Chapter 3
Programs

The Design and Implementation of Multimedia Software

David Bernstein
Jones and Bartlett Publishers

www.jbpub.com
Thus far 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 discusses a way to unify different types of Java programs (which 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

<table>
<thead>
<tr>
<th>Environment</th>
<th>Top-Level Container</th>
<th>Entry Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Operating System</td>
<td>JFrame</td>
</tr>
<tr>
<td>Applet</td>
<td>Browser</td>
<td>JApplet</td>
</tr>
</tbody>
</table>
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 Lifecycle

- 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).
import java.awt.event.*;
import java.util.*;
import javax.swing.*;

public class BadInteractiveRandomMessageSwingApplication
    implements ActionListener, Runnable
{
    // Attributes
    private JLabel label;

    // The pseudo-random number generator
    private static Random rng = new Random();

    // String "constants"
    private static final String CHANGE = "Change";

    // The messages
    private static final String[] MESSAGES = {
        "What a great book.", "Bring on the exercises.",
        "Author, author!", "I hope it never ends."};

    public static void main(String[] args) throws Exception
    {
        SwingUtilities.invokeLater(
            new BadInteractiveRandomMessageSwingApplication());
    }
public void actionPerformed(ActionEvent event)
{
    String actionCommand;

    actionCommand = event.getActionCommand();
    if (actionCommand.equals(CHANGE))
    {
        label.setText(createRandomMessage());
    }
}

private static String createRandomMessage()
{
    return MESSAGES[rng.nextInt(MESSAGES.length)];
}

public void run()
{
    JButton button;
    JFrame window;
    JPanel contentPane;
    String s;

    // Select a message at random
    s = createRandomMessage();

    // Construct the "window"
    window = new JFrame();
window.setSize(600,400);
window.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

// Get the container for all content
contentPane = (JPanel)window.getContentPane();
contentPane.setLayout(null);

// Add the message component to the container
label = new JLabel(s, SwingConstants.CENTER);
label.setBounds(50,50,500,100);
contentPane.add(label);

// Add the button to the container
button = new JButton(CHANGE);
button.setBounds(450,300,100,50);
contentPane.add(button);
button.addActionListener(this);

// Make the "window" visible
window.setVisible(true);
}
import java.awt.event.*;
import java.util.*;
import javax.swing.*;

public class BadInteractiveRandomMessageJApplet
    extends JApplet
    implements ActionListener
{
    // Attributes
    private JLabel label;

    // The pseudo-random number generator
    private static Random rng = new Random();

    // String "constants"
    private static final String CHANGE = "Change";

    // The messages
    private static final String[] MESSAGES = {
        "What a great book.","Bring on the exercises.",
        "Author, author!","I hope it never ends."};

    public BadInteractiveRandomMessageJApplet()
    {
        super();
    }
}
public void actionPerformed(ActionEvent event)
{
    String actionCommand;
    actionCommand = event.getActionCommand();
    if (actionCommand.equals(CHANGE))
    {
        label.setText(createRandomMessage());
    }
}

private static String createRandomMessage()
{
    return MESSAGES[rng.nextInt(MESSAGES.length)];
}

public void init()
{
    JButton button;
    JPanel contentPane;
    String s;

    // Select a message at random
    s = createRandomMessage();

    // Get the container for all content
    contentPane = (JPanel) getContentPane();
    contentPane.setLayout(null);
// Add a component to the container
label = new JLabel(s, SwingConstants.CENTER);
label.setBounds(50,50,500,100);
contentPane.add(label);

// Add the button to the container
button = new JButton(CHANGE);
button.setBounds(450,300,100,50);
button.addActionListener(this);
contentPane.add(button);
The differences between applications and applets are problematic for multimedia programmers, who must frequently create applications and applets that provide the same functionality.
Dealing with this Problem

Develop distinct applications and applets that share classes (which is not difficult since the general structures of applets and applications are very similar).

What are the shortcomings?

Create a unified system that, to the extent possible, makes it possible to used the same ‘glue code’ (i.e., code that connects the various cooperating classes) in applets and applications.
Dealing with this Problem

- Develop distinct applications and applets that share classes (which is not difficult since the general structures of applets and applications are very similar).

  Code Duplication

- Create a unified system that, to the extent possible, makes it possible to used the same ‘glue code’ (i.e., code that connects the various cooperating classes) in applets and applications.
Requirements

F3.1 Applets and applications must have a common programming interface.

