

## MODULE 8p - Applets - Trials B

The applet to be developed in this series of trials is called AppletB. The first version is a simple modification of AppletA but should be saved in AppletB.java

```
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Button;                                // new statement

public class AppletB extends Applet                    // new class name
{ private String s = "Greetings";

    private Button jack = new Button("Jack");          // new statement
    private Button jill = new Button("Jill");           // new statement

    public void init()
    { this.add(this.jack);                              // new statement
      this.add(this.jill);                              // new statement
      System.out.printf("Done init\n");
    }

    public void start()
    { System.out.printf("Done start\n");
    }

    public void paint(Graphics g)
    { g.drawRect(15, 15, 270, 70);
      g.drawString(this.s, 100, 60);
      System.out.printf("Done paint\n");
    }

    public void stop()
    { System.out.printf("Done stop\n");
    }

    public void destroy()
    { System.out.printf("Done destroy\n");
    }
}
```

Compile this using the javac command:

```
$ javac AppletB.java
```

Two new data fields augment String s and the inherited data fields. The new data fields, jack and jill, are both of type Button and these buttons are going to appear in the central region. When declaring a Button object is it usual to provide a label to go on the button and the Strings "Jack" and "Jill" are used here. Button is not in the main Java package and has to be imported from java.awt

To add a button to an applet, use the add() method which the applet automatically inherits from its ancestors. The add() method adds an

item to the component array referred to by the component data field. In simple cases it is customary to add buttons (and other items) in the `init()` method as shown.

The associated HTML needs to refer to `AppletB.class` so modify the old `AppletA.html` to `AppletB.html` as in:

```
<HTML>
  <BODY>
    <APPLET code="AppletB.class" width=300 height=100>
      Java is not available.
    </APPLET>
  </BODY>
</HTML>
```

Give the following appletviewer command:

```
$ appletviewer AppletB.html &
```

The applet window is now adorned by two obvious buttons.

Notice that although the buttons were added in `init()` BEFORE the rectangle was drawn in `paint()` they obscure the top edge of the rectangle. It is as though buttons are added to a transparent sheet which covers the central region.

Try pressing the buttons. They change their appearance when clicked but nothing else happens. More facilities are required.

Repeat the previous experiments with Stop, Start, etc. concluding with Quit.

#### MORE ABOUT CLASS Button

Compared with class `Applet`, the line of succession from the root class `Object` to class `Button` is very short:

Object - Component - Button

`Button` inherits from class `Component` which is also an ancestor of class `Applet`. Accordingly, class `Button` and class `Applet` (and any user class which extends `Applet`) will have some data fields and methods in common.

As noted in the context of class `Applet`, most of the data fields are not published but one can infer their presence and suggest names for some of them. Here are four such hypothesised identifiers:

1. `label` - is a `String`, the label on the button.
2. `size` - gives the size of the button (both width and height in pixels).
3. `background` - is the background colour ('color' in American).

4. listener - is guessed to be an array of special objects set up to spring into action when, for example, the button is pressed.

Of these, size, background and listeners are inherited from class Component. All have been seen before in the context of class Applet.

As well as data fields, class Button incorporates a number of methods (both inherited and non-inherited). A particularly important method in class Button is:

`addActionListener()` - used for adding an object of type `ActionListener` to the putative listener array.

`ActionListener` is an interface (not a class) which specifies that any class which implements the interface must incorporate a method `actionPerformed()`.

By supplying a suitable object of type `ActionListener` and adding it to a button by `addActionListener()` one can arrange for something to happen when the button is pressed.

In some sense the `ActionListener` listens out for the action of the button being pressed and, on detecting such an action, it invokes the `actionPerformed()` method.

#### A FIRST VARIATION - INTRODUCTION

The first variation on `AppletB` puts the above ideas into practice.

In this version of `AppletB`, class `JackL` is set up as the `ActionListener` class for Button `jack` and an instantiation of this class is added to the `jack`'s putative listener array by:

```
this.jack.addActionListener(new JackL());
```

It is important NOT to omit the `jack` by writing:

```
this.addActionListener(new JackL());
```

This is a perfectly legitimate statement but it adds the `ActionListener` to the applet's listener array and not to the Button's listener array. The `ActionListener` would then be listening out for some action on the applet and it is not immediately clear what such an action might be.

Using proper terminology, acts such as pressing a button or clicking a mouse provoke an 'event' and methods in listener objects 'handle events'.

The event provoked by pressing a button is an `ActionEvent` (a Java class) and an instance of this class is handed to the `actionPerformed()` method which therefore requires a formal parameter of matching type. In the version of `AppletB` below, the `actionPerformed()` method simply writes `Jack pressed` in the log and makes no use of the `ActionEvent` argument.

