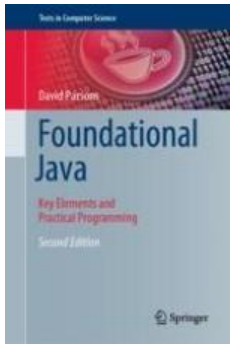


Chapter 17
Building GUIs with the JFC Swing Library

Foundational Java

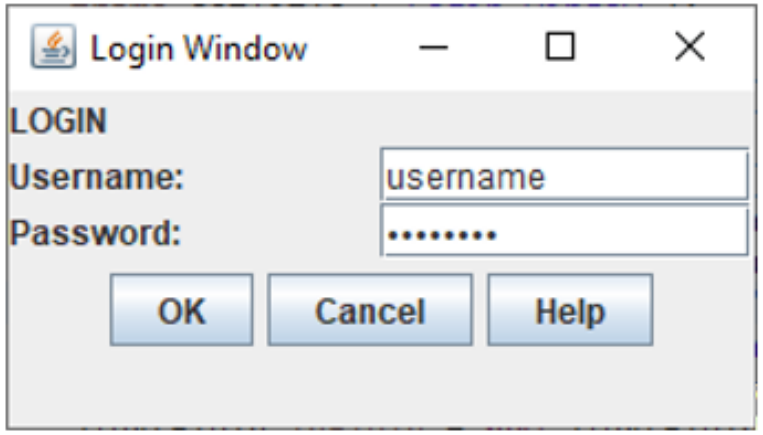
Key Elements and Practical Programming

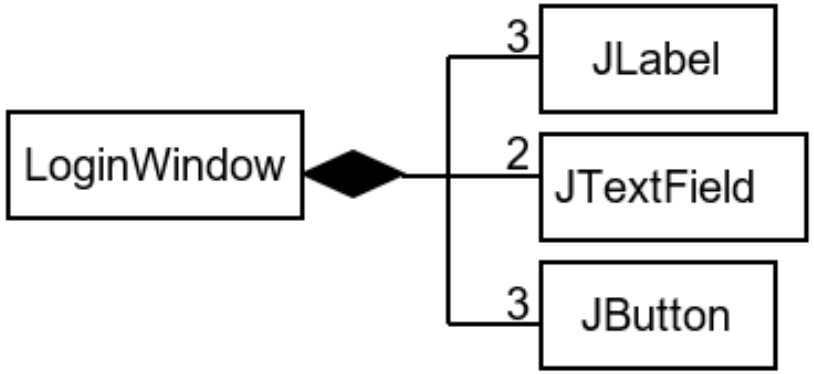
Graphical User Interfaces

- WIMP interaction
- WYSIWYG presentation
- Common look and feel across applications
  - Window appearance
  - Menus
- Reduced learning time for the user

GUI Programming

- GUI programming can be complicated
- OO languages make this easier
- GUI components have corresponding objects





UI Objects **Code Objects**

AWT, Swing, Java FX

- Java has three generations of UI framework
  - Abstract Windowing Toolkit (AWT)
  - Swing
  - JavaFX
- AWT was a very basic set of GUI components
  - built on existing native windowing systems
  - Inconsistent across platforms
  - ‘lowest common denominator’ components

Swing

- Part of the Java Foundation Classes (JFC)
- Cross-platform GUI components and services
  - Graphical libraries
  - Accessibility
  - Drag and drop support etc.
- ‘Swing’ provides a a fully featured set of classes for building Java graphical interfaces
- Some parts of AWT also important in Swing

JavaFX

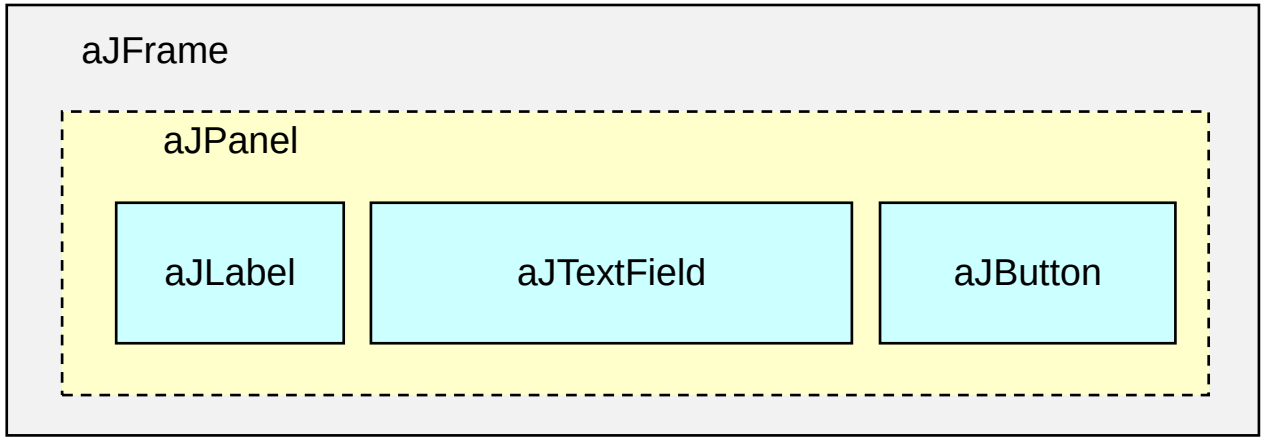
- First released in 2008
- Aimed to provide a more flexible and platform agnostic UI tool for rich clients across multiple platforms, including mobile
- Java as a language has moved away from developing new rich client applications, with most Java installations running as back end processing
- JavaFX never gained a high profile
  - it continues to be developed and used for new Java UI development.

