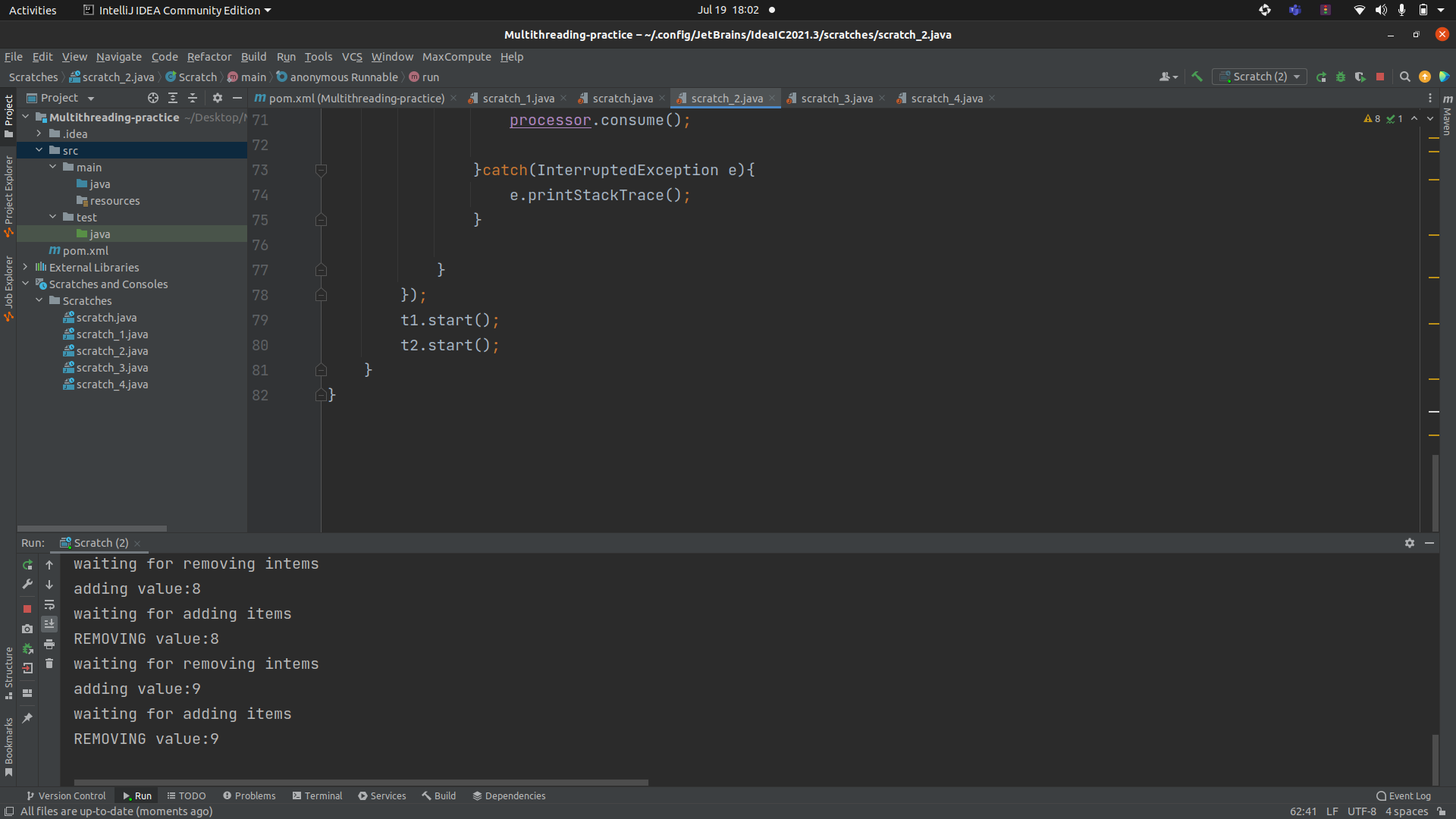
