|  |  |
| --- | --- |
| **Grade:** | X / 100 |
| **Instructor’s Comments:** | To be filled by the instructor |

This lab template is developed per data collection requirement for ABET accreditation process. Your cooperation of using it is highly appreciated. ***Please do not delete this page!***

**Grading Criteria**:

Each lab is worth 100 points. The first 20 points are toward your submission of required documents, and the rest 80 points are for the problem of each lab assignment.

|  |  |  |
| --- | --- | --- |
| **Item** | **Description** | **Points** |
| Required documents | 1) The lab report (this document)  **2) Any source code files (Don’t forget to compress them into a “.zip” file!)** | 20 |
| For all exercises/problems | Completeness of your work  1) Complete solution  2) Inclusion of screenshot of executing your code – you must include at least one whether your code works or not (if applicable)  Correctness of your logic/solution  Coding style (below are common mistakes)  1) Proper alignment of your code  2) Proper naming convention  3) Meaningful naming | 80 |

**ITS 330 – Advanced Operating Systems**

**Final Project**

**Due: May 2nd, 2021**

**Laquon Hamilton**

(***Create a table of contents before your solution***)

Contents

[Problem 1. 3](#_Toc70629066)

[Problem 2. 9](#_Toc70629067)

[Problem 3. 9](#_Toc70629068)

[Problem 4. 9](#_Toc70629069)

[Problem 5. 10](#_Toc70629070)

[Problem 6. 10](#_Toc70629071)

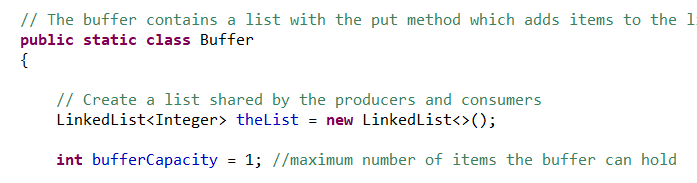
[Problem 7. 10](#_Toc70629072)

[Problem 8. 10](#_Toc70629073)

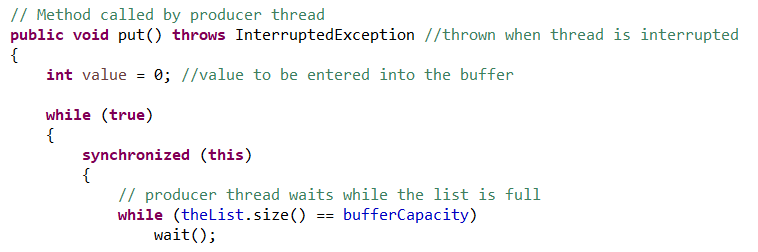
[Problem 9. 10](#_Toc70629074)

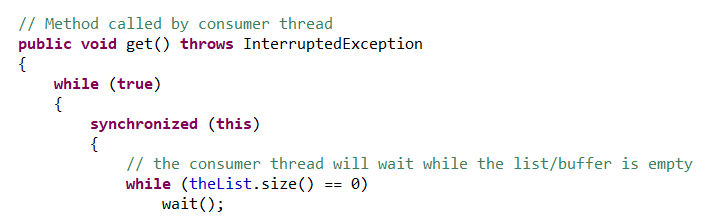
# Problem 1.

**Task 1** – Set the buffer capacity to one value only.



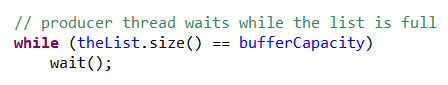
**Task 2 –** Both producers and consumers will wait until they can perform their tasks.

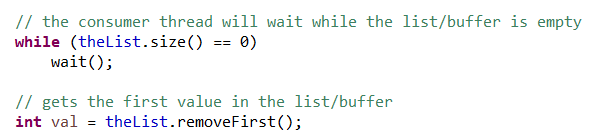




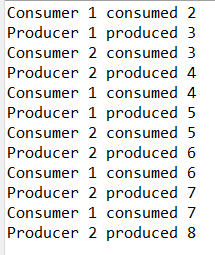
**Task 3 –** The buffer is a first in first out linked list. The producer will only produce when it is empty, and the consumer will only consume when the buffer has values.