Components, Containers and Frames

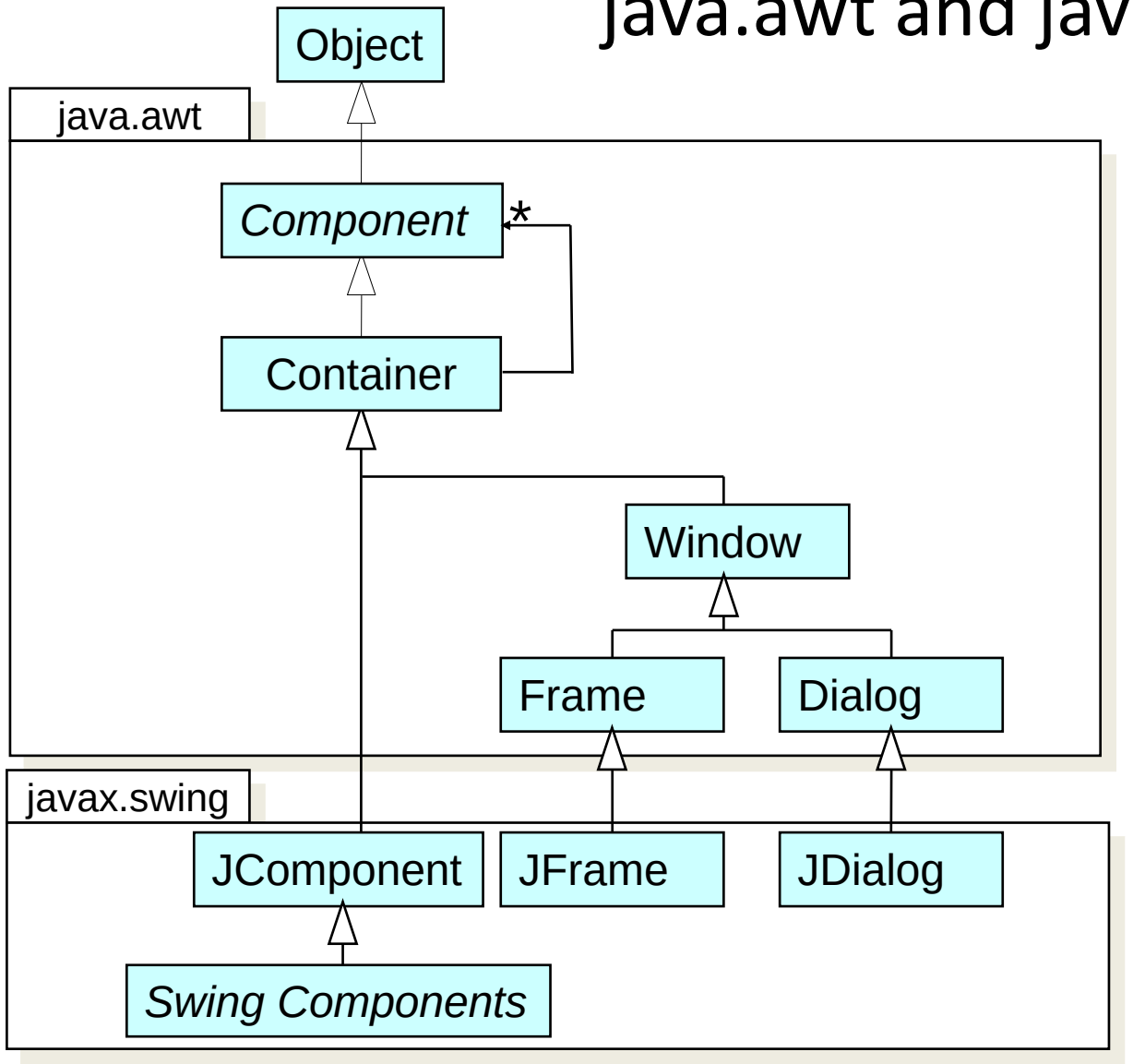
- Components are the objects that allow the user to interact with a graphical user interface
  - buttons, menus, text fields etc.
- A container is a component that can contain other components
  - ‘JFrame’, ‘JPanel’ etc.
- Most of Swing components have names beginning with ‘J’

Containers

- Containers (which are also components) hold one or more components in parent-child relationships
  - A component that contains another component is said to be a parent of that component
    - ‘aJPanel’ is the parent (container) of ‘aJLabel’, ‘aJTextField’ and ‘aJButton’
    - ‘aJFrame’ is the parent of ‘aJPanel’
    - ‘aJFrame’ has no parent



java.awt and javax.swing



- Swing components in javax.swing inherit from AWT components in java.awt

Container Features

- Components can be added and removed from containers
  - add(java.awt.Component)
  - remove(java.awt.Component)
- Containers have an optional layout manager
  - Controls the size and location of children
  - There will be a default layout manager applied
  - Can be changed or removed using
    - setLayout(java.awt.LayoutManager)

Creating a Main Window Frame

- In Swing, use javax.swing.JFrame
- We can pass the frame's title to the constructor

```
JFrame frame = new JFrame("My JFrame");
```

- Or use the 'setTitle' method

```
frame.setTitle("My JFrame");
```

Selecting Closing Behaviour

- The setDefaultCloseOperation method defines what to do when the user attempts to close the frame
- The javax.swing.WindowConstants class specifies four options
  - DO_NOTHING_ON_CLOSE
  - HIDE_ON_CLOSE (default) ← If you use the default, closing the window leaves it running but invisible
  - DISPOSE_ON_CLOSE ← Best option in most cases
  - EXIT_ON_CLOSE ← Also in JFrame class

```
frame.setDefaultCloseOperation(Jframe.EXIT_ON_CLOSE);
```

Sizing the JFrame

- The ‘setBounds’ method sets the initial position and dimensions of the frame
- First two parameters specify the top left hand corner position relative to screen
  - Top left hand corner of the screen is 0,0
- Other parameters define the initial width and height of the frame respectively

```
frame.setBounds(0, 0, 500, 200);
```

- Alternatively, ‘pack’ will size the frame to fit the components inside it

```
frame.pack();
```

