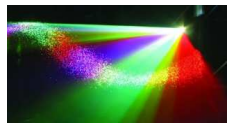
Chapter 3
Programs

The Design and Implementation of Multimedia Software

David Bernstein

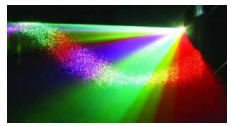
Jones and Bartlett Publishers

www.jbpub.com



About this Chapter

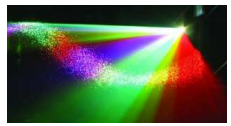
- Thus far the the phrase “software product” has been used instead of the word “program”.
- This chapter develops a (somewhat) formal definition of the word “program”.
- This chapter also designs and implements a particular notion of a “program” that is especially important in the context of multimedia software products.



A Definition

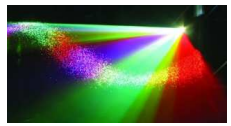
Definition

A *program* in an object-oriented programming language is a group of cooperating classes with a well-defined *entry point* (i.e., a method that should be executed first) and, perhaps, a re-entry point and/or an exit point.



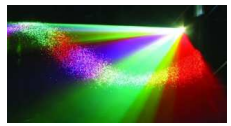
Java Programs with GUIs

	Environment	**Top-Level Container**	**Entry Point**
Applet	Browser	`JApplet`	`init()` then `start()`
Application	Operating System	`JFrame`	`main()`
MIDlet	Operating System	`Screen`	`startApp()`
Servlet	(HTTP) Server	N/A	`init()` then `service()`



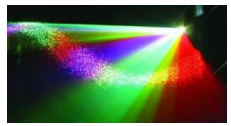
Application Lifecycle

- When an application is started the `main()` method is executed in a non-daemon thread called the *main thread*.
- A single-threaded application terminates when the `System.exit()` method is called, in response to a platform-specific event such as a `SIGINT` or a `Ctrl-C`, or when the main thread 'drops out of' the `main()` method.
- A multi-threaded application terminates when the `System.exit()` method is called, in response to a platform specific event, or when all non-daemon threads have died.



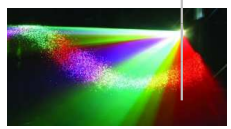
Applet Lifeycyle

- When an HTML page containing an `<applet>` element is loaded for the first time, the appropriate object (i.e., the descendent of the `Applet` class referred to in the `<applet>` element) is constructed, its `init()` and `start()` methods are called in a thread other than the event dispatch thread.
- Each time the user leaves the page containing the applet, the `stop()` method is called (again, not in the event dispatch thread).
- Similarly, each time the user re-loads the page containing the applet, the `start()` method is called.
- When the browser is shut down, the `destroy()` method is called (again, not in the event dispatch thread).



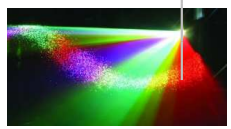
An Application

```
// Java libraries
import javax.swing.*;

// Multimedia libraries
import event.*;

public class BadTimedMessageSwingApplication implements MetronomeListener,
                                                        Runnable
{
  private static final String[] MESSAGES = {
     "What a great book.","Bring on the exercises.",
     "Author, author!","I hope it never ends."};

  private int       index;
  private JLabel    label;
  private Metronome metronome;

  public static void main(String[] args)
  {
    try
    {
      SwingUtilities.invokeAndWait(new BadTimedMessageSwingApplication());
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
  }
```



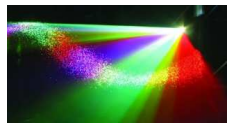
An Application (cont.)

```
public void handleTick(int millis)
{
  index = (index + 1) % MESSAGES.length;
  label.setText(MESSAGES[index]);
}

public void run()
{
  // Setup the window
  JFrame frame = new JFrame();
  frame.setSize(400,  200);

  // Setup the content pane
  JPanel contentPane = (JPanel)frame.getContentPane();
  contentPane.setLayout(null);

  // Add a component to the container
  label = new JLabel("  ", SwingConstants.CENTER);
  label.setBounds(0, 0, 400, 200);
  contentPane.add(label);

  metronome = new Metronome(1000);
  metronome.addListener(this);
  metronome.start();
```

