CAVENDISH CAMPUS

School of Electronics and Computer Science

Modular Undergraduate Programme
First Semester 2008 – 2009

Module Code: 3SFE605

Module Title: Concurrent Programming

Date: Monday, 11th May 2009

Time: 14:00 – 16:00

Instructions to Candidates:

Answer THREE questions.
Each question is worth 33 marks.

Question 1

(a) The Java programming language supports concurrent programming by means of the concept of *threads*. Give a brief definition of a *thread*. Describe how a programmer can create a thread in a Java program by sub-classing, illustrate your answer by a suitable code fragment. **[5 marks]**

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

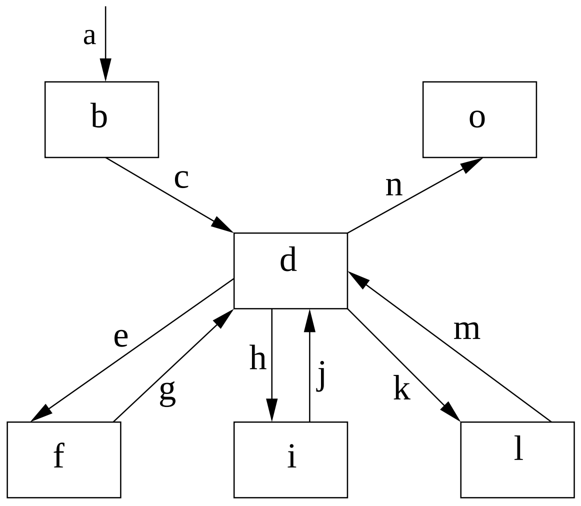

Figure 1: The life-cycle states and transitions of a Java thread.

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. **[19 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. **[9 marks]**

Question 2

(a) Describe the concurrent programming concept known as a *monitor* as originally proposed by C.A.R. Hoare. **[5 marks]**

(b) Describe in detail Java's implementation of the monitor concept. You should illustrate the basic structure of a Java *"monitor"* class by including suitable code fragments. **[17 marks]**

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

(i) Describe in detail the sequence of states of the object `mb` and the threads `p` and `r` during its execution; assuming that `r` calls `mb`'s `retrieve()` method before `p` calls its `post()` 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]**

Question 3

(a) Explain the concept of a *design pattern*. How do design patterns assist in developing concurrent object-oriented programs? **[7 marks]**

(b) Give a brief description of Java's "`interface`" and "`abstract`" class features. In addition, give reasons for why they are useful when developing software using the *design patterns* approach. **[10 marks]**

(c) Describe the overall approach of the *Immutability* design pattern, and the six design steps that are used when applying the *Immutability* design pattern to develop a class. **[7 marks]**

(d) Apply the *Immutability* design pattern to produce a Java class called `Personal_ID`, that can be used to represent a person's identity using their *name* and *date of birth*. Examples of possible values for each of these attributes, using the Java `String` type, are `"Jim Jones"` and `"21/3/1970"`. In addition it should have *accessors* for each attribute called `name` and `dob`.

[9 marks]

Question 4

(a) **(i)** When considering the *correctness* of a concurrent program in terms of desirable properties, we classify these properties as being either a *"safety"* or *"liveness"* property. Explain what is meant by *"safety"* and *"liveness"* properties, and give an example of each. **[3 marks]**

(ii) Describe the ways in which a Java thread can fail to be live. **[5 marks]**

(b) Many design patterns for achieving liveness are based on the technique known as *"splitting synchronization"*. One technique for doing this is known as *lock splitting*, describe this approach. **[7 marks]**

(c) The Java `Rectangle` class is given in Appendix B. This class provides the basis for displaying (i.e., moving and resizing) rectangular shapes, that for example, could be used to implement terminal windows.

You can assume that `moveRectangle()` never deals with its dimensions and `resizeRectangle()` never deals with its location.

The `Rectangle` class is seen as inefficient, i.e., has poor *liveness* characteristics.

(i) Produce a new version with improved performance, by applying the *lock splitting* liveness design pattern.

Note: you only need to give appropriate annotated code fragments which illustrate the main features, **not** a complete program. **[10 marks]**

(ii) Give an annotated diagram which illustrates the structure of your "lock splitting" version of the `Rectangle` class. The diagram should include the `LockSplittingRectangle` class and any new classes and any relationships/links between the `LockSplittingRectangle` class and any new classes. **[8 marks]**

Question 5

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

(b) Given the following Java interface for a semaphore:

```
public interface Semaphore_Int
{
     final int MIN_VALUE = 0 ;
     public void claim() ;
     public void release() ;
}
```

Write a Java `Semaphore` class that implements the `Semaphore_Int` interface. The class should be able to operate as either a binary or general semaphore. The class' "constructor" should take two parameters, one that indicates its initial value and the other that specifies its upper limit. **[10 marks]**

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

(d) Using your `Semaphore` class, write a Java program for the Dining Philosophers problem which avoids deadlock by using a Butler.

Describe how your `Semaphore` class can be used to model parts of the program.

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: assume that all threads will execute forever. **[12 marks]**

Appendix A

Program for Question 2: comprises four classes Poster, Retriever, MessageBoard and MessageSystem.

```
class Poster extends Thread
{
  private final MessageBoard messageboard;

  public Poster(MessageBoard mb)
  {
      messageboard = mb;
  }

  public void run()
  {
     messageboard.post( new String("Hello mate.") );
     messageboard.post( new String("Good Luck.") );
  }

}

class Retriever extends Thread
{
   private final MessageBoard messageboard;

   public Retriever(MessageBoard mb)
   {
       messageboard = mb;
   }

   public void run()
   {
      Object message = null ;

      message = messageboard.retrieve();
      message = messageboard.retrieve();
   }
}
```

[Continued Overleaf]

```
class MessageBoard
{
  private Object  message = null;
  private boolean message_posted = false;

  public synchronized Object retrieve()
  {
     while ( !message_posted ) {
        try {
             wait();
        } catch(InterruptedException e){ }
     }
     message_posted = false;
     notify();
     return message ;
  }

  public synchronized void post(Object new_message)
  {
     while ( message_posted ) {
        try {
             wait();
        } catch(InterruptedException e){ }
     }
     message = new_message;
     message_posted = true;
     notify();
  }
}

class MessageSystem
{
  public static void main(String args[]) {
      MessageBoard mb = new MessageBoard() ;
      Poster        p = new Poster(mb) ;
      Retriever     r = new Retriever(mb) ;

      p.start();
      r.start();
  }
}
```

Appendix B

Program Code for Question 3: consisting of the Rectangle class.

```
public class Rectangle
{
  protected double x = 0.0;
  protected double y = 0.0;

  protected double width = 0.0;
  protected double height = 0.0;

  public synchronized double x() {  return x;  }

  public synchronized double y() {  return y;  }

  public synchronized double width()  {  return width;   }

  public synchronized double height() {  return height;  }

  public synchronized void adjustLocation()
  {
    x = Calculation1();
    y = Calculation2();
  }

  public synchronized void adjustDimensions()
  {
    width  = Calculation3();
    height = Calculation4();
  }

  protected double Calculation1() {  ...  }
  protected double Calculation2() {  ...  }
  protected double Calculation3() {  ...  }
  protected double Calculation4() {  ...  }

}
```