Showing the JFrame

- The ‘setVisible’ method takes a boolean parameter that can show or hide the frame
- Passing it ‘true’ will make the frame visible

```
frame.setVisible(true);
```

- The frame can be moved, resized, minimised, maximised and closed

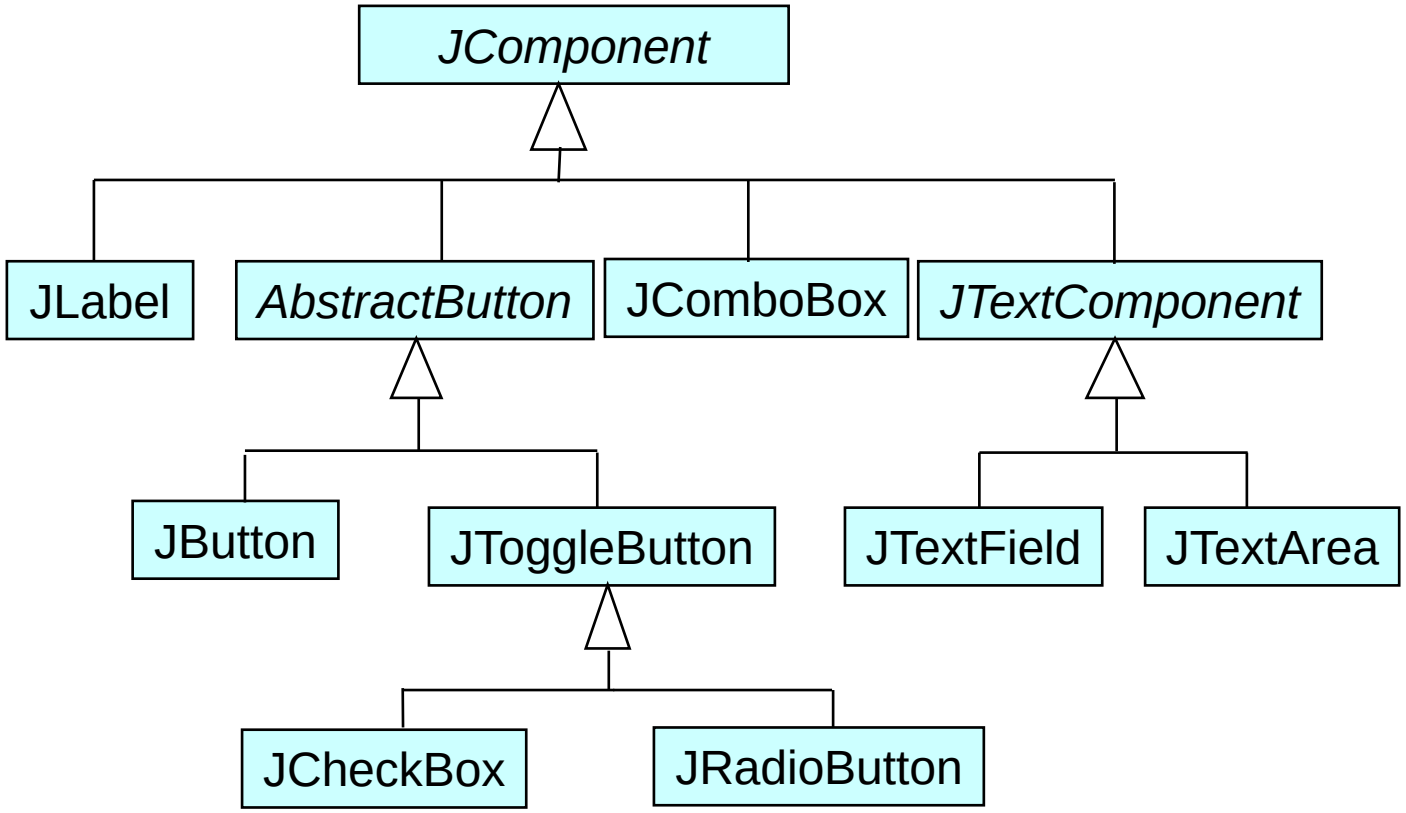
```
JFrame frame = new JFrame("My JFrame");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setBounds(0, 0, 500, 200);
frame.setVisible(true);
```

Swing Component Classes

- Some of the most commonly used components
  - JLabel
  - JTextField (single line of text that can be edited)
  - JTextArea (multiple lines of text that can be edited)
  - JButton
  - JCheckbox (individual)
  - JRadioButton (usually as part of a ButtonGroup)
  - JComboBox

Partial Swing Component Hierarchy

- Components are subclasses of 'JComponent'
  - All share inherited fields and methods



The JLabel class

- Text label
- Constructor can set the text of the label

```
JLabel textLabel = new JLabel("Component label");
```

- Default text alignment is left aligned
- Another version of the constructor can set the alignment using a SwingConstants parameter
  - LEFT, RIGHT, CENTER

```
JLabel centeredLabel = new JLabel("I am a label", SwingConstants.CENTER) ;
```

Adding Components to the Frame

- Components need to be added to the frame
- Can be done in a ‘main’ method, but better to create subclasses of JFrame
  - Add components in the constructor or other methods
- Components are added to Containers using ‘add’ methods, e.g.

```
JLabel myLabel = new JLabel ("Hello");
add(myLabel);
```

The Content Pane

- Components are not added directly to the JFrame
  - They are added to its content pane (A JRootFrame)
  - Content pane can be accessed using getContentPane
  - Convenience methods enable you to add components via the JFrame

```
JFrame frame = new JFrame();
Container container = frame.getContentPane();
container.add(new JLabel("Hello"));
```