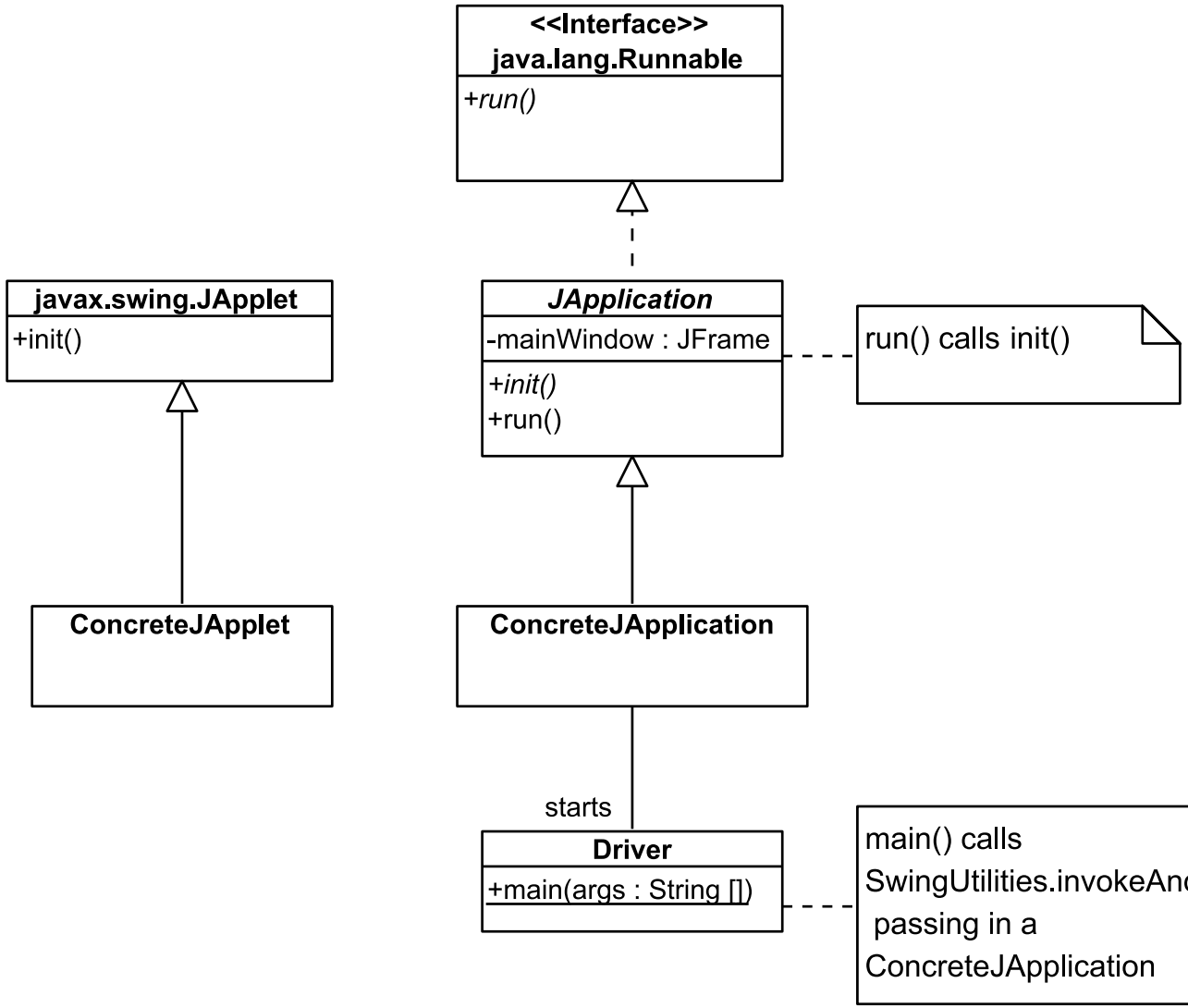


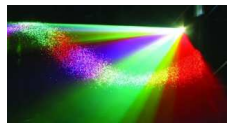
An Application (cont.)

```
    frame.setVisible(true);
  }
}
```



