UNIVERSITY OF
WESTMINSTER

SCHOOL OF
COMPUTER SCIENCE & ENGINEERING

Module Title:	**Concurrent Programming**
Module Code:	**6SENG002W**
In-Class Test:	**17th January, 2022**
Start Time:	**11:00**
Submission Deadline:	**13:15**
RAF Submission Deadline:	**13:50**

INSTRUCTIONS FOR CANDIDATES

There are EIGHT questions in the test.

Answer ALL EIGHT questions.

Questions 1 - 4 are worth 10 marks each.

Questions 5 - 8 are worth 15 marks each.

YOU MUST SUBMIT YOUR ANSWERS BEFORE THE SUBMISSION DEADLINE.

Question 1

Explain what each of the following concurrency concepts mean:

(a) race conditions

(b) synchronisation

(c) interleaving

(d) mutual exclusion

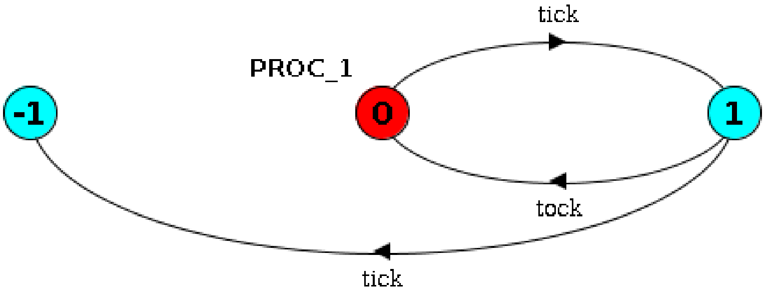
(e) deadlock

[10 marks]
[TOTAL 10]

Question 2

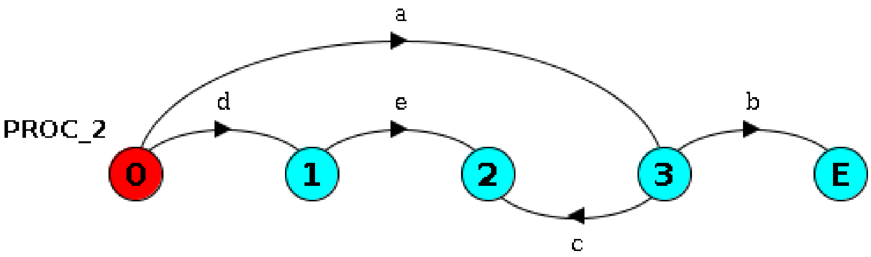
For the following two *Labelled Transition System (LTS)* graphs give the corresponding FSP process definitions.

(a)



[4 marks]

(b)



[6 marks]
[TOTAL 10]

Question 3

(a) Briefly describe the different ways to create a Java thread. **[4 marks]**

(b) When a user defines their own thread class, a constructor method is sometimes used, what should always be called from within the constructor? **[2 marks]**

(c) How do you define the body of a thread? **[2 marks]**

(d) How do you initiate thread execution? **[2 marks]**

[TOTAL 10]

Question 4

(a) Describe the main features of the monitor concurrent programming language mechanism that is used to ensure safe resource sharing in a concurrent program. **[6 marks]**

(b) In a Java version of a monitor, the monitor methods often include a `while` loop. What is the purpose of these while loops? **[4 marks]**

[TOTAL 10]

Question 5

Given the following FSP system.

```
const N    = 3
range DATA = 1..N

Producer = ( produce[2] -> in[2] -> Producer ) .

BUFFER   = ( in[ i : DATA ] -> out[ i ] -> BUFFER ) .

Consumer = ( out[ x : DATA ] -> consume[x] -> Consumer ) .

||SYSTEM = ( Producer || BUFFER || Consumer ) .
```

(a) Draw the alphabet diagram for SYSTEM. **[5 marks]**

(b) For each action state:

- the type of action: synchronous or asynchronous.
- all the processes that perform it.

[10 marks]

[TOTAL 15]

Question 6

(a) Draw a diagram that illustrates the relationships between the life-cycle states of a Java thread. **[6 marks]**

(b) Give a brief description of each state. **[9 marks]**

[TOTAL 15]

Question 7

With reference to the Java code for the Producer Consumer Problem given in Appendix A, answer the following questions.

(a) Explain why the `Buffer` class (lines 1 – 34) is a “secure” *monitor*. **[4 marks]**

(b) In the `Buffer`’s `put` method, what is the purpose of the `while`-loop (lines: 21 – 26)? What is the effect of the `while`-loop? **[4 marks]**

(c) Assume that the `Buffer`’s `new_data` variable is false, the `Producer` thread `p1` is “sleeping”, and the `Consumer` thread `c1` starts to execute line 70:

```
70          int data = buffer.take() ;
```

What then happens to the `Consumer` thread `c1`? **[4 marks]**

(d) In the definition of the `Buffer`’s `take` method (lines 6 – 17), which lines of the method does the monitor’s synchronisation lock change its status, e.g. from locked to unlocked, or from unlocked to locked? Further, state which lock status change or changes happen on those lines. **[3 marks]**

[TOTAL 15]

Question 8

(a) Describe the features of the *semaphore* concurrent programming mechanism. **[5 marks]**

(b) What is the *Dining Philosophers* problem (for 5 Philosophers)? Explain how *deadlock* can occur and how it can be avoided by the use of a *Butler*. **[5 marks]**

(c) Explain what semaphores would be needed to construct a deadlock free solution to the Dining Philosophers problem using a Butler, and how they would be used. **[5 marks]**

[TOTAL 15]

Appendix A

This appendix contains the Java code for a simple version of the Producer Consumer problem.

```
class Buffer
{
  private int contents = -1 ;
  private boolean new_data = false ;

  public synchronized int take( )
  {
     while ( !new_data )
     {
        try {
              wait() ;
        } catch(InterruptedException e){ }
     }
     new_data = false ;
     notifyAll() ;
     return contents ;
  }

  public synchronized void put( int value )
  {
     while ( new_data )
     {
       try {
             wait() ;
        } catch(InterruptedException e){ }
     }

     contents = value ;
     new_data = true ;

     notifyAll() ;
  }

} // Buffer
```

```
class Producer extends Thread
{
  private final Buffer buffer ;

  public Producer( Buffer buffer )
  {
     super( "Producer" ) ;
     this.buffer = buffer ;
  }

  public void run()
  {
    for (int i = 0; i < 10; i++)
    {
       buffer.put( i ) ;
       System.out.println(getName() + " put:   " + i) ;
       try {  sleep( 1000 ) ;  }
       catch (InterruptedException e) { }
    }
  }
} // Producer

class Consumer extends Thread
{
  private final Buffer buffer ;

  public Consumer( Buffer buffer )
  {
     super( "Consumer" ) ;
     this.buffer = buffer ;
  }

  public void run()
  {
    for (int i = 0; i < 10; i++)
    {
      int data = buffer.take() ;
      System.out.println(getName() + " taken: " + data) ;
    }
  }
} // Consumer
```

```
class ProducerConsumerProblem
{
  public static void main( String args[] )
  {
     Buffer buffer = new Buffer() ;

     Producer p1 = new Producer( buffer ) ;
     Consumer c1 = new Consumer( buffer ) ;

     p1.start() ;
     c1.start() ;
  }
} // ProducerConsumerProblem
```

END OF THE IN-CLASS TEST PAPER