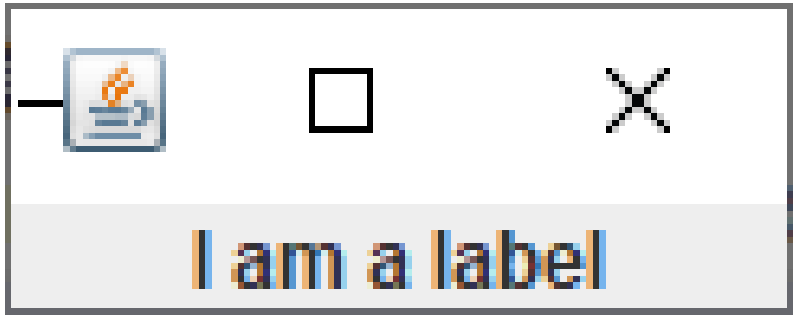
OR

```
frame.add(new JLabel("Hello"));
```

Adding a Label to a Frame

- Subclass of JFrame
- JLabel added in the constructor

```
public class LabelFrame extends JFrame
{
  private JLabel label;

  public LabelFrame(String title)
  {
    super(title);
    label = new JLabel("I am a label", CENTER) ;
    add(label);
  }
```



serialVersionUID Warning

- You will get the following warning from Eclipse on this class
- "The serializable class LabelFrame does not declare a static final serialVersionUID field of type long".
  - This is not important if you do not plan to Serialize objects of these classes
- If you do want to Serialize them, then then add a version number field like this (be sure to increment the version number if the class is changed).

```
private static final long serialVersionUID = 1L;
```

- If you do not plan to Serialize any of these classes and just want the warning to go away, add the following annotation above any classes that give the warning:

```
@SuppressWarnings("serial")
```

  - Whenever you suppress a warning, add a comment above the line to explain why you are doing so.

Manually Placing Components

- A JFrame object will have a default layout manager of the BorderLayout class
- Components can be placed manually within the frame
- Remove the default layout manager

```
setLayout(null);
```

- Can precisely set the size and position of our components

Component Size and Position

- Subclasses of Component inherit common methods
- Include methods to control the position and size of a component

```
public void setLocation(java.awt.Point p)
public void setSize(java.awt.Dimension d)
public void setBounds(int x, int y, int width, int height)
public void setBounds(java.awt.Rectangle r)
```

Other JComponent Methods

- Components can also be hidden or made visible using the ‘setVisible’ method

```
void setVisible(boolean)
```

- They can be enabled or disabled

```
void setEnabled(boolean)
```

- setToolTip controls what gets displayed when the mouse hovers over that component

```
void setToolTip(String s)
```

The JTextField class

- A JTextField allows text to be typed in and edited
- There is a default constructor with no parameters that will set the JTextField's initial width to zero columns

```
JTextField textField = new JTextField();
```

- An alternative constructor allows the number of text columns in the field to be specified

```
JTextField textField = new JTextField(20);
```

The JButton class

- JButtons are very simple objects because their only behaviour is to be pressed
- The JButton class has two constructors, one that creates a blank button and another one that creates a button with a text label
- This example creates a button with the label ‘Press Me!’

```
JButton    button = new JButton("Press Me!");
```

Placing Components Example

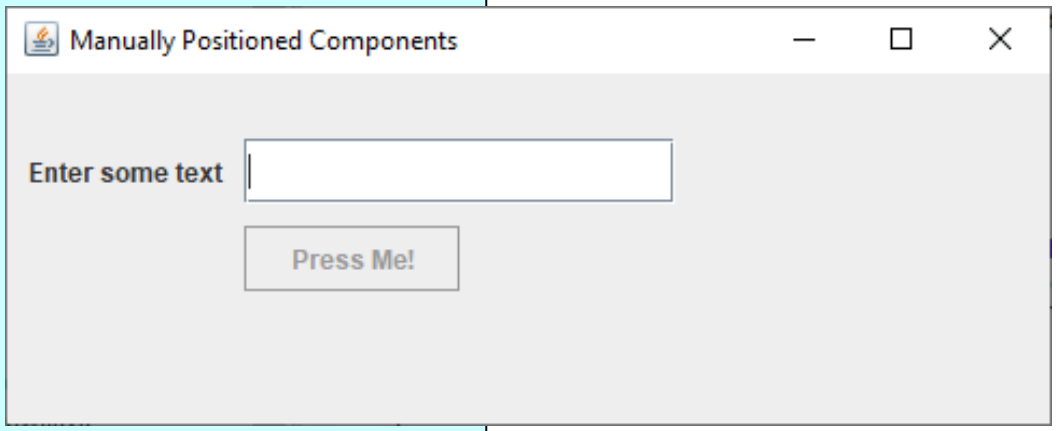
```
public class NoLayout extends JFrame
{
  private JLabel label;
  private JTextField textField;
  private JButton button;

  public NoLayout()
  {
    setLayout(null);
    label = new JLabel("Enter some text");
    textField = new JTextField(20);
    textField.setToolTipText("Enter some text here");
    button = new JButton("Press Me!");
    button.setEnabled(false);
    add(label);
    add(textField);
    add(button);
    label.setBounds(10, 20, 300, 50);
    textField.setBounds(110, 30, 200, 30);
    button.setBounds(110,70,100,30);
  }
```

turn off the default layout manager

create the components

add them to the window

manually position and size them



Exercise 17.1

- Develop the previous example so that it becomes a login window, with one field to enter a username and another to enter a password
- The password entry field should be an object of the JPasswordField class, rather than a JTextField
- Provide appropriate labels for the fields
- Change the text on the button to say “Log In”, and position the components manually within the frame
- Instead of using “setBounds”, try using “setLocation” and “setSize” to place the components in the frame

Colors and Fonts

- Colors and Fonts can be used to configure Components using the following methods

```
void setBackground(Color c)
void setForeground(Color c)
void setFont(Font f)
```

