COM1003 Java Programming

Richard Clayton

A Very Graphic Story

Graphics and Graphical User Interfaces in Java

A History Lesson

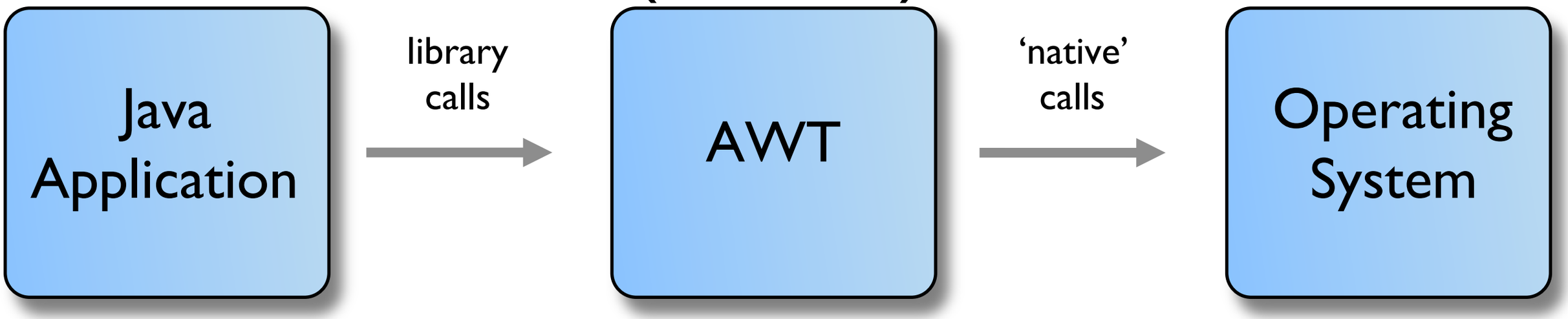
In the early days of Java, there was the AWT.

