UNIVERSITY OF WESTMINSTER

COLLEGE OF DESIGN, CREATIVE AND DIGITAL INDUSTRIES
School of Computer Science and Engineering
ONLINE EXAMINATION 2020/21

Module Code:	**6SENG002W, 6SENG004C**
Module Title:	**Concurrent Programming**
Module Leader:	**Paul Howells**
Release Time:	**Monday, 11th January 2021, 10:00**
Submission Deadline:	**Monday, 11th January 2021, 13:30**

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) Describe the Finite State Process (FSP) *abstract model* of a process. **[4 marks]**

(b) The following FSP process models a 4 minute egg timer.

```
EGG_TIMER( N = 240 ) = COUNTDOWN[ N ] ,

COUNTDOWN[ i : 0..N ]
    = (  when( i > 0 )   tick -> COUNTDOWN[ i - 1 ]
       | when( i == 0 )  soundBuzzer -> STOP
       | reset -> COUNTDOWN[ N ]
      ) .
```

(i) State what actions are possible for the `COUNTDOWN` process and explain your selection of actions when:

- `i` has the value 30
- `i` has the value 0

[10 marks]

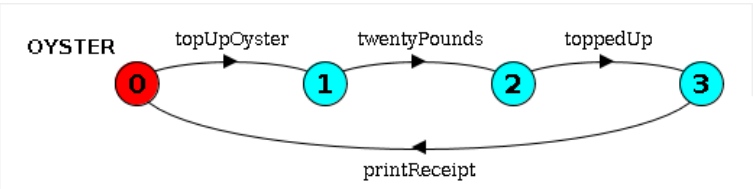
(ii) Explain the meaning of process *Alphabet*. Give the alphabet for the `EGG_TIMER` process. **[2 marks]**

(iii) Define an FSP process called `HARD_EGG_TIMER` using the above `EGG_TIMER` process without modifying its code, to cook a hard boiled egg, that is for 5 minutes (300 seconds). **[3 marks]**

(iv) Explain what *deadlock* means for an FSP process. Describe the `EGG_TIMER` process's deadlock trace? What change would you need to make to `EGG_TIMER` so that it *successfully terminates* instead of deadlocking? **[6 marks]**

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

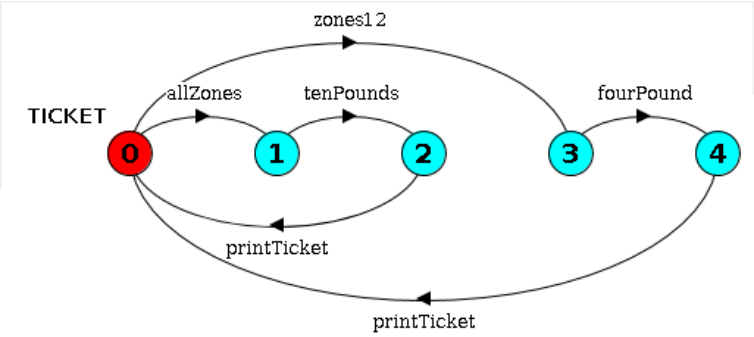
(i) `OYSTER` LTS graph



[3 marks]

[Continued Overleaf]

(ii) `TICKET` LTS graph:



[5 marks]

[TOTAL 33]

Question 2

The following is a specification of two trains travelling between two stations in opposite directions, the journey requires them to share a single section of track, see Figure 1 below.

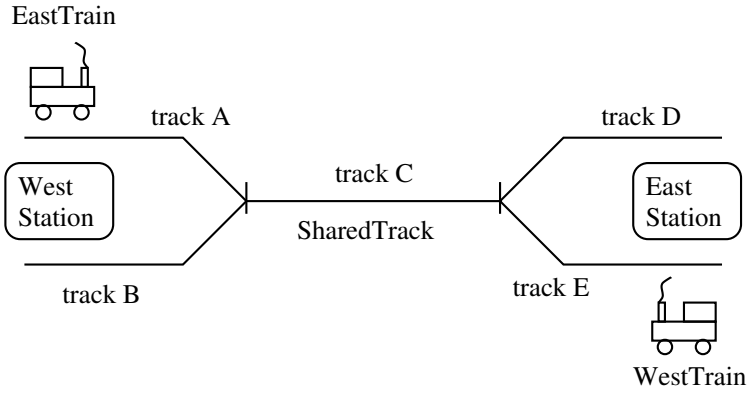


Figure 1: Railway System

- The east bound train `EastTrain` leaves the `WestStation` travels over three sections of track *A, C* and *D* to arrive at the `EastStation`.
- The west bound train `WestTrain` leaves the `EastStation` travels over three sections of track *E, C* and *B* to arrive at the `WestStation`.
- To avoid a crash the trains must be given **mutually exclusively** use of the shared section of track, i.e. *C*.
- The system consists of the **two** train processes and the **one** shared section of track process.

(a) Define three Finite State Process (FSP) language processes to model the `EastTrain`, `WestTrain` and `SharedTrack`.

Then using these three processes define a composite process that models the trains and then the complete railway system. **[24 marks]**

(b) Give the *structure* diagram for the complete Railway system. **[5 marks]**

(c) Explain how you have ensured that the two train processes `EastTrain` and `WestTrain` have *mutually exclusive* access to the shared rail track section. **[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*. **[6 marks]**

(b) Describe and explain the life-cycle states of a Java thread. Illustrate your answer by means of a diagram illustrating the relationships between the states and suitable Java code fragments that cause the state transitions. **[14 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. **[13 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 B.

(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) Describe the features of the *semaphore* concurrent programming mechanism. **[6 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. **[6 marks]**

[TOTAL 33]

Appendix A

Program for Question 3: consisting of two classes `Racer` and `RaceStarter`.

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

Appendix B

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