CAVENDISH CAMPUS

School of Electronics and Computer Science

Modular Undergraduate Programme
Second Semester 2012 – 2013

Module Code: ECSE603

Module Title: Concurrent Programming

Date: Tuesday, 7th May 2013

Time: 10:00 – 12:00

Instructions to Candidates:

Answer THREE questions.
Each question is worth 33 marks.

Question 1

(a) What is the Finite State Process (FSP) view of a process and how are these processes modelled? **[8 marks]**

(b) Given the following FSP process:

```
MINUTE_ALARM( N = 60 ) = COUNTER[ N ] ,

COUNTER[ i : 0..N ]
    = (  when( i > 0 )  tick -> COUNTER[ i - 1 ]
       | when( i == 0 )  sound.alarm -> STOP
      ) .
```

(i) Explain the meaning of the following FSP language features used in this process: "`N = 60`", "`|`", "`when ( i == 0 )`" and "`STOP`". **[8 marks]**

(ii) With reference to the above process explain the meaning of the following terms:

- *Transition*
- *Trace* **[4 marks]**

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

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

(i)
```
RHYME = ( one -> two -> buckle
            -> my -> shoe -> STOP ).
```
[4 marks]

(ii)
```
CHANGE = ( fivep -> onep ->
             twop -> (  onep -> onep -> CHANGE
                      | twop -> CHANGE ) ).
```
[9 marks]

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- A shared bank account called `BANK_ACCOUNT`, that can have money withdrawn from it or deposited into it.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]**

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.

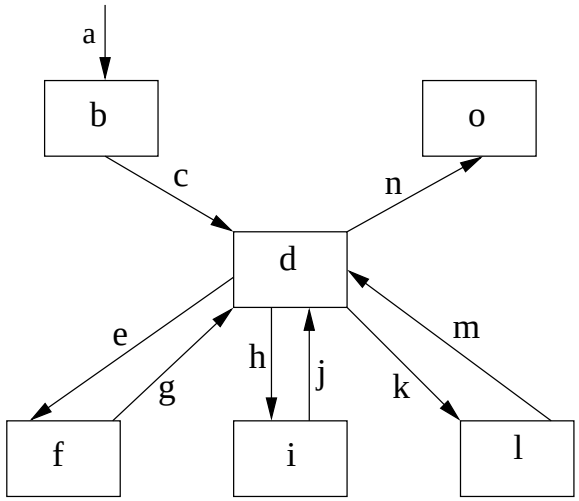

Figure 1: The life-cycle states and transitions of a Java thread.

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]**

Question 4

(a) Describe the main features of C.A.R. Hoare's concurrent programming language mechanism known as a *monitor*. **[5 marks]**

(b) Describe in detail Java's implementation of the general *monitor* concepts. You should illustrate your discussion with relevant fragments of Java code. **[16 marks]**

(c) Describe the main differences between C.A.R. Hoare's "standard" definition of a *monitor* and Java's implementation of the monitor mechanism. **[5 marks]**

(d) In Appendix A there is a Java program which provides a simple simulation of sending a simple *SMS text message* to a mobile phone.

With reference to this program describe in detail the sequence of states of the object `billsPhone` and the threads `ben` and `bill` during its execution; assuming that `bill` calls `billsPhone`'s `readtext()` method before `ben` calls its `sendtext()` method. **[7 marks]**

Question 5

(a) Describe the features of the first specific concurrent programming mechanism, known as a *semaphore*. **[5 marks]**

(b) Write a Java class that operates as a general semaphore. The semaphore class' "constructor" should have two arguments:

- the initial value of the semaphore; and
- the upper limit of the semaphore.

[10 marks]

(c) Give a brief description of the Producer/Consumer problem. What types of semaphores are required to solve this problem, i.e. avoid deadlock, etc. In addition explain the purpose of each semaphore. **[5 marks]**

(d) Using your semaphore class modify the version of the Producer/Consumer program given in Appendix B, so that it is a safe and live solution of the problem.

Note: you do not have to copy out the whole program, but only those parts that are sufficient to indicate your additions and modifications. You should make use of the line numbers to do this. **[13 marks]**

Appendix A

Program Code for Question 4(d)

The 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("Gd Luk. 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  billsPhone = new MobilePhone();
65        Texter       ben        = new Texter( billsPhone );
66        Recipient    bill       = new Recipient( billsPhone );
67
68        ben.start();
69        bill.start();
70      }
71  }
```

Appendix B

Program Code for Question 5(d)

Consisting of four classes `Buffer`, `Producer`, `Consumer` and `ProducerConsumer`.

```
1    class Buffer
2    {
3      public Object[] buffer = null ;
4      public final int size ;
5      public int in  = 0 ;
6      public int out = 0 ;
7
8      public Buffer( int size ) {
9        this.size = size ;
10       buffer    = new Object[size] ;
11     }
12   }

13   class Producer extends Thread
14   {
15     private Buffer b = null ;
16
17     public Producer( Buffer buff ) {  b = buff ;  }
18
19     public void run() {
20       for (int i = 0; i < 10; i++ ) {
21         b.buffer[b.in] = produceItem() ;
22         b.in = (b.in + 1) % b.size ;
23       }
24     }
25
26     private Object produceItem() {  // produce item  }
27   }
```

[Continued Overleaf]

```
28  class Consumer extends Thread
29  {
30    private Buffer b    = null ;
31    private Object item = null ;
32
33    public Consumer( Buffer buff ) {  b = buff ;  }
34
35    public void run() {
36      for (int i = 0; i < 10; i++ ) {
37        item = b.buffer[b.out] ;
38        b.buffer[b.out] = null ;
39        b.out = (b.out + 1) % b.size ;
40
41        consumeItem(item);
42      }
43    }
44
45    private void consumeItem(Object item) { // consume item  }
46  }

47  class ProducerConsumer
48  {
49    private static final int N = 5 ;
50
51    public static void main( String args[] )
52    {
53      Buffer buffer = new Buffer(N) ;
54
55      Thread p = new Producer(buffer) ;
56      Thread c = new Consumer(buffer) ;
57
58      p.start() ;
59      c.start() ;
60    }
61  }
```