**Operating Systems and Concurrency**

**Student Name:** Raphael Salaja

**AIT Student number:** A00269349

**Assignment:** Assignment 4

**Date of Submission:** 16/11/2021

**FORM A1**

# STUDENT PLAGIARISM DISCLAIMER FORM

****

## PLAGIARISM DISCLAIMER

STUDENT NAME: Raphael Salaja

STUDENT NUMBER: A00269349

PROGRAMME: BSc (Hons) in Software Design with Virtual Reality and Gaming

YEAR: 3

MODULE: Operating Systems and Concurrency

LECTURER: Thiago Braga Rodrigues

ASSIGNMENT TITLE: Lab 4 Assignment

DUE DATE: 17 November2021

DATE SUBMITTED: 16 November 2021

ADDITIONAL INFORMATION:

I understand that plagiarism is a serious academic offence, and that AIT deals with it according to the AIT Policy on Plagiarism.

I have read and understand the AIT Policy on Plagiarism and I agree to the requirements set out therein in relation to plagiarism and referencing. I confirm that I have referenced and acknowledged properly all sources used in preparation of this assignment. I understand that if I plagiarise, or if I assist others in doing so, that I will be subject to investigation as outlined in the AIT Policy on Plagiarism.

I understand and agree that plagiarism detection software may be used on my assignment. I declare that, except where appropriately referenced, this assignment is entirely my own work based on my personal study/or research. I further declare that I have not engaged the services of another to either assist in, or complete this assignment.

## Signed:Raphael Salaja

**Dated: 10 November 2021**

# Contents

[STUDENT PLAGIARISM DISCLAIMER FORM 2](#_Toc87445626)

[PLAGIARISM DISCLAIMER 2](#_Toc87445627)

[Signed:Raphael Salaja 2](#_Toc87445628)

[Contents 3](#_Toc87445629)

[Setup 4](#_Toc87445630)

[Main 4](#_Toc87445631)

[Consumer 4](#_Toc87445632)

[Producer 4](#_Toc87445633)

[Output 4](#_Toc87445634)

[Question 1 4](#_Toc87445635)

[Main 4](#_Toc87445636)

[Consumer 4](#_Toc87445637)

[Producer 5](#_Toc87445638)

[Output 6](#_Toc87445639)

[Question 2 7](#_Toc87445640)

[Output 7](#_Toc87445641)

[Question 3 8](#_Toc87445642)

[Producer 8](#_Toc87445643)

[Output 8](#_Toc87445644)

[Question 4 9](#_Toc87445645)

[Main 9](#_Toc87445646)

[Consumer 9](#_Toc87445647)

[Producer 9](#_Toc87445648)

[Output 9](#_Toc87445649)

[Question 5 10](#_Toc87445650)

[Main 10](#_Toc87445651)

[Consumer 10](#_Toc87445652)

[Producer 10](#_Toc87445653)

[Output 10](#_Toc87445654)

# Setup

Create a ArrayBlockingQueue size 10 Create a Producer thread. Create a Consumer thread. Start both threads in Main.java and see the output.

## Main

package ie.ait.week5.blockingqueue;  
import java.util.concurrent.ArrayBlockingQueue;  
import java.util.concurrent.*BlockingQueue*;  
public class Main {  
 public static void main(String[] args) {  
 *BlockingQueue*<Integer> buffer = new ArrayBlockingQueue<Integer>(10);  
 Thread p = new Producer(buffer);  
 p.start();  
 Thread c = new Consumer(buffer);  
 c.start();  
 }  
}

