UNIVERSITY OF WESTMINSTER

FACULTY OF
SCIENCE & TECHNOLOGY
Department of Computer Science

Module:	**Concurrent Programming**
Module Code:	**6SENG002W**
Module Leader:	P. Howells
Date:	**SAMPLE EXAM** 2018
Start:	N/A
Time allowed:	2 Hours

Instructions for Candidates:
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.
Each question is worth 33 marks.
Only the three questions with the highest marks count.

DO NOT TURN OVER THIS PAGE
UNTIL THE INVIGILATOR INSTRUCTS YOU TO DO SO.

Module: Concurrent Programming
Module Code: 6SENG002W
Date: SAMPLE EXAM 2018

Question 1

The following Finite State Process (FSP) processes are used to model a Drinks Vending Machine (DVM), a tea drinking customer (TeaCustomer), a coffee drinking customer (LatteCustomer) and the complete system (System).

```
DVM
= ( teaButton   -> pay70p -> deliverTea   -> takeTea   -> DVM
   |
    latteButton -> pay90p -> deliverLatte -> takeLatte -> DVM ).

TeaCustomer = ( teaButton -> pay70p -> takeTea
                          -> drinkTea -> TeaCustomer ) .

LatteCustomer = ( latteButton -> pay90p -> takeLatte
                              -> drinkLatte -> LatteCustomer ) .

|| System = ( DVM || TeaCustomer || LatteCustomer ) .
```

Given the above FSP process definitions:

(a) **(i)** State the *alphabets* of the three processes: DVM, TeaCustomer and LatteCustomer. **[3 marks]**

(ii) Using your answer to part **(i)** draw the *Alphabet* diagram for the composite process System. **[7 marks]**

(iii) Based on your Alphabet diagram for System, state for each action whether it is *synchronous* or *asynchronous* and the processes that perform it. **[5 marks]**

(b) Give the *trace tree* and the *Labelled Transition System* (LTS) Graph for the following processes:

(i) LatteCustomer **[6 marks]**

(ii) DVM **[8 marks]**

(c) Explain how the states of the state machine representing the composite process System are formed from the states of the state machines of its three sub processes TeaCustomer, LatteCustomer and DVM. Given an example. **[4 marks]**

[TOTAL 33]

Question 2

A free music concert is to be held in a Concert Hall, the specification of the system is as follows:

- The Concert Hall has a seating capacity of 100.
- The Concert Hall has one Entrance door and one Exit door.
- The Concert Hall must keep track of how many people are in it, so there is one Doorman controlling the Entrance and a second Doorman controlling the Exit.
- The two doormen record when someone enters or leaves the Concert Hall by using a shared counter.
- The Entry doorman process called `EntryDoorman`, who reads the value of the counter, add 1 and then updates its value when someone enters the Concert Hall.
- The Exit doorman process called `ExitDoorman`, who reads the value of the counter, subtracts 1 and then updates its value when someone leaves the Concert Hall.
- To ensure that an accurate record of the number of people in the Concert Hall is maintained, the two doormen must have **mutually exclusively** access to the shared counter.

(a) Using the Finite State Process (FSP) language define the three processes to model the shared counter, Entry Doorman and Exit Doorman. **[25 marks]**

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

(c) Briefly describe how you have ensured that the two processes modelling the two Doormen have *mutually exclusive* access to the counter. **[4 marks]**

[TOTAL 33]

Question 3

(a) The Java programming language facilitates concurrent programming by providing the *threading* mechanism. Describe and define the general concept of a *thread*. **[5 marks]**

(b) The life-cycle of a Java thread spans its creation, execution and final termination. Describe all of the logical states it may be placed in during its life-cycle. In addition describe how and why a thread enters and exits these possible states. Your answer should be illustrated by a diagram and where appropriate, you should refer to the program code given in Appendix B. **[18 marks]**

(c) With reference to the program given in Appendix A, briefly describe the Java scheduling algorithm. Explain how the Java scheduler would schedule the `Racer` threads, and give an example of the output that could be produced. **[10 marks]**

[TOTAL 33]

Question 4

Appendix B contains a Java program which provides a simple simulation of sending an *SMS text message* to a mobile phone.