Note that both `ActionListener` and `ActionEvent` must be imported. They are imported from the `java.awt.event` package.

In an analogous way, class JillL is set up as the ActionListener class for Button jill and an instantiation of this class is added to jill's putative listener array by:

```
        this.jill.addActionListener(new JillL());
```

#### A FIRST VARIATION - PROGRAM

Modify the source code in AppletB.java so that it appears thus:

```
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Button;
import java.awt.event.ActionListener;           // new statement
import java.awt.event.ActionEvent;             // new statement

public class AppletB extends Applet
{ private String s = "Greetings";

    private Button jack = new Button("Jack");
    private Button jill = new Button("Jill");

    public void init()
    { this.add(this.jack);
      this.jack.addActionListener(new JackL()); // new statement
      this.add(this.jill);
      this.jill.addActionListener(new JillL()); // new statement
      System.out.printf("Done init\n");
    }

    public void start()
    { System.out.printf("Done start\n");
    }

    public void paint(Graphics g)
    { g.drawRect(15, 15, 270, 70);
      g.drawString(this.s, 100, 60);
      System.out.printf("Done paint\n");
    }

    public void stop()
    { System.out.printf("Done stop\n");
    }

    public void destroy()
    { System.out.printf("Done destroy\n");
    }
}

class JackL implements ActionListener           // new class
{ public void actionPerformed(ActionEvent e)
  { System.out.printf("Jack pressed\n");
  }
}
```

```

class JillL implements ActionListener           // new class
{ public void actionPerformed(ActionEvent e)
  { System.out.printf("Jill pressed%n");
  }
}

```

Compile this using the javac command:

```
$ javac AppletB.java
```

Run the appletviewer program with this new applet:

```
$ appletviewer AppletB.html &
```

The appearance of the applet window is exactly as before and the messages Done init Done start and Done paint are all written in the log. The difference now is what happens when the buttons are pressed...

Press Jack and notice that Jack pressed appears in the log.

Press Jill and notice that Jill pressed appears in the log.

As a general rule, users of applets are not much interested in any kind of log and would prefer messages to be written to the applet. Arranging for this to happen when the buttons are pressed is the goal of the next version of AppletB...

#### A SECOND VARIATION - INTRODUCTION

One possibility might be to modify the bodies of the actionPerformed() methods so that they assigned to String s but this isn't entirely straightforward. These methods are not in the same class as s so s is out of scope and one cannot have assignments like s = "Jack pressed" or even this.s = "Jack pressed" and, moreover, s cannot be private.

One might change s from being a private instance variable to being a public class variable so the following assignment would work:

```
AppletB.s = "Jack pressed";
```

This could be achieved by changing the declaration of s from:

```
private String s = "Greetings"
```

to:

```
public static String s = "Greetings"
```

This is not good news. There is a better way...

#### A SECOND VARIATION - PROGRAM

Modify the source code in AppletB.java so that it appears thus:

```

import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Button;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class AppletB extends Applet
{ private String s = "Greetings";

    private Button jack = new Button("Jack");
    private Button jill = new Button("Jill");

    public void init()
    { this.add(this.jack);
      this.jack.addActionListener(new JackL());
      this.add(this.jill);
      this.jill.addActionListener(new JillL());
      System.out.printf("Done init\n");
    }

    public void start()
    { System.out.printf("Done start\n");
    }

    public void paint(Graphics g)
    { g.drawRect(15, 15, 270, 70);
      g.drawString(this.s, 100, 60);
      System.out.printf("Done paint\n");
    }

    public void stop()
    { System.out.printf("Done stop\n");
    }

    public void destroy()
    { System.out.printf("Done destroy\n");
    }

    class JackL implements ActionListener           // JackL WITHIN AppletB
    { public void actionPerformed(ActionEvent e)
      { AppletB.this.s = "Jack pressed";           // new statement
        System.out.printf("%s\n", AppletB.this.s); // modified statement
      }
    }

    class JillL implements ActionListener           // JillL WITHIN AppletB
    { public void actionPerformed(ActionEvent e)
      { AppletB.this.s = "Jill pressed";           // new statement
        System.out.printf("%s\n", AppletB.this.s); // modified statement
      }
    }
} // closing } of AppletB

```

The principal modification is that the two ActionListener classes, JackL and JillL have been incorporated INSIDE class AppletB; the

final closing bracket of AppletB encloses these two classes which have been INDENTED to reflect their status as 'member classes'

A feature of Java is that the members defined in a class definition can include not only data fields and methods but also classes (as classes within a class).

