# JAVA NOTES

# GRAPHICAL USER INTERFACES

Terry Marris 14 June 2001

# 1 FRAMES

## 1.1 LEARNING OUTCOMES

By the end of this lesson the student should be able to

- create a closeable frame
- place a title on a frame
- size and position a frame

## 1.2 INTRODUCTION

We build on the topics introduced in Java Notes Part One - Fundamentals.

So far we have not been concerned with windows, menus and dialogues, the stuff of modern user interfaces. But we are now.

We see what a frame is. We create a frame that can close and terminate program execution. We see how to size and position a frame on the screen. We introduce events, listeners and adapters. We note that a frame may have regions or borders.

## 1.3 FRAMES

A Java frame is a top-level window that is not contained inside any other Java window. It contains graphical user interface (GUI) components such as labels, text boxes and buttons.

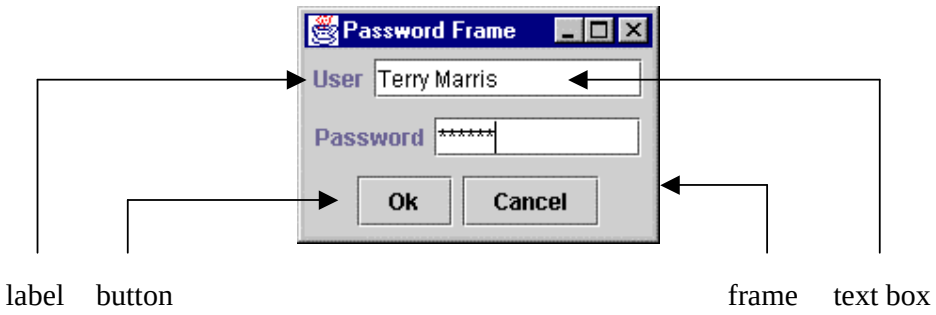



**Figure 1.1** *A Java Frame is a container*

By default is has five regions or borders: North, South, West, East and Center.

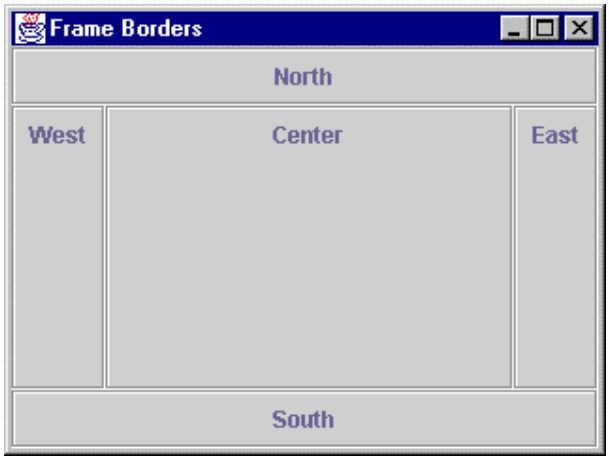



**Figure 1.2** *By default, a frame has five regions or borders*

## 1.4 FRAME SIZE AND POSITION

When creating a frame we need to specify its size and (optionally) its location on the screen. A frame of a given size might be too small for a large screen, but too large for a small screen. So we need to specify the size of the frame in relation to the size of the screen. We need to use toolkit methods.

A toolkit usually provides system dependent information, such as the size of the monitor screen.

The *Toolkit* class has a method named *getScreenSize()* that returns a *Dimension* object. A *Dimension* object stores a height and a width. We obtain a *Toolkit* instance and name it *tk*. We use the *tk* object's *getScreenSize()* method to initialise a *Dimension* object named *screen*. Then we use the frame's *setSize(int, int)* method to make the frame fill the entire screen.

```
import java.awt.*;

...

Toolkit tk = Toolkit.getDefaultToolkit();
Dimension screen = tk.getScreenSize();
setSize(screen.width, screen.height);
```

The co-ordinates of its top left-hand corner define the position of a frame. For a frame that fills the entire screen these co-ordinates are (0, 0). If you wanted to position a frame, width 300 pixels (a pixel is a dot that represents the smallest part of a picture that can be displayed on the screen) height 200 pixels, with its top left hand corner at (50, 100) you might code

```
int width = 300;
int height = 200;
int xCoord = 50;
int yCoord = 100;
setSize(width, height);
setLocation(xCoord, yCoord);
```

The *Toolkit* class is found in the *java.awt* package.

AWT stands for Abstract Windowing Toolkit.

