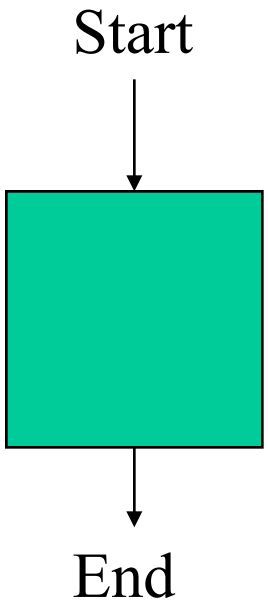
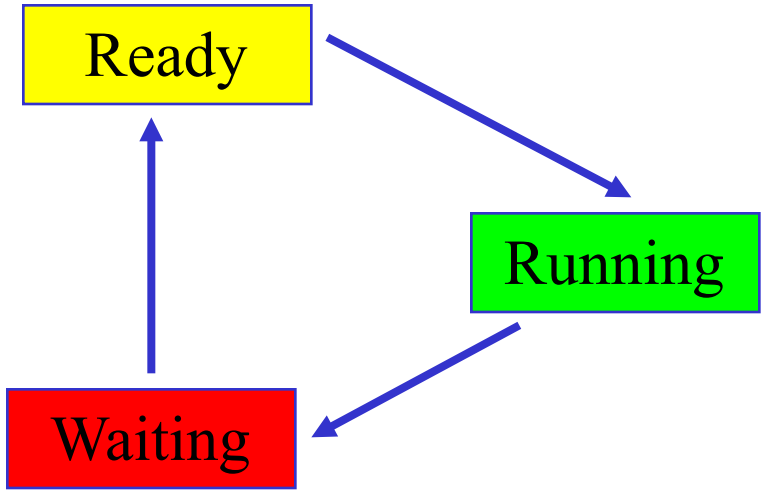
Threads

