CAVENDISH CAMPUS

School of Informatics

Modular Undergraduate Programme
First Semester 2006 – 2007

Module Code: 3SFE605

Module Title: Concurrent Programming

Date: Tuesday, 22nd May 2007

Time: 14:00 – 16:00

Instructions to Candidates:

Answer THREE questions.
Each question is worth 33 marks.

Question 1

(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) Describe the possible states of a JDK 1.5 Java thread. Explain how a Java thread changes state. Your answer should be illustrated by a diagram and fragments of Java code which produce the state changes. **[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 2

(a) Describe the concurrent programming concept known as a *monitor*. **[3 marks]**

(b) Describe in detail Java's implementation of the monitor concept. Your answer should be illustrated by fragments of Java code. **[17 marks]**

(c) With reference to the Java program given in Appendix A. Describe in detail the sequence of states of the object `sv` and the threads `w` and `r` during the following scenarios.

(i) From their creation to their termination, when `w` calls `sv`'s `assign` method before `r` calls its `read` method. **[8 marks]**

(ii) Assume the object and threads have been created and started as in part **(i)**, that `r` has just started executing `sv`'s `read` method and that `w` is blocked trying to execute the `assign` method.

Then describe in detail the sequence of states from this point to the thread's termination. **[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) Describe the Java language features *interfaces* and *abstract classes*, and how they assist in the construction of design patterns. **[8 marks]**

(c) Describe the main aims of the *safety preservation* design patterns for concurrent object-oriented programs. Give a brief description of the strategies of three safety design patterns. **[3 marks]**

(d) 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]**

(e) Apply the *Immutability* design pattern to produce a Java class called `OrderedPair`, that can be used to represent an ordered pair of characters, for example (a, A), using the Java type `char`. In addition it should have accessors for each element of the pair called `first` and `second`. **[8 marks]**

Question 4

(a) Explain why there is a need to "balance *safety* and *liveness* requirements" when applying design patterns to concurrent object-oriented programs. Discuss the role that synchronization plays in achieving this balance. **[8 marks]**

(b) Many design patterns for achieving liveness are based on the technique known as *splitting synchronization*. In practice this technique is achieved by applying the technique know as *splitting classes*; describe this approach. **[9 marks]**

(c) **(i)** The `Cube` class given in Appendix B, provides the basis for a simple 3D cube for a graphics application. (Assume that `moveCube()` never deals with its dimensions and `resizeCube()` never deals with its location.)

The `Cube` class is seen as inefficient, i.e., has poor *liveness* characteristics. With the aim of improving performance, produce a new version `ClassSplittingCube` by applying the *splitting classes* 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 "class splitting" version of the `Cube` class. The diagram should include the `ClassSplittingCube` class and any new classes and any relationships/links between the `Cube` class and any new classes. **[6 marks]**

Question 5

(a) Describe the concurrent programming mechanism known as a *semaphore*. **[7 marks]**

(b) What are the advantages and disadvantages of using semaphores? **[6 marks]**

(c) Given the following Java `interface` for a semaphore:

```
public interface Semaphore
{
     public void claim() ;
     public void release() ;
}
```

Write a Java class called `BinarySemaphore`, that implements this semaphore interface and operates as a *binary* semaphore. **[12 marks]**

(d) Using your Java binary semaphore class `BinarySemaphore` (from part **(c)**) show, by giving suitable code fragments, how it could be used to achieve the *mutual exclusion* of a critical section by two threads. **[8 marks]**

Appendix A

Program Code for Question 2(c)

The program comprises four classes: `SharedVar`, `Writer`, `Reader` and `System`.

```
class SharedVar
{
  private int contents;
  private boolean new_value = false;

  public synchronized int read()
  {
     while ( !new_value )
     {
        try {  wait();  }
        catch(InterruptedException e){ }
     }
     new_value = false;
     notifyAll();
     return contents;
  }

  public synchronized void assign(int value)
  {
     while ( new_value )
     {
        try {  wait();  }
        catch(InterruptedException e){ }
     }
     contents = value;
     new_value = true;
     notifyAll();
  }
}
```

[Continued Overleaf]

```
class Writer extends Thread
{
  private SharedVar sharedvar;

  public Writer(SharedVar sv) {  sharedvar = sv;  }

  public void run() {  sharedvar.assign(1);  }
}


class Reader extends Thread
{
   private SharedVar sharedvar;
   private int value;

   public Reader(SharedVar sv) {  sharedvar = sv;  }

   public void run() {  value = sharedvar.read();  }
}


class System
{
  public static void main(String args[])
  {
      SharedVar sv = new SharedVar();
      Thread w = new Writer(sv);
      Thread r = new Reader(sv);

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

Appendix B

Program Code for Question 4(c)

The following code fragments are for the Cube class.

```
public class Cube
{
  protected double x = 0.0;
  protected double y = 0.0;
  protected double z = 0.0;

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

  public synchronized double x() { return x; }
  public synchronized double y() { return y; }
  public synchronized double z() { return z; }

  public synchronized double width()  { return width; }
  public synchronized double height() { return height; }
  public synchronized double depth()  { return depth; }

  public synchronized void moveCube()
  {
    x = calcX();
    y = calcY();
    z = calcZ();
  }

  public synchronized void resizeCube()
  {
    width  = calcWidth();
    height = calcHeight();
    depth  = calcDepth();
  }

  protected double calcX() {  // ...  }
  // etc
}
```