INTRODUCTION TO COMPUTER PROGRAMMING

Richard Pierse

Class 9: Writing *Java Applets*

Introduction

Applets are *Java* programs that execute within *HTML* pages. There are three stages to creating a working *Java Applet*. Firstly, the *applet* itself must be written in the *Java* language and compiled into a class. Secondly, a web page must be written in *HTML* that includes a link to the *Java applet* class, using one of the tags `<APPLET>` or `<OBJECT>`. Finally, both the *HTML* page and the *applet* need to be uploaded to a web server.

Writing a *Java Applet*

The structure of a *Java applet* differs in some respects from that of a stand-alone *Java* application. Firstly, the main class of a *Java applet* must be derived as an extension of the standard *Java* class `java.applet.Applet`. This class does not include a `main()` method as with stand-alone applications. Secondly, the method `init()` should always be included in the main class to initialise the *applet*. In a stand-alone application, initialisation is done in the class constructor. However, the class constructor of an *applet* may be called before the hosting program (web browser) is ready to initialise the *applet*, so the `init()` method is provided instead.

Other methods of the class `java.applet.Applet` that may be overridden by your subclass are:

```
    public void start() {}
```

which is called to tell the *applet* to start doing its task, and

```
    public void paint(Graphics g) {}
```

which is called to draw the *applet* on the screen. This latter method is passed a `Graphics` object, which can be used to display text or draw shapes in the *applet* window.

A Simple *Applet*

Here is the skeleton for a simple *Java applet*.

```
import java.applet.*;           // required for applet
import java.awt.*;              // required for graphics
public class MyApplet extends java.applet.Applet
{
    public void init() {}
    public void paint(Graphics g)
    {
        g.setColor(Color.red);
        g.drawString(”Welcome to Java!!”,50,60);
    }
}
```

This *applet* uses the `setcolor` and `drawString` methods of the `Graphics` class to set the pen colour to red and then write a line to the *applet* window. The first two `import` lines load the necessary libraries: `java.applet`, which is required for any *applet* and `java.awt`, which is required in order to use `Graphics` objects.

Creating *Java Applets* in *JCreator*

JCreator provides a user-friendly environment in which to build new *applets*. Select `New...` from the main menu and choose the *Basic Java applet* template. A new project is created having two files: a template for the *Java applet* itself and a simple *HTML* file to display the *applet*. These files should be edited to include content for your *applet*. Then go to the `Build` menu and *compile* the project to create the *Java* class and then *execute* the project to test the *applet*. Finally, the `.class` and `.html` files can be uploaded to a web server using *WS_FTP* to make them available on the *WWW*.

The *HTML* `<APPLET>` and `<OBJECT>` Tags

A *Java applet* can be included in an *HTML* page using either the `<APPLET>` or the `<OBJECT>` tag. Both tags are equivalent although the `<OBJECT>` tag is preferred and can be used to refer to compiled objects other than *Java* classes. The format is:

```
 <APPLET CODE=”my.class” HEIGHT=30 WIDTH=40>
</APPLET>
```

or

```
<OBJECT CODE=”my.class” HEIGHT=30 WIDTH=40>
</OBJECT>
```

The `CODE` parameter indicates the name of the *Java applet*. The parameters `HEIGHT` and `WIDTH` are *required* and specify the initial height and width of the object on the page. (This initial size can subsequently be changed by the *applet* itself).

Two other parameters that may be needed are `CODEBASE` and `ARCHIVE`. The browser expects to look for the *applet* code in the current directory. If the *applet* (and any files associated with it), are located elsewhere, then the location should be specified in the `CODEBASE` parameter. For example, the value

```
CODEBASE=”../myapplets”
```

causes the browser to look for the file `my.class` in the directory `myapplets` located one level higher up the tree than the current directory.