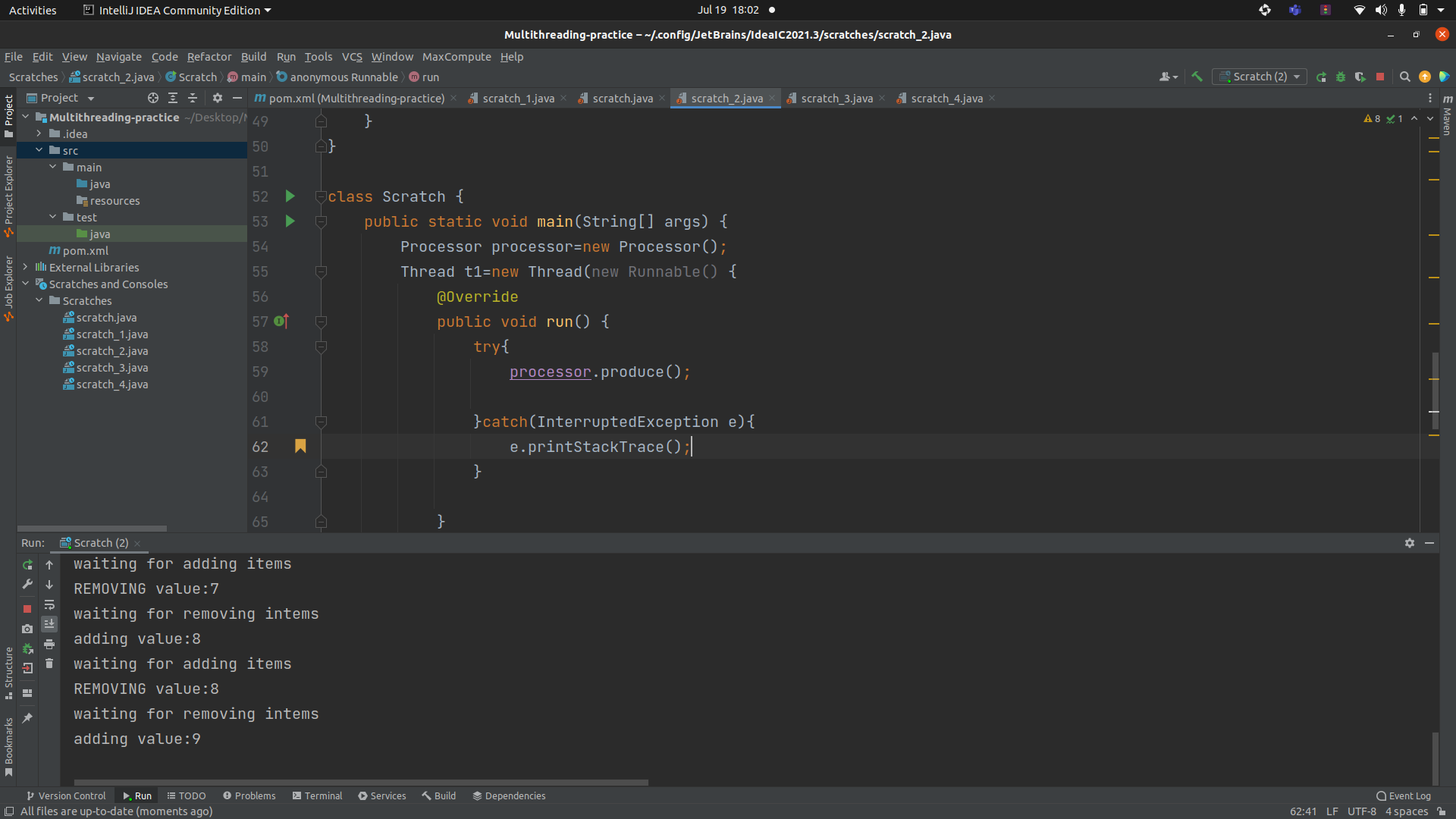