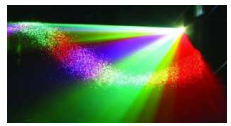
Initial Design





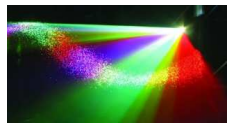
Alternative 1 - The run() Method

```
public final void run()
{
  constructMainWindow();
  init();
  mainWindow.setVisible(true);
}
```



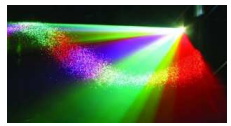
Alternative 1 - The constructMainWindow() Method

```
mainWindow = new JFrame();
mainWindow.setTitle("Multimedia Software - jblearning.com
mainWindow.setResizable(false);

JPanel contentPane = (JPanel)mainWindow.getContentPane();
contentPane.setLayout(null);
contentPane.setDoubleBuffered(false);
```



Alternative 1 - The `init()` Method

```
public abstract void init();
```



An Applet-Like Lifecycle

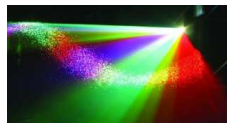
- The Issue:

  At this point, only the entry points are similar.

  Ideally, the transition methods would also be simialr.

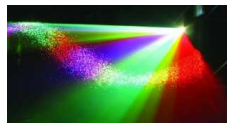
- Resolution:

  Make `JApplication` a `WindowListener` on its main window.



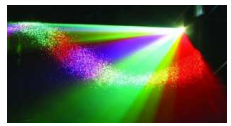
Step 1

```
mainWindow.setDefaultCloseOperation(
    JFrame.DO_NOTHING_ON_CLOSE);
mainWindow.addWindowListener(this);
```



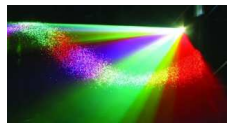
Step 2a

```
public void windowOpened(WindowEvent event)
{
  resize();
  start();
}
```



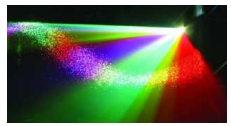
Step 2b

```
public void windowDeiconified(WindowEvent event)
{
  start();
}
```



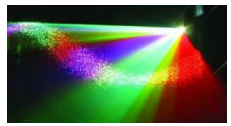
Step 2c

```
public void windowIconified(WindowEvent event)
{
  stop();
}
```



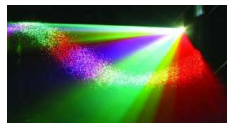
Step 2d

```
public void windowClosing(WindowEvent event)
{
  exit();
}
```