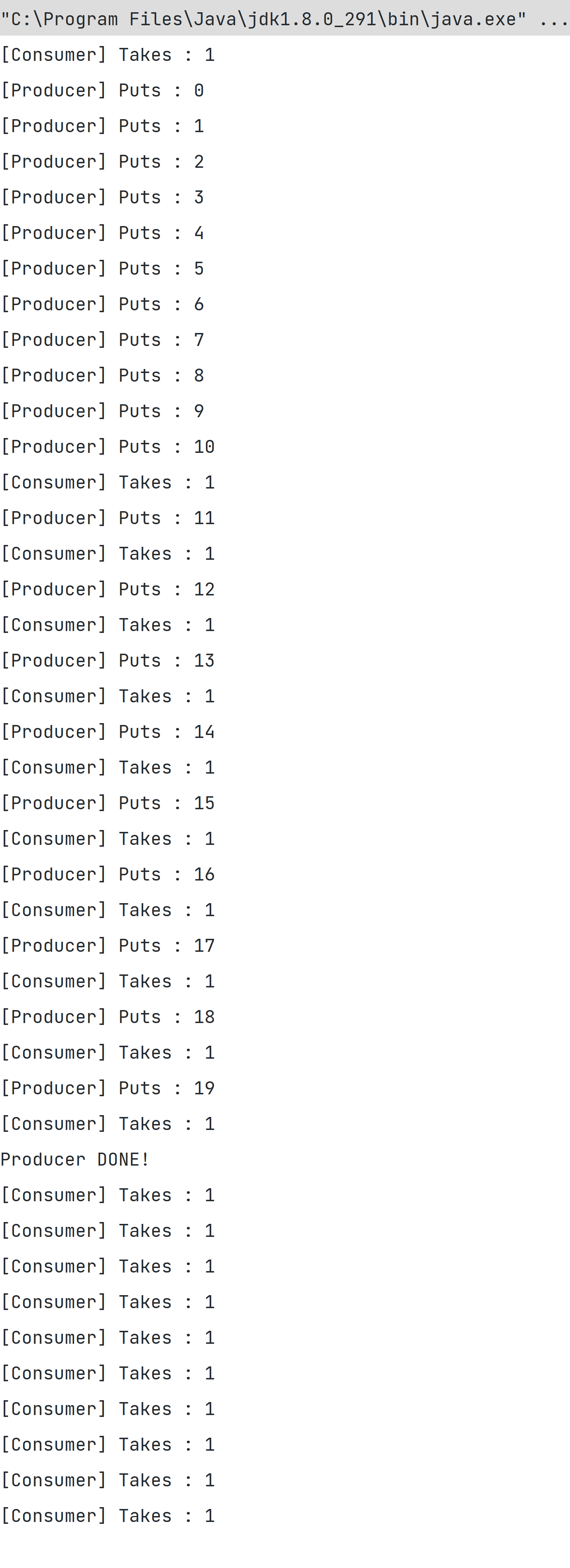
## Consumer

package ie.ait.week5.blockingqueue;  
import java.util.concurrent.*BlockingQueue*;  
public class Consumer extends Thread implements *Runnable* {  
 private final *BlockingQueue*<Integer> queue;  
 @Override  
 public void run() {  
 while (true) {  
 try {  
 process(queue.take());   
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 private void process(Integer take) throws InterruptedException {  
 System.*out*.println("[Consumer] Takes : " + take);  
 Thread.*sleep*(500);  
 }  
 public Consumer(*BlockingQueue*<Integer> queue) {  
 this.queue = queue;  
 }  
}

## Producer

package ie.ait.week5.blockingqueue;  
import java.util.concurrent.*BlockingQueue*;  
public class Producer extends Thread implements *Runnable* {  
 private final *BlockingQueue*<Integer> queue;  
 @Override  
 public void run() {  
 try {  
 process();  
 } catch (InterruptedException e) {  
 Thread.*currentThread*().interrupt();  
 }  
 }  
 private void process() throws InterruptedException {  
 try {  
 for (int i = 0; i < 20; i++) {  
 System.*out*.println("[Producer] Puts : " + i);  
 queue.put(i);  
 try {  
 *sleep*((int) (Math.*random*() \* 100));   
 } catch (InterruptedException e) {  
 }  
 }  
 System.*out*.println("Producer DONE! ");  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 public Producer(*BlockingQueue*<Integer> queue) {  
 this.queue = queue;  
 }  
}

## Output



# Question 1

Run with a fast Producer and slow Consumer. What can you notice in this case?.

## Main

package ie.ait.week5.blockingqueue;  
import java.util.concurrent.ArrayBlockingQueue;  
import java.util.concurrent.*BlockingQueue*;  
public class Main {  
 public static void main(String[] args) {  
 *BlockingQueue*<Integer> buffer = new ArrayBlockingQueue<Integer>(10);  
 Thread p = new Producer(buffer);  
 p.start();  
 Thread c = new Consumer(buffer);  
 c.start();  
 }  
}