- The java.awt.Color class represents a Color object that can be passed to components
- The java.awt.Font class maps characters to *glyphs* (shapes)

java.awt.Color

- Thirteen standard colors are available through static final fields in the Color class
  - black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white and yellow
- These fields are specified in both upper and lower case so you can use either (upper case is preferred)

```
Color.darkGray
```

or

```
Color.DARK_GRAY
```

- Other colors can be created by passing Red, Green, and Blue (RGB) values to a Color constructor
  - values in the range zero to 255

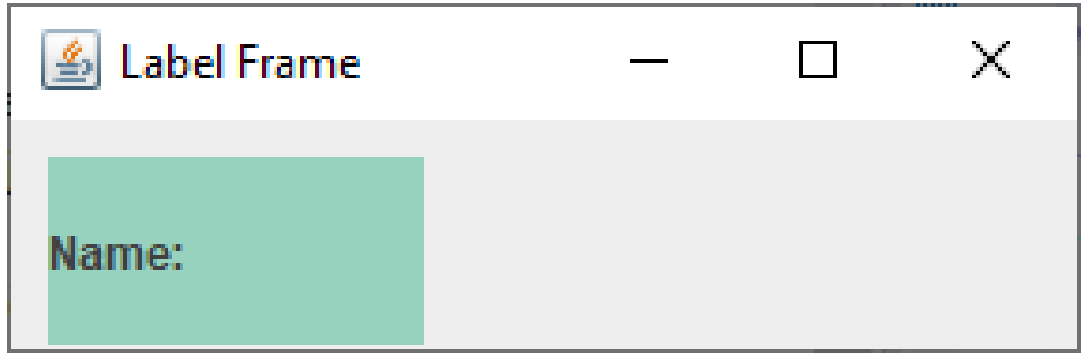
```
new Color(n, n, n);
```

- Color objects are immutable once created

Color Example

- Sets foreground and background colors of a JLabel
  - One constant, one RGB object
- To set a JLabel background, set the ‘opaque’ value to ‘true’

```
JLabel label = new JLabel("Name:");
label.setOpaque(true);
label.setForeground(Color.darkGray);
label.setBackground(new Color(150, 210, 190));
```



Font Names

- The Font constructor takes three parameters
  - Font name, character style, size
- The font name can be a logical name
  - Serif, SansSerif, Monospaced, Dialog, DialogInput, Symbol
- Or platform font family
  - You can get a String array of available font names from the GraphicsEnvironment object, using the ‘getAvailableFontFamilyNames’ method

Fonts Styles and Sizes

- Font styles
  - Font.BOLD, Font.ITALIC and Font.PLAIN
  - BOLD and ITALIC can be combined using the '|' operator Font.BOLD|Font.ITALIC
- The third constructor parameter represents the font size in points

```
new Font("Helvetica", Font.BOLD, 12);
```
← 12 point, bold Helvetica font

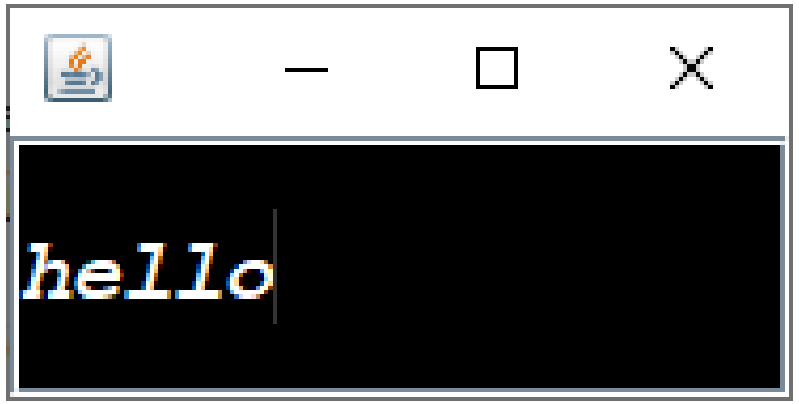
```
new Font("SansSerif", Font.ITALIC, 20);
```
← 20 point, italic SansSerif font

```
new Font("Courier", Font.BOLD | Font.ITALIC, 10);
```
← 10 point, bold and italic Courier font

Color and Font Example

- In this code fragment, a JTextField has both its Colors and its Font set

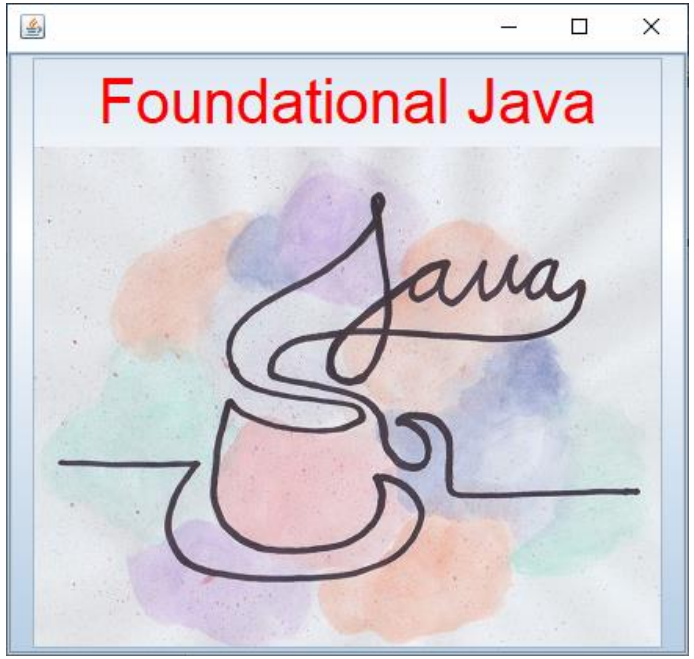
```
JTextField field = new JTextField();
field.setBackground(Color.black);
field.setForeground(Color.white);
field.setFont(new Font("Courier", Font.ITALIC | Font.BOLD, 20));
```