## 1.5 IMPLEMENTING A WINDOW LISTENER

We need to be able to close a frame and terminate the application, that is, the program that is running. Typically, we close a frame and terminate the program by clicking the button marked × in the frame's top right hand corner - the frame-close icon. We need to detect when the user clicks on the frame-close icon and then take appropriate action.

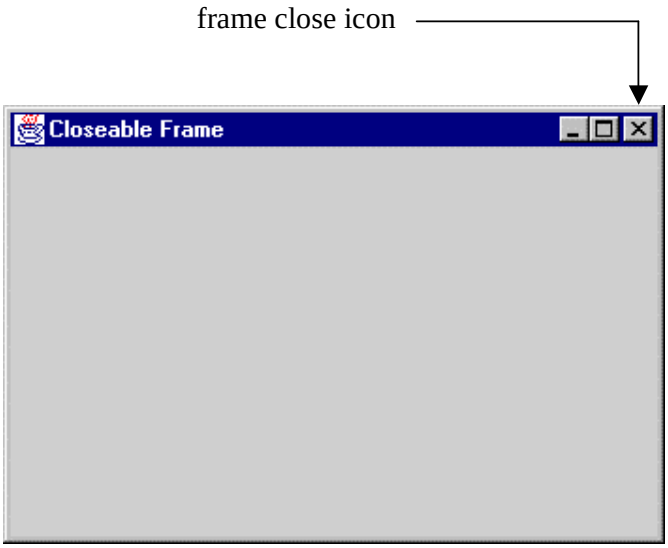



**Figure 1.3** *Closeable Frame*

Windows 95 constantly monitors the environment for events such as mouse clicks. Windows then reports these events to the programs that are running. Each program decides what, if anything, to do in response to these events.

Events are transmitted from event sources (e.g. a mouse being clicked on a window's close icon) to event listeners (e.g. a window listener).

The *WindowListener* deals with window events. It is an interface class - a class that lists, but does not implement, its methods.

The method we are particularly interested in is the *windowClosing(WindowEvent)* method.

```
public interface WindowListener {
     ...
public void windowClosing(WindowEvent e);
     ...
}
```

If we implement the *WindowListener* interface, we are obliged to provide an implementation for all its methods - even if we want to use just one of them.

The *WindowAdapter* class, provided for us in the *java.awt* package, implements all the *WindowListener* methods, but in a minimal way.

So we extend the *WindowAdapter* class and override the method that deals with receiving the window-closing event by providing the code to close the frame and stop the program from running.

Our *Terminator* class, shown below, does just this.

```
import java.awt.event.*

class Terminator extends WindowAdapter {
  public void windowClosing(WindowEvent e)
  {
    System.exit(0);    // terminates program execution
  }
}
```

Our *windowClosing()* method closes down a frame and terminates program execution.

## 1.6 ASSOCIATING A FRAME WITH A WINDOW LISTENER

We need to register our frame as a listener for window events. Then, when a window-closing event occurs, our frame can close itself down and terminate program execution. To do this we use our frame's *addWindowListener()* method.

```
addWindowListener(new Terminator());
```

But the usual idiom for closing a frame and terminating program execution is to embed the code for doing so as the argument to *addWindowListener()* like this:

```
addWindowListener(new WindowAdapter()
  {
    public void windowClosing(WindowEvent e)
    {
      System.exit(0);
    }
  });
```

## 1.7 THE CLOSEABLE FRAME

Here, shown below, is our first frame, named *CloseableFrame*. It displays the frame shown in Figure 1.3.

```
/* CloseableFrame.java
   Terry Marris  14 June 2001
*/


import java.awt.*;
import java.awt.event.*;
import javax.swing.*;


public class CloseableFrame extends JFrame {
  public CloseableFrame()
  {
    setTitle("Closeable Frame");

    Toolkit tk = Toolkit.getDefaultToolkit();
    Dimension screen = tk.getScreenSize();
    setSize(screen.width, screen.height);

    addWindowListener(new WindowAdapter()
      {
        public void windowClosing(WindowEvent e)
        {
          System.exit(0);
        }
      });
  }


  public static void main(String[] s)
  {
    JFrame aFrame = new CloseableFrame();
    aFrame.show();
  }
}
```

*JFrame* is defined in the *javax.swing* package. The *javax* package contains the graphical user interface (GUI) toolkit known as *Swing* (don't ask me why). Swing is an extension to the Abstract Windowing Toolkit (AWT), somewhat slower to execute (not a problem on modern machines) but richer in facilities.

