# Quiz Submissions - ICS 440 - Quiz #2

窗体顶端

## **Nalongsone Danddank (username: jf3893pd)**

## **Attempt 1**

Written: Oct 18, 2021 7:57 PM - Oct 18, 2021 8:57 PM

## **Submission View**

Released: Oct 29, 2020 7:08 PM

|  |  |  |
| --- | --- | --- |
| ****Question 1**** |  | 0 / 1 point |

While inside the ****wait()**** method, the calling thread releases the lock it held, and then re-acquires it before returning?

Question options:

|  |  |
| --- | --- |
|  | Yes |
|  | No |

|  |  |  |
| --- | --- | --- |
| ****Question 2**** |  | 1 / 1 point |

In the code below, would the ****waitWhileFull()**** method (lines 45-51) be more efficient if line 47 was changed to:  
    ****if (isFull()) {****  
?

1: //...  
 2: private final int[] slots;  
 3: private int head;  
 4: private int tail;  
 5: private int count;  
 6: private final Object lockObject;  
 7: //...  
 8:   
 9: /\*\*   
10: \* Returns true if added, false for timeout.  
11: \*/  
12: public boolean add(int item, long msTimeout) throws InterruptedException {  
13: synchronized ( lockObject ) {  
14: if (waitWhileFull(msTimeout)) {  
15: slots[tail] = item;  
16: tail = (tail + 1) % slots.length;  
17: count++;  
18: lockObject.notifyAll();  
19: return true;  
20: } else {  
21: return false;  
22: }  
23: }  
24: }  
25:   
26: public void add(int item) throws InterruptedException {  
27: waitWhileFull();  
28: synchronized ( lockObject ) {  
29: slots[tail] = item;  
30: tail = (tail + 1) % slots.length;  
31: count++;  
32: lockObject.notifyAll();  
33: }  
34: }  
35:   
36: /\*\*   
37: \* Returns true if no longer full, false for a timeout.   
38: \*/  
39: public boolean waitWhileFull(long msTimeout) throws InterruptedException {  
40: // In here is code that works correctly   
41: // and synchronizes on lockObject  
42: // ...  
43: }  
44:   
45: public void waitWhileFull() throws InterruptedException {  
46: synchronized ( lockObject ) {  
47: while (isFull()) {  
48: lockObject.wait();  
49: }  
50: }  
51: }

Question options:

|  |  |
| --- | --- |
|  | Yes, with a ****while**** we risk an infinite loop. |
|  | Yes, we were notified, having to check if it's full again with a ****while**** is slightly wasteful. |
|  | No, it needs to be a ****while****, not an ****if**** to protect against early notification. |
|  | No, it is equally correct and equally efficient with either a ****while**** or an ****if****. |

|  |  |  |
| --- | --- | --- |
| ****Question 3**** |  | 0 / 1 point |

Which of the follow (select one or more) can happen if threadA is not holding any locks and calls doStuff()?

private void doStuff() throws InterruptedException {  
 wait(5000);  
}

Question options:

|  |  |  |  |
| --- | --- | --- | --- |
|  | |  |  | | --- | --- | | A) | threadA waits until notified by another thread | |
|  | |  |  | | --- | --- | | B) | threadA waits until 5 seconds have passed | |
|  | |  |  | | --- | --- | | C) | an IllegalMonitorStateException is thrown | |
|  | |  |  | | --- | --- | | D) | threadA waits until interrupted and throws an InterruptedException | |
|  | |  |  | | --- | --- | | E) | threadA sleeps for 5 seconds - even if notified earlier | |

|  |  |  |
| --- | --- | --- |
| ****Question 4**** |  | 1 / 1 point |

In the code below, are there any issues with the add(int) method (lines 26-34)?

