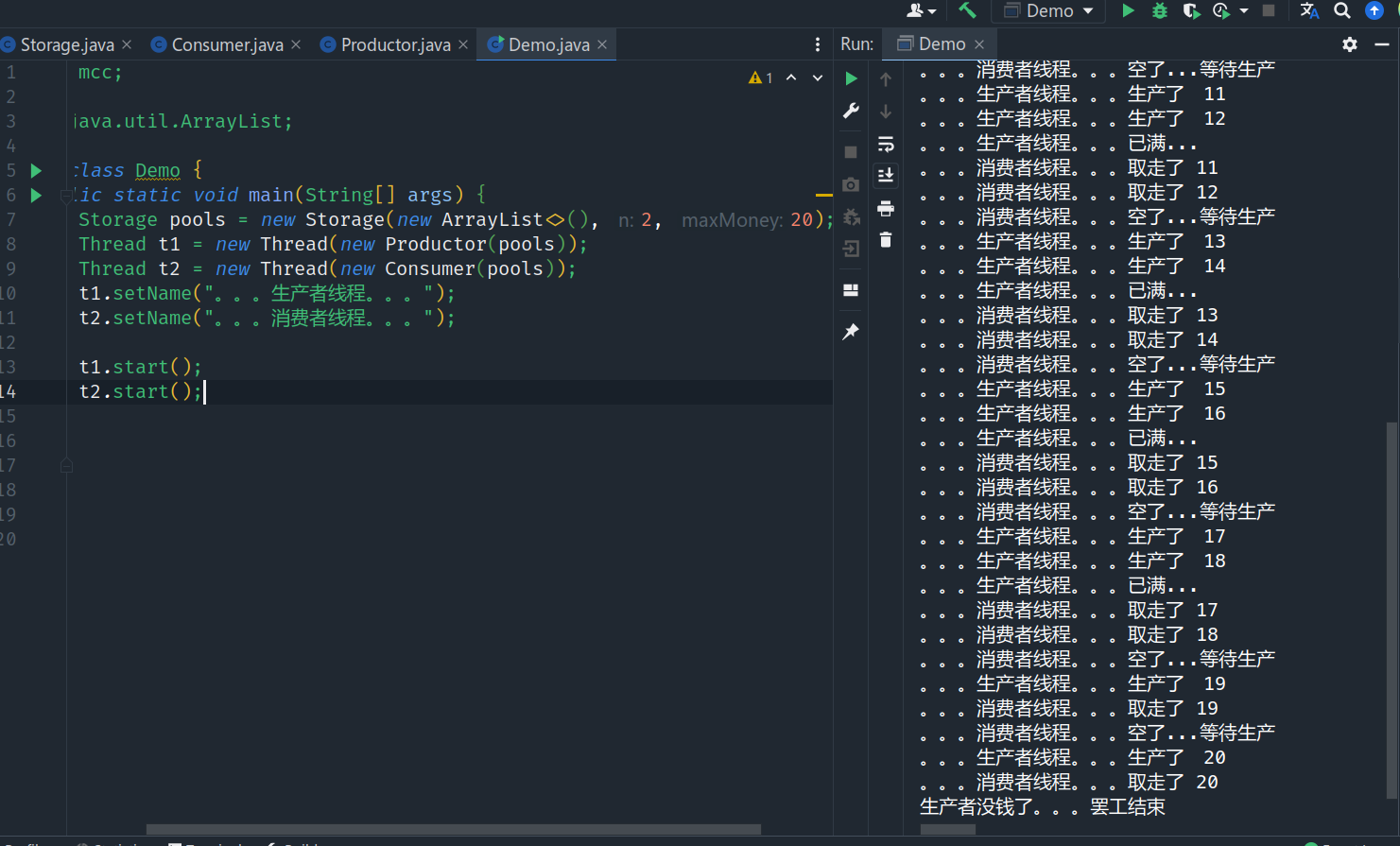
**消费者类**

*package* mcc;  
  
*/\*\*  
 \** ***@author*** *ljjtpcn  
 \*/  
public class* Consumer *implements* Runnable {  
 *public* Storage storage;  
  
 *public Consumer*(Storage storage) {  
 *this.*storage = storage;  
 }  
  
  
 @Override  
 *public void* run() {  
 storage*.*getNumber();  
 }  
}

**测试类**

*package* mcc;  
  
*import* java*.*util*.*ArrayList;  
  
*public class* Demo {  
 *public static void* main(String[] args) {  
 Storage pools = *new* Storage(*new* ArrayList<>(), 2, 20);  
 Thread t1 = *new* Thread(*new* Productor(pools));  
 Thread t2 = *new* Thread(*new* Consumer(pools));  
 t1*.*setName("。。。生产者线程。。。");  
 t2*.*setName("。。。消费者线程。。。");  
  
 t1*.*start();  
 t2*.*start();  
  
  
 }  
}

**实验截图：**

****