Our *CloseableFrame* inherits its behaviour from *JFrame*. In addition it

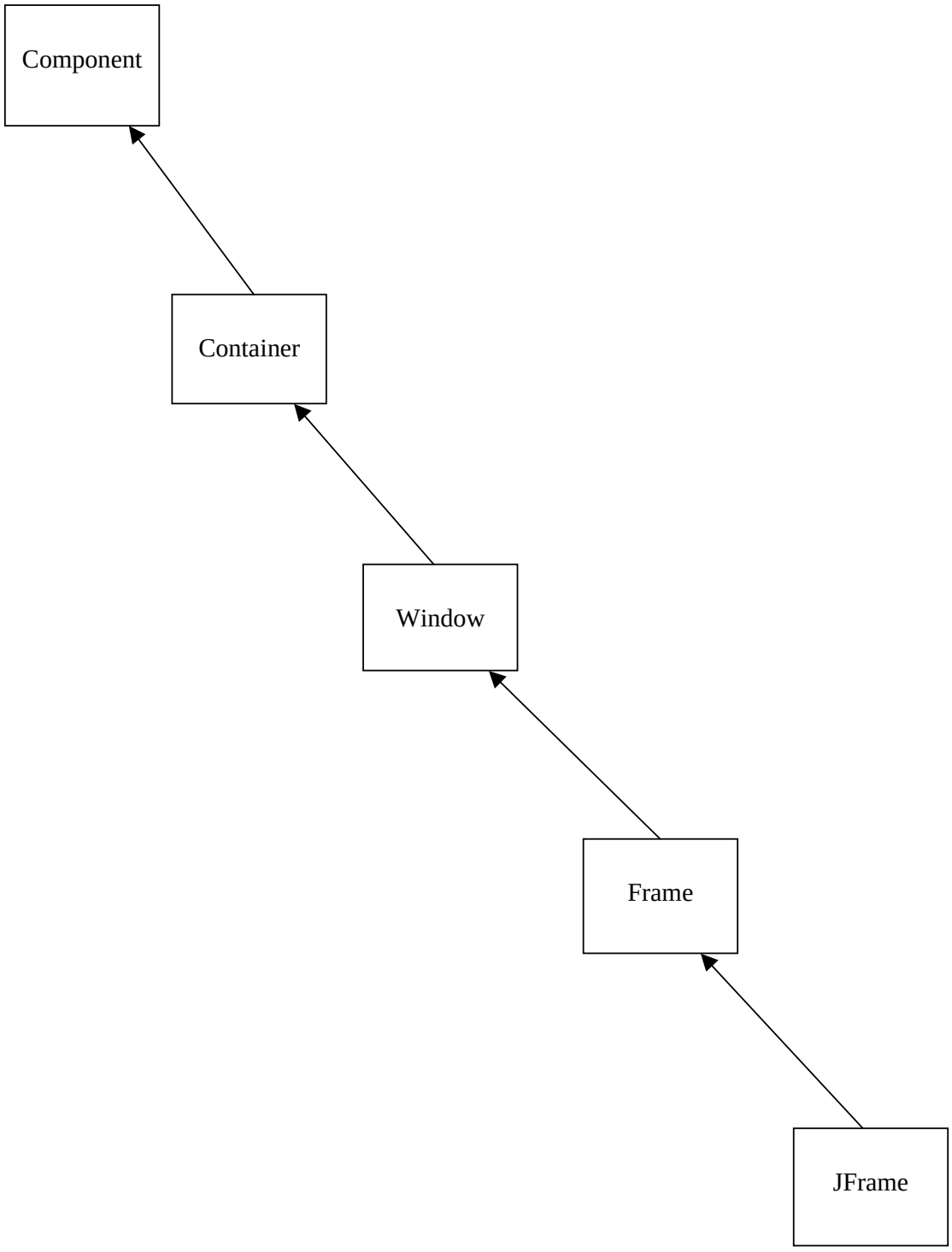
- sets its title to *Closeable Frame* in its title bar
- sets the size of the frame to fill the entire screen
- registers itself as a listener for window closing events.

To actually display a frame on the screen we

- create the frame object with a call to *new*
- call the frame's *show()* method

## 1.8 DOCUMENTATION

The inheritance hierarchy for the *JFrame* class is shown below.




This means that many of the *JFrame* methods may be found in the *Component* and *Window* classes.