Icons on Labels and Buttons

- AbstractButton and JLabel have an icon property as well as text
  - This includes subclasses of AbstractButton
  - JCheckBox, JMenuItem, JButton, JRadioButton, ...
- Text can be oriented on any side of the image or on top of it
  - horizonalTextPosition, verticalTextPosition

```
JButton button = new JButton
  ("Foundational Java", new ImageIcon("Java.png"));
button.setHorizontalTextPosition(SwingConstants.CENTER);
button.setVerticalTextPosition(SwingConstants.TOP);
button.setFont(new Font("helvetica", Font.PLAIN ,40));
button.setForeground(Color.red);
```



Exercise 17.2

- Change the Colors and Fonts of the components you added to a frame in Exercise 17.1
- Include a change to the background color of the JFrame's content pane

JTextArea

- A JTextArea is similar to a JTextField except that it can have multiple rows and columns
- This constructor allows you to set the number of rows and columns:

```
public JTextArea(int rows, int columns)
```

- JTextAreas are simple text entry components
- More sophisticated multi line text editing can be performed with JTextPanes or JEditorPanes

Check Boxes and Radio Buttons

- A check box is square
  - Can be checked or unchecked independently
- A radio button is round
  - Usually appears as part of a group
  - Only one radio button can be checked at any one time
- However both have the same superclass (JToggleButton) so do the same things
  - Have to manage their differences manually

JCheckBox

- Check boxes are created using the JCheckBox class
  - Usually instantiated with text labels as parameters

```
JCheckBox check = new JCheckbox("Tick me");
```

  - Alternative constructor initialises the state of the check box using a boolean parameter

```
JCheckBox mailCheck = new JCheckBox("Don't send me stuff", true);
```

  - Using a 'false' parameter would create an unchecked box

Radio Buttons and Button Groups

- To create radio buttons, we use the JRadioButton class
- Can select one radio button in a group

```
JRadioButton radio1 = new JRadioButton("Radio 1", true);
JRadioButton radio2 = new JRadioButton("Radio 2");
```

- Radio boxes need to be added to a ButtonGroup to operate together

```
ButtonGroup buttons = new ButtonGroup();
buttons.add(radio1);
buttons.add(radio2);
```

JComboBox

- Displays a single item with a button for a drop down a list of items
- The simplest JComboBox constructor takes no parameters

```
JComboBox dropDownList = new JComboBox();
```

- Can add objects using the ‘add’ method:

```
dropDownList.add("cucumber");
```

  – You can provide your own implementation of the ListCellRenderer interface

JSlider

- Swing is not dependent on the native UI components
- Can provide specialised components like sliders
- Represents a slider control
  - Sound volume, image zoom etc.
  - Can be vertical or horizontal
  - Can show labels or tick marks

```
JSlider sliderControl = new JSlider();
```

JPanel

- A lightweight container used inside other containers
  - Can divide up the area of a frame and add different panels to different areas
- Each panel can host its own subset of components

```
JFrame frame = new JFrame();
JPanel panel = new JPanel();
JTextField textField = new JTextField(20);
JButton button = new JButton("Press me!");
panel.add(textField);
panel.add(button);
frame.add(panel);
```

Setting the Look and Feel

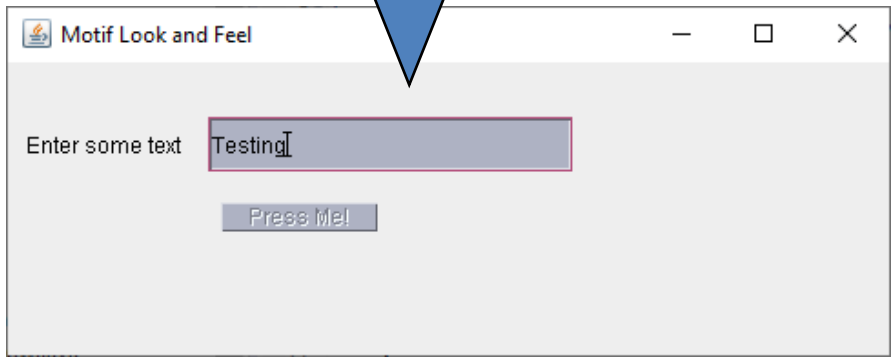
- Swing emulates the look-and-feel of different platforms
- The default is the cross platform look and feel
- Can also set the look and feel to match the local system or the ‘motif’ look and feel

```
UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
```

```
UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
```

default

```
UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");
```



Exercise 17.3

- Add components of the JTextField and JComboBox classes to a frame
- Add one JCheckBox and two JRadioButtons in a single ButtonGroup
- Position the components using ‘setLocation’ and ‘setSize’
- Set the look and feel to use the system look and feel

Exercise 17.4

- Modify your solution to Exercise 17.3 so that your components are added to a JPanel rather than directly to a JFrame
- Add the JPanel to the JFrame
- Remove all your ‘setLocation’ and/or ‘setBounds’ methods
- You should find that the components are automatically organized within the window
  - This is because of the layout manager used with JPanels, as explained in the next section

Layout Managers

- Manually sizing and placing components is inflexible
- Layout managers organize the components according to a standard pattern
- The Component class provides methods to give the layout manger information about their display sizes
  - Return Dimension objects

```
getMinimumSize()
getPreferredSize()
getMaximumSize()
```

Reorganizing Components

- Layout managers handle much of the work of GUI creation
  - When the window is resized, the layout manager calculates the new sizes and positions of the components
  - If components are added or removed, the others are repositioned and resized appropriately

Layout Manager Types

- There were five different layout managers in AWT
  - FlowLayout
  - BorderLayout
  - GridLayout
  - CardLayout
  - GridBagLayout
- More have been added with, and since, Swing
  - BoxLayout
  - ScrollPaneLayout
  - ViewportLayout
  - SpringLayout

