Ryan Fazal
Prof. Jinwoo Kim
CSCI 375
Project #1

For this project, I had decide to use java to create my project. For this we had to implement a producer and consumer. However the problem that arises, is that the producer may try to add a buffer. This buffer by the producer is adding data with the system. With all this data adage, the memory of the system becomes full. This causes, in this case, the consumer to not be removed data. This is because the consumer wouldn't want to remove data from a buffer that is empty. For this task, we must figure out a way to reduce the amount of data implanted by the buffer of the producer. To do so, we must create a program to where the data implemented does not overfill the buffer. To do so, the producer must be overridden to undergo a sleep function. Or be able on its own, to remove unnecessary data that is just taking over space. Instead the consumer is allowed to add data. And has the right to alert the producer that new data will be able to added. And vice versa. If the producer/consumer didn't undergo this procedure, that may set the program at a deadlock. In order to prevent this, we must first implement a linked list. This list will implement a queue in which all the jobs will be stored depending either the producer or consumer. Second we would implement a variable capacity. This acts as a storage check. This means that the list will be checked if It is indeed full or not. Lastly, we will implement a mechanism that will take out and put in data. We must remember that within the PC class, its added so that the producer/consumer does not fail upon itself. This is if it finds that the list is full at that moment. For the producer, we start off with a value set at zero. For the lab below, I have given it a capacity of 5. This is so that the loop is synchronized between both the producer/consumer methods. However they run one at a time. An additional loop located on the inner part of the program allows the list to be checked, whether it is full or not. If full, the computer is placed in a state of waiting. If it is empty, a value is added. For the consumer thread, it does the exact opposite. It checks to see if the computer is empty. If it is, the control is given to the producer to start producing more applications. However if it is full, a lop is created. This loop goes back and removes an item. At each each, whether producer/consumer, we use the function "notify" to tell the computer when the process for one is completed. Another command "sleep()" is used so that everything for both producer/consumer is outputted in a list manner. This means that it'll only show the necessary steps, rather than an entire dialogue. For the program provided below, I have provided a "//." Once inputting this "//," I have including an explanation regarding that particular process of the program. Also for this program, I have set the thread values for 100, and the integer value to 5. That way I can create a program with a unique output for the producer/consumer. After these slashes, I have included an explanation as to what is occurring at this point of the program. At the end of the program, I have included the original program, and a screenshot of the result produced.

I did not have a compiler available to run this program on my computer. Instead, I used an online compiler. The website of choice was "www.tutorialspoint.com". Below I have placed the website's citation.

"Compile and Execute Java Online." www.tutorialspoint.com, Tutorials Point, www.tutorialspoint.com/compile_java_online.php.

```
// I start off with a Java program. This is in order to create a solution for the producer

import java.util.LinkedList;

 public class Thread example

{

public static void main (String[] args) // this allows the process to only be allowed in the static block, rather than the main(). This means that it's not accessed by the rest of the command. It creates arguments in that sense.

throws interruptedexception

{

// The object of a class is represented by a "()". That's if it has both produce/consume methods

final PC pc = new PC();

// The producer thread is created

Thread t1 = new Thread (new Runnable() // Here the new thread is created. In this case, it's the producer first.

{

@Override // this gives the producer access to stop it's process if the buffer is full

public void run() // this allows the producer to stop the process

{

Try

{

 pc.produce(); // once the function has processed, the producer produces the result

 }

 catch(InterruptedException e) // this gives the result of the method call (i.e producer)

 {

 e.printStackTrace(); // this can print out if an error occurs along the function
```

```
 }
 }
 });
 // Now I create the consumer thread
Thread t2 = new Thread(new Runnable()
{
@Override
public void run()
 {
 try
 {
pc.consume(); // once the function has processed, the consumer consumes a result
 }
 catch(InterruptedException e) // this gives the result of the method call (i.e consumer)
 {
e.printStackTrace();
 }
 }
 });
 // Now i must start the commands for both the producer and consumer threads
 t1.start();
 t2.start();
 // Here t1 must finish before t2
 // This then leads to t1 joining first then t2
```

```
 t1.join();
t2.join();
 }
 // This class must have a list, which is for the producer. This list adds items.
 // After the consumer removes the items
 public static class PC
 {
 // a list is created to be shared by both the producer and consumer
// Since it's the producer/consumer included, the size of the list is equal to five.
LinkedList<Integer> list = new LinkedList<>();
 int capacity = 5;
// the producer thread now calls the function, after setting it's capacity.
 public void produce() throws InterruptedException
 {
 int value = 0;
while (true)
{
 synchronized (this)
 {
 // the producer thread now wait to see if the list is checked
// this is if it's full or not
while (list.size () == capacity)
 wait();
System.out.println("The producer has produced - "+ value); //This creates for ex. (The producer has produced -  99)
```

```
 // the jobs inserted are added to the list

list.add(value++);

//  the consumer thread is notified that it can start consuming (if there's no loop created)

 notify();

// Notifying the thread makes the process of the program easier to check

Thread.sleep(100);

 }

 }

 }

 // Just like after the implementation of the producer, the consumer thread calls upon the function

public void consume() throws InterruptedException

{

while (true)

 {

synchronized (this)

 {

// the consumer thread waits while list (check if it's empty)

 while (list.size()==0)

 wait();

 // must wait to check the first job is received

 int val = list.removeFirst();

System.out.println("The consumer has consumed - " + val); // This creates for ex. (The consumer has consumed - 99)
```

```
// the producer thread is awaken
 notify();
 // Now the producer is placed to sleep after the buffer is checked
Thread.sleep(100);
 }
}
}
}
}
```

