UNIVERSITY OF WESTMINSTER

SCHOOL OF COMPUTER SCIENCE & ENGINEERING

protected

Module Title:	**Concurrent Programming**
Module Code:	**6SENG002W**
In-Class Test:	**EXAMPLE ICT**
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) process

(b) synchronous action

(c) interference

(d) mutual exclusion protocol

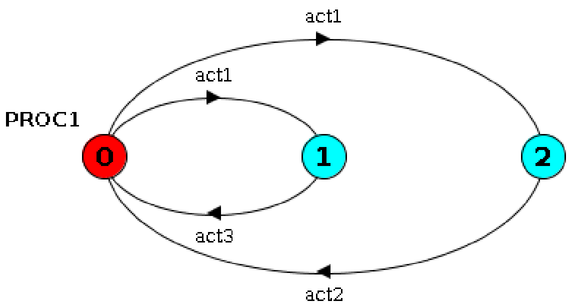
(e) livelock **[10 marks]**

[TOTAL 10]

Question 2

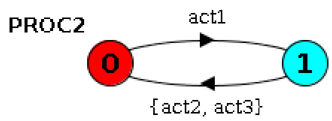
For the following three *Labelled Transition System (LTS)* graphs give the corresponding FSP process definitions.

(a)



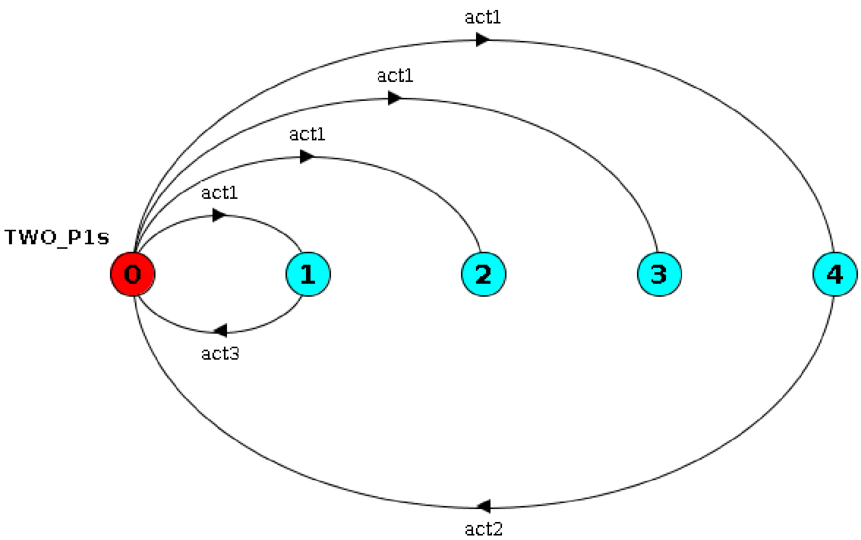
[2 marks]

(b)



[3 marks]

(c)



[5 marks]
[TOTAL 10]

Question 3

(a) Explain the why a Java thread is described as a lightweight process. **[4 marks]**

(b) Assume that two Java threads called `myThread1` and `myThread2` have been created in a Java program. Each thread prints out its name 3 times. Explain what would happen in the following two code fragments:

```
// Code 1:
          myThread1.start() ;
          myThread2.start() ;

// Code 2:
          myThread1.run() ;
          myThread2.run() ;
```

[6 marks]

[TOTAL 10]

Question 4

(a) In relation to the concurrent programming mechanism known as a *semaphore*, what is the difference between a *mutex*, a *binary semaphore* and a *general semaphore*? **[4 marks]**

(b) For each of the types of semaphores referred to in part (a), give one example of a type of concurrent programming problem that can be solved using it, i.e. 3 different examples. **[6 marks]**

[TOTAL 10]

Question 5

Given the following FSP system.

```
SAFE   = ( unlock -> lock -> SAFE ).

WORKER = ( unlock -> accessDocs -> lock -> WORKER ).

SPY    = ( lock -> accessDocs -> unlock -> SPY ).

|| MI6 =  (    w1 : WORKER || w2 : WORKER || s : SPY
            || { w1, w2, s } :: SAFE ) .
```

(a) Draw the alphabet diagram for MI6. **[6 marks]**

(b) For each action state:

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

[9 marks]

[TOTAL 15]

Question 6

In relation to a multi-threaded Java program, its threads and thread life-cycle states, answer the following questions.