**Task 4 –** There are multiple producers and consumers. Each one only accesses the buffer one at a time.





Code:

/\*

 ITS-330

 Final Project

 Problem ---

 4/29/2021

 Laquon Hamilton

\*/

// Java program to implement solution of producer/consumer problem.

import java.util.LinkedList; //used to store values into the buffer

import java.util.concurrent.\*; //used for the execution of the threads

public class BoundedBufferSolution

{

public static void main(String[] args) throws InterruptedException

{

// Object of the buffer that contains the put and get methods

Buffer theBuffer = new Buffer();

ExecutorService theProducers = Executors.newFixedThreadPool(2); //thread pool for the producers

ExecutorService theConsumers = Executors.newFixedThreadPool(2); //thread pool for the consumers

Runnable newProducer = new Runnable() //runnable method for the producers

{

      @Override

      public void run()

      {

        // TODO Auto-generated method stub

try

{

theBuffer.put(); //calls the put method to enter a value into the buffer

}

catch (InterruptedException e)

{

e.printStackTrace();

}

      }

    };

    Runnable newConsumer = new Runnable() //runnable method for the consumers

    {

      @Override

      public void run()

      {

        // TODO Auto-generated method stub

try

{

   //identifies the thread as belonging to the first consumer

theBuffer.get(); //calls the get method to obtain the value from the buffer and sends the

            //thread number to the buffer

}

catch (InterruptedException e)

{

e.printStackTrace();

}

      }

    };

// Start both threads

theProducers.execute(newProducer);

theConsumers.execute(newConsumer);

// Shutdown the threads when they are finished

theProducers.shutdown();

theConsumers.shutdown();

}

public static int getRandNum() //generate random thread numbers for producers and consumers

{

  int temp = (Math.random() <= 0.5) ? 1 : 2;

  return temp;

}

// The buffer contains a list with the put method which adds items to the list and the get method which removes items

public static class Buffer

{

// Create a list shared by the producers and consumers

LinkedList<Integer> theList = new LinkedList<>();

int bufferCapacity = 1; //maximum number of items the buffer can hold

// Method called by producer thread

public void put() throws InterruptedException //thrown when thread is interrupted

{

int value = 0; //value to be entered into the buffer

while (true)

{

synchronized (this)

{

// producer thread waits while the list is full

while (theList.size() == bufferCapacity)

wait();

System.out.println("Producer " + getRandNum() + " produced " + value);

// put values into the list/buffer

theList.add(value++);

// notifies the consumer thread that it can now start consuming

notify();

// Let the thread sleep for enough time to allow for demonstrating the producer consumer process

Thread.sleep(2000);

}

}

}

// Method called by consumer thread

public void get() throws InterruptedException

{

while (true)

{

synchronized (this)

{

// the consumer thread will wait while the list/buffer is empty

while (theList.size() == 0)

wait();

// gets the first value in the list/buffer

int val = theList.removeFirst();

System.out.println("Consumer " + getRandNum() + " consumed " + val);

// Wake up the producer thread

notify();

// Let the thread sleep for enough time to allow for demonstrating the producer consumer process

Thread.sleep(1000);

}

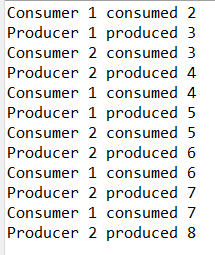
}

}

}

}

Screenshot



# Problem 2.

Code

Screenshot

# Problem 3.

Code:

Screenshot

# Problem 4.

Code

Screenshot

# Problem 5.

Code

Screenshot

# Problem 6.

Code

Screenshot

# Problem 7.

Code

Screenshot

# Problem 8.

Code

Screenshot

# Problem 9.

Code

Screenshot