UNIVERSITY OF WESTMINSTER

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
TIMED ASSESSMENT REFER/DEFER 2020/21

Module Code:	**6SENG002W, 6SENG004C**
Module Title:	**Concurrent Programming**
Module Leader:	**Paul Howells**
Exam Start Time:	**Thursday, 8th July 2021, 10:00 BST**
Recommended Exam End Time:	**Thursday, 8th July 2021, 12:00 BST**
Submission Window:	**1 hour and 30 minutes**
Submission Deadline:	**Thursday, 8th July 2021, 13:30 BST**

Instructions to Candidates:

Please read the instructions below before starting the paper

- Module specific information is provided below by the Module Leader
- The Module Leader will be available during the exam release time to respond to any queries via the Discussion Board in the Timed Assessment area of the module's Blackboard site
- As you will have access to resources to complete your assessment any content you use from external source materials will need to be referenced correctly. Whenever you directly quote, paraphrase, summarise, or utilise someone else's ideas or work, you have a responsibility to give due credit to that person. Support can be found at:
  https://www.westminster.ac.uk/current-students/studies/study-skills-and-training/research-skills/referencing-your-work
- This is an individual piece of work so do not collude with others on your answers as this is an academic offence
- Plagiarism detection software will be in use
- Where the University believes that academic misconduct has taken place the University will investigate the case and apply academic penalties as published in Section 10 Academic Misconduct regulations.
- ***Once completed please submit your paper via the Assignment submission. In case of problems with submission, you will have two opportunities to upload your answers and the last uploaded attempt will be marked. Note that instructions on how to compile and submit your handwritten and/or typed solutions will have been sent to you separately.***
- ***Work submitted after the deadline will not be marked and will automatically be given a mark of zero***

Module Specific Information

PLEASE WRITE YOUR STUDENT ID CLEARLY AT THE TOP OF EACH PAGE

You are advised (but not required) to spend the first ten minutes of the examination reading the questions and planning how you will answer those you have selected.

ANSWER **THREE** QUESTIONS.
All questions carry equal marks.
Only the THREE questions with the HIGHEST MARKS will count towards the FINAL MARK for the EXAM.

Question 1

(a) When modelling concurrent systems and processes using the abstract Finite State Process (FSP) language, it is necessary to take an *abstract* view of a processes. Describe this FSP abstract view of a process. **[4 marks]**

(b) Given the following FSP process:

```
COUNT ( N = 2 )  =  COUNT[0],

COUNT[ i : 0..N ]
    =  (   when( i < N )  inc -> COUNT[i+1]
         | when( i > 0 )  dec -> COUNT[i-1]
       ).
```

(i) Explain the meaning of the following FSP language features used in this process: "`->`", "`|`", "`when ( i < N )`" and "`COUNT[i+1]`". **[8 marks]**

(ii) Explain the meaning of the following terms and give an example of each for the above process:

- *Alphabet*
- *Transition* **[4 marks]**

(c) For each of the following FSP processes give the corresponding:

- *Labelled Transition System Graph*
- *Trace Tree*.

(i)
```
P1  = (   a -> b -> c -> STOP
        | d -> e -> f -> STOP ) .
```
[6 marks]

(ii)
```
P2 = (   a -> ( b -> STOP | c -> STOP )
       | d -> e -> STOP ) .
```
[7 marks]

(d) What are the most significant differences between the FSP process `P3` and the process `P2` given in part **(c)**.

```
P3 = (   a -> ( b -> P3 | b -> c -> P3 )
       | d -> e -> P3  ) .
```

[4 marks]

[TOTAL 33]

Question 2

The following is a specification of a husband and wife shared bank account system consisting of people processes sharing a bank account.

- A shared bank account called `BANK_ACCOUNT`, that can have money withdrawn from it or deposited into it.
- A "stay at home" husband process called `JIM`, that repeatedly withdraws money from the account.
- A "a career minded" wife process called `KATE`, that repeatedly deposits money into the account.
- The husband and wife processes share the bank account and must obviously have **mutually exclusively** access to it when making deposits or withdrawals.
- The system consists of the **two** human processes, and the **one** bank account process.

(a) Define three Finite State Process (FSP) language processes to model the `BANK_ACCOUNT`, `JIM` and `KATE`. **[26 marks]**

(b) Using your three types of processes define a composite process that models the complete system. **[4 marks]**

(c) Briefly explain how you have ensured that the two processes `JIM` and `KATE` have *mutually exclusive* access to the shared bank account process `BANK_ACCOUNT`. **[3 marks]**

[TOTAL 33]

Question 3

(a) Describe the two methods by which a programmer can create a thread in a Java program. How would you decide which method to use? Illustrate your answer by means of suitable code fragments. **[8 marks]**

(b) Figure 1 represents the *un-labelled* states (nodes) and transitions (arcs) of the life-cycle of a Java thread.