1: //...  
 2: private final int[] slots;  
 3: private int head;  
 4: private int tail;  
 5: private int count;  
 6: private final Object lockObject;  
 7: //...  
 8:   
 9: /\*\*   
10: \* Returns true if added, false for timeout.  
11: \*/  
12: public boolean add(int item, long msTimeout) throws InterruptedException {  
13: synchronized ( lockObject ) {  
14: if (waitWhileFull(msTimeout)) {  
15: slots[tail] = item;  
16: tail = (tail + 1) % slots.length;  
17: count++;  
18: lockObject.notifyAll();  
19: return true;  
20: } else {  
21: return false;  
22: }  
23: }  
24: }  
25:   
26: public void add(int item) throws InterruptedException {  
27: waitWhileFull();  
28: synchronized ( lockObject ) {  
29: slots[tail] = item;  
30: tail = (tail + 1) % slots.length;  
31: count++;  
32: lockObject.notifyAll();  
33: }  
34: }  
35:   
36: /\*\*   
37: \* Returns true if no longer full, false for a timeout.   
38: \*/  
39: public boolean waitWhileFull(long msTimeout) throws InterruptedException {  
40: // In here is code that works correctly   
41: // and synchronizes on lockObject  
42: // ...  
43: }  
44:   
45: public void waitWhileFull() throws InterruptedException {  
46: synchronized ( lockObject ) {  
47: while (isFull()) {  
48: lockObject.wait();  
49: }  
50: }  
51: }

Question options:

|  |  |
| --- | --- |
|  | Yes, between lines 27 and 28 there is a chance that another thread could add an item making it full again. |
|  | Yes, if the item couldn't be added, false must be returned. |
|  | Yes, line 32 should be notify(), not notifyAll(). |
|  | Yes, on line 27 ****waitWhileFull()**** requires that a timeout value be passed in. |
|  | No, that method will work just fine. |

|  |  |  |
| --- | --- | --- |
| ****Question 5**** |  | 0 / 1 point |

Which of the follow (select one or more) can happen if threadA is not holding any locks and calls doFoo()?

private synchronized void doFoo() throws InterruptedException {  
 wait(5000);  
}

Question options:

|  |  |  |  |
| --- | --- | --- | --- |
|  | |  |  | | --- | --- | | A) | an IllegalMonitorStateException is thrown | |
|  | |  |  | | --- | --- | | B) | threadA sleeps for 5 seconds - even if notified earlier | |
|  | |  |  | | --- | --- | | C) | threadA waits until notified by another thread | |
|  | |  |  | | --- | --- | | D) | threadA waits until 5 seconds have passed | |
|  | |  |  | | --- | --- | | E) | threadA waits until interrupted and throws an InterruptedException | |

|  |  |  |
| --- | --- | --- |
| ****Question 6**** |  | 1 / 1 point |

In the code below, what does line 16 use % for?

1: //...  
 2: private final int[] slots;  
 3: private int head;  
 4: private int tail;  
 5: private int count;  
 6: private final Object lockObject;  
 7: //...  
 8:   
 9: /\*\*   
10: \* Returns true if added, false for timeout.  
11: \*/  
12: public boolean add(int item, long msTimeout) throws InterruptedException {  
13: synchronized ( lockObject ) {  
14: if (waitWhileFull(msTimeout)) {  
15: slots[tail] = item;  
16: tail = (tail + 1) % slots.length;  
17: count++;  
18: lockObject.notifyAll();  
19: return true;  
20: } else {  
21: return false;  
22: }  
23: }  
24: }  
25:   
26: public void add(int item) throws InterruptedException {  
27: waitWhileFull();  
28: synchronized ( lockObject ) {  
29: slots[tail] = item;  
30: tail = (tail + 1) % slots.length;  
31: count++;  
32: lockObject.notifyAll();  
33: }  
34: }  
35:   
36: /\*\*   
37: \* Returns true if no longer full, false for a timeout.   
38: \*/  
39: public boolean waitWhileFull(long msTimeout) throws InterruptedException {  
40: // In here is code that works correctly   
41: // and synchronizes on lockObject  
42: // ...  
43: }  
44:   
45: public void waitWhileFull() throws InterruptedException {  
46: synchronized ( lockObject ) {  
47: while (isFull()) {  
48: lockObject.wait();  
49: }  
50: }  
51: }

Question options:

|  |  |
| --- | --- |
|  | To keep ****tail**** from passing ****head**** and overwriting items which have not yet been removed. |
|  | It shouldn't be used at all, just do ****tail++**** |
|  | To wrap around to ****slot[0]**** if we increment ****tail**** too far. |
|  | To calculate a percentage of the number of slots |

|  |  |  |
| --- | --- | --- |
| ****Question 7**** |  | 1 / 1 point |

In the code below, is the ****waitWhileFull()**** method (lines 45-51) multithread-safe as written?