A *Thread* is an instance of program execution.
The general term is a *Process*

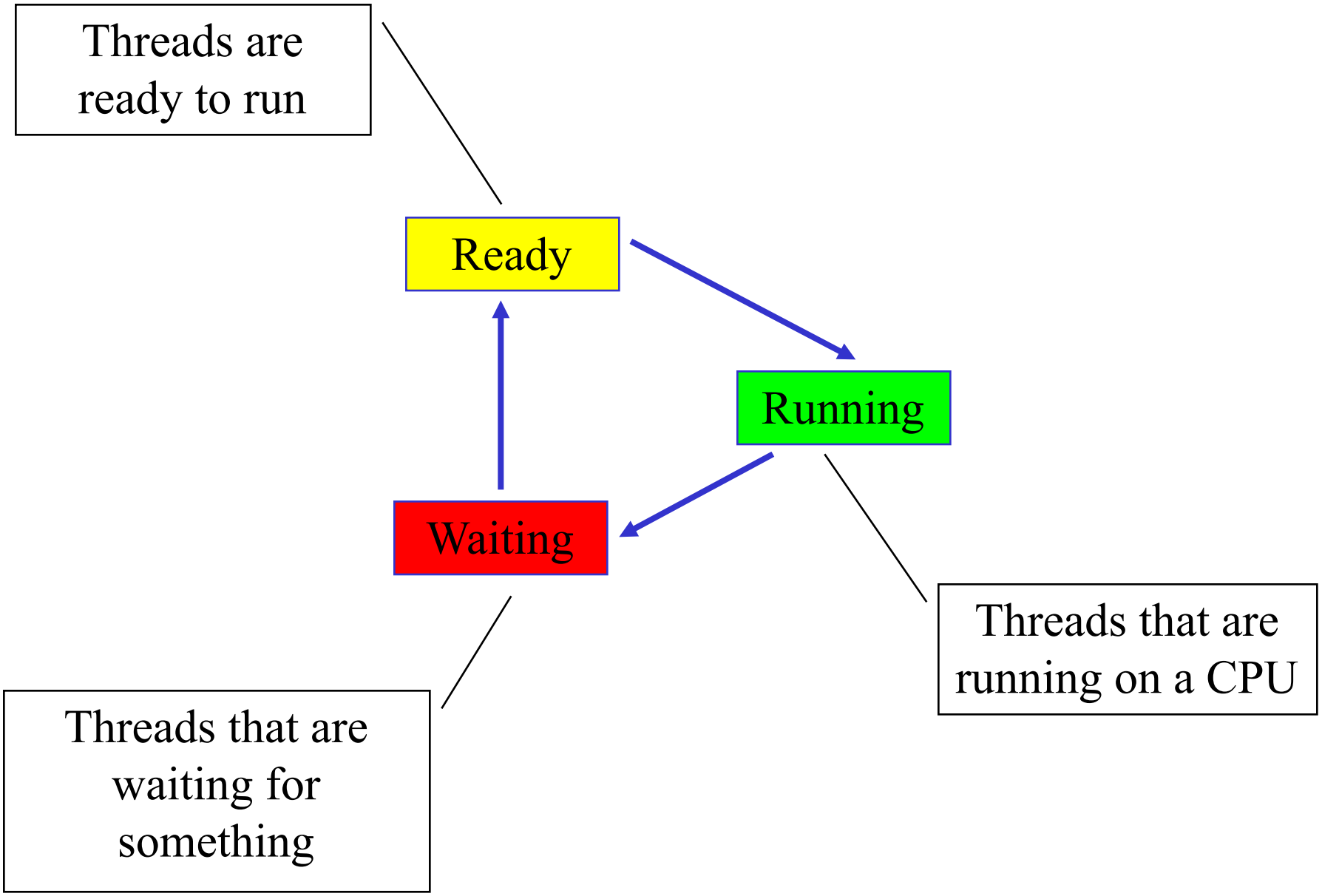


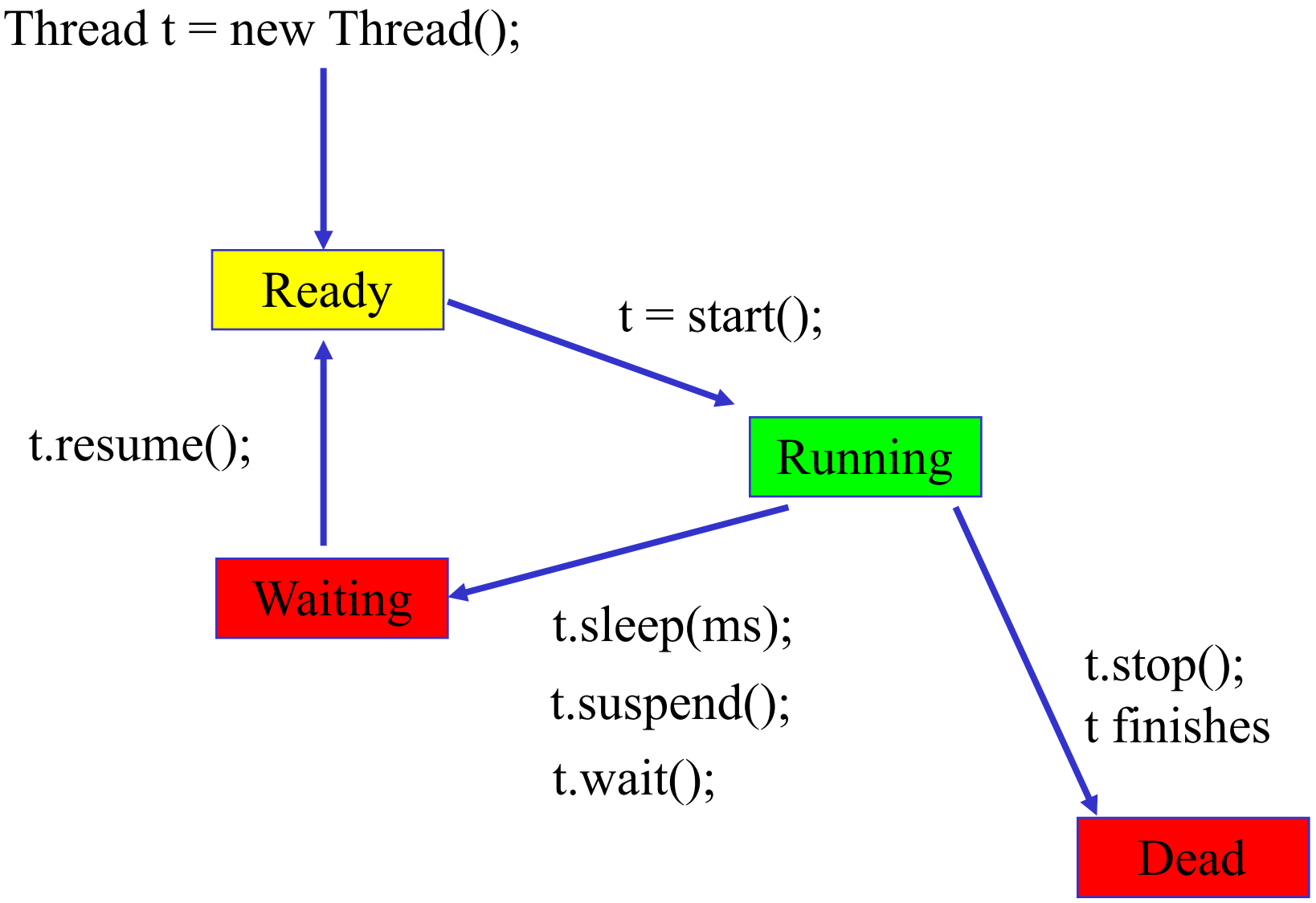
A single Java application could be considered a Thread, but a Java application may contain multiple Threads.