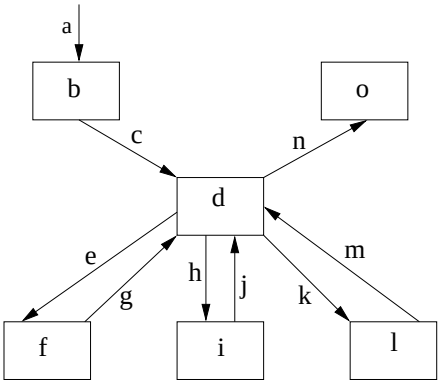

Figure 1: The life-cycle states and transitions of a Java thread.

Identify & describe the labelled (*a* to *o*) states and transitions in Figure 1.

Note you should not re-draw the diagram, but just refer to the labels *a* to *o* in your answer. **[18 marks]**

(c) Within the Java Virtual Machine (JVM) threads use *low-level* actions to interact with the main memory, i.e., transferring values of variables between the *main memory*, the thread's *working copy* and the thread's *execution engine*. Describe these *low-level* actions. **[7 marks]**

[TOTAL 33]

Question 4

(a) In 1974 the computer scientist C.A.R. Hoare proposed the "standard" definition of the concurrent programming concept known as a *monitor*. Describe the main features of a monitor. **[5 marks]**

(b) The Java programming language supports the concept of a monitor. Describe in detail how the monitor concept has been achieved in Java. Illustrate your answer by means of fragments of Java code. **[17 marks]**

(c) With reference to the Java program given in Appendix A.

(i) Describe the sequence of states of the object `mb` and the threads `p` and `r` during their execution; assuming that `p` calls the `post()` method **before** `r` calls the `retrieve()` method. **[6 marks]**

(ii) If the `MessageSystem` class created several `Poster` threads and several `Retriever` threads rather than just one of each, deadlock could occur.

Assuming deadlock has occurred, explain what has happened to the `Poster` and `Retriever` threads. What is the simplest change that could be made to the two `MessageBoard` methods `post()` and `retrieve()` that would stop this happening. **[5 marks]**

[TOTAL 33]

Question 5

(a) One of the first specialised concurrent programming mechanisms invented was the *semaphore*. Describe the features of semaphores. **[9 marks]**

(b) What is the *Dining Philosophers* problem (for 5 Philosophers)? Explain how *deadlock* can occur and how it can be avoided by the introduction of a *Butler*. **[6 marks]**

(c) You are given a Java class that correctly implements a `Semaphore`; where the constructor has the following form:

```
Semaphore( int max_value, int initial_value )
```

Use this semaphore class to write a Java program for the Dining Philosophers problem which avoids deadlock by using a Butler.

You must provide a `Philosopher` class, which represents the behaviour of a philosopher and a `DiningPhilosopher` class which creates the philosophers, forks and butler.

Note: you may assume that all threads execute forever. **[18 marks]**

[TOTAL 33]

Appendix A

Program for Question 4: comprises four classes `Poster`, `Retriever`, `MessageBoard` and `MessageSystem`.

```
1    class Poster extends Thread
2    {
3      private final MessageBoard messageboard;
4
5      public Poster(MessageBoard mb)
6      {
7          messageboard = mb;
8      }
9
10     public void run()
11     {
12        messageboard.post( new String("Hello mate.") );
13        messageboard.post( new String("Good Luck.") );
14     }
15
16   }
17
18
19   class Retriever extends Thread
20   {
21      private final MessageBoard messageboard;
22
23      public Retriever(MessageBoard mb)
24      {
25          messageboard = mb;
26      }
27
28      public void run()
29      {
30         Object message = null ;
31
32         message = messageboard.retrieve();
33         message = messageboard.retrieve();
34      }
35   }
```

[Continued Overleaf]

```
36  class MessageBoard
37  {
38    private Object  message = null;
39    private boolean message_posted = false;
40
41    public synchronized Object retrieve()
42    {
43       while ( !message_posted ) {
44          try {
45               wait();
46          } catch(InterruptedException e){ }
47       }
48       message_posted = false;
49       notify();
50       return message ;
51    }
52
53    public synchronized void post(Object new_message)
54    {
55       while ( message_posted ) {
56          try {
57               wait();
58          } catch(InterruptedException e){ }
59       }
60       message = new_message;
61       message_posted = true;
62       notify();
63    }
64  }
65
66  class MessageSystem
67  {
68    public static void main(String args[]) {
69        MessageBoard mb = new MessageBoard() ;
70        Poster        p = new Poster(mb) ;
71        Retriever     r = new Retriever(mb) ;
72
73        p.start();
74        r.start();
75    }
76  }
```

END OF THE EXAM PAPER