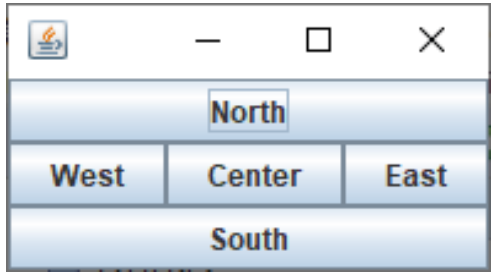
BorderLayout Manager

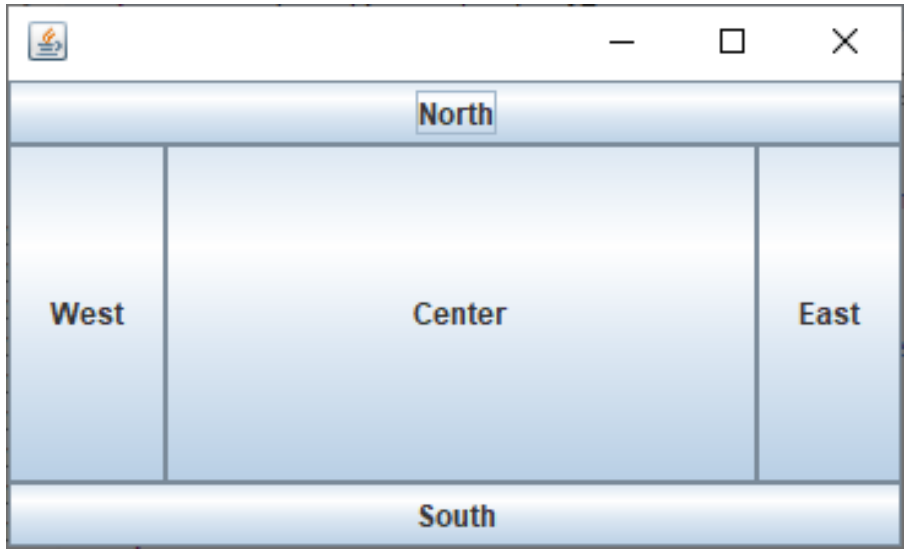
- Default layout manager for a JFrame's content pane
- Arranges components along each edge of the container, with a fifth component residing in the center
- Locations (BorderLayout constants)
  - NORTH, SOUTH, EAST, WEST, CENTER
- Components are resized according to the pattern
  - Any extra space is given to the "Center" componentYou cannot have more than five components added to a Border Layout but you can have fewer. The other components will fill the available space.

BorderLayout Manager

- Adding components to a JFrame's BorderLayout

```
JFrame frame = new JFrame();
frame.add(new JButton("South"), BorderLayout.SOUTH);
frame.add(new JButton("Center"), BorderLayout.CENTER);
frame.add(new JButton("East"), BorderLayout.EAST);
frame.add(new JButton("West"), BorderLayout.WEST);
```

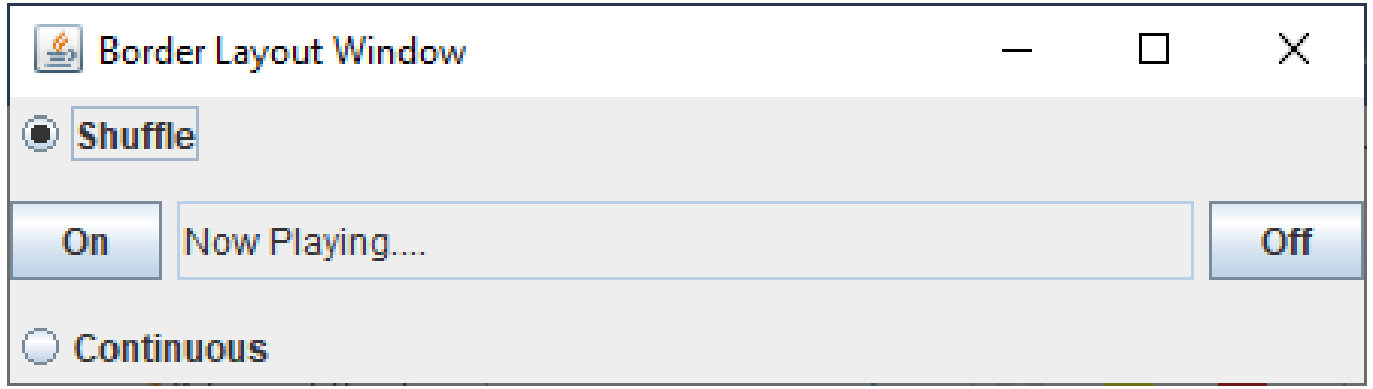




BorderLayout Constraints

- Can configure the horizontal and vertical gap between components
- Can use the constructor or ‘set’ methods

```
public class BorderLayoutWindow extends JFrame
{
    public BorderLayoutWindow()
    {
        setLayout(new BorderLayout(5,10));
```



FlowLayout Manager

- Lays the components out like text; left to right, wrapping onto the next line where necessary
- Components are given their “preferred size”
  - Components are not resized, just rearranged
- This is the default layout manager for a JPanel

```
public class MusicPlayerPanel extends JPanel  {
  public MusicPlayerPanel() {
    JLabel textLabel = new JLabel("Volume");
    // etc…
    add(textLabel);
    // etc…
  }
}
```