## Consumer

package ie.ait.week5.blockingqueue;  
import java.util.concurrent.*BlockingQueue*;  
public class Consumer extends Thread implements *Runnable* {  
 private final *BlockingQueue*<Integer> queue;  
 @Override  
 public void run() {  
 while (true) {  
 try {  
 process(queue.take());  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 private void process(Integer take) throws InterruptedException {  
 System.*out*.println("[Consumer] Takes : " + take);  
 Thread.*sleep*(100);  
 }  
 public Consumer(*BlockingQueue*<Integer> queue) {  
 this.queue = queue;  
 }  
}

## Producer

package ie.ait.week5.blockingqueue;  
import java.util.concurrent.*BlockingQueue*;  
public class Producer extends Thread implements *Runnable* {  
 private final *BlockingQueue*<Integer> queue;  
 @Override  
 public void run() {  
 try {  
 process();  
 } catch (InterruptedException e) {  
 Thread.*currentThread*().interrupt();  
 }  
 }  
 private void process() throws InterruptedException {  
 try {  
 for (int i = 0; i < 20; i++) {  
 System.*out*.println("[Producer] Puts : " + i);  
 queue.put(i);  
 try {  
 *sleep*((int) (Math.*random*() \* 100));   
 } catch (InterruptedException e) {  
 }  
 }  
 System.*out*.println("Producer DONE! ");  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 public Producer(*BlockingQueue*<Integer> queue) {  
 this.queue = queue;  
 }  
}

## Output

Table

Description automatically generated with low confidence

# Question 2

Run with slow Producer and fast Consumer. Now make the Consumer sleep for4 seconds between iterations. What is the difference?

The Consumer will have to wait till for producer at each step

# Question 3

Include a parameter in the constructor for the Producer that specifies the startinteger and now loops from this value.

## Producer

package ie.ait.week5.blockingqueue;  
import java.util.concurrent.*BlockingQueue*;  
public class Producer extends Thread implements *Runnable* {  
 private final *BlockingQueue*<Integer> queue;  
 private final int start;  
 @Override  
 public void run() {  
 try {  
 process();  
 } catch (InterruptedException e) {  
 Thread.*currentThread*().interrupt();  
 }  
 }  
 private void process() throws InterruptedException {  
 try {  
 for (int i = start; i < 20 + start; i++) {  
 System.*out*.println("[Producer] Puts : " + i);  
 queue.put(i);  
 try {  
 *sleep*((int) (Math.*random*() \* 1000));  
 } catch (InterruptedException e) {  
 }  
 }  
 System.*out*.println("Producer DONE! ");  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 public Producer(*BlockingQueue*<Integer> queue, int start) {  
 this.queue = queue;  
 this.start = start;  
 }  
}

# Question 4

Start a few Producers and only one Consumer. What happens in this case?

## Main

package ie.ait.week5.blockingqueue;  
import java.util.concurrent.ArrayBlockingQueue;  
import java.util.concurrent.*BlockingQueue*;  
public class Main {  
 public static void main(String[] args) {  
 *BlockingQueue*<Integer> buffer = new ArrayBlockingQueue<Integer>(10);  
 Thread p\_1 = new Producer(buffer, 5);  
 Thread p\_2 = new Producer(buffer, 1);  
 Thread p\_3 = new Producer(buffer, 3);  
 Thread p\_4 = new Producer(buffer, 4);  
 p\_1.start();  
 p\_2.start();  
 p\_3.start();  
 p\_4.start();  
 Thread c = new Consumer(buffer);  
 c.start();  
 }  
}

## Output

A picture containing graphical user interface

Description automatically generatedA picture containing graphical user interface

Description automatically generated

Table

Description automatically generated with medium confidence

# Question 5

Use a method poll instead of the take method. Print the output from that method. What happens is the queue is empty?

The method does not throw an when the Queue is empty, it returns null instead.

## Consumer

package ie.ait.week5.blockingqueue;  
import java.util.concurrent.*BlockingQueue*;  
public class Consumer extends Thread implements *Runnable* {  
 private final *BlockingQueue*<Integer> queue;  
 @Override  
 public void run() {  
 try {  
 Thread.*sleep*(1000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 while (true) {  
 try {  
 Integer x = queue.poll();  
 if (x == null) {  
 System.*out*.println(" CONSUMER DONE ");  
 break;  
 }  
 process(x);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 private void process(Integer take) throws InterruptedException {  
 System.*out*.println("[Consumer] Takes : " + take);  
 Thread.*sleep*(500);  
 }  
 public Consumer(*BlockingQueue*<Integer> queue) {  
 this.queue = queue;  
 }  
}