

**The course “Application development with Java”**

Practical assignment 2

Student name: Roberts

Student surname: Dubovskis

St. code: 62085

**RIGA**

**2022**

Contents

[Exercise 1 3](#_Toc101020797)

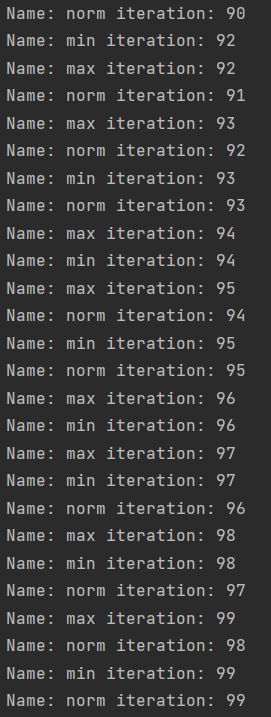
[Exercise 2 4](#_Toc101020798)

[Exercise 3 5](#_Toc101020799)

[Exercise 4 7](#_Toc101020800)

[Conclusion 9](#_Toc101020801)

# Exercise 1



Img.1 (result)

**PriorityMain.java**

package lv.tsi.priority.task1;  
  
public class PriorityMain {  
  
 public static void main(String[] args) {  
 PriorityThread t1 = new PriorityThread("min");  
 PriorityThread t2 = new PriorityThread("norm");  
 PriorityThread t3 = new PriorityThread("max");  
  
 t1.setPriority(2);  
 t2.setPriority(4);  
 t3.setPriority(8);  
  
 t1.start();  
 t2.start();  
 t3.start();  
 }  
  
}

**PriorityThread.java**

package lv.tsi.priority.task1;  
  
public class PriorityThread extends Thread{  
 public PriorityThread(String name) {  
 super(name);  
 }  
  
 @Override  
 public void run(){  
 for (int i = 0; i < 100; i++) {  
 System.*out*.printf("Name: %s iteration: %d \n",getName(),i);  
 try {  
 Thread.*sleep*(10);  
 } catch (InterruptedException e) { }  
 }  
 }  
}

# Exercise 2

Img.2 (result)

**SyncMain.java**

package lv.tsi.sync;  
public class SyncMain {  
 public static void main(String[] args) throws InterruptedException {  
 final TaskManager tm1 = new TaskManager();  
 TaskThread t1 = new TaskThread("First", 1, tm1);  
 TaskThread t2 = new TaskThread("Second", 2, tm1);  
  
 t1.start();  
 Thread.*sleep*(2);  
 t2.start();  
 }  
}

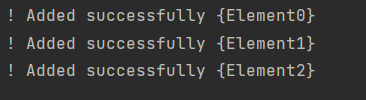
**TaskManager.java**

package lv.tsi.sync;  
  
public class TaskManager {  
 synchronized void performTask(int taskId){  
 System.*out*.printf("name: %s value of taskId: %d\n", Thread.*currentThread*().getName(), taskId);  
 try {  
 Thread.*sleep*(4);  
 } catch (InterruptedException e) { }  
 System.*out*.printf("name: %s value of taskId: %d\n", Thread.*currentThread*().getName(), taskId);  
 }  
  
 // "synchronized" makes that first go first Thread then second. No random.  
}

**TaskThread.java**

package lv.tsi.sync;  
  
public class TaskThread extends Thread{  
 private final TaskManager taskManager;  
 private final int taskId;  
 public TaskThread(String name,int taskId, TaskManager taskManager) {  
 super(name);  
 this.taskId = taskId;  
 this.taskManager = taskManager;  
 }  
  
 @Override  
 public void run(){  
 taskManager.performTask(taskId);  
 }  
}

# Exercise 3



Img.3 (result)

**BlockingMain.java**

package lv.tsi.queue;  
  
import java.util.concurrent.BlockingQueue;  
import java.util.concurrent.LinkedBlockingDeque;  
  
public class BlockingMain {  
  
 public static void main(String[] args) {  
 final BlockingQueue<String> blockingQueue = new LinkedBlockingDeque<>();  
 AddingThread t1 = new AddingThread(blockingQueue);  
 TakingThread t2 = new TakingThread(blockingQueue);  
 t1.start();  
 t2.start();  
  