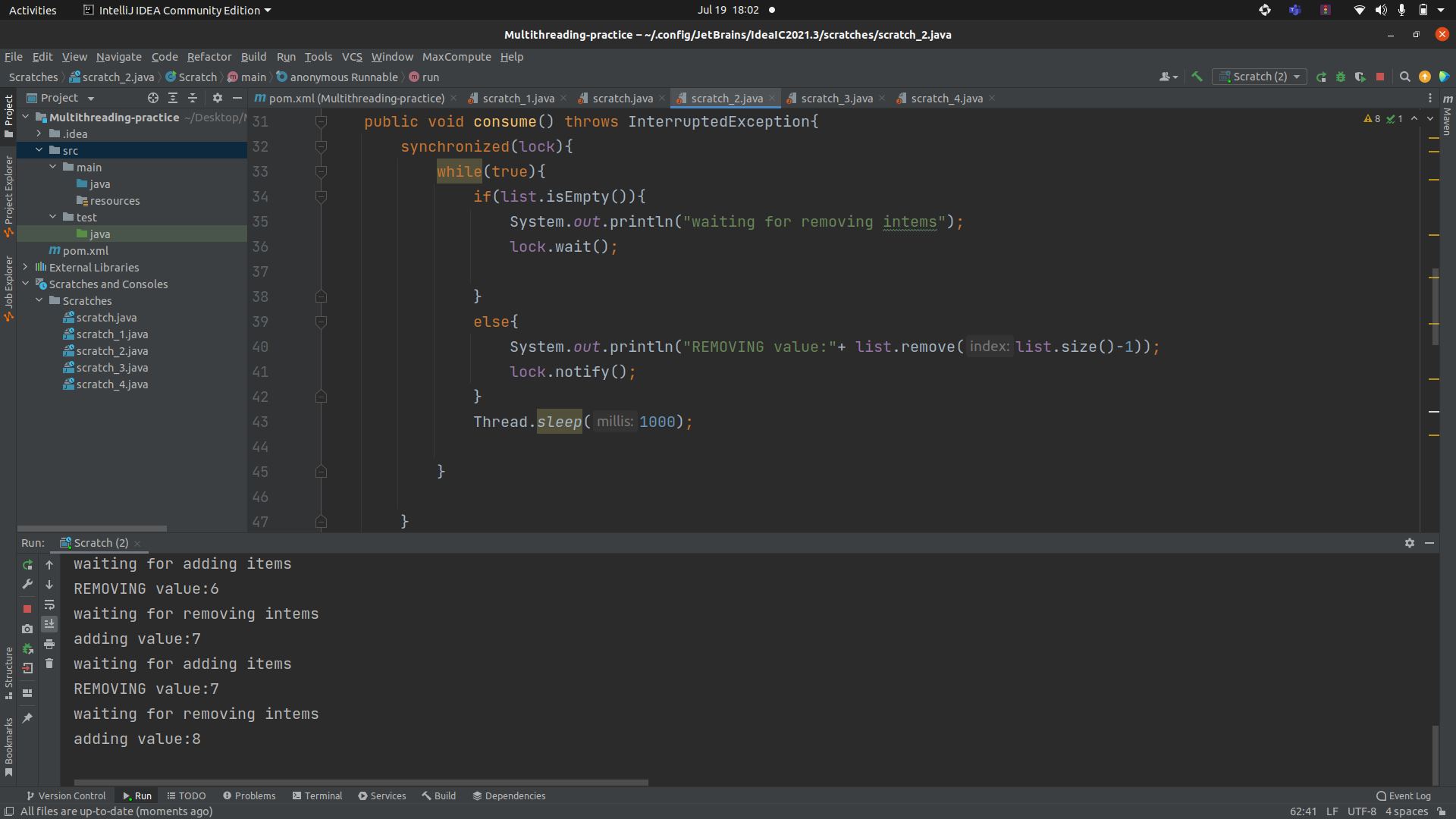
