5COSC001W - Solutions to Tutorial 11 Exercises

1 Testing more your Knowledge on Threads

Answers 2 and 4 are the only correct ones. Answer 2 is correct because `myT.start()` is never called. The thread never runs as nobody starts it! Answer 4 is correct because `myT.start()` is never called.

Answers 1 and 3 are incorrect because the `myT` thread is never started, so `run()` is never executed.

To understand this further, type the following code:

```java
public class Whiffler implements Runnable {
    Thread myT ;

    public void start(){
        myT = new Thread( this );
    }

    public void run(){
        while( true ){
            doStuff();
        }
        System.out.println("Exiting run");
    }
    // more class code follows....

    void doStuff() {
        System.out.println("dostuff is executing");
    }


    public static void main(String[] args) {
        Whiffler w = new Whiffler();
        w.start();
    }
}
```

Hint: Comment out the line `System.out.println("Exiting run");` to compile the code and verify the rest of your guesses.

2 Testing more your Knowledge on Threads

```java
import java.util.concurrent.locks.*;

public class PlumWithHighLevelLocks {
  public static void main(String args[]) {
     Producer p = new Producer();
     p.start();

     Consumer c = new Consumer(p);
     c.start();
  }
}

class Producer extends Thread {
    private String [] buffer = new String [8];
    private int pi = 0;  // produce index
    private int gi = 0;  // get index

    ReentrantLock prod_cons_lock = new ReentrantLock();
    Condition prod_cons_condition = prod_cons_lock.newCondition();


    public void run() {
       // just keep producing
       for(;;) produce();
    }

    private final long start = System.currentTimeMillis();
    private final String banana() {
        return "" + (int) (System.currentTimeMillis() - start);
    }

    void produce() {
       // activate lock
       prod_cons_lock.lock();

       // while there isn't room in the buffer
       while ( pi-gi+1 > buffer.length ) {
           try {
               prod_cons_condition.await();
           }
           catch(Exception e) {}
       }
       buffer[pi%8] = banana();
       System.out.println("produced["+(pi%8)+"] " + buffer[pi%8]);
       pi++;

       prod_cons_condition.signalAll();
```

```
        prod_cons_lock.unlock();
     }

     synchronized String consume(){
        // activate lock
         prod_cons_lock.lock();

        // while there's nothing left to take from the buffer
        while (pi==gi) {
            try {
                prod_cons_condition.await();
            }
            catch(Exception e) {}
        }

        String data = buffer[ gi++ % 8 ];
        prod_cons_condition.signalAll();
        prod_cons_lock.unlock();

        return data;
     }
}

class Consumer extends Thread {
   Producer whoIamTalkingTo;
   // java idiom for a constructor
   Consumer(Producer who) {
       whoIamTalkingTo = who;
   }

   public void run() {
      java.util.Random r = new java.util.Random();
      for(;;) {
          String result = whoIamTalkingTo.consume();
          System.out.println("consumed: "+result);
          // next line is just to make it run a bit slower.
          int randomtime = Math.abs(r.nextInt() % 250);
          try{sleep(randomtime);} catch(Exception e){}
      }
   }
}
```

3 Swing and Threads

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
```

```java
class AsynchronousTask extends SwingWorker<Integer, Void> {
    // JLabel to publish the results to
    JLabel resultLabel;
    int result;

    // the constructor of the class
    AsynchronousTask(JLabel label) {
        resultLabel = label;
    }

    @Override
    public Integer doInBackground() {
        // simulate a time consuming task
        System.out.println("Started time consuming task, " +
                           " it might take a while...");

        try {
            Thread.sleep(5000);  // 5 secs sleep - too tired
        }
        catch (InterruptedException ex) {
            System.out.println("task was interrupted");
            ex.printStackTrace();
        }

        // calculate some result - create an overflow!
        int sum = 0;
        for (int i=1; i<=10000000; i++)
            sum += i;

        result = sum;

        return sum;
    }

    // execute in the event dispatch thread after this swing worker
    // is finished
    public void done() {
        resultLabel.setText(result+"");
    }

}

// window event Handler class
class MyWindowListener extends WindowAdapter {
    public void windowClosing(WindowEvent e) {
        System.out.println("Closing window!");
        System.exit(0);
    }
```

```java
}

// button event handler class
class MyActionListener implements ActionListener {
    AsynchronousTask task;

    MyActionListener(AsynchronousTask task) {
        this.task = task;
    }

    public void actionPerformed(ActionEvent e) {
        task.execute();
    }
}


class SimpleSwingWorkerExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame();

        JLabel label = new JLabel("Result: ");
        JButton button = new JButton("AsyncTask");


        // put components next to each other in the x-direction
        Container c = frame.getContentPane();
        c.setLayout(new BoxLayout(c, BoxLayout.X_AXIS));

        // add label and button in the frame
        c.add(button);
        c.add(label);

        // create the task object (don't execute it yet)
        AsynchronousTask task = new AsynchronousTask(label);

        // register an event handler for frame events
        frame.addWindowListener(new MyWindowListener());

        // register an event handler for button events
        button.addActionListener(new MyActionListener(task));

        frame.setVisible(true);
        //frame.setSize(400, 400);
        frame.pack();

        /* demonstration of waiting for the second thread to
           finish, this is not necessary for this example */
        int result = 0;
        try {
            result = task.get();
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }

        // display the result on the terminal as well!
        System.out.println("result is:" + result);
    }
}
```

4 Interrupted Threads

```
/* this example is adopted from the Oracle Java tutorial */
public class SimpleThreads {

    // Display a message, preceded by
    // the name of the current thread
    static void threadMessage(String message) {
        String threadName =
            Thread.currentThread().getName();
        System.out.format("%s: %s%n",
                          threadName,
                          message);
    }

    private static class MessageLoop
        implements Runnable {
        public void run() {
            String importantInfo[] = {
                "Mares eat oats",
                "Does eat oats",
                "Little lambs eat ivy",
                "A kid will eat ivy too"
            };
            try {
                for (int i = 0;
                     i < importantInfo.length;
                     i++) {
                    // Pause for 4 seconds
                    Thread.sleep(4000);
                    // Print a message
                    threadMessage(importantInfo[i]);
                }
            } catch (InterruptedException e) {
                threadMessage("I wasn't done!");
            }
        }
```

```java
    }

    public static void main(String args[])
        throws InterruptedException {

        // Delay, in milliseconds before
        // we interrupt MessageLoop
        // thread (default one hour).
        long patience = 1000 * 60 * 60;

        // If command line argument
        // present, gives patience
        // in seconds.
        if (args.length > 0) {
            try {
                patience = Long.parseLong(args[0]) * 1000;
            } catch (NumberFormatException e) {
                System.err.println("Argument must be an integer.");
                System.exit(1);
            }
        }

        threadMessage("Starting MessageLoop thread");
        long startTime = System.currentTimeMillis();
        Thread t = new Thread(new MessageLoop());
        t.start();

        threadMessage("Waiting for MessageLoop thread to finish");
        // loop until MessageLoop
        // thread exits
        while (t.isAlive()) {
            threadMessage("Still waiting...");
            // Wait maximum of 1 second
            // for MessageLoop thread
            // to finish.
            t.join(1000);
            if (((System.currentTimeMillis() - startTime) > patience)
                  && t.isAlive()) {
                threadMessage("Tired of waiting!");
                t.interrupt();
                // Shouldn't be long now
                // -- wait indefinitely
                t.join();
            }
        }
        threadMessage("Finally!");
    }
}
```