The Value of Threads can be seen by looking at *Process States*



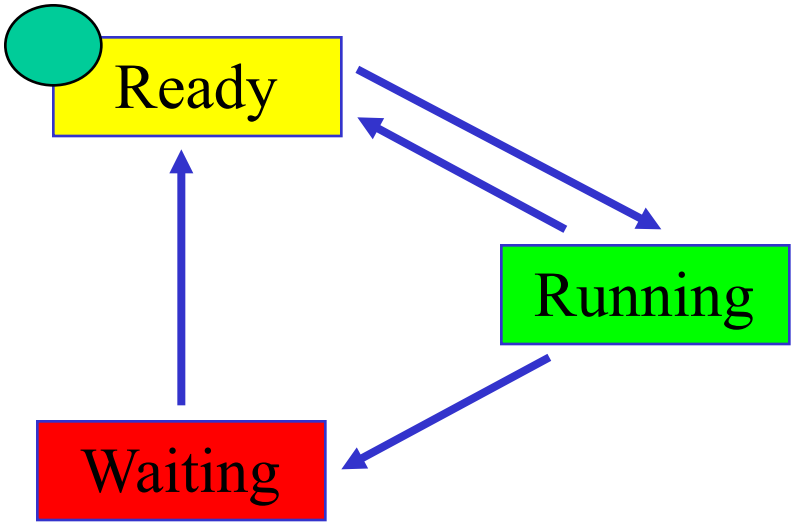
The operating system (or the JVM) is responsible for moving processes (Threads) from state to state.


Threads are ready to run
Ready
Running
Waiting
Threads that are running on a CPU
Threads that are waiting for something


Thread t = new Thread();
Ready
t = start();
t.resume();
Running
Waiting
t.sleep(ms);
t.suspend();
t.wait();
t.stop();
t finishes
Dead

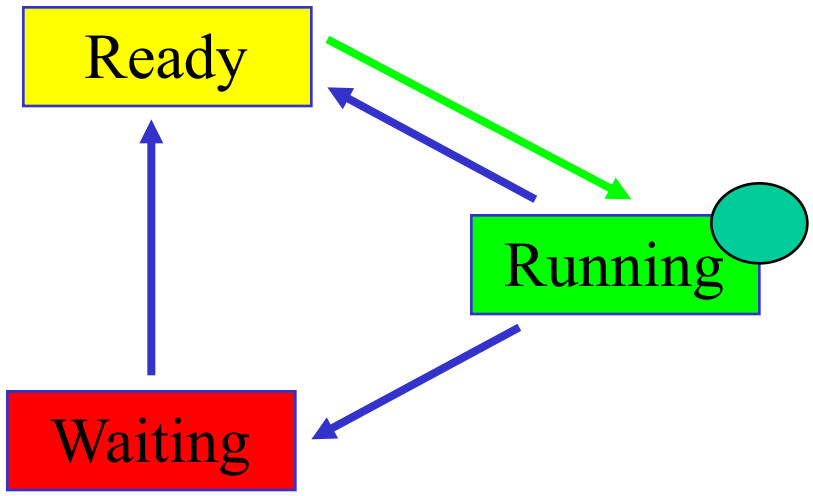
An example of a single Thread (a single Java application):

Program placed in memory, ready to run.