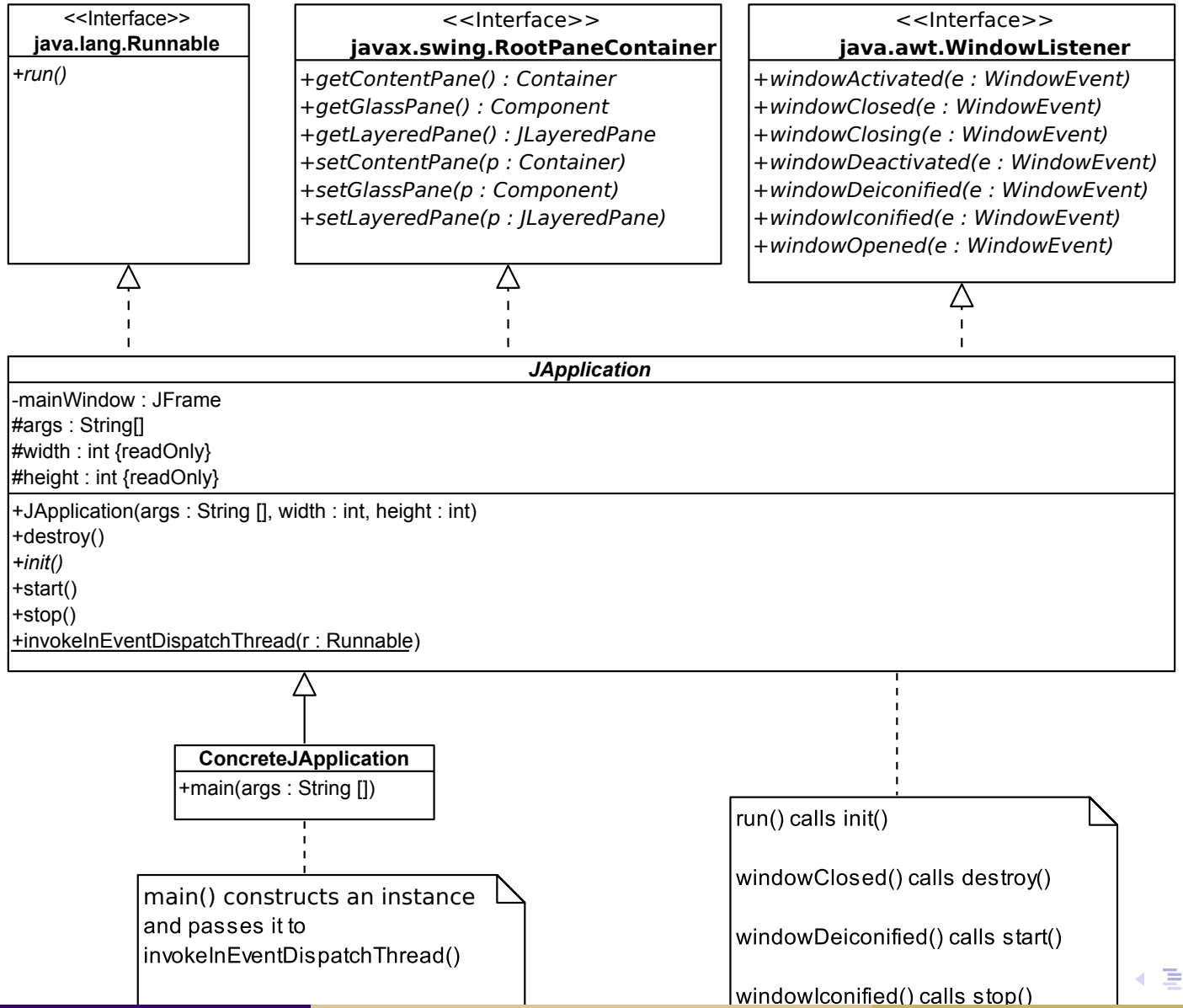


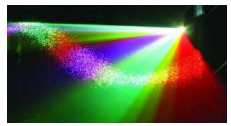
Step 2e

```
public void windowClosed(WindowEvent event)
{
  destroy();
  System.exit(0);
}
```



The Final Design





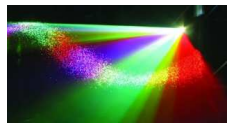
An Applet-Like Top-Level Container

- The Issue:

  A program shouldn't use its top-level container directly.

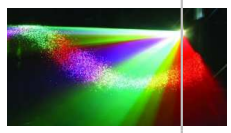
- Resolution:

  Make `JApplication` a `RootPaneContainer`.



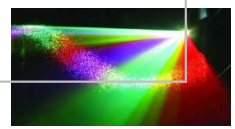
Pane Getters

```
public Container getContentPane()
{
  return mainWindow.getContentPane();
}

public Component getGlassPane()
{
  return mainWindow.getGlassPane();
}

public JLayeredPane getLayeredPane()
{
  return mainWindow.getLayeredPane();
}

public JRootPane getRootPane()
```



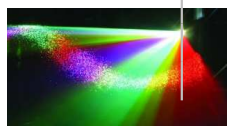
Pane Setters

```
public void setContentPane(Container contentPane)
{
  mainWindow.setContentPane(contentPane);
}

public void setGlassPane(Component glassPane)
{
  mainWindow.setGlassPane(glassPane);
}

public void setLayeredPane(JLayeredPane layeredPane)
{
  mainWindow.setLayeredPane(layeredPane);
}
```