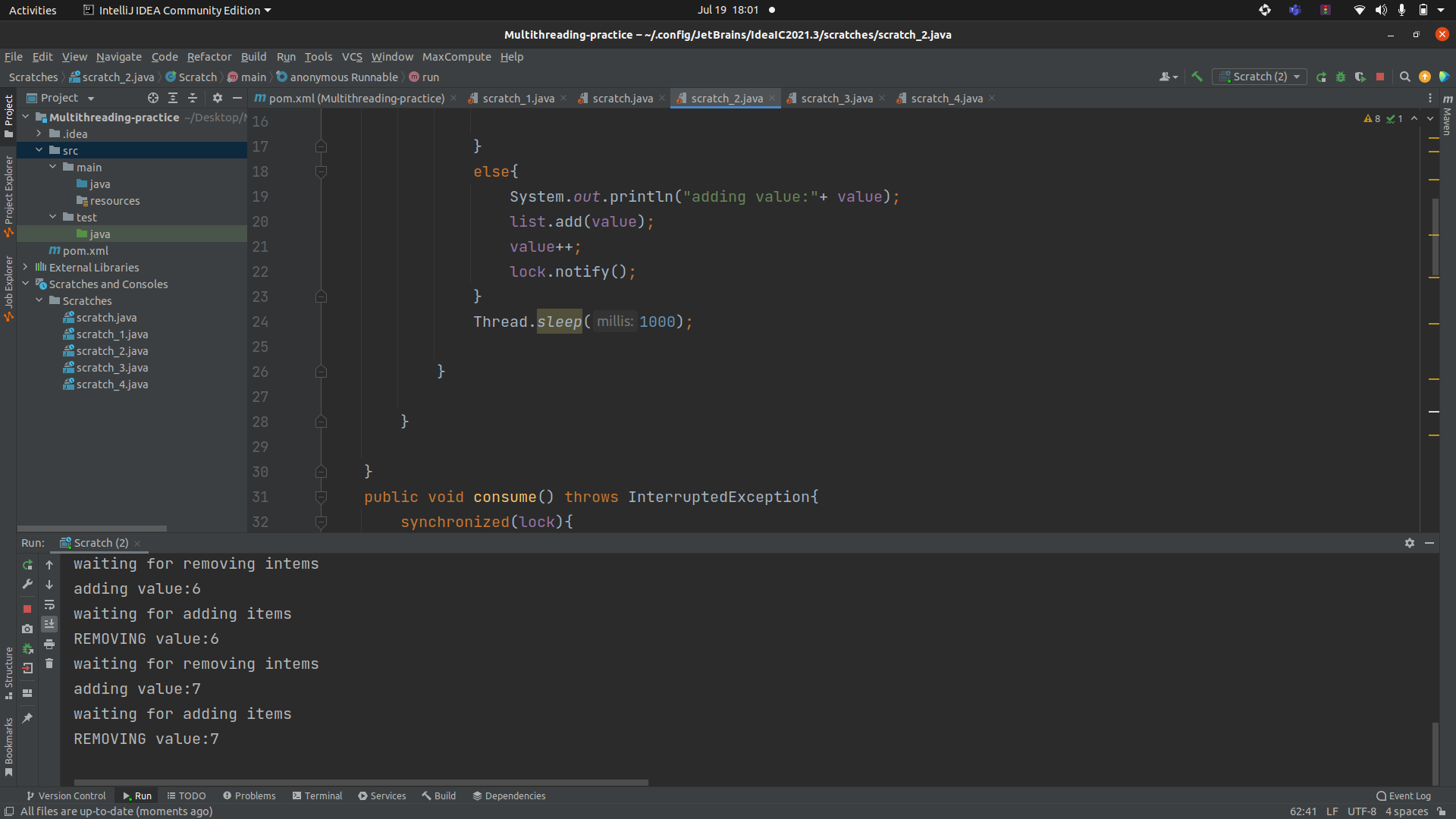
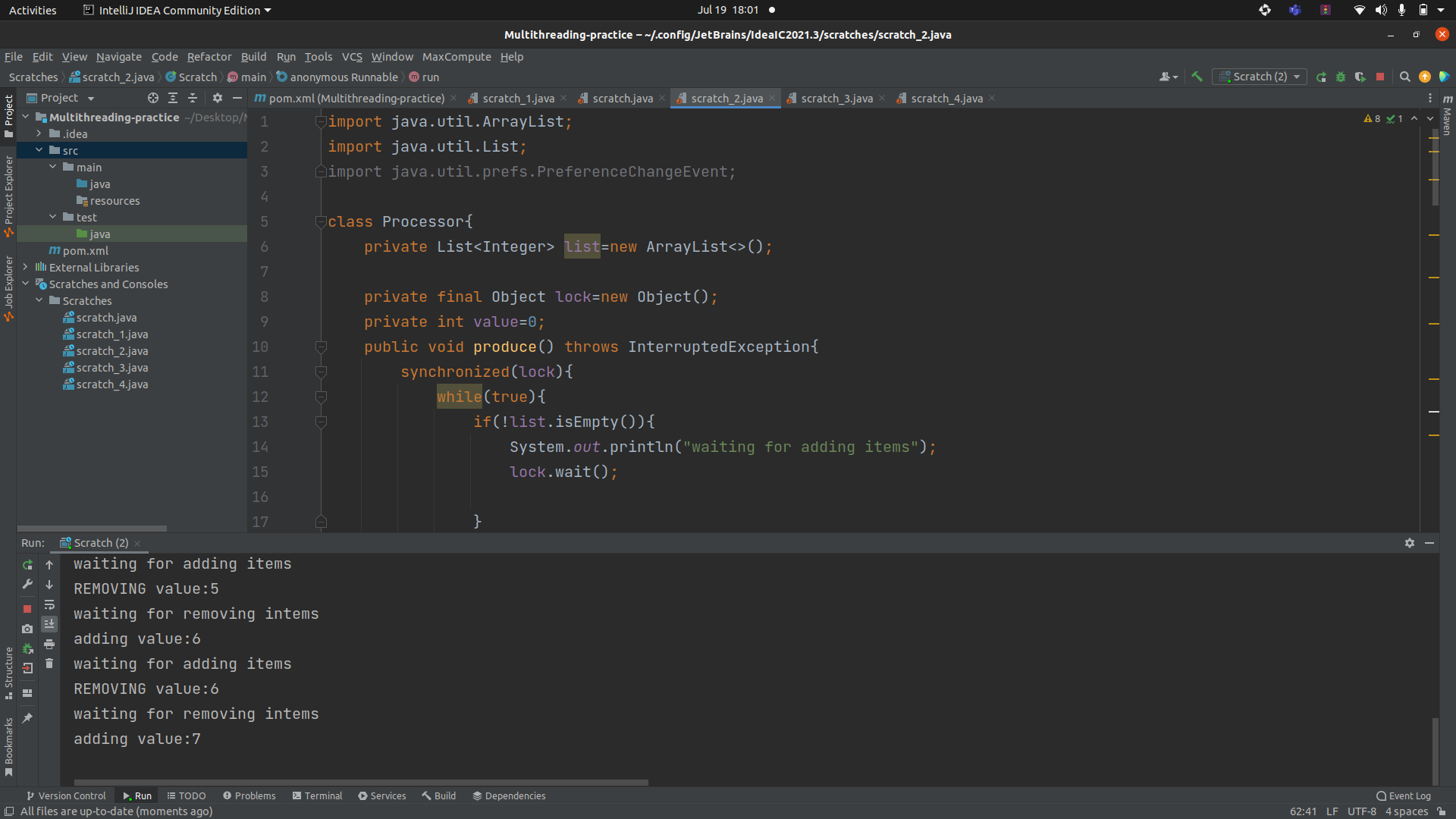