(a) What can a thread do after it has been created? **[2 marks]**

(b) For all thread life-cycle states, state whether a thread can or cannot execute code in it? **[2 marks]**

(c) If a thread attempts to acquire a synchronisation lock but fails, explain the state transition it undergoes. **[3 marks]**

(d) If several threads are waiting to execute how does the Java Virtual Machine (JVM) decide which one to execute next? **[3 marks]**

(e) If a thread is in the `TIMED_WAITING` state what can happen to it? **[5 marks]**

[TOTAL 15]

Question 7

(a) Describe the Readers & Writers problem, and explain the concurrency issues that must be solved. **[5 marks]**

(b) Semaphores can be used to construct a correct solution for the Readers & Writers problem. Explain what semaphores are needed and what they are used for in a solution. **[10 marks]**

[TOTAL 15]

Question 8

With reference to C.A.R Hoare's solution of the Dining Philosophers problem using his own version of a monitor is given in Appendix A, answer the following questions.

(a) What is used to indicate how many forks are available for each philosopher to pick up? **[1 mark]**

(b) What does a philosopher do if it does not have sufficient forks available to pick up and eat? **[2 marks]**

(c) Which philosophers update the number of forks available to philosopher 2? **[2 marks]**

(d) What does a philosopher do when it puts down its forks? **[3 marks]**

(e) What programming feature is declared by this statement:

```
Line 6:        Condition[] OK_to_eat ;
```

and what is the equivalent feature in Java? **[4 marks]**

(f) On lines 23 and 25, the "`signal( )`" method is used. What does this do and what is the Java equivalent of this method? **[3 marks]**

[TOTAL 15]

Appendix A

This appendix contains C.A.R Hoare's solution of the Dining Philosophers problem.

```
Monitor ForksMonitor
{
  final int NP = 5 ;
  int[] Fork ;

  Condition[] OK_to_eat ;

  public void Pick_up_Forks( int i )
  {
    if ( Fork[i] < 2 )  wait( OK_to_eat[i] ) ;

    Fork[(i+1) % NP] = Fork[(i+1) % NP] - 1 ;

    Fork[(i-1) % NP] = Fork[(i-1) % NP] - 1 ;
  }

  public void Put_down_Forks( int i )
  {
    Fork[(i+1) % NP] = Fork[(i+1) % NP] + 1 ;

    Fork[(i-1) % NP] = Fork[(i-1) % NP] + 1 ;

    if ( Fork[(i+1) % NP] == 2 )  signal( OK_to_eat[(i+1) % NP] ) ;

    if ( Fork[(i-1) % NP] == 2 )  signal( OK_to_eat[(i-1) % NP] ) ;
  }

  beginbody
     Fork = new int[NP] ;

     Condition OK_to_eat = new Condition[NP] ;

     for ( int i = 0; i < NP; i++ )  Fork[i] = 2 ;

  endbody
} // end Monitor ForkMonitor
```

```
Process Philosopher( int i, ForkMonitor fm )
{
   while ( true )
   {
     // Think() ;
     // SitDown() ;
     fm.Pick_up_Forks( i ) ;
     // Eat() ;
     fm.Put_down_Forks( i ) ;
     // GetUp() ;
   }
}



Program DiningPhilosophers
{
    ForkMonitor fm = new ForkMonitor ;

    parbegin
             Philosopher( 0, fm ) ;
             Philosopher( 1, fm ) ;
             Philosopher( 2, fm ) ;
             Philosopher( 3, fm ) ;
             Philosopher( 4, fm ) ;
    parend ;
}
```

END OF THE IN-CLASS TEST PAPER