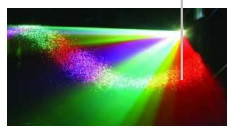
An Example

```
// Java libraries
import javax.swing.*;

// Multimedia libraries
import app.*;
import event.*;

public class TimedMessageJApplication extends JApplication
                                      implements MetronomeListener
{
  private static final String[] MESSAGES = {
     "What a great book.","Bring on the exercises.",
     "Author, author!","I hope it never ends."};

  private int       index;
  private JLabel    label;
  private Metronome metronome;
  public static void main(String[] args)
  {
    JApplication demo = new TimedMessageJApplication(args, 400, 200);
    invokeInEventDispatchThread(demo);
  }

  public TimedMessageJApplication(String[] args, int width, int height)
  {
    super(args, width, height);
    index = -1;
  }
```



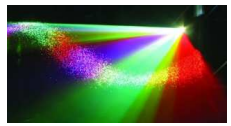
An Example (cont.)

```
public void handleTick(int millis)
{
  index = (index + 1) % MESSAGES.length;
  label.setText(MESSAGES[index]);
}
public void init()
{
  // Setup the content pane
  JPanel contentPane = (JPanel)getContentPane();
  contentPane.setLayout(null);

  // Add a component to the container
  label = new JLabel("  ", SwingConstants.CENTER);
  label.setBounds(0, 0, 400, 200);
  contentPane.add(label);

  metronome = new Metronome(1000);
  metronome.addListener(this);
}

public void start()
{
  metronome.start();
}

public void stop()
```

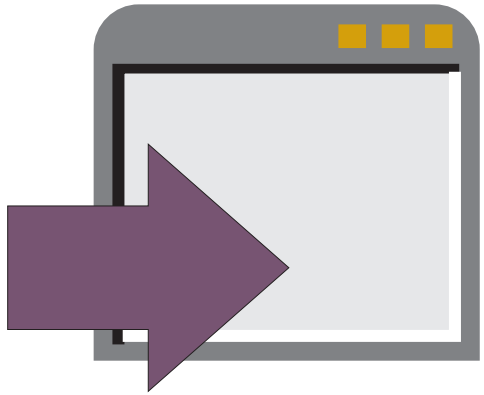


An Example (cont.)

```
  {
    metronome.stop();
  }
}
```

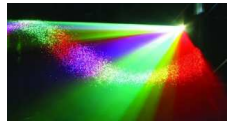


A Demo



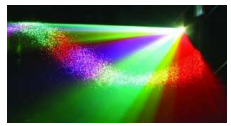
In `examples/chapter/`:

```
java -cp multimedia2.jar:examples.jar TimedMessageJapplication
```



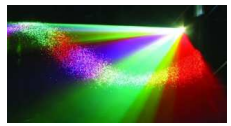
The Issue

- Most multimedia programs, be they applications or applets, need to 'load' resources of various kinds (e.g, artwork, preferences) at run-time.
- This can be problematic because of the different ways in which applets and applications can be 'organized' (e.g., in a `.jar` file, in a packaged set of classes, in an un-packaged set of classes) and 'delivered/installed' (e.g., by an HTTP server, by an installer, as files on a CD/DVD).
- Hence, it can be very difficult for a program to know where resources are.



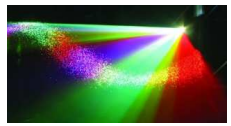
How Does the Interpreter Do It?

- The Java interpreter obtains the byte codes that constitute a class using a `class loader`.
- We can do the same thing using *reflection*.



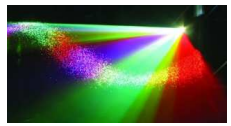
Reflection Basics

- Every interface, class and object in Java has an associated `Class` object that can be used to obtain information about it.
- This information is encapsulated as `Constructor`, `Field`, `Method` , and `Type` objects.



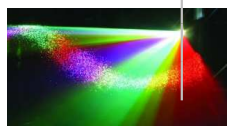
Creating a `ResourceFinder`

- Use the `getResource()` and `getResourceAsStream()` methods in `Class` objects.
- Allow it to use either its class loader or another object's class loader.



