班级：软工11602 姓名：吴思特 序号： 21

上机1 线程调试

**一、上机目的**

1、了解线程的基本过程；

2、对三种情况的线程进行各种调试；

3、调试三种线程的错误；

**二、上机准备**

软件：eclipse；

**三、上机内容**

调试代码：

**ThreadMain.java:**

package com.ym.test;

import java.util.ArrayList;

import java.util.List;

public class ThreadMain {

public static void main(String[] args) {

//第一个的演示示例

List<String> list = new ArrayList<>(1000);

for(int cnt=0;cnt<1000;cnt++){

list.add("abc");

}

ThreadSafeDemo1 demo1 = new ThreadSafeDemo1(list);

ThreadSafeDemo1 demo2 = new ThreadSafeDemo1(list);

Thread thread1 = new Thread(demo1);

Thread thread2 = new Thread(demo2);

thread1.run();

thread2.run();

//第二个的演示示例

/\* List<String> list = new ArrayList<>();

ThreadSafeDemo2 demo1 = new ThreadSafeDemo2(list);

ThreadSafeDemo2 demo2 = new ThreadSafeDemo2(list);

ThreadSafeDemo2 demo3 = new ThreadSafeDemo2(list);

ThreadSafeDemo2 demo4 = new ThreadSafeDemo2(list);

ThreadSafeDemo2 demo5 = new ThreadSafeDemo2(list);

ThreadSafeDemo2 demo6 = new ThreadSafeDemo2(list);

ThreadSafeDemo2 demo7 = new ThreadSafeDemo2(list);

ThreadSafeDemo2 demo8 = new ThreadSafeDemo2(list);

ThreadSafeDemo2 demo9 = new ThreadSafeDemo2(list);

ThreadSafeDemo3 demo10 = new ThreadSafeDemo3(list);

new Thread(demo1).start();

new Thread(demo2).start();

new Thread(demo3).start();

new Thread(demo4).start();

new Thread(demo5).start();

new Thread(demo6).start();

new Thread(demo7).start();

new Thread(demo8).start();

new Thread(demo9).start();

new Thread(demo10).start();\*/

//第三个的演示示例

/\* List<String> list = new ArrayList<>(10);

for(int cnt=0;cnt<10;cnt++){

list.add("abc");

}

ThreadSafeDemo3 demo1 = new ThreadSafeDemo3(list);

ThreadSafeDemo3 demo2 = new ThreadSafeDemo3(list);

Thread thread1 = new Thread(demo1);

Thread thread2 = new Thread(demo2);

thread1.start();

thread2.start();\*/

}

}

**ThreadSafeDemo1.java:**

package com.ym.test;

import java.util.Iterator;

import java.util.List;

import java.util.Random;

import java.util.concurrent.TimeUnit;

public class ThreadSafeDemo1 implements Runnable {

private List<String> list;

public ThreadSafeDemo1(List<String> list){

this.list = list;

}

@Override

public void run() {

Random random = new Random();

System.out.println("线程："+ random.nextInt());

synchronized(this){

Iterator<String> iterator = list.iterator();

int cnt = 0;

while(iterator.hasNext()){

iterator.next();

iterator.remove();

cnt++;

try {

TimeUnit.MILLISECONDS.sleep(10);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

System.out.println("移除元素的数量："+cnt);

}

}

}

**ThreadSafeDemo2.java:**

**package** com.ym.test;

**import** java.util.List;

**public** **class** ThreadSafeDemo2 **implements** Runnable {

**private** List<String> list;

**public** ThreadSafeDemo2(List<String> list){

**this**.list = list;

}

@Override

**public** **void** run() {

**while**(**true**){

list.add("版权归作者所有，任何形式转载请联系作者。作者：KY主创们（来自豆瓣来源：https://www.douban.com/note/630084526/不过，专门研究“人猫交流（cat-human communication)”的学者认为这种印象也不完全是精确的——是的，这是一个真实存在的研究学科。研究宠物行为的学科曾经一度被取消，因为人们认为这种研究的本质是一种拟人论（anthropomorphism），但现在研究这个学科的学者已经不再这样认为了（即宠物行为研究有其自身本质的意义）。");