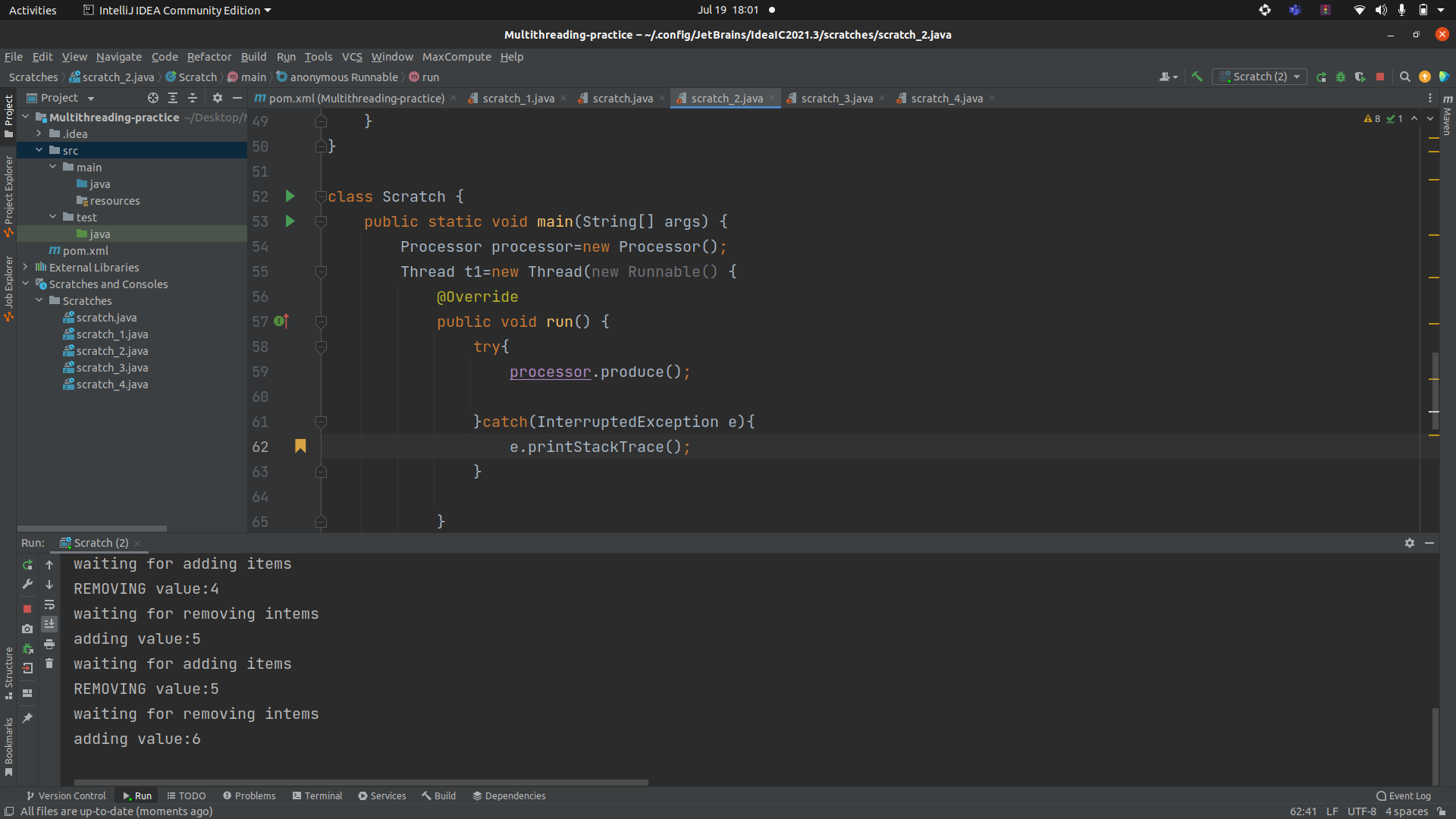
**Advance java track - Assignment-1**

