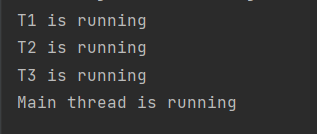
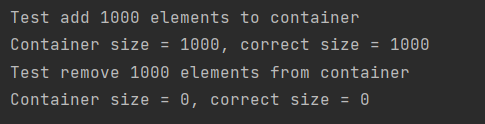
1. 继承Thread 使用Runnable接口
2. Dead lock
3. 暂停，并执行其他线程
4. Sleep setPriority
5. Thread.currentThread ().getId ()
6. C
7. B
8. B
9. C
10. B
11. C

问答题

package chapter30;  
  
public class SynThread {  
 public static void main(String[] args) {  
  
 Thread t1 = new Thread(() -> {  
 // T1 执行代码  
 System.out.println("T1 is running");  
 });  
  
 Thread t2 = new Thread(() -> {  
 try {  
 // 等待 T1 执行完成  
 t1.join();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 // T2 执行代码  
 System.out.println("T2 is running");  
 });  
  
 Thread t3 = new Thread(() -> {  
 try {  
 // 等待 T2 执行完成  
 t2.join();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 // T3 执行代码  
 System.out.println("T3 is running");  
 });  
  
 t1.start();  
 t2.start();  
 t3.start();  
  
 try {  
 // 等待 T3 执行完成  
 t3.join();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 // 主线程执行代码  
 System.*out*.println("Main thread is running");  
 }  
}



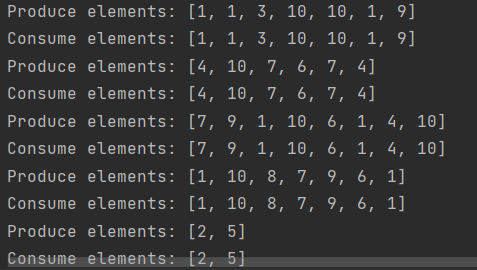
编程题1



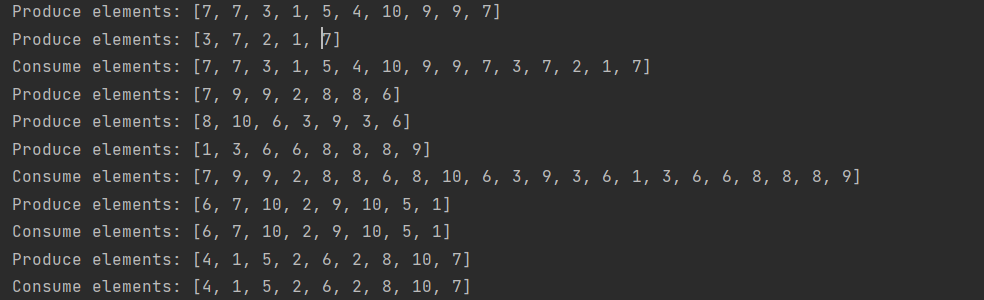
package chapter30;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class SynchronizedContainer<T> {  
 private List<T> elements = new ArrayList<>();  
 */\*\*  
 \* 添加元素  
 \** ***@param*** *e 要添加的元素  
 \*/* public synchronized void add(T e){  
 elements.add(e);  
 }  
 */\*\*  
 \* 删除指定下标的元素  
 \** ***@param*** *index 指定元素下标  
 \** ***@return*** *被删除的元素  
 \*/* public synchronized T remove(int index){  
 return elements.remove(index);  
 }  
 */\*\*  
 \* 获取容器里元素的个数  
 \** ***@return*** *元素个数  
 \*/* public synchronized int size(){  
 return elements.size();  
 }  
 */\*\*  
 \* 获取指定下标的元素  
 \** ***@param*** *index 指定下标  
 \** ***@return*** *指定下标的元素  
 \*/* public synchronized T get(int index){  
 return elements.get(index);  
 }  
}

编程题2

SyncQueue1



SyncQueue2



package chapter30code;  
  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.Collections;  
import java.util.List;  
  
*/\*\*  
 \* 一个线程安全同步队列，模拟多线程环境下的生产者消费者机制  
 \* 一个生产者线程通过 produce 方法向队列里产生元素  
 \* 一个消费者线程通过 consume 方法从队列里消费元素  
 \** ***@param*** <*T*> *元素类型  
 \*/*public class SyncQueue1<T> {  
 */\*\*  
 \* 保存队列元素  
 \*/* private ArrayList<T> list = new ArrayList<>();  
  
 */\*\*  
 \* produce和consume互斥锁  
 \*/* private final Object lock = new Object();  
  
  
//*TODO 这里加入需要的数据成员  
 /\*\*  
 \* 生产数据  
 \** ***@param*** *elements 生产出的元素列表，需要将该列表元素放入队列  
 \** ***@throws*** *InterruptedException  
 \*/* public void produce(List<T> elements) throws InterruptedException {  
//*TODO 这里需要实现代码* synchronized (lock){  
 while(list.size() != 0){  
 lock.wait();  
 }  
 list.addAll(elements);  
 System.*out*.println("Produce elements: " + elements);  
 lock.notifyAll();  
 }  
 }  
 */\*\*  
 \* 消费数据  
 \** ***@return*** *从队列中取出的数据  
 \** ***@throws*** *InterruptedException  
 \*/* public List<T> consume() throws InterruptedException {  
//*TODO 这里需要实现代码* synchronized (lock){  
 while(list.size() == 0){  
 lock.wait();  
 }  
 List<T> ret = new ArrayList<>(list);  
 list.clear();  
 lock.notifyAll();  
 System.*out*.println("Consume elements: " + ret);  
 return ret;  
 }  
 }  
}

package chapter30code;  
  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.Collections;  
import java.util.List;  
  
*/\*\*  
 \* 一个线程安全同步队列，模拟多线程环境下的生产者消费者机制  
 \* 一个生产者线程通过 produce 方法向队列里产生元素  
 \* 一个消费者线程通过 consume 方法从队列里消费元素  
 \** ***@param*** <*T*> *元素类型  
 \*/*public class SyncQueue2<T> {  
 */\*\*  
 \* 保存队列元素  
 \*/* private ArrayList<T> list = new ArrayList<>();  
  
 */\*\*  
 \* produce和consume互斥锁  
 \*/* private final Object lock = new Object();  
  
  
//*TODO 这里加入需要的数据成员  
 /\*\*  
 \* 生产数据  
 \** ***@param*** *elements 生产出的元素列表，需要将该列表元素放入队列  
 \** ***@throws*** *InterruptedException  
 \*/* public void produce(List<T> elements) throws InterruptedException {  
//*TODO 这里需要实现代码* synchronized (lock){  
 list.addAll(elements);  
 System.*out*.println("Produce elements: " + elements);  
 lock.notifyAll();  
 }  
 }  
 */\*\*  
 \* 消费数据  
 \** ***@return*** *从队列中取出的数据  
 \** ***@throws*** *InterruptedException  
 \*/* public List<T> consume() throws InterruptedException {  
//*TODO 这里需要实现代码* synchronized (lock){  
 while(list.size() == 0){  
 lock.wait();  
 }  
 List<T> ret = new ArrayList<>(list);  
 list.clear();  
 lock.notifyAll();  
 System.*out*.println("Consume elements: " + ret);  
 return ret;  
 }  
 }  
}