```
import javax.swing.*;
import java.awt.Component;
import java.awt.Graphics;

public class ResizableIcon extends ImageIcon
{
    public ResizableIcon(String name)
    {
        super(name);
    }

    public void paintIcon(Component c, Graphics g, int x, int y)
    {
        int width = c.getWidth();
        int height = c.getHeight();
        g.drawImage(getImage(),0,0,width,height,c);
    }
}
```

```
import javax.swing.*;
import java.awt.BorderLayout;
import java.awt.Font;

public class ShowTime extends JPanel implements TimerListener
{
    private JLabel myLabel;

    public ShowTime()
    {
        setLayout(new BorderLayout());
        myLabel = new JLabel("   ", SwingConstants.LEFT);
        myLabel.setFont(new Font("SansSerif",Font.BOLD,24));
        add(myLabel,BorderLayout.CENTER);
    }

    public void timerStarted(Timer timer)
    {

    }

    public void timerStopped(Timer timer)
    {

    }

    public void timerTicked(Timer timer)
    {
        long timeLeft = timer.timeRemaining();
        long secs = timeLeft / 1000;
        long decs = timeLeft % 1000;
        String s = ""+secs+"."+decs;
        myLabel.setText(s);
        repaint();
    }
}
```

```
import javax.swing.*;
import java.awt.GridLayout;
import java.awt.Font;

public class ShowTimeGif extends JPanel implements TimerListener
{
    private JLabel[] myLabels;
    private Icon[] myIcons;
    private static final int LCOUNT = 6;
    private static final int DOTPOS = 3;

    public ShowTimeGif(String rootName)
    {
        setLayout(new GridLayout(1,LCOUNT));
        myLabels = new JLabel[LCOUNT];
        myIcons = getIcons(rootName);

        for(int k=0; k < DOTPOS; k++){
            myLabels[k] = new JLabel(myIcons[0]);
        }
        myLabels[DOTPOS] = new JLabel(" ", SwingConstants.LEFT);
        myLabels[DOTPOS].setFont(new Font("SansSerif",Font.BOLD,24));

        for(int k=DOTPOS+1; k < LCOUNT; k++){
            myLabels[k] = new JLabel(myIcons[0]);
        }
        for(int k=0; k < LCOUNT; k++){
            add(myLabels[k]);
        }

    }

    public Icon[] getIcons(String rootName)
    {
        Icon[] list = new Icon[10];
        for(int k=0; k <= 9; k++){
            String name = ""+k+rootName+".GIF";
            Icon ic = new ResizableIcon("images/"+name);
            list[k] = ic;
        }
        return list;
    }

    public void timerStarted(Timer timer)
    {

    }

    public void timerStopped(Timer timer)
    {

    }

    public void timerTicked(Timer timer)
    {
        long timeLeft = timer.timeRemaining();
        int secs = (int) timeLeft / 1000;
        int decs = (int) timeLeft % 1000;
        String s = ""+secs+"."+decs;

        myLabels[0].setIcon(myIcons[secs/100]);
        myLabels[1].setIcon(myIcons[(secs % 100)/10]);
        myLabels[2].setIcon(myIcons[secs % 10]);

        myLabels[4].setIcon(myIcons[decs/100]);
        myLabels[5].setIcon(myIcons[(decs%100)/10]);
        repaint();
    }
}
```

```
import javax.swing.*;

public class TimeGui extends JFrame
{

    public TimeGui(Timer t, String gifName)
    {
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        ShowTimeGif panel = new ShowTimeGif(gifName);
        setContentPane(panel);
        t.addListener(panel);
        pack();
        setVisible(true);
    }
}
```

```
import java.util.*;

/**
 * Simple example of Timer that reports
 * progress to TimerListeners. This is a count-down
 * timer.