**Ans 1.**

import java.util.ArrayList;  
import java.util.List;  
import java.util.prefs.PreferenceChangeEvent;  
  
class Processor{  
 private List<Integer> list=new ArrayList<>();  
  
 private final Object lock=new Object();  
 private int value=0;  
 public void produce() throws InterruptedException{  
 synchronized(lock){  
 while(true){  
 if(!list.isEmpty()){  
 System.*out*.println("waiting for adding items");  
 lock.wait();  
  
 }  
 else{  
 System.*out*.println("adding value:"+ value);  
 list.add(value);  
 value++;  
 lock.notify();  
 }  
 Thread.*sleep*(1000);  
  
 }  
  
 }  
  
 }  
 public void consume() throws InterruptedException{  
 synchronized(lock){  
 while(true){  
 if(list.isEmpty()){  
 System.*out*.println("waiting for removing intems");  
 lock.wait();  
  
 }  
 else{  
 System.*out*.println("REMOVING value:"+ list.remove(list.size()-1));  
 lock.notify();  
 }  
 Thread.*sleep*(1000);  
  
 }  
  
 }  
  
 }  
}  
  
class Scratch {  
 public static void main(String[] args) {  
 Processor processor=new Processor();  
 Thread t1=new Thread(new Runnable() {  
 @Override  
 public void run() {  
 try{  
 processor.produce();  
  
 }catch(InterruptedException e){  
 e.printStackTrace();  
 }  
  
 }  
 });  
 Thread t2=new Thread(new Runnable() {  
 @Override  
 public void run() {  
 try{  
 processor.consume();  
  
 }catch(InterruptedException e){  
 e.printStackTrace();  
 }  
  
 }  
 });  
 t1.start();  
 t2.start();  
 }  
}

**Output screen:**



**Ans 2.**

import java.util.\*;  
class ThreadX implements Runnable{  
 int num = 0;  
 public void positiveEven(){  
 int max=30, min=-10;  
 Random rand = new Random();  
 num = rand.nextInt(max - min) + min;  
 if(num%2==0){  
 if(num>= 0){  
 System.*out*.println("Thread X with even num = "+ num);  
 }  
 else{  
 num \*= -1;  
 System.*out*.println("Thread X with even num = "+ num);  
 }  
 } else{  
 if(num>=0){  
  
 System.*out*.println(" Thread X with i even num = "+ (num+1));  
 }  
 else {  
 num \*= -1;  
 System.*out*.println(" Thread X with i even num = "+ (num+1));  
 }  
  
 }  
 }  
  
 public void run( ) {  
  
 for(int i = 1; i <= 5; i++) {  
  
 try {  
 positiveEven();  
 Thread.*sleep*(2000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 System.*out*.println("Exiting Thread X ...");  
 }  
}  
class ThreadY implements Runnable{  
 int num = 0;  
 public void positiveOdd(){  
 int max=30, min=-10;  
 Random rand = new Random();  
 num = rand.nextInt(max - min) + min;  
 if(num%2!=0){  
 if(num>= 0){  
 System.*out*.println("Thread y with odd num = "+ num);  
 }  
 else{  
 num \*= -1;  
 System.*out*.println("Thread y with odd num = "+ num);  
 }  
 } else{  
 if(num>=0){  
  
 System.*out*.println(" Thread y with i odd num = "+ (num+1));  
 }  
 else {  
 num \*= -1;  
 System.*out*.println(" Thread y with i odd num = "+ (num+1));  
 }  
  
 }  
  
  
 }  
  
 public void run( ) {  
 for(int j = 1; j <= 5; j++) {  
  
 try {  
 positiveOdd();  
 Thread.*sleep*(2000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
// System.out.println("Thread Y with j = "+ j);  
 }  
 System.*out*.println("Exiting Thread Y ...");  
 }  
}  
  
class ThreadZ implements Runnable{  
 int num=0;  
 public void negative(){  
 int max=30, min=-10;  
 Random rand = new Random();  
 num = rand.nextInt(max - min) + min;  
 if(num>=0) {  
 num \*= -1;  
 System.*out*.println("Thread Z with k with -ve num = " + num);  
 }  
 else{  
 System.*out*.println("Thread Z with k with -ve num = " + num);  
 }  
 }  
  
  
 public void run( ) {  
 for(int k = 1; k <= 5; k++) {  
 try {  
 negative();  
 Thread.*sleep*(2000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 System.*out*.println("Exiting Thread Z ...");  
 }  
}  
class Scratch{  
 public static void main(String args[]) {  
 ThreadX x = new ThreadX();  
 Thread t1 = new Thread(x);  
 ThreadY y = new ThreadY();  
 Thread t2 = new Thread(y);  
 ThreadZ z = new ThreadZ();  
 Thread t3 = new Thread(z);  
 t1.start();  
 t2.start();  
 t3.start();  
 try {  
 t1.join();  
 t2.join();  
 t3.join();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 System.*out*.println("... Multithreading is over ");  
 }  
}

**Output screen:**