1: //...  
 2: private final int[] slots;  
 3: private int head;  
 4: private int tail;  
 5: private int count;  
 6: private final Object lockObject;  
 7: //...  
 8:   
 9: /\*\*   
10: \* Returns true if added, false for timeout.  
11: \*/  
12: public boolean add(int item, long msTimeout) throws InterruptedException {  
13: synchronized ( lockObject ) {  
14: if (waitWhileFull(msTimeout)) {  
15: slots[tail] = item;  
16: tail = (tail + 1) % slots.length;  
17: count++;  
18: lockObject.notifyAll();  
19: return true;  
20: } else {  
21: return false;  
22: }  
23: }  
24: }  
25:   
26: public void add(int item) throws InterruptedException {  
27: waitWhileFull();  
28: synchronized ( lockObject ) {  
29: slots[tail] = item;  
30: tail = (tail + 1) % slots.length;  
31: count++;  
32: lockObject.notifyAll();  
33: }  
34: }  
35:   
36: /\*\*   
37: \* Returns true if no longer full, false for a timeout.   
38: \*/  
39: public boolean waitWhileFull(long msTimeout) throws InterruptedException {  
40: // In here is code that works correctly   
41: // and synchronizes on lockObject  
42: // ...  
43: }  
44:   
45: public void waitWhileFull() throws InterruptedException {  
46: synchronized ( lockObject ) {  
47: while (isFull()) {  
48: lockObject.wait();  
49: }  
50: }  
51: }

Question options:

|  |  |
| --- | --- |
|  | Yes |
|  | No |

|  |  |  |
| --- | --- | --- |
| ****Question 8**** |  | 0 / 1 point |

How can we tall if a call to wait(long msTimeout) returned because is was notified or because it timed out?

Question options:

|  |  |
| --- | --- |
|  | we can't tell which occurred without checking other variables (and even then we can't be 100% sure) |
|  | it returns true is a timeout occurred |
|  | it returns true if notified |

|  |  |  |
| --- | --- | --- |
| ****Question 9**** |  | 1 / 1 point |

While waiting for a condition to become true, we only need to invoke wait() once as we can be sure that the notification we receive always indicates that our condition has been met?

Question options:

|  |  |
| --- | --- |
|  | Yes |
|  | No |

|  |  |  |
| --- | --- | --- |
| ****Question 10**** |  | 1 / 1 point |

What is ****SwingUtilities.invokeLater()**** used for?

Question options:

|  |  |
| --- | --- |
|  | to safely interact with Swing components from a non-UI (event handling) thread |
|  | to wait for a fixed period of time before updating a Swing component |
|  | to prevent text from flickering |
|  | to disable a JComponent for the specified number of milliseconds |

|  |  |  |
| --- | --- | --- |
| ****Question 11**** |  | 0 / 1 point |

While inside the ****notifyAll()**** method, the calling thread releases the lock it held, and then re-acquires it before returning?

Question options:

|  |  |
| --- | --- |
|  | Yes |
|  | No |

|  |  |  |
| --- | --- | --- |
| ****Question 12**** |  | 0.333 / 1 point |

Why was the ****resume()**** method on ****Thread**** deprecated? (choose one or more)

Question options:

|  |  |
| --- | --- |
|  | it is no longer needed since ****stop()**** was deprecated |
|  | it is no longer needed since ****suspend()**** was deprecated |
|  | it allowed "dirty reads" to occur |
|  | it is no longer needed now that we can call ****notifyAll()**** instead of ****notify()**** |
|  | it was never clear if ****pause()**** was actually called |
|  | it allowed objects to become corrupted |

|  |  |  |
| --- | --- | --- |
| ****Question 13**** |  | 0 / 1 point |

The wait-notify mechanism of Java provides which benefit?

