

GUI Applications Using AWT

Introduction

So far all our applications have been of a command prompt variety. There has been no sign of moveable windows with buttons, menus and edit boxes.

However, if we are to create a modern software package with a significant amount of user interaction then a **Graphical User Interface** (GUI) is almost a necessity.

Java allows us to achieve this by supplying a set of related classes for creating the basic components of such a system. Hence, we can use existing *Button*, *Label* or *TextField* classes to create the objects that populate our new software.

These GUI components need to be positioned within our application's window. With platform-specific software such components are usually placed at specific positions within the window and are of specified dimensions. However, when using Java, we have to keep in mind that our application may be run on many different platforms with varying screen resolutions, so absolute positioning and sizing of components can cause serious problems. Instead, Java employs a **layout manager** to decide on the position and size of the GUI components appearing in the application window.

Once positioned, components must be made to react to user input. For example, we will want something to happen when the user clicks on a button. A component is made to react to user input by adding a **listener** which states exactly which events a particular component should *listen* for. For each event we want a component to react to, we need to write a Java function. The function will be executed when the event occurs. For example, if an application contained a button and a label, we might make the button *listen* for being pressed and then write a routine, linked to the button pressed event, which changes the label's text to "*Button pressed*".

This chapter introduces the various features mentioned above; subsequent chapters then expand on the options available.

Creating an Application Window

The GUI components that we need are defined by Java in the **Abstract Windowing Toolkit** or **AWT** package. To use objects from the classes defined there we need to include the statement

```
import java.awt.*;
```

in our programs.

Any GUI application will appear within a window. Java's *Frame* class allows us to create such a basic window. One way to do this is to add a *Frame* object to *main()*. The logic required by the program is:

```
Create a Frame object
Set its size
Make the window (frame) visible
```

The code overleaf (LISTING-13.1) shows how this is done.

LISTING-13.1

Creating a Window
Using a *Frame* Object

```
import java.awt.*;

public class Window
{
    public static void main(String args[])
    {
        Frame fm = new Frame();
        fm.setSize(300,100);
        fm.show();
    }
}
```

Activity 13.1

Type in the above program, compile and run it.

You should be able to see the window. Try moving, resizing and closing the window.

Notice that the close options don't work. You'll have to click on the DOS window and press Ctrl-C to cancel the program.

An alternative approach is to make our own class a descendant of the *Frame* class. This, more usual approach, is demonstrated in LISTING-13.2 below.

LISTING-13.2

Creating a Window
Using a *Frame* Subclass.

```
import java.awt.*;

public class Window2 extends Frame
{
    public static void main(String args[])
    {
        Window2 fm = new Window2();
        fm.setSize(300,100);
        fm.show();
    }
}
```

Activity 13.2

Type in and test this second version of the program.

As you can see, there is very little difference in the two approaches when it comes to coding *main()*. This second version creates a *Window2* object rather than the *Frame* object of the earlier version. In both cases the screen should appear as shown in FIG-13.1.

FIG-13.1

A Basic Application
Window



In general, we'll stick to this second approach, but highlight the differences between the two styles on occasion.

Displaying the Window

The size, position and visibility of the application window can be set using the following methods:

void setSize(int w, int h) Sets the application window to *w* pixels wide by *h* pixels high.

void setBounds(int x, int y, int w, int h) Sets the application window to *w* pixels wide by *h* pixels high. The top left corner of the window is at screen coordinates (*x,y*).

void show() Causes the window to be made visible.

void hide() Causes the window to become invisible.

The window is initially hidden, so a call to *show()* is necessary to display it.

An alternative to *show()* and *hide()* is *setVisible()* which can perform both the same tasks.

void setVisible(boolean b) If *b* is *true*, the window is shown. If *b* is *false*, the window is hidden.

boolean isVisible() Returns *true* if the window is visible; otherwise *false* is returned.

Activity 13.4

Modify your *Window2* program to use the *setBounds()* and *setVisible()* operations. These should replace the calls to *setSize()* and *show()*.

Position the application window so that it occupies the centre of the screen.

Accessing the Window Title

The window title can be retrieved or changed later.

String getTitle() Returns the string displayed in the window's title.

void setTitle(String s) Sets the title to *s*.

Window Resizing

We can enable or disable resizing of the window by the user and also determine which of these two states the window is currently in.

void setResizable(boolean b) The window can be resized if *b* is *true*, otherwise the window cannot be resized by the user.

boolean isResizable() Returns *true* if the window can be resized; otherwise *false* is returned.

Activity 13.5

Add the line

```
fm.setResizable(false);
```

to your program.

Execute the program and try to resize the window by dragging the edges of the window.

Try to maximise and minimise the window.w

Changing the Background Colour

The background colour of your window can be changed using the *setBackground()* method:

void setBackground(Color c) Changes the applications background colour to *c*.

Color getBackground() Returns the current colour of the background.

Notice that the parameter to this method is an object from the *Color* class. This class is defined within Java

The Color Class

Constants

Color white
Color lightGray
Color gray
Color darkGray
Color black
Color red
Color pink
Color orange
Color yellow
Color green
Color magenta
Color cyan
Color blue

These named colours can be used when a method requires a *Color* object as an argument.

Constructors

Color(int red, int green, int blue)

Creates a *Color* object whose colour is constructed from the specified amount of red, green and blue. These values should lie in the range 0 to 255.

Other Methods

<i>int getRed()</i>	Returns the value of the colour's red component.
<i>int getGreen()</i>	Returns the value of the colour's green component.
<i>int getBlue()</i>	Returns the value of the colour's blue component.
<i>Color brighter()</i>	Returns a colour one shade lighter than the current colour.
<i>Color darker()</i>	Returns a colour one shade darker than the current colour.

In our previous example we could make the window's background colour red by adding the line

```
fm.setBackground(Color.red);
```

Alternatively, we could create our own colour using the *Color* class constructor to supply an argument to *setBackground()*:

```
fm.setBackground(new Color(214,20,76));
```