CSE 428, Spring 2002. Midterm 2, 28 March 2001

The exam consists of 4 exercises, totalling 100 points. Write your answers directly on these pages. **Please remember to write your name, Student ID, and section number (Section 2 for the morning section and Section 1 for the afternoon section) at the top of each sheet.**

Exercise 1 (10 pts)

Consider the following three alternative definitions of a method `readString()`.

1.
```
public String readString() {
   String s = ‘‘’’;
   while (true) {
       try {
           System.Out.print( "Enter string :");
           s = in.readLine();
           return s;
       } catch (IOException e) {
       System.out.println( "Input error occurred. Please rerenter :");
       } // try
    } // while
   return s;
 } // readString
```

2.
```
public String readString() {
   String s= ‘‘’’;
   try {
       while (true) {
          System.Out.print( "Enter string :");
          s = in.readLine();
          return s;
       } // while
   } catch (IOException e) {
       System.out.println( "Input error occurred. Please rerenter :");
   } // try
   return s;
 } // readString
```

3.
```
public String readString() throws IOException {
   String s= ‘‘’’;
   while (true) {
      System.Out.print( "Enter string :");
      s = in.readLine();
      return s;
   } // while
   return s;
 } // readString
```

Suppose that during the execution of `readString()` an IO exception occurs. Please describe what will happen then, for each of the three definitions above.

1.

2.

3.

Exercise 2 (20 pts)

The following classes represent a Restaurant open 24 hours a day, and a group of customers of the restaurant, respectively.

Please complete the definition of `Restaurant` and `Group` by replacing the "`...`" with suitable code (possibly empty). Assume that a group can enter the restaurant only if the number of tables available is enough to accommodate the number of people in the group (each table accommodates 4 people), otherwise it will have to wait. Please avoid "busy waiting".

Note: You can ignore the treatment of `InterruptedException`.

```
class  Restaurant {

   private int tablesAvailable; // Number of tables available.
                                // Each table can accomodate 4 people.

   Restaurant(int totalNumerOfTables) { tablesAvailable = totalNumerOfTables; }

   public ... enter(int numberOfTables) {
          ...
          tablesAvailable = tablesAvailable - numerOfTables;
          ...
   }

   public ... exit(int numerOfTables) {
          ...
          tablesAvailable = tablesAvailable + numerOfTables;
          ...
   }
}

class Group extends Thread {

   private int numberOfPeople; // Number of people composing the group.
   private Restaurant restaurant;

   Group(int n, Restaurant r) { numberOfPeople = n; restaurant = r; }

   public run() {
         < do some activity >
         ...  // enter the restaurant
         < eat >
         ... // exit the restaurant
         < do some other activity >
   }

}
```

Exercise 3 (35 pts)

We want to define in Java a thread `JobsExercutor` that takes "jobs" from two input queues, and execute them. We assume that the jobs are put in the queues by other threads and that the execution of a job always terminates. We want that if a job is put in a queue, then it will be eventually executed (fairness). In particular, `JobsExercutor` **should not be suspended forever on an empty queue if the the other queue has (or receives) jobs**.

For simplicity, we assume that the queues are bounded and that their maximal length is 1, namely that they instances of the one position buffer we saw in class.

In the following we give a sketch of the code of `JobsExercutor` and of the `Buffer` we saw in class.

```
class Buffer {  // A one position buffer of Jobs.
    private Job content;
    private boolean empty;

    public Buffer() { empty = true; }

    public synchronized void put(Job i) {
         while (!empty) wait();
         content = i;
         empty = false;
         notifyAll();
    }
    public synchronized Job get() {
         while (empty) wait();
         empty = true;
         notifyAll();
         return content;
    }
}// end Buffer


class JobsExercutor extends Thread {

    private Buffer firstQueue; // first queue
    private Buffer secondQueue; // second queue;

    JobsExercutor(Buffer f, Buffer s) { firstQueue = f; secondQueue = s; }

    public run() {
        Job nextJob;
        while (true) {
           ...  // takes the next job from a queue in a fair way
           < execute nextJob >
        }
   }

}
```

1. Please fill in the missing parts of `JobsExercutor` and, if needed, modify `Buffer`, so to ensure that `JobsExercutor` behaves in the way described above. Busy waiting is accepted as a solution. **Note:** We are not interested in the nature of the jobs nor of job execution. Please leave <execute nextJob> as it is. You can also ignore the treatment of `InterruptedException`.

2. Does your solution avoid busy waiting or not? If it does not, please discuss how you could possibly modify the program so to avoid busy waiting. (If it does avoid busy waiting already and it is fair as requested, then you will receive full credit on this part of the exercise.)

Exercise 4 (35 pts)

The following applet prints the sentence "Hello World" on the frame in the point of coordinates (50,100). Please modify it in the following way:

- Add a menu choice with the items "Hello" and "World", and
- Add suitable event listeners so that when the mouse is clicked in a certain point of the frame, the word corresponding to the current menu choice is printed in that point.

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class HelloWorld extends Applet {

   public void init ( ) {
      setBackground(Color.white);
      Graphics g = getGraphics();
      g.drawString(‘‘Hello World’’, 50, 100);
   }
}
```