 *
 * @author Owen Astrachan
 */

public class Timer implements Runnable
{
    /**
     * Create timer with default duration of 3 seconds
     */
    public Timer()
    {
        this(3);
    }

    /**
     * Create timer with specified duration in seconds
     * @param duration is number of seconds to countdown
     */
    public Timer(int duration)
    {
        myListeners = new ArrayList();
        myRunning = false;
        myThread = null;
        myTickPause = 1000;
        myDuration = duration*1000;
    }

    /**
     * Sets pause in milliseconds, this is rate
     * at which timer fires
     * @param pause is # milliseconds between timer firings
     */

    public void setPause(int pause)
    {
        myTickPause = pause;
    }

    /**
     * Add a listener to receive timer events
     * @param tl is the listener added
     */
    public void addListener(TimerListener tl)
    {
        myListeners.add(tl);
    }

    /**
     * Remove a listener (error if not listening)
     * @param tl is the listener removed
     */
    public void removeListener(TimerListener tl)
    {
        myListeners.remove(tl);
    }

    /**
     * Starts this timer should notify listeners, but doesn't
     */
    public void start()
    {
        myThread = new Thread(this);
        myStartTime = new Date();
```



```
        myThread.setDaemon(true);
        myThread.start();
    }

    /**
     * Stops timer, notifies listeners that timer has stopped
     */
    public void stop()
    {
        myRunning = false;
        myStopTime = new Date();
        for(int k=0; k < myListeners.size(); k++) {
            ( (TimerListener) myListeners.get(k)).timerStopped(this);
        }
    }

    /**
     * Does work, notifies listeners when ticking occurs
     */
    public void run()
    {
        myRunning = true;

        while (myRunning) {

            try {
                pause();
            }
            catch (InterruptedException e) {
                myRunning = false;
                return;
            }

            handleTicks();

            if (isExpired()) {
                stop();
            }
        }
    }

    /**
     * Returns true if timer has expired.
     * @return true iff this timer has expired
     */

    public boolean isExpired()
    {
        myStopTime = new Date();
        return timeRemaining() <= 0;
    }

    /**
     * Returns # milliseconds left before timer expires.
     * @return milliseconds remaining before isExpired() returns true
     */
    public long timeRemaining()
    {
        myStopTime = new Date();
        long secs = myStopTime.getTime() - myStartTime.getTime();
        long value = myDuration - secs;
        return value < 0 ? 0 : value;
    }

    private void pause() throws InterruptedException
    {
        myThread.sleep(myTickPause);
    }
```

```
    /**
     * Helper function to notify listeners
     */
    private void handleTicks()
    {
        for(int k=0; k < myListeners.size(); k++) {
            ( (TimerListener) myListeners.get(k)).timerTicked(this);
        }
    }

    private boolean myRunning;
    private long    myDuration;
    private Date    myStartTime;
    private Date    myStopTime;
    private Thread  myThread;
    private int     myTickPause;

    private List myListeners;
}
```

```
import javax.swing.*;
import java.awt.BorderLayout;
import java.awt.event.*;
import java.util.Random;

public class TimerController extends JFrame
{
    private Timer myTimer;

    public TimerController()
    {
        myTimer = new Timer(19);
        myTimer.setPause(30);
        JPanel panel = new JPanel(new BorderLayout());
        panel.add(makeButtons(), BorderLayout.CENTER);
        setContentPane(panel);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        pack();
        setVisible(true);
    }

    private JPanel makeButtons()
    {
        final JButton start = new JButton("start");
        final JButton stop = new JButton("stop");
        final JButton add = new JButton("add gui");
        final Random rando = new Random();

        final String[] gifNames = {
            "ARIALB", "SPACEMN", "BIGWAVE"
        };


        start.addActionListener(new ActionListener()
            {
                public void actionPerformed(ActionEvent e){
                    myTimer.start();
                }
            });
        stop.addActionListener(new ActionListener()
            {
                public void actionPerformed(ActionEvent e){
                    myTimer.stop();
                }
            });
        add.addActionListener(new ActionListener()
            {
                public void actionPerformed(ActionEvent e){
                    TimeGui gui = new TimeGui(myTimer,
                                              gifNames[rando.nextInt(3)]);
                }
            });

        JPanel panel = new JPanel();
        panel.add(start);
        panel.add(stop);
        panel.add(add);
        return panel;
    }

    public static void main(String[] args)
    {
        TimerController controller = new TimerController();
    }
}
```

```
public interface TimerListener
{
    public void timerStarted(Timer timer);
    public void timerStopped(Timer timer);
    public void timerTicked(Timer timer);
}
```