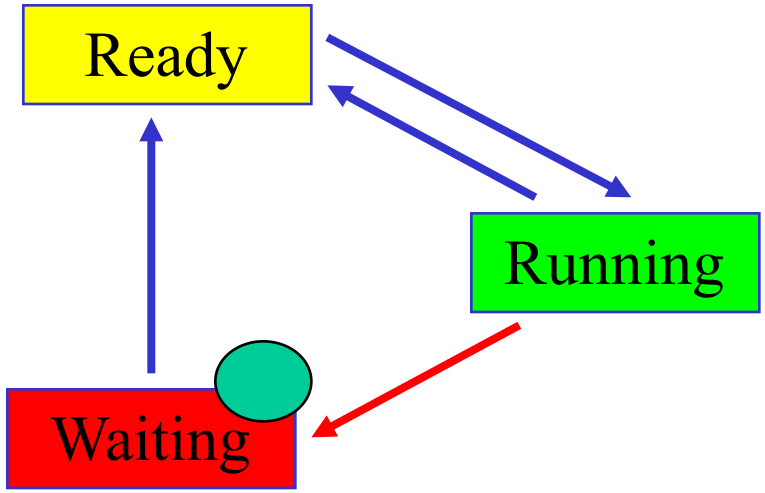
An example of a single Thread (a single Java application):

Program starts running.



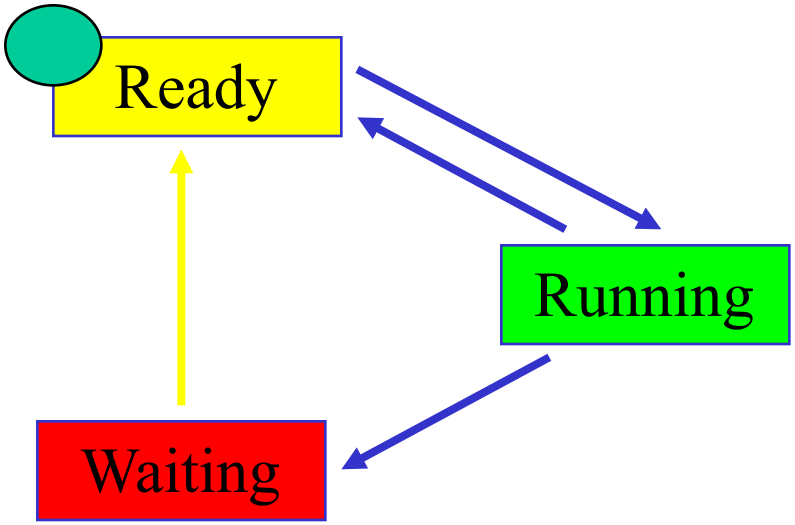
An example of a single Thread (a single Java application):

Program makes an I/O request.

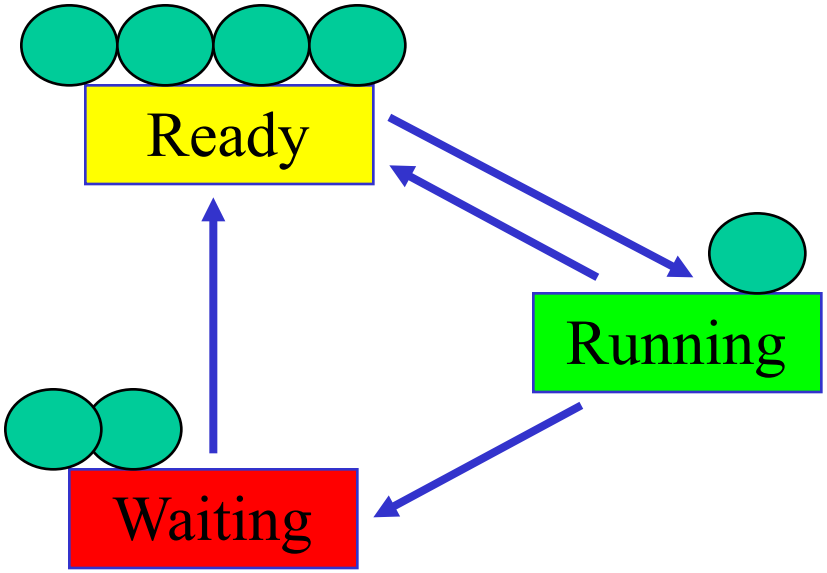


An example of a single Thread (a single Java application):

I/O is finished; program is ready to run again.