 }  
  
}

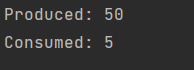
**TakingThread.java**

package lv.tsi.queue;  
  
import java.util.concurrent.BlockingQueue;  
import java.util.concurrent.LinkedBlockingDeque;  
  
public class TakingThread extends Thread {  
 private BlockingQueue<String> blockingQueue1 = new LinkedBlockingDeque<>();  
  
 public TakingThread(BlockingQueue<String> blockingQueue) {  
 this.blockingQueue1 = blockingQueue;  
 }  
  
 // Task don't say to fill BlockingQueue and it is empty.  
 //6. Rewrite method run:  
 //a. Call sleep for 1 second  
 //b. Take the element from the queue and output information to the screen  
 @Override  
 public void run() {  
 try {  
 Thread.*sleep*(1000);  
 } catch (InterruptedException e) {  
 }  
 for (int i = 0; i < 3; i++) {  
 var a = blockingQueue1;  
 for (int j = 0; j < 3; j++) {  
 if (!a.isEmpty()) {  
 if (!(a.peek().contains("Element" + i))) {  
 a.poll();  
 } else {  
 System.*out*.printf("Added successfully {%s}\n", a.element());  
 break;  
 }  
 }  
 }  
  
 }  
 }  
}

**AddingThread.java**

package lv.tsi.queue;  
  
import java.util.concurrent.BlockingQueue;  
import java.util.concurrent.LinkedBlockingDeque;  
  
public class AddingThread extends Thread {  
 private BlockingQueue<String> blockingQueue = new LinkedBlockingDeque<>();  
  
 public AddingThread(BlockingQueue<String> blockingQueue) {  
 this.blockingQueue = blockingQueue;  
 }  
  
 @Override  
 public void run() {  
 for (int i = 0; i < 3; i++) {  
 blockingQueue.add("Element" + i);  
 var a = blockingQueue;  
 for (int j = 0; j < 3; j++) {  
 if (!a.isEmpty()) {  
 if (!(a.peek().contains("Element" + i))) {  
 a.poll();  
 } else {  
 System.out.printf("! Added successfully {%s}\n", a.element());  
 break;  
 }  
 }  
 }  
  
 }  
 }  
}

# Exercise 4



Img.4 (result)

**ProducerConsumerMain.java**

package lv.tsi.producer\_consumer;  
  
public class ProducerConsumerMain {  
 public static void main(String[] args) {  
 Goods g1 =new Goods();  
 Consumer c1 =new Consumer(g1);  
 Producer p1 =new Producer(g1);  
 c1.start();  
 p1.start();  
  
 }  
}

**Producer.java**

package lv.tsi.producer\_consumer;  
  
public class Producer extends Thread{  
 Goods goods = new Goods();  
  
 public Producer(Goods goods) {  
 this.goods = goods;  
 }  
 @Override  
 public void run(){  
 while (true){  
 goods.produce(50);  
 System.*out*.println("Produced: 50");  
 }  
 }  
}

**Goods.java**

package lv.tsi.producer\_consumer;  
  
public class Goods {  
 private int amount = 0;  
  
 synchronized void produce(int newAmount) {  
 while (amount > 0) {  
 try {  
 wait();  
 } catch (InterruptedException e) {  
 }  
 }  
 amount = newAmount;  
 notify();  
  
  
 }  
  
 synchronized int consume() {  
 while (amount < 1) {  
 try {  
 wait();  
 } catch (InterruptedException e) {  
 }  
 }  
 amount-=5;  
 notify();  
 return 5;  
 }  
  
  
}

**Consumer.java**

package lv.tsi.producer\_consumer;  
  
public class Consumer extends Thread {  
 Goods goods = new Goods();  
  
 public Consumer(Goods goods) {  
 this.goods = goods;  
 }  
 @Override  
 public void run() {  
 System.*out*.println("Consumed: " + goods.consume());  
 }  
}

# Conclusion

All tasks have been accomplished.