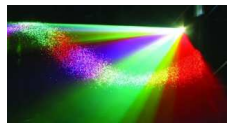
Structure of the ResourceFinder

```
package io;
import java.io.*;
import java.net.*;
import java.util.*;


public class ResourceFinder
{
  private static final Map<Class<?>, ResourceFinder> pool = new HashMap<Class<?>, ResourceFinder>();

  private final Class<?>  c;


  private ResourceFinder(final Class<?> c)
  {
    this.c = c;
  }

  public static ResourceFinder createInstance()
  {
    return createInstance(ResourceFinder.class);
  }

  public static ResourceFinder createInstance(final Object o)
  {
    return createInstance(o.getClass());
  }
```



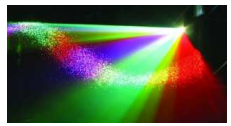
Structure of the ResourceFinder (cont.)

```
  public static ResourceFinder createInstance(final Class<?> c)
  {
    ResourceFinder result = pool.get(c);
    if (result == null)
    {
      result = new ResourceFinder(c);
      pool.put(c, result);
    }

    return result;
  }
}
```



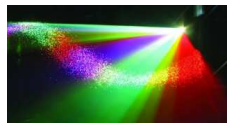
The `findInputStream()` Method

```
public InputStream findInputStream(String name)
{
   InputStream    is;

   is    = c.getResourceAsStream(name);

   return is;
}
```



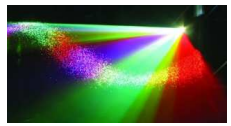
The `findInputStream()` Method

```
public URL findURL(String name)
{
   URL            url;

   url   = c.getResource(name);

   return url;
}
```



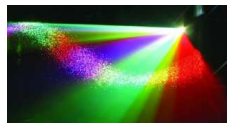
Another Issue

- IDEs often keep the source code and byte code in different directories.
- IDEs often copy resources and/or set the `classpath` at runtime.
- IDEs are often used to create `.jar` files "automatically".
- Combined, these conveniences can make it difficult to know where a resource will be at runtime.



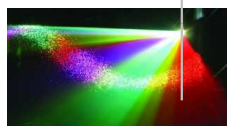
A "Trick"

- Put the resources in a **package** (e.g., named `resources`).
- Include an empty class (e.g., named `Marker`) in the resources package that a `ResourceFinder` can use.



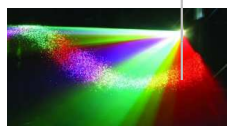
An Example

```
// Java libraries
import java.io.*;
import java.util.*;
import javax.swing.*;

// Multimedia libraries
import app.*;
import event.*;
import io.*;

public class TimedMessageDemo extends    JApplication
                              implements MetronomeListener
{
  private ArrayList<String> messages = new ArrayList<String>();

  private int               index;
  private JLabel            label;
  private Metronome         metronome;
  public static void main(String[] args)
  {
    JApplication demo = new TimedMessageDemo(args, 400, 200);
    invokeInEventDispatchThread(demo);
  }

  public TimedMessageDemo(String[] args, int width, int height)
  {
    super(args, width, height);
    index = -1;
```



An Example (cont.)

```
  ResourceFinder rf = ResourceFinder.createInstance(new resources.Marker());
  InputStream    is = rf.findInputStream("messages.txt");
  BufferedReader in     = new BufferedReader(new InputStreamReader(is));

  String line;
  try
  {
    while ((line = in.readLine()) != null)
    {
      messages.add(line);
    }
  }
  catch (IOException ioe)
  {
    messages.add("Best book ever!");
  }
}

public void handleTick(int millis)
{
  index = (index + 1) % messages.size();
  label.setText(messages.get(index));
}

public void init()
{
  // Setup the content pane
```



An Example (cont.)

```
    JPanel contentPane = (JPanel)getContentPane();
    contentPane.setLayout(null);

    // Add a component to the container
    label = new JLabel("  ", SwingConstants.CENTER);
    label.setBounds(0, 0, 400, 200);
    contentPane.add(label);

    metronome = new Metronome(1000);
    metronome.addListener(this);
  }

  public void start()
  {
    metronome.start();
  }

  public void stop()
  {
    metronome.stop();
  }
}
```