Applets may comprise more than a single class. In this case, all the classes comprising the *applet* can be compiled together into a *Java* archive. Archives are stored in files with the suffix `.JAR`. These files can be compressed so that they will load faster in the web browser. The location of the *Java* archive containing the *applet* is specified with the `ARCHIVE` parameter as in:

```
ARCHIVE=”myfiles.jar”
```

Passing Parameters to *Applets*

Many *applets* use parameters to control the behaviour and screen appearance of the *applet*. Parameters are passed to the *applet* from the browser using `<PARAM>` tags between the `<APPLET>` and `</APPLET>` tags. The format is:

```
    <PARAM NAME=”speed” VALUE=”20”>
```

This passes to the *applet* a parameter called `speed` which is set to the value `20`.

If *Java* has been disabled in the web browser, then the `<PARAM>` tags are simply ignored, but any text included between the `<APPLET>` and `</APPLET>` will be displayed. (This text is ignored if *Java* is enabled). This provides a mechanism for notifying users with disabled browsers that they are missing out on an *applet*.

Reading Parameters in *Applets*

Within the applet itself, parameters can be read using the `getParameter` method. In the above example, the value of the parameter `speed` could be read with the following code:

```
    String temp; int speed;
    temp = getParameter(”speed”);
    if(temp == null) speed = 1;
    else speed = Integer.parseInt(temp);
```

The parameter value is first read into the `String` variable `temp`. If the speed parameter had not been defined by a `<PARAM>` tag in the *HTML* code calling the *applet*, then this string would contain a `null` value. In this case, the integer variable `speed` is set to the default value of `1`. Otherwise, the string variable `temp` is converted to an integer using the `Integer.parseInt` method and the result stored in `speed`.

Modifying Our Simple *Applet*

To modify our simple *applet* to read a parameter `”message”` to set the message to display and a parameter `”colour”` to select the text colour of the message, we could use the following code:

```
int colour; String message;
public void init()
{
    String temp;
    temp = getParameter(”message”);
    if(temp == null)
        message = ”Welcome to Java!!”;
    else message = temp;
    temp = getParameter(”colour”);
    if(temp == null) colour = 0;
    else colour = Integer.parseInt(temp);
}
public void paint(Graphics g)
{
    if(colour == 1) g.setColor(Color.blue);
    else if(colour == 2)
        g.setColor(Color.green);
    else g.setColor(Color.red);
    g.drawString(message,50,100);
}
```

The variables `colour` and `message` are defined as data members of the *applet* class. This makes them "global" so that they are available to all methods in the class. They are defined in the `init()` method called when the *applet* is initialised and then used in the `paint()` method when the *applet* is displayed.

Note that only three values are recognised for the parameter `”colour”`: `1` = blue, `2` = green, any other value = red.

To call this *applet* from a web page, we could use the *HTML* code:

```
<APPLET CODE=”MyApplet.class”
    HEIGHT=500 WIDTH=300>
    <PARAM NAME=”message” VALUE=”Hi there!”>
    <PARAM NAME=”colour” VALUE=”1”>
</APPLET>
```

A Note on *JavaScript*

One feature you will certainly come across if you take a look at many actual *HTML* pages is the use of *JavaScript*. *JavaScript* is a language that looks rather similar to *Java* (both being based on *C* and *C*++) but is much less powerful. Invented by *Netscape* for their browser, it is mainly used for achieving small dynamic effects such as making hyperlinks change appearance when a mouse hovers over them, or pre-checking *HTML* forms before they are submitted.

One fundamental difference between *Java* and *JavaScript* is that *JavaScript* source code appears directly in *HTML* documents within `<SCRIPT>` and `</SCRIPT>` tags. This code is interpreted at run time by the web browser. In contrast, *Java applets* are compiled before being uploaded to the web and the browser merely executes the compiled *bytecode*. This means that *Java applets* will generally run much faster than their equivalent *JavaScript* counterparts. Also, the source code for a *Java applet* is hidden from the end-user in a way not possible in *JavaScript*.

Some Example *Applets*

I have created a web page which illustrates the use of our simple *applet* and also includes some other (more interesting) *applets* taken from the web site:

www.javaboutique.internet.com

Some of these *applets* contain *Java* source code, which you can study. They illustrate the wide range of uses of the *Java* applet.

Look at the source code of the *HTML* file `index.html` in the ***java*** directory. This will show you the parameters that are needed to run the various applets. You can try to incorporate these (or other *applets* from the *Java* boutique site) into your own HTML pages.