MODULE 6p - Applets - Trials A

Here is the source of the simplest possible applet. Key this into the file AppletA.java

```
import java.applet.Applet;

public class AppletA extends Applet
 { // No data fields
   // No methods
 }
```

Now compile it using the javac command:

```
$ javac AppletA.java
```

As a class with no data fields and no methods, AppletA cannot be expected to do very much, but notice that it extends class Applet. This latter class is not in the main Java package (java.lang) and an import statement is required to import the class from the java.applet package as shown.

By extending class Applet, class AppletA inherits the data fields and methods not only of Applet but of all the ancestors of Applet right back to the root class Object. By way of interest the line of succession is:

Object - Component - Container - Panel - Applet - AppletA

Applets don't incorporate a method main() and are not run by using the java command. Applets are intended to be run from Web browsers and it is necessary to write some HTML. Key this source into the file AppletA.html

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

Web browsers are still rather sluggish when processing applets and many cannot cope with applets at all. The APPLET tag in the HTML specifies (after code=) the name of the class file in quotes and specifies the width and height in pixels. If the browser cannot process the applet the message Java not available will be displayed.

To test an applet it is best to use the appletviewer command:

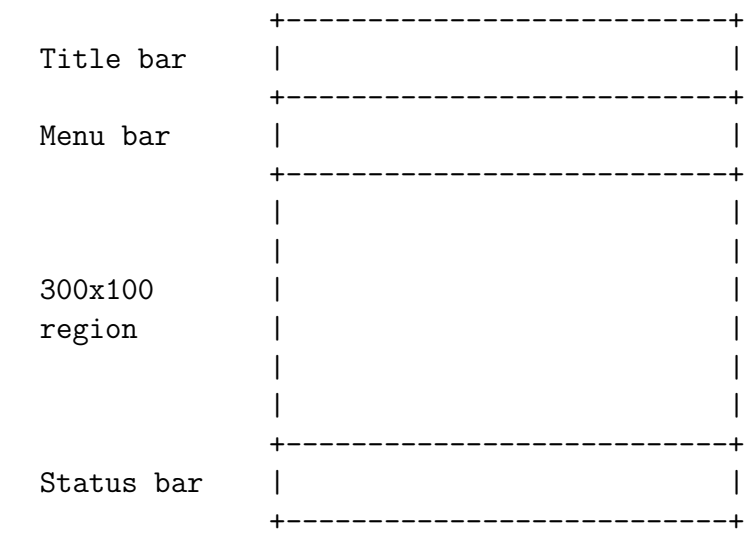
```
$ appletviewer AppletA.html &
```

In outline, the sequence of events initiated by this command are:

1. The appletviewer program is loaded and starts running.

2. The program notes AppletA.html as the command line argument and looks at the contents of this file. It sees the APPLET tag.

3. Given the information in the APPLET tag the appletviewer sets up an applet window. This is divided horizontally into four regions:



   The title bar announces: Applet Viewer: AppletA.class
   The menu bar has a single menu: Applet
   The central region is 300 pixels wide and 100 pixels high
   The status line at the bottom says: Applet started.

NB: the window is 300 pixels wide but more than 100 pixels high.

4. Next, the appletviewer program instantiates an AppletA object.

5. The appletviewer program then sees that no methods have been specified in class AppletA and hangs up.

DATA FIELDS

By the principle of data hiding or encapsulation, the identifiers of most of the data fields inherited by AppletA are not published but one can guess at their presence and suggest names for some of them. Here are a few:

1. size - refers to an object which gives the size of the applet (both width and height in pixels).

2. background - refers to an object which describes the background colour ('color' in American).

3. graphics - refers to an object which incorporates details of the central region of the applet and enables lines and shapes to be drawn on it without running off the edge.

4. layout - refers to an object which specifies how items such as buttons, checkboxes and labels are to be laid out on the applet. There are no such items in this applet.

5. component - refers to an array (initially of 4 elements) in which each element refers to a Component. A component is an item such as a button, checkbox or label on the applet. Each component is an object in its own right. There are no such items in this applet.

6. listener - is also guessed to refer to an array and here each element refers to a special object called a Listener. A listener springs into action when, for example, the mouse is clicked or the applet loses focus. There are no listeners associated with this applet.

Even this frivolous first applet has a size and a background colour but, since no items have been laid out and no listeners have been set up, the layout is irrelevant and component and listener have nothing to refer to.