**java.awt.Component**

| **Methods** | | |
|---|---|---|
| boolean | isVisible() | checks if this component is set to be visible. Components are initially visible with the exception of top level components such as JFrame. |
| void | setVisible(boolean b) | shows or hides the component depending on whether b is true or false. |
| boolean | isShowing() | checks if this component is showing on the screen. For this it must be visible and be inside a container that is showing. |
| boolean | isEnabled() | checks if this component is enabled. An enabled component can receive keyboard input. Components are initially enabled. |
| void | setEnabled(boolean b) | enables or disables a component. |
| Point | getLocation() | returns the location of the top left corner of this component, relative to the top left corner of the surrounding container. (A Point object p encapsulates an x and a y co-ordinate that are accessible by p.x and p.y). |
| Point | getLocationOnScreen() | returns the location of the top left corner of this component using the screen co-ordinates. |
| void | setBounds(int x, int y, int width, int height) | moves and resizes this component. The location of the top-left corner is given by x and y, and the new size is given by the width and height parameters. |
| void | setLocation(int x, int y)<br>setLocation(Point p) | moves the component to the new location. The x and y co-ordinates (or p.x and p.y) use the co-ordinates of the container if the component is not a top-level component, or the co-ordinates of the screen if the component is a top level component e.g. a JFrame. |
| Dimension | getSize() | gets the current size of this component. |
| void | setSize(int width, int height)<br>setSize(Dimension d) | resizes the component to the specified width and height. |

**java.awt.Window**

| **Methods** | | |
|---|---|---|
| void | addWindowListener( WindowListener l) | adds the given window listener to receive window events from this window. |
| void | pack() | causes this window to fit the preferred size of the its components. |
| void | show(0 | makes this window visible. |
| void | toFront() | shows this window on top of any other windows |
| void | toBack() | moves this window to the back of the stack of windows on the desktop and rearranges all other windows accordingly. |

**java.awt.Frame**

| **Methods** | | |
|---|---|---|
| void | setResizable(boolean b) | determines whether the user can resize this frame. |
| void | setTitle(String s) | sets the text in the title bar for the frame to the String s. |
| void | setIconImage(Image image) | sets the image you want to appear as the icon for the frame. |

**java.awt.Toolkit**

| **Methods** | | |
|---|---|---|
| static Toolkit | getDefaultToolkit() | returns the default toolkit. |
| Dimension | getScreenSize() | gets the size of the user's screen. |
| Image | getImage(String fileName) | loads an image from the file with the name fileName. |

## 1.9 REVIEW

## 1.10 FURTHER READING

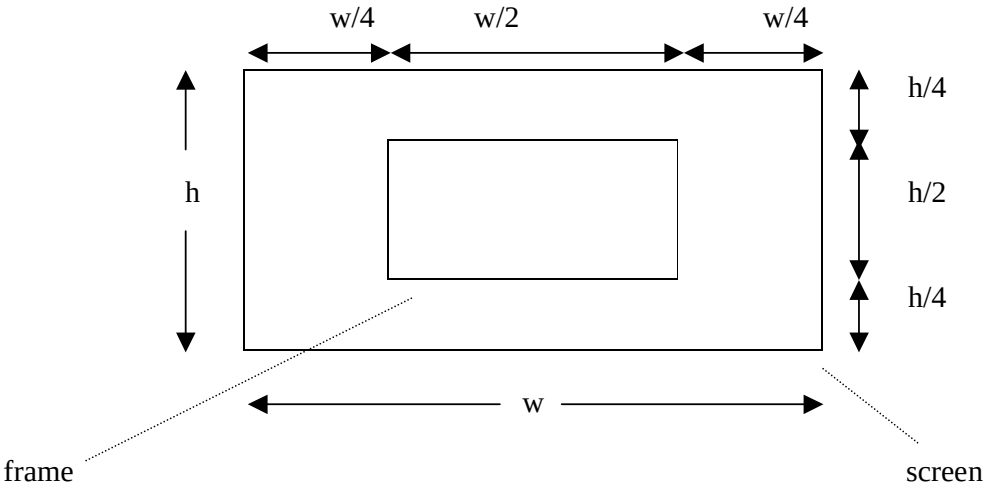
HORSTMANN & CORNELL *Core Java 2 Volume 1* pp 264
*www.java.sun.com/docs/books/tutorial*

In the next chapter we see how to place GUI components such as panels in a frame.

## 1.11 EXERCISES

**1** Try out the *CloseableFrame* class shown in §1.7 above. Explain what each line of coding does.

**2** Create a closeable frame that is initially ¼ the area of the whole screen. The diagram of a frame within the screen shown below should help.




w = screen width, h = screen height, w/2 = frame width, h/2 = frame height