F3.2 Applets and applications must have a common lifecycle.

F3.3 Applets and applications must have a common way to obtain start-up parameters.

N3.4 Transition methods in both applets and applications must be called in the event dispatch thread.
Alternative 1

```
<<Interface>>
java.lang.Runnable
+run()

javax.swing.JApplet
+init()

JApplication
-mainWindow : JFrame
+init()
+run()

ConcreteJApplet

ConcreteJApplication

starts

Driver
+main(args : String [])

main() calls
SwingUtilties.invokeAndWait()
passing in a
ConcreteJApplication
```
public final void run()
{
    constructMainWindow();
    init();
    mainWindow.setVisible(true);
}
Alternative 1 - The `constructMainWindow()` Method

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

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

David Bernstein (jbpub.com)
Alternative 1 - The init() Method

public abstract void init();
import java.util.*;
import javax.swing.*;
import app.JApplication;

public class BadRandomMessageJApplication
    extends JApplication
{
    // Attributes
    private JLabel label;
    
    // The pseudo-random number generator
    private static Random rng = new Random();
    
    // The messages
    private static final String[] MESSAGES = {
        "What a great book.", "Bring on the exercises.",
        "Author, author!", "I hope it never ends."};

    public static void main(String[] args) throws Exception
    {
        SwingUtilities.invokeLater(
            new BadRandomMessageJApplication(600,400));
    }

    public BadRandomMessageJApplication(int width, int height)
    {
super(width, height);
}

private static String createRandomMessage()
{
    return MESSAGES[rng.nextInt(MESSAGES.length)];
}

public void init()
{
    JPanel contentPane;
    String s;

    // Select a message at random
    s = createRandomMessage();

    // Get the container for all content
    contentPane = (JPanel) getContentPane();
    contentPane.setLayout(null);

    // Add a component to the container
    label = new JLabel(s, SwingConstants.CENTER);
    label.setBounds(50, 50, 500, 100);
    contentPane.add(label);
}
Alternative 1 - Shortcomings

- Requirement 3.3 is not satisfied because the RootPaneContainer for an applet (which is the JApplet itself) has access to the start-up parameters whereas the RootPaneContainer for an application does not.

- Requirement 3.4 is also not satisfied because the transition methods (i.e., the `init()`, `start()`, `stop()` and `destroy()` methods) in a JApplet are not called in the event dispatch thread.

- It still encourages code duplication.
package app;

import javax.swing.*;

public interface MultimediaRootPaneContainer
    extends RootPaneContainer
{
    public abstract String getParameter(String name);
}
Use the Decorator Pattern

```
<<Interface>>
MultimediaApp
+destroy()
+init()
+setMultimediaRootPaneContainer(container : MultimediaRootPaneContainer)
+start()
+stop()
```

```
JApplet
```

```
JApplication
```

```
MultimediaApplet
-decorated : MultimediaApp
```

```
MultimediaApplication
-decorated : MultimediaApp
-params : Properties
```

Delegates to decorated
Alternative 2 - MultimediaApp

```java
package app;

import javax.swing.*;

public interface MultimediaApp
{
    public abstract void destroy();

    public abstract void init();

    public abstract void setMultimediaRootPaneContainer(
        MultimediaRootPaneContainer container);

    public abstract void start();

    public abstract void stop();
}
```
package app;

import javax.swing.*;

public abstract class AbstractMultimediaApp implements MultimediaApp {
    protected MultimediaRootPaneContainer rootPaneContainer;

    public void destroy() {
    }

    public void init() {
    }

    public void setMultimediaRootPaneContainer(
        MultimediaRootPaneContainer container)
    {
        rootPaneContainer = container;
    }

    public void start() {
    }
}
public void stop()
{
}
}
package app;