Multiple Threads (perhaps in a single Java application):

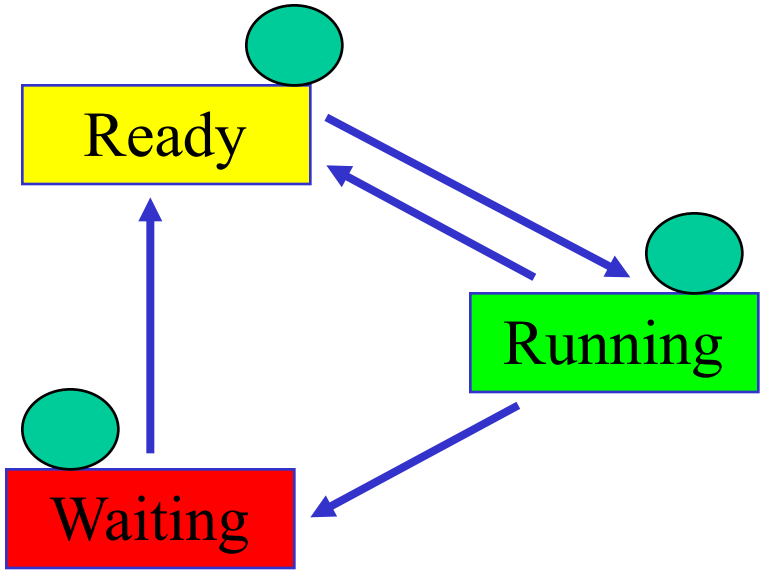


Example:

A game with a timer.

•the timer starts with an initial number of seconds to go, then

- •displays the seconds remaining
- •decreases the seconds remaining by 1
- •sleeps for 1 second.

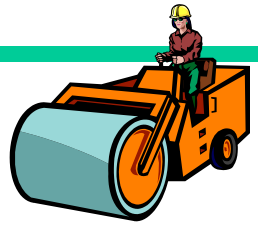


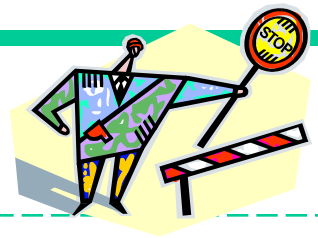
```
import java.awt.*;
import javax.swing.*;
public class TimerJFrame extends JFrame implements Runnable {
   private int secondsRemaining;
   private JTextArea text = new JTextArea();
   public TimerJFrame (int seconds) {
      secondsRemaining = seconds;
      setTitle("Time Remaining...");
      setSize(150,150);
      setLocation  (400,200);
      Container cp = getContentPane();
      text.setFont(new Font("Arial",2,72));
      cp.add(text);
      text.append(Integer.toString(secondsRemaining));
      setVisible(true);
      setDefaultCloseOperation(EXIT_ON_CLOSE);
      Thread timer = new Thread(this);
      timer.start();
   }
```

```
 public void run() {
  System.out.println("The game has started...");
  while (secondsRemaining > 0) {
    try {
         Thread.sleep(1000);
         secondsRemaining--;
         text.setText(Integer.toString(secondsRemaining));
         setVisible(true);
      }
        catch (InterruptedException ie) {
           System.out.println("Timer is interrupted");
        }
      }
   JOptionPane.showMessageDialog(null,"Time is up!");
}
```

```
public class LoggingThread extends Thread {
  private LinkedList linesToLog = new LinkedList();
  private volatile boolean terminateRequested;

  public void run() {
    try {
      while (!terminateRequested) {
        String line;
        synchronized (linesToLog) {
          while (linesToLog.isEmpty())
            linesToLog.wait();
          line = (String) linesToLog.removeFirst();
        }
        doLogLine(line);
      }
    } catch (InterruptedException ex) {
      Thread.currentThread().interrupt();
    }
  }
  private void doLogLine(String line) {
    // ... write to wherever
  }
  public void log(String line) {
    synchronized (linesToLog) {
      linesToLog.add(line);
      linesToLog.notify();
    }
  }
}
```




STOP