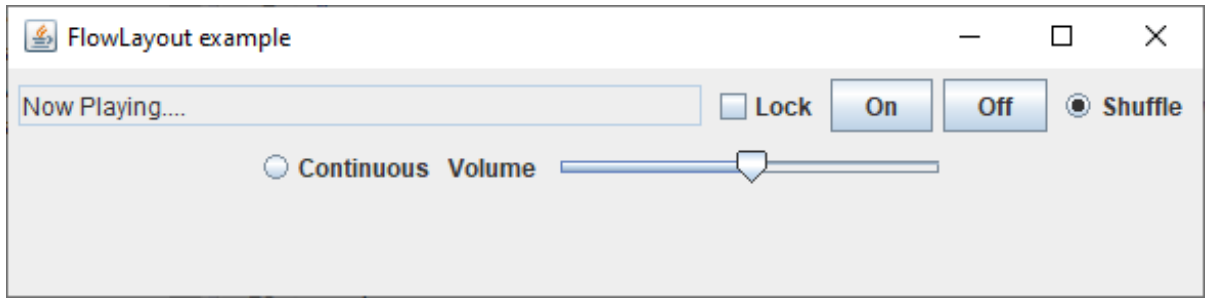
Will add to a FlowLayout

Adding Panels to a Frame

- BorderLayout may appear restrictive
- However region can contain panels
  - JPanel is a subclass of JComponent
- Adding a MusicPlayerPanel to a JFrame
  - the 'add' method will add the panel to the central region of the layout

```
MusicPlayerPanel playerPanel = new MusicPlayerPanel();
add(playerPanel);
```

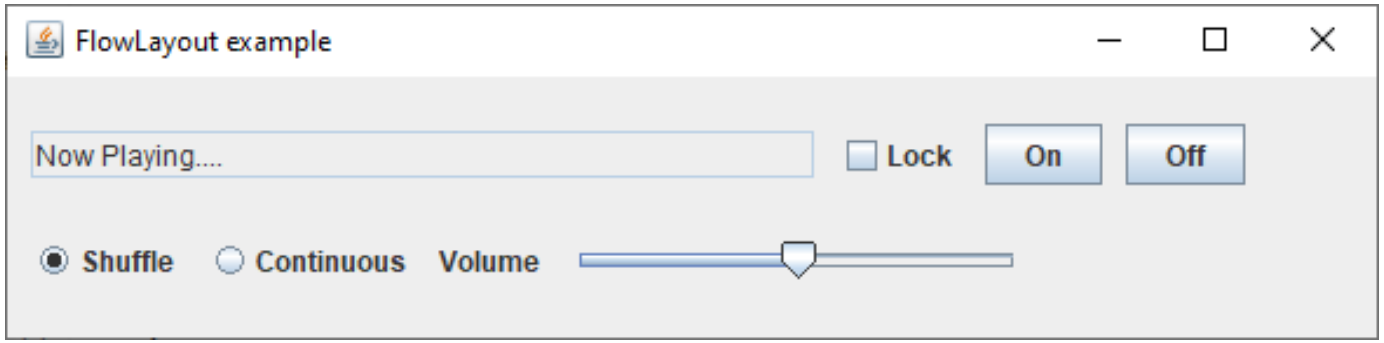
  - Other components or panels can be added to other regions of the BorderLayout



FlowLayout Constraints

- Horizontal and vertical gaps can be set
- Alignment can also be changed
  - LEFT, CENTER, RIGHT…
- e.g. set the alignment to the left, with a horizontal gap of 10 and a vertical gap of 20

```
MusicPlayerPanel playerPanel = new MusicPlayerPanel();
playerPanel.setLayout(new FlowLayout(FlowLayout.LEFT, 10, 20));
add(playerPanel);
```



GridLayout Manager

- Components are laid out left to right and top to bottom
  - Can specify the number of rows or columns
  - Components will resize to fit
- Constructor has two integer parameters for the rows or columns
  - If the first parameter is non-zero then the layout will use that number of rows
  - If the first parameter zero, then the layout will use the second (columns) parameter instead

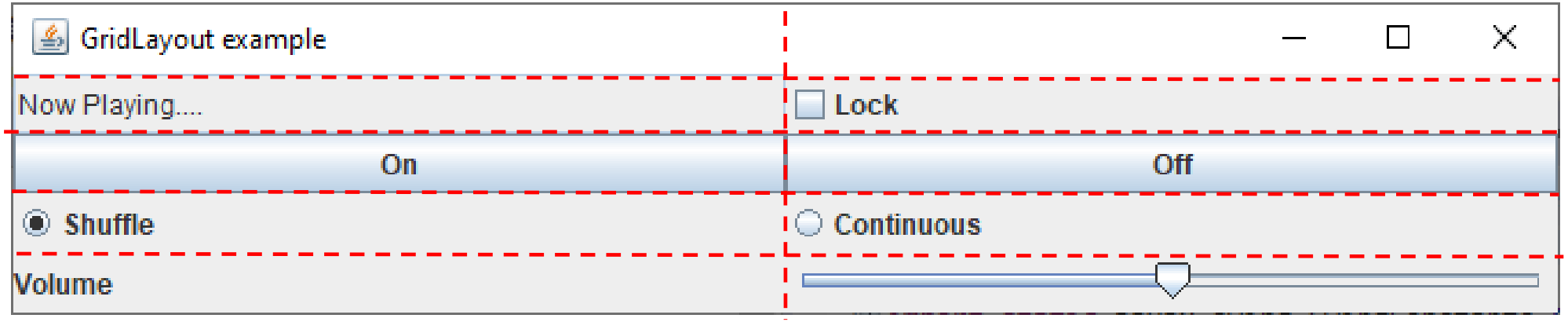
GridLayout Example

- This example will use two columns with an automatic setting for the number of rows

```
playerPanel.setLayout(new GridLayout(0, 2));
```

- Further parameters can be added to set the horizontal and vertical gaps

```
playerPanel.setLayout(new GridLayout(0, 2, 5, 5));
```



Exercise 17.5

- Change the panel from exercise 17.4 to use a 4 column GridLayout

Summary

- Components and Containers
- Component types
  - labels, buttons, radio buttons, text fields combo boxes…
- Colors and Fonts
- Changing the look and feel
- Manual layout of components in a container
- JPanels
- Automatic layout managers
  - BorderLayout (the default layout manager for JFrames)
  - FlowLayout (the default layout manager for JPanels)
  - GridLayout