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

public abstract class MultimediaApplet
    extends JApplet
    implements MultimediaRootPaneContainer
{
    private MultimediaApp app;

    public MultimediaApplet(MultimediaApp app)
    {
        super();
        this.app = app;
        setLayout(null);
        app.setMultimediaRootPaneContainer(this);
    }

    protected MultimediaApp getMultimediaApp()
    {
        return app;
    }
}
Satisfying Requirement 3.4

```java
public void destroy()
{
    if (SwingUtilities.isEventDispatchThread()) app.destroy();
    else
    {
        try {SwingUtilities.invokeLater(new DestroyRunnable());}
        catch (Exception e) {} 
    }
}

public void init()
{
    if (SwingUtilities.isEventDispatchThread()) app.init();
    else
    {
        try {SwingUtilities.invokeLater(new InitRunnable());}
        catch (Exception e) {e.printStackTrace();}
    }
}

public void start()
{
    if (SwingUtilities.isEventDispatchThread()) app.start();
    else
    {
        try {SwingUtilities.invokeLater(new StartRunnable());}
        catch (Exception e) {e.printStackTrace();}
    }
}
```
Satisfying Requirement 3.4 (cont.)

```java
private class DestroyRunnable implements Runnable {
    public void run() {
        app.destroy();
    }
}

private class InitRunnable implements Runnable {
    public void run() {
        app.init();
    }
}

private class StartRunnable implements Runnable {
    public void run() {
        app.start();
    }
}

private class StopRunnable implements Runnable {
    public void run() {
        app.stop();
    }
}
```
Alternative 2 - An Example

```java
import java.util.*;
import javax.swing.*;
import app.*;

public class RandomMessageApp
    extends AbstractMultimediaApp
{
    // Attributes
    private JLabel label;

    // The pseudo-random number generator
    private static Random rng = new Random();

    // The messages
    private static final String[] MESSAGES = {
        "What a great book.",
        "Bring on the exercises.",
        "Author, author!",
        "I hope it never ends."};

    private static String createRandomMessage()
    {
        return MESSAGES[rng.nextInt(MESSAGES.length)];
    }

    public void init()
    {
        JPanel contentPane;
        ```
String s;

// Select a message at random
s = createRandomMessage();

// Get the container for all content
contentPane = (JPanel)rootPaneContainer.getContentPane();
contentPane.setLayout(null);

// Add a component to the container
label = new JLabel(s, SwingConstants.CENTER);
label.setBounds(50,50,500,100);
contentPane.add(label);
import app.*;

public class RandomMessageMultimediaApplet
    extends MultimediaApplet
{
    public RandomMessageMultimediaApplet()
    {
        super(new RandomMessageApp());
    }
}
Alternative 2 - The Applet Demo

In examples/chapter:

RandomMessage.html
package app;

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

public abstract class MultimediaApplication
extends JApplication
implements MultimediaRootPaneContainer
{
    private MultimediaApp app;
    private Properties params;

    public MultimediaApplication(String[] args,
                                  MultimediaApp app,
                                  int width, int height)
    {
        super(width, height);

        this.app = app;
        app.setMultimediaRootPaneContainer(this);

        params = new Properties();
        for (int i=0; i<args.length; i++)
        {
            params.put(Integer.toString(i), args[i]);
        }
    }
}
public void destroy()
{
    app.destroy();
}

protected MultimediaApp getMultimediaApp()
{
    return app;
}

public String getParameter(String name)
{
    return params.getProperty(name);
}

public void init()
{
    app.init();
}

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

public void stop()
Alternative 2 - MultimediaApplication (cont.)

```java
{
    app.stop();
}
```
Alternative 2 - The Application

```java
import app.*;
import java.util.*;
import javax.swing.*;

public class RandomMessageMultimediaApplication
    extends MultimediaApplication
{
    public static void main(String[] args) throws Exception
    {
        SwingUtilities.invokeLater(
            new RandomMessageMultimediaApplication(args, 600, 400));
    }

    public RandomMessageMultimediaApplication(String[] args,
                                               int width, int height)
    {
        super(args, new RandomMessageApp(), width, height);
    }
}
```
Alternative 2 - The Application Demo

In `examples/chapter`:

```
java -cp RandomMessage.jar RandomMessageMultimediaApplication
```
Alternative 2 - One Remaining Issue

- The Issue:
  
  A MultimediaApplet has its transition methods called by the browser when the page containing the JApplet is loaded/unloaded.

  The transition methods in MultimediaApplication objects should be called at corresponding times.

- Resolution:
  
  Make JApplication a WindowListener on its main window.
The `constructMainWindow()` Method

```java
mainWindow.setDefaultCloseOperation(
        JFrame.DO_NOTHING_ON_CLOSE);
mainWindow.addWindowListener(this);
```
The `windowOpened()` Method

```java
public void windowOpened(WindowEvent event) {
    resize();
    start();
}
```
Alternative 2 - One Remaining Issue (cont.)

The `windowDeiconfied()` Method

```java
public void windowDeiconfied(WindowEvent event) {
    start();
}
```
The `windowIconified()` Method

```java
public void windowIconified(WindowEvent event) {
    stop();
}
```
The `windowClosing()` Method

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

```java
private void exit()
{
    int response;

    response = JOptionPane.showConfirmDialog(mainWindow,
                                           "Exit this application?",
                                           "Exit?",
                                           JOptionPane.YES_NO_OPTION);

    if (response == JOptionPane.YES_OPTION)
    {
        mainWindow.setVisible(false);
        stop();
        mainWindow.dispose();
    }
}
```
Alternative 2 - One Remaining Issue (cont.)

The `windowClosed()` Method

```java
public void windowClosed(WindowEvent event) {
    destroy();
    System.exit(0);
}
```
Final Design of the Unified System

- **MultimediaApplet**
  - `decorated : MultimediaApp`
  - `MultimediaApplet(decorated : MultimediaApp)`

- **MultimediaApplication**
  - `decorated : MultimediaApp`
  - `params : Properties`
  - `MultimediaApplication(args : String [], decorated : MultimediaApp)`

- **RootPaneContainer**
  - `getContentPane() : Container`

- **MultimediaRootPaneContainer**
  - `getParameter(name : String) : String`

- Delegates methods in MultimediaApp to decorated

- Delegates methods in RootPaneContainer to mainWindow

- WindowListener for mainWindow
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.
package io;

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

public class ResourceFinder
{
    private Class c;

    private ResourceFinder()
    {
        c = this.getClass();
    }

    private ResourceFinder(Object o)
    {
        // Get the Class for the Object that wants the resource
        c = o.getClass();
    }

    public static ResourceFinder createInstance()
    {
        return new ResourceFinder();
    }
}
Structure of the `ResourceFinder` (cont.)

```java
public static ResourceFinder createInstance(Object o) {
    return new ResourceFinder(o);
}
```
public InputStream findInputStream(String name) {
    InputStream is;
    is = c.getResourceAsStream(name);
    return is;
}
The `findInputStream()` Method

```java
public URL findURL(String name)
{
    URL url;
    url = c.getResource(name);
    return url;
}
```
Structure of the `StopWatchApp()`

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

public class StopWatchApp
    extends AbstractMultimediaApp
    implements ActionListener, MetronomeListener
{
    private boolean running;
    private JLabel label;
    private Metronome metronome;

    private static final String START = "Start";
    private static final String STOP = "Stop";
}
```
public void actionPerformed(ActionEvent event)
{
    String actionCommand;

    actionCommand = event.getActionCommand();
    if (actionCommand.equals(START))
    {
        label.setText("0");
        metronome.reset();
        metronome.start();
        running = true;
    }
    else if (actionCommand.equals(STOP))
    {
        metronome.stop();
        running = false;
    }
}

public void handleTick(int millis)
{
    label.setText(""+millis/1000);
}
public void init()
{
    JButton start, stop;
    JPanel contentPane;

    running = false;

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

    label = new JLabel("0");
    label.setBounds(250,100,100,100);
    contentPane.add(label);

    start = new JButton(START);
    start.setBounds(50,300,100,50);
    start.addActionListener(this);
    contentPane.add(start);

    stop = new JButton(STOP);
    stop.setBounds(450,300,100,50);
    stop.addActionListener(this);
    contentPane.add(stop);

    metronome = new Metronome(1000, true);
    metronome.addListener(this);
}
The \texttt{start()} and \texttt{stop()} Methods

```java
public void start()
{
    if (running) metronome.start();
}

public void stop()
{
    if (running) metronome.stop();
}
```
import app.*;

import javax.swing.*;

public class StopWatchMultimediaApplication extends MultimediaApplication {

    public static void main(String[] args) throws Exception {
        SwingUtilities.invokeLater(new StopWatchMultimediaApplication(args, 600, 400));
    }

    public StopWatchMultimediaApplication(String[] args, int width, int height) {
        super(args, new StopWatchApp(), width, height);
    }
}
import app.*;

public class StopWatchMultimediaApplet
    extends MultimediaApplet
{
    public StopWatchMultimediaApplet()
    {
        super(new StopWatchApp());
    }
}
StopWatch Demonstration

In examples/chapter:
StopWatchMultimedia.html

java -cp StopWatchMultimedia.jar StopWatchMultimediaApplication