#### DETAILS OF THE MEMBER CLASSES

By making JackL and JillL member classes within AppletB, the identifier `s` is no longer out of scope in the `actionPerformed()` methods and `s` can continue to be declared NON-static.

Within the `actionPerformed()` methods one cannot write `this.s` (as in the `paint()` method) because 'this' would refer to a JackL or JillL object which is NOT where `s` is declared.

One solution is to use the syntax `AppletB.this.s` (as shown) where 'this' is prefixed by AppletB to indicate which 'this' is meant!

Compile the program using the `javac` command:

```
$ javac AppletB.java
```

Run the `appletviewer` program with this new applet:

```
$ appletviewer AppletB.html &
```

#### EXPERIMENTS WITH THE NEW APPLLET

Try pressing the Jack button.

The result is something of a disappointment. Jack pressed appears in the log but not on the applet which continues to show Greetings. The reason is that the `paint()` method hasn't been invoked.

Previous experience has demonstrated several ways of stirring the `paint()` method into life:

1. Cover and uncover the applet with another window.
2. Change the size of the applet window.
3. Iconify and deiconify the window.
4. Select Stop from the menu and then Start.

Try any of these and Jack pressed will appear. Clearly it would be nice to make the message appear without having to take special steps and the next variant of the program shows how to achieve this goal.

There is one more experiment which gives insight into what the `paint()`

method does. Carry out the following steps carefully:

5. Ensure that Jack pressed is in the central region.
6. Press the Jill button.
7. Cover up the right-hand side of the applet with another window in such a way that Jack is visible but pressed is covered.
8. Now move the covering window out of the way so as to reveal the whole applet again.

You should notice that the `paint()` method appears to do just half a job! It repairs the central region only where it was covered up. The result might look something like Jackressed

Using proportionally-spaced lettering, Jill is somewhat shorter than Jack so the 'pressed' of Jill pressed is somewhat to the left of the 'pressed' of Jack pressed and therefore appears in the new position. Jack wasn't covered up and so isn't replaced by Jill.

#### INVOKING THE `paint()` METHOD

Up to now, the `paint()` method has been invoked by the appletviewer first at start-up [just after `init()` and `start()`] and subsequently when there is a need to repair or restore the applet.

It has been suggested in an earlier footnote that the way the appletviewer invokes the `paint()` method is by a statement like:

```
handle.paint(handle.getGraphics());
```

In this, the identifier `handle` is hypothesised as the appletviewer's reference to the instantiation of the applet (`AppletB` now).

There is no reason why an analogous statement cannot be included in the `actionPerformed()` methods to invoke the `paint()` method directly rather than waiting for the appletviewer to decide that some repair work is necessary.

#### A THIRD VARIATION

Add an extra statement to the `actionPerformed()` method in `JackL` so that it appears thus:

```
class JackL implements ActionListener
{ public void actionPerformed(ActionEvent e)
  { AppletB.this.s = "Jack pressed";
    AppletB.this.paint(AppletB.this.getGraphics()); // new statement
    System.out.printf("%s\n", AppletB.this.s);
  }
}
```

Don't bother to change `JillL`. Note that, instead of `handle`, the reference



to AppletB from within class JackL is AppletB.this

Compile using javac and run the appletviewer program again.

Press the Jack button and note the result. There is now an immediate change to the central region where Jack pressed does indeed appear but not quite as we might have wished. Unfortunately Jack pressed appears superimposed on Greetings and there is a nasty mess. A final variation will cure this...

#### A FOURTH VARIATION

The proper way to cause Jack pressed to be written on the central region is to invoke the repaint() method rather than the paint() method.

The repaint() method does not require any arguments so there is no need to employ getGraphics()

In short, the actionPerformed() method in JackL should be changed to:

```
class JackL implements ActionListener
{ public void actionPerformed(ActionEvent e)
  { AppletB.this.s = "Jack pressed";
    AppletB.this.repaint();                // modified statement
    System.out.printf("%s\n", AppletB.this.s);
  }
}
```

The same call of repaint() can be added to JillL.

Compile using javac and run the appletviewer program again.

Press the Jack button and, at last, Jack pressed appears in the central region nice and tidily.

#### FOOTNOTE ABOUT repaint()

Like the paint() method, repaint() is inherited by any class that extends class Applet. Unlike the paint() method, the default inherited version of repaint() actually does something. Amongst other things, it calls another inherited method update() which first clears the central region and then calls the paint() method.

It has been noted in previous experiments that when, for example, the applet is iconified and deiconified the central region reappears with the correct message unsullied by earlier messages. This suggests that rather more than a simple call of paint() occurs and, more likely, repaint() [or perhaps update() directly] is called to ensure that the central region is cleared prior to a fresh display.