METHODS

Of all the methods that are inherited by ANY class that extends Applet, the five most important are:

init() start() paint() stop() destroy()

Any or all of these methods can be overridden to do useful things but the inherited default versions do nothing.

Other important inherited methods are:

add() - used for adding an item to the applet, in practice by adding an appropriate object to the component array.

add???Listener() - used for adding a listener, possibly by adding an appropriate object to the putative array of listeners. [Various possibilities can stand in the place of ???, and addMouseListener() provides an example.]

EXPERIMENTS WITH THE FRIVOLOUS APPLET

1. Open the Applet menu. Select the Stop entry.

   The status bar announces Applet stopped.

2. Select the Start entry on the menu.

   The status bar announces Applet started.

3. Select the Restart entry on the menu.

   This is supposed to be equivalent to (1) followed by (2) and the status bar again concludes with Applet started.

4. Select the Quit entry on the menu.

   This is how to finish with an applet.

A FIRST VARIATION

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

```
import java.applet.Applet;
import java.awt.Graphics;

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

   public void paint(Graphics g)
    { g.drawString(this.s, 100, 60);
    }
 }
```

Compile it using the javac command:

```
$ javac AppletA.java
```

A second import statement is required for the class Graphics which is in the java.awt package. This class is required since it is the type of the argument of the paint() method. Note that awt stands for abstract windowing toolkit.

This version of AppletA includes a data field and a method. The data field s augments the inherited data fields and the method paint() overrides the inherited paint() method.

The data field is of type String and is initialised to "Greetings".

The method paint() causes "Greetings" to be drawn on the central region. The explanation can wait a moment. First, using the same HTML as before,

run the appletviewer program on this new applet:

```
$ appletviewer AppletA.html &
```

The appearance of the applet window is just as before except that the String "Greetings" has been drawn roughly in the middle.

This time, after instantiating an AppletA object, the appletviewer program invokes the paint() method which requires an argument. The actual argument handed over by the appletviewer program is the object referred to by the (inherited) data field graphics which provides a reference to the central region of the applet window.

The formal argument g then has access to all the methods of this object and these include drawString() which draws a string in the central region. This method is invoked by the statement:

```
g.drawString(this.s, 100, 60);
```

The first argument of drawString is the String to be drawn (here this.s refers to "Greetings"). The second two arguments show the position of the start of the string relative to the top left-hand corner of the central region. Thus the string begins 100 pixels to the right of and 60 pixels below the top left-hand corner.

After invoking the paint method, the running program hangs up.

EXPERIMENTS WITH THE NEW APPLET

Repeat the previous experiments. Little has changed except that when the Stop entry on the menu is selected not only does the status bar reflect Applet stopped but the drawn String disappears too. On selecting Start it reappears.

Select the Quit entry in the menu to finish with the applet.

A SECOND VARIATION

It has been noted that five important methods in ANY applet are:

init() start() paint() stop() destroy()

Each of these methods is invoked from the appletviewer and, for each, there are well-defined moments for being invoked.

For example, the stop() method is invoked when the Stop entry in the menu is selected. Obviously the appletviewer cannot invoke a method which doesn't exist so stop() (and the others) are mandatory in any applet. If the user doesn't supply overriding versions then the inherited versions will be invoked. These are guaranteed to do nothing.

In the final version of AppletA, overriding versions of all five methods are supplied. Each contains a println() statement so that a log will be built up showing just when each method is invoked.

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

```
import java.applet.Applet;
import java.awt.Graphics;

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

   public void init()
    { System.out.println("Done init");
```

```
    }

   public void start()
    { System.out.println("Done start");
    }

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

   public void stop()
    { System.out.println("Done stop");
    }

   public void destroy()
    { System.out.println("Done destroy");
    }
 }
```

One other augmentation is the extra statement in the paint() method:

```
g.drawRect(15, 15, 270, 70);
```

This draws a rectangle whose top left-hand corner is 15 pixels to the right of and 15 pixels below the top left-hand corner of the central region and, further, the rectangle is 270 pixels wide and 70 high.

Recall that the central region is 300 pixels wide and 100 pixels high and note that 15 + 270 + 15 = 300 and 15 + 70 + 15 = 100 so the rectangle is roughly centred in the available space.

Again use javac to compile the applet:

```
$ javac AppletA.java
```

Now give the command:

```
$ appletviewer AppletA.html &
```

The only difference in the appearance of the applet window now is the enveloping rectangle but, separately, the first three lines of the log appear after the appletviewer command.