Question options:

|  |  |
| --- | --- |
|  | the ability to use locks to control concurrent access to variables |
|  | an efficient means for inter-thread signaling |
|  | a way to have a thread wait to be restarted |
|  | the ability to be interrupted while sleeping |

|  |  |  |
| --- | --- | --- |
| ****Question 14**** |  | 4 / 5 points |

Given the following code, write a new method named ****waitUntilValueIs()**** that returns ****void**** and takes a single ****int**** parameter named ****valueToMatch****. Just write the code for this one method by adding to the small bit of code you are given to start - please keep the answer indented for readability.

public class IntegerBox {  
 private int value;  
 private final Object lockObject;  
  
 public IntegerBox(int value) {  
 lockObject = new Object();  
 this.value = value;  
 }  
  
 public int getValue() {  
 synchronized ( lockObject ) {  
 return value;  
 }  
 }  
  
 public void setValue(int newValue) {  
 synchronized ( lockObject ) {  
 if ( newValue != value ) {  
 value = newValue;  
 lockObject.notifyAll();  
 }  
 }  
 }  
  
 public boolean setValueIfValueMatches(  
 int newValue,  
 int valueToMatch) {  
  
 synchronized ( lockObject ) {  
 if ( value == valueToMatch ) {  
 setValue(newValue);  
 return true;  
 }  
 return false;  
 }  
 }  
}

public void  ****waitUntilValueIs(int valueToMatch)****    {

****synchronized**** (lockObject) {

           while(valueToMatch != this.value) {

                lockObject.wait();

            }

      }

}

|  |  |
| --- | --- |
| [IMG_256Hide Feedback](javascript://" \o ") | |
|  |  |
| Multiple Choice questions: 8.3/16; points: 49.3/95, curved up: 68.4/95; Code question: 4/5 points; Overall curved score: 72.4/100 - missing "throws InterruptedException" in method declaration | |

|  |  |  |
| --- | --- | --- |
| ****Question 15**** |  | 0 / 1 point |

In the code below, are there any issues with the add(int, long) method (lines 12-24)?

1: //...  
 2: private final int[] slots;  
 3: private int head;  
 4: private int tail;  
 5: private int count;  
 6: private final Object lockObject;  
 7: //...  
 8:   
 9: /\*\*   
10: \* Returns true if added, false for timeout.  
11: \*/  
12: public boolean add(int item, long msTimeout) throws InterruptedException {  
13: synchronized ( lockObject ) {  
14: if (waitWhileFull(msTimeout)) {  
15: slots[tail] = item;  
16: tail = (tail + 1) % slots.length;  
17: count++;  
18: lockObject.notifyAll();  
19: return true;  
20: } else {  
21: return false;  
22: }  
23: }  
24: }  
25:   
26: public void add(int item) throws InterruptedException {  
27: waitWhileFull();  
28: synchronized ( lockObject ) {  
29: slots[tail] = item;  
30: tail = (tail + 1) % slots.length;  
31: count++;  
32: lockObject.notifyAll();  
33: }  
34: }  
35:   
36: /\*\*   
37: \* Returns true if no longer full, false for a timeout.   
38: \*/  
39: public boolean waitWhileFull(long msTimeout) throws InterruptedException {  
40: // In here is code that works correctly   
41: // and synchronizes on lockObject  
42: // ...  
43: }  
44:   
45: public void waitWhileFull() throws InterruptedException {  
46: synchronized ( lockObject ) {  
47: while (isFull()) {  
48: lockObject.wait();  
49: }  
50: }  
51: }

Question options:

|  |  |
| --- | --- |
|  | Yes, if the item couldn't be added, false must be returned. |
|  | Yes, line 18 should be notify(), not notifyAll(). |
|  | No, that method will work just fine. |
|  | Yes, on line 14 the waitWhileFull() method which does not take a timeout should be used. |
|  | Yes, between lines 14 and 15 there's a chance that another thread could add an item making it full again. |

|  |  |  |
| --- | --- | --- |
| ****Question 16**** |  | 1 / 1 point |

All implementations of the ****List**** interface are multithread-safe?

Question options:

|  |  |
| --- | --- |
|  | Yes |
|  | No |

|  |  |  |
| --- | --- | --- |
| ****Question 17**** |  | 1 / 1 point |

All implementations of the ****Collection**** interface are multithread-safe?

Question options:

|  |  |
| --- | --- |
|  | Yes |
|  | No |

|  |  |
| --- | --- |
|  |  |
|  | |

|  |
| --- |
| ****Attempt Score:****  12.33 / 21 |
| ****Overall Grade**** (highest attempt)****:****  12.33 / 21 |

窗体底端