Output

Result

```
$javac Threadexample.java
$java -Xmx128M -Xms16M Threadexample
The producer has produced - 0
The producer has produced - 1
The producer has produced - 2
The producer has produced - 3
The producer has produced - 4
The consumer has consumed - 0
The consumer has consumed - 1
The consumer has consumed - 2
The consumer has consumed - 3
The consumer has consumed - 4
The producer has produced - 5
The producer has produced - 6
The producer has produced - 7
The producer has produced - 8
The producer has produced - 9
The consumer has consumed - 5
The consumer has consumed - 6
The consumer has consumed - 7
The consumer has consumed - 8
The consumer has consumed - 9
The producer has produced - 10
The producer has produced - 11
The producer has produced - 12
The producer has produced - 13
The producer has produced - 14
The consumer has consumed - 10
The consumer has consumed - 11
The consumer has consumed - 12
The consumer has consumed - 13
The consumer has consumed - 14
The producer has produced - 15
The producer has produced - 16
The producer has produced - 17
The producer has produced - 18
The producer has produced - 19
The consumer has consumed - 15
The consumer has consumed - 16
The consumer has consumed - 17
The consumer has consumed - 18
The consumer has consumed - 19
The producer has produced - 20
The producer has produced - 21
The producer has produced - 22
The producer has produced - 23
The producer has produced - 24
The consumer has consumed - 20
The consumer has consumed - 21
The consumer has consumed - 22
The consumer has consumed - 23
The consumer has consumed - 24
The producer has produced - 25
The producer has produced - 26
The producer has produced - 27
The producer has produced - 28
The producer has produced - 29
The consumer has consumed - 25
The consumer has consumed - 26
The consumer has consumed - 27
The consumer has consumed - 28
The consumer has consumed - 29
The producer has produced - 30
```

```
The producer has produced - 30
The producer has produced - 31
The producer has produced - 32
The producer has produced - 33
The producer has produced - 34
The consumer has consumed - 30
The consumer has consumed - 31
The consumer has consumed - 32
The consumer has consumed - 33
The consumer has consumed - 34
The producer has produced - 35
The producer has produced - 36
The producer has produced - 37
The producer has produced - 38
The producer has produced - 39
The consumer has consumed - 35
The consumer has consumed - 36
The consumer has consumed - 37
The consumer has consumed - 38
The consumer has consumed - 39
The producer has produced - 40
The producer has produced - 41
The producer has produced - 42
The producer has produced - 43
The producer has produced - 44
The consumer has consumed - 40
The consumer has consumed - 41
The consumer has consumed - 42
The consumer has consumed - 43
The consumer has consumed - 44
The producer has produced - 45
The producer has produced - 46
The producer has produced - 47
The producer has produced - 48
The producer has produced - 49
The consumer has consumed - 45
The consumer has consumed - 46
The consumer has consumed - 47
The consumer has consumed - 48
```