WHAT HAPPENS AND WHEN

After the applet window is set up, the following sequence of events occurs:

1. The appletviewer instantiates the AppletA object. In effect there is the statement:

   AppletA handle = new AppletA();

   The identifier handle enables the appletviewer to access the data fields and methods in the applet. For example handle.graphics (see footnote below) enables the appletviewer to use the reference to the central region and handle.init() enables the appletviewer to invoke the init() method.

2. Shortly after instantiating the AppletA object, the appletviewer invokes:

   handle.init();

   This results in Done init being printed in the log.

3. Next the appletviewer invokes:

   handle.start();

This results in Done start being printed in the log.

4. Next the appletviewer invokes:

```
handle.paint(handle.graphics);
```

   The actual argument handle.graphics is handed over to the formal argument g of the paint() method. The body of the paint() method draws the enveloping rectangle and then draws the String.

   The third statement in the body of paint() results in Done paint being printed in the log (possibly more than once as explained later).

5. The appletviewer then hangs up but...

Now select Reload in the applet menu...

6. This causes the appletviewer to close the applet and reload it. The closing process involves two stages, first:

```
handle.stop();
```

   This results in Done stop being printed in the log and the status bar also records Applet stopped. The second stage in the closing process is:

```
handle.destroy();
```

   This results in Done destroy being printed in the log and any resources used by the applet are then freed.

7. If Quit had been selected instead of Reload this would be the end of the story. With Reload there is a fresh beginning and the appletviewer invokes handle.init() handle.start() and then handle.paint(handle.graphics) and again hangs up.

   The messages Done init Done start Done paint are printed once again. As before Done paint may be printed more than once. An explanation is provided in the section following the footnote.

FOOTNOTE ABOUT handle.graphics

The data field graphics was introduced as a hypothesis; this is a suggested identifier for something that isn't published. Although its proper identifier is unknown its value can be obtained by using the method getGraphics() and this method IS published. Accordingly, the actual argument passed to paint() by the appletviewer is handle.getGraphics() rather than handle.graphics

For simplicity, handle.paint(handle.graphics) is used in this introduction to applets rather than handle.paint(handle.getGraphics()) which is more likely to be the truth.

THE FIVE PRINCIPAL METHODS

The five methods are invoked by the appletviewer program at well-defined moments. Roughly speaking:

init()	is invoked on initial entry (or after reloading) and is intended for once-and-for-all-setting-up statements.
start()	is invoked after init() (or after restarting) and is intended for statements which follow initialisation but come before paint() is invoked.
paint()	is invoked after init() and start() but also whenever any repair work is required to the central region. Most obviously it is invoked after part (or all) of the region has been obscured and revealed again or after the applet window has been resized. Accordingly, paint() is typically invoked more frequently than the other methods.
stop()	is invoked after selecting Stop in the Applet menu. It is also invoked when Quit is selected [in which case

destroy() follows immediately].

destroy() is invoked on selecting Quit.

Some experiments will illuminate the processes further.

EXPERIMENTS AND OBSERVATIONS

When conducting the following experiments keep an eye on the log check which method is being invoked and in which circumstances.

1. Cover up part of the applet window and then reveal it again.

   All the pixels are restored. Whenever such restoration is needed, handle.paint() is automatically invoked.

2. Change the size of the applet window.

   This too causes handle.paint() to be invoked, possibly twice.

3. Select the Stop entry.

   This invokes handle.stop() and the central region is cleared.

4. Select the Start entry on the menu.

   This invokes handle.start() and then handle.paint(handle.graphics) to restore the central region.

5. Iconify the applet window.

   Besides iconifying the window this is equivalent to selecting Stop and handle.stop() is invoked.

6. Deiconify the icon.

   Besides deiconifying the window this is equivalent to selecting Start and handle.start() and handle.paint(handle.graphics) are invoked in turn.

7. Select the Reload entry on the menu.

As explained before, the appletviewer invokes handle.stop() and handle.destroy() and then there is a fresh beginning. The methods handle.init() handle.start() and handle.paint(handle.graphics) are invoked in turn.

Accordingly, five messages are printed.

8. Select the Restart entry on the menu.

   This is supposed to be equivalent to Stop followed by Start but with some versions of appletviewer there is a bug and Restart is exactly the same as Reload. Check the messages carefully.

9. Select the Quit entry in the menu.

   The appletviewer invokes handle.stop() and handle.destroy() and removes the applet window altogether.