System.***out***.println("添加一个后尺寸："+list.size());

}

}

}

**ThreadSafeDemo3.java:**

**package** com.ym.test;

**import** java.util.List;

**public** **class** ThreadSafeDemo3 **implements** Runnable {

**private** List<String> list;

**public** ThreadSafeDemo3(List<String> list){

**this**.list = list;

}

@Override

**public** **void** run() {

**while**(**true**){

**if**(list.size()>0){

list.remove(0);

System.***out***.println("删除一个后尺寸："+list.size());

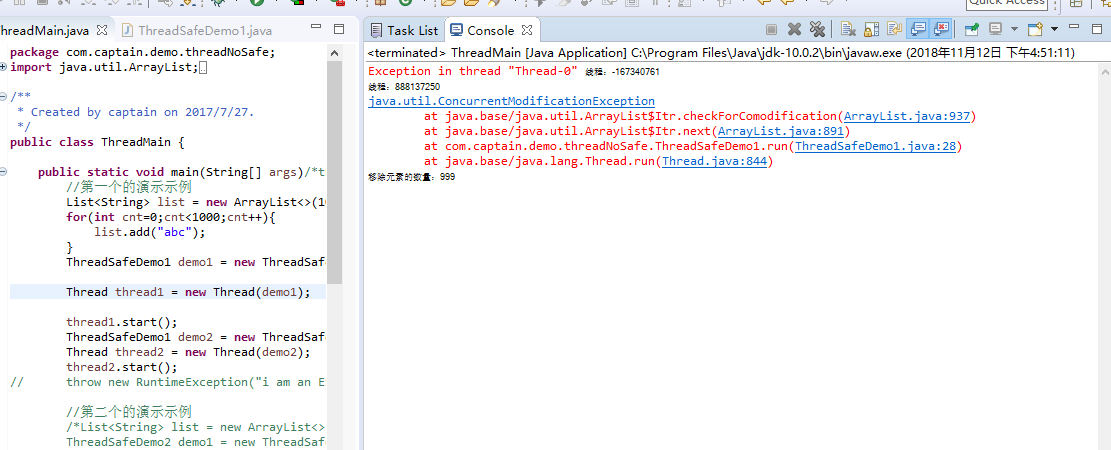
}

}

}

}

异常：



异常原因：

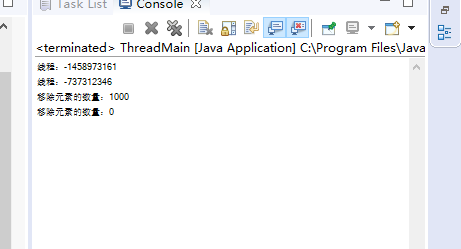
**Synchronized，**代表这个方法加锁,相当于不管哪一个线程（例如线程A），运行到这个方法时,都要检查有没有其它线程B（或者C、D等）正在用这个方法，有的话要等正在使用synchronized方法的线程B（或者C、D）运行完这个方法后再运行此线程A,没有的话,直接运行。它包括两种用法：synchronized方法和synchronized块。

synchronized方法控制对类成员变量的访问：每个类实例对应一把锁，每个synchronized方法都必须获得调用该方法的类实例的锁方能执行，否则所属[线程阻塞](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=71644860&ss_c=ssc.citiao.link" \t "https://baike.sogou.com/_blank)，方法一旦执行，就独占该锁，直到从该方法返回时才将锁释放，此后被阻塞的线程方能获得该锁，重新进入可执行状态。这种机制确保了同一时刻对于每一个类实例，其所有声明为synchronized的成员函数中至多只有一个处于可执行状态（因为至多只有一个能够获得该类实例对应的锁），从而有效避免了类成员变量的访问冲突（只要所有可能访问类成员变量的方法均被声明为synchronized）。

异常修改：

将**Synchronized（this）改为Synchronized（list），让函数访问的成员变量变为一个数组而不是两个变量**

**修改后的结果：**



**四、上机心得**

通过这次学习，我知道了接口的作用，**也知道了Synchronized的用法是什么**