(a) Briefly describe the main features of the *monitor* concurrent programming language mechanism, as described by C.A.R. Hoare in his "classic" paper published in 1974. **[5 marks]**

(b) The Java language designers choose to incorporate the *monitor* mechanism in Java, in an attempt to help programmers ensure the safe sharing of resource in multi-threaded Java programs. Describe in detail how the *monitor* mechanism has been implemented in Java. Your answer should be illustrated by reference to the program code given in Appendix B. **[16 marks]**

(c) Describe the main differences between Java's version of the *monitor* mechanism and the "classic" version as described by C.A.R. Hoare. **[5 marks]**

(d) With reference to the program given in Appendix B, describe in detail the sequence of states of the object `jillsphone` and the threads `steve` and `jill` during its execution; assuming that `jill` calls `jillsphone`'s `readtext()` method before `steve` calls its `sendtext()` method. **[7 marks]**

[TOTAL 33]

Question 5

(a) Describe the features of the *semaphore* concurrent programming mechanism. **[9 marks]**

(b) What are the advantages and disadvantages of using semaphores? Explain why *monitors* are generally considered to be "better" than semaphores? **[7 marks]**

(c) **(i)** Assuming that you have available a Java `Semaphore` class, that implements a semaphore.

Give suitable Java code fragments to illustrate how semaphores can be used to achieve *mutual exclusion* of a critical section by two Java threads. **[14 marks]**

(ii) With reference to your code given in answer to part **(i)**; explain how you have used the semaphore mechanism to achieve mutual exclusion of the critical section. **[3 marks]**

[TOTAL 33]

Appendix A

The following thread racer program consists of two classes: `Racer` and `RaceStarter`.

```
1    class Racer extends Thread
2    {
3      Racer(int id)
4      {
5        super( "Racer[" + id + "]" ) ;
6      }
7
8      public void run()
9      {
10       for ( int i = 1 ; i < 40 ; i++ )  {
11         if ( i % 10 == 0 )
12         {
13           System.out.println( getName() + ", i = " + i ) ;
14           yield() ;
15         }
16       }
17     }
18   }
19
20   class RaceStarter
21   {
22     public static void main( String args[] )
23     {
24       Racer[] racer = new Racer[4] ;
25
26       for ( int i = 0 ; i < 4 ; i++ )  {
27           racer[i] = new Racer(i) ;
28       }
29
30       racer[0].setPriority(7) ;
31       racer[3].setPriority(2) ;
32
33       for ( int i = 0 ; i < 4 ; i++ )  {
34           racer[i].start() ;
35       }
36     }
37   }
```

Appendix B

The following mobile phone texting program comprises four classes: `Texter`, `Recipient`, `MobilePhone` and `SMS`.

```
1    class Texter extends Thread
2    {
3      private final MobilePhone  friendsphone ;
4
5      public Texter( MobilePhone phone )
6      {
7          friendsphone = phone ;
8      }
9
10     public void run()
11     {
12        String myTxt = new String("see u @ uni. Spk l8r.") ;
13        friendsphone.sendtext( myTxt ) ;
14     }
15   }
16
17   class Recipient extends Thread
18   {
19      private final MobilePhone myphone ;
20
21      public Recipient( MobilePhone phone )
22      {
23          myphone = phone ;
24      }
25
26      public void run()
27      {
28        String textmessage = myphone.readtext() ;
29      }
30   }
```

[Continued Overleaf]

```
31  class MobilePhone
32  {
33    private String  textmessage = null ;
34    private boolean got_message = false ;
35
36   public synchronized void sendtext ( String message )
37   {
38       while ( got_message ) {
39          try {
40               wait() ;
41          } catch(InterruptedException e){ }
42       }
43       textmessage = message ;
44       got_message = true ;
45       notify() ;
46   }
47
48    public synchronized String readtext( )
49    {
50       while ( !got_message ) {
51          try {
52               wait() ;
53          } catch(InterruptedException e){ }
54       }
55       got_message = false ;
56       notify() ;
57       return textmessage ;
58    }
59  }
60
61  class SMS
62  {
63    public static void main( String args[] )  {
64        MobilePhone jillsphone = new MobilePhone() ;
65        Texter           steve = new Texter( jillsphone ) ;
66        Recipient        jill  = new Recipient( jillsphone ) ;
67
68        steve.start() ;
69        jill.start() ;
70      }
71  }
```