

Appendix A: Distributing Java programs

Java is designed so that compiled byte code can run on a variety of platforms. This is, in fact, one of its most appealing features for it makes distributing Java programs relatively painless. There are two basic ways to do this. First, programs may be constructed as Java “applets,” programs that may be distributed by an http server and run by the client from within, say, a web browser. However, the types of operations such a program can perform are, without some further work, somewhat limited. For instance, applets are typically not allowed to print or perform basic file operations such as saving and loading. This is typically not an issue when creating mathematical illustrations of the kind we’ve seen in these notes.

Alternately, one can distribute an application through Java’s Web Start technology. This is a fairly attractive option for the types of operations performed are not restricted and the user can receive updates to the code easily.

In this appendix, we will illustrate how to implement both of these options.

1: Applets

To write an applet, one extends the class `javax.swing.JApplet` and embeds it in a web page through the use of an HTML tag. This is a little different from writing an application since it not clear, at first glance, where the entry point is into the code; that is, there is no method `public static void main(String[] args)`. Instead, the `JApplet` is being run by another program, usually, a web browser.

In general, a `JApplet` is not explicitly instantiated and a constructor for `JApplet` is not written; instead, `JApplet` has a method `public void init()` that is run by the browser at the time the `JApplet` is created. If we are converting an application into a `JApplet`, the usual theme is to move the code from the `main` method of the application into the `JApplet`’s `init` method.

Also, `JApplet` is a subclass of `Panel`. The web browser implicitly creates a `Frame` that contains `JApplet`. In general, it is not wise to override the `paint` method of the `JApplet`. Instead, we use it as a container into which we place components with customized `paint` methods just as we place such components in a `JFrame` when writing an application.

Here is an example illustrating how to convert the application `CubicGraph` from Chapter 3 into a `JApplet`. You may wish to review that example quickly.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
public class CubicGraphApplet extends JApplet {
    public void init() {
        CubicPanel panel = new CubicPanel();
        getContentPane().add(panel, BorderLayout.CENTER);
    }
class CubicPanel extends JPanel {
    public void paintComponent(Graphics gfx) {
        super.paintComponent(gfx);
        setBackground(Color.white);
        Graphics2D g = (Graphics2D) gfx;
        g.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                           RenderingHints.VALUE_ANTIALIAS_ON);
        AffineTransform transform = new AffineTransform();
        transform.translate(150, 150);
        transform.scale(1, -1);
        transform.scale(75, 50);
        g.setPaint(Color.lightGray);
        GeneralPath path = new GeneralPath();
        for (int i = -2; i <= 2; i++) {
            path.moveTo(i, -3);
            path.lineTo(i, 3);
        }
        for (int i = -3; i <= 3; i++) {
            path.moveTo(-2, i);
            path.lineTo(2, i);
        }
        g.draw(transform.createTransformedShape(path));
        int steps = 100;
        float dx = 4.0f/steps;
        g.setPaint(Color.black);
        path = new GeneralPath();
        path.moveTo(-2, valueAt(-2));
        for (float x = -2+dx; x <= 2; x += dx)
            path.lineTo(x, valueAt(x));
        g.draw(transform.createTransformedShape(path));
    }
    public float valueAt(float x) {
        return x*x*x-x;
    }
}
}
```

Notice that all we have done in the `JApplet`'s `init` method is instantiate a `JPanel` and add it to the `JApplet`. The rest of the code stays the same.

To run this program, we need to construct an HTML page to be viewed with a web browser. It can be as simple as a file containing only the following HTML tag:

```
<applet code=CubicGraphApplet width = 300 height = 300
codebase=.>
</applet>
```

This file should be contained in the same directory as the `CubicGraphApplet.class` file.

If you view this web page in your browser, the applet will appear in the upper left corner of the web page. Java's Software Development Kit contains a program called `appletviewer` that is useful for testing purposes. The command `appletviewer file.html` displays all the applets referenced by tags inside `file.html`.

In the past, distributing Java programs as applets has been complicated by the fact that every web browser contains its own, sometimes idiosyncratic, version of the Java virtual machine. Different versions would have different bugs and require different strategies to get the program to work correctly. In addition, the time it took for various virtual machines to support recent updates to the Java language was often considerable.

Sun has addressed these issues by creating the Java Plug-in, a browser plug-in available for most common web browsers, written and distributed by Sun along with the Software Development Kit. Requesting that viewers use the Java Plug-in gives greater confidence that programs will run in the expect way.

Sun also provides a tool `HtmlConverter` that converts the HTML tags in a page into tags that require the Java Plug-in. Just issue the command `HtmlConverter file.html`. When applied to the HTML file above, the result looks like this:

```
<!--"CONVERTED_APPLET"-->
<!-- HTML CONVERTER -->
<OBJECT
  classid = "clsid:CAFEEFAC-0014-0001-0002-ABCDEFEDCBA"
  codebase = "http://java.sun.com/products/plugin/autodl/jinstall-1_4_1_02-windows-
i586.cab#Version=1,4,1,20"
  WIDTH = 300 HEIGHT = 300 >
  <PARAM NAME = CODE VALUE = CubicGraph >
  <PARAM NAME = CODEBASE VALUE = . >
  <PARAM NAME = "type" VALUE = "application/x-java-applet;jpi-version=1.4.1_02">
  <PARAM NAME = "scriptable" VALUE = "false">
  <COMMENT>
    <EMBED
      type = "application/x-java-applet;jpi-version=1.4.1_02"
      CODE = CubicGraph
      JAVA_CODEBASE = .
      WIDTH = 300
      HEIGHT = 300
      scriptable = false
      pluginspage = "http://java.sun.com/products/plugin/index.html#download">
    </EMBED>
  </EMBED>
</COMMENT>
</OBJECT>
<!--
<APPLET CODE = CubicGraph JAVA_CODEBASE = . WIDTH = 300 HEIGHT = 300>
</APPLET>
-->
<!--"END_CONVERTED_APPLET"-->
```

A user who has not installed the Java Plug-in will be directed to the proper page from which it may be downloaded.

As a final example, here is how the LineMover application from Chapter 6 may be converted to an applet:

```
package appa;
import figure.*;
import java.awt.*;
import javax.swing.*;
public class LineMoverApplet extends JApplet implements Mover {
    GraphicalPoint p0, p1;
    GraphicalLine line;
    public void init() {
        FigurePanel panel = new FigurePanel(-3, -3, 3, 3);
        getContentPane().add(panel, BorderLayout.CENTER);
        p0 = new GraphicalPoint(0, 0);
        p0.setColor(Color.red);
        p0.setSize(3);
        p1 = new GraphicalPoint(1, 1);
        p1.setColor(Color.red);
        p1.setSize(3);
        line = new GraphicalLine(p0.x, p0.y, p1.x, p1.y);
        line.setColor(Color.blue);
        Grid grid = new Grid();
        grid.setColor(Color.lightGray);
        panel.add(grid);
        panel.add(new Axes());
        panel.add(line);
        panel.add(p0); panel.add(p1);
        panel.addMoveable(p0, this); panel.addMoveable(p1, this);
    }
    public void move(Moveable m, double x, double y) {
        ((GraphicalPoint) m).setPoint(x, y);
        line.setPoints(p0.x, p0.y, p1.x, p1.y);
    }
}
```

2: Using Java Web Start to distribute applications

Java Web Start provides a means to distribute Java applications and gives a practical alternative to applets. In this section, we'll see how to use it.

First, your web server needs to be reconfigured to recognize a new mime type. How to do this varies from server to server. Using the Apache server, it is as simple as adding the line

```
application/x-java-jnlp-file JNLP
```

to the file mime.types.

The application now needs to be bundled together into a jar file. This is a special type of file (its name stands for "Java archive") created by a Java program called jar. Its syntax is patterned on the

Unix command `tar`. For instance, to put all the `.class` files from the directory `mydir` into the jar file `myjar.jar`, issue the command

```
java cvf myjar.jar mydir/*.class
```

As part of Web Start's security provisions, your jar file needs to be digitally signed by you in the following way.

1. Create a public/private key pair using `keytool`:

```
keytool -genkey -alias david
```

After prompting you for a few pieces of information, this program will create a file called `.keystore` in your home directory containing the key pair referenced by the name `david`.

2. Create a self-signed certificate using `keytool`:

```
keytool -selfcert -alias david
```

3. Sign the `.jar` file using another Java program called `jarsigner`:

```
jarsigner myjar.jar david
```

The certificate you have created by these steps is not enough to prevent a rather dire warning when a user downloads your code using Web Start. This requires a trustworthy certificate issued by a certificate authority (VeriSign is one such company) and costing real money. For the purposes of these notes, however, the certificate created here will suffice.

The next step is to create a `.jnlp` file to be distributed over the internet. The following is an example of the file `sphere.jnlp` located at <http://localhost/david/sphere.jnlp>:

```
<?xml version="1.0" encoding="UTF-8" ?>
<jnlp
  spec="1.0+"
  codebase="http://localhost/david" href="sphere.jnlp">
  <information>
    <title>Spherical Tilings</title>
    <vendor>David Austin</vendor>
    <description>A tiling of the sphere shown dynamically</description>
    <homepage href="http://localhost/david/index.html"/>
    <offline-allowed/>
  </information>
  <security>
    <all-permissions/>
  </security>
  <resources>
    <j2se version="1.4+"/>
    <jar href="sphere.jar"/>
  </resources>
  <application-desc main-class="spheretilings.SpherePanel">
    <argument>illuminated</argument>
  </application-desc>
</jnlp>
```

To download the program using Java Web Start, the user needs to have Java Web Start installed locally. In fact, Web Start is included with the Java JRE. On Windows machines, installing the JRE (or SDK) also installs Web Start. Under Linux, a file named something like `javaws-1.2.0_02-linux-i586-i.zip`, residing in the `jre` directory, should be copied to a directory under the user's home directory and unzipped. Running the file `install.sh` installs Java Web Start.

When the user visits the page `http://localhost/david/sphere.jnlp`, the program is downloaded, stored on the user's machine and started. In addition, Web Start provides a user interface to manage applications downloaded in this way.

Java Web Start may be used to distribute applets as well. Just change `application-desc` to `applet-desc` and add the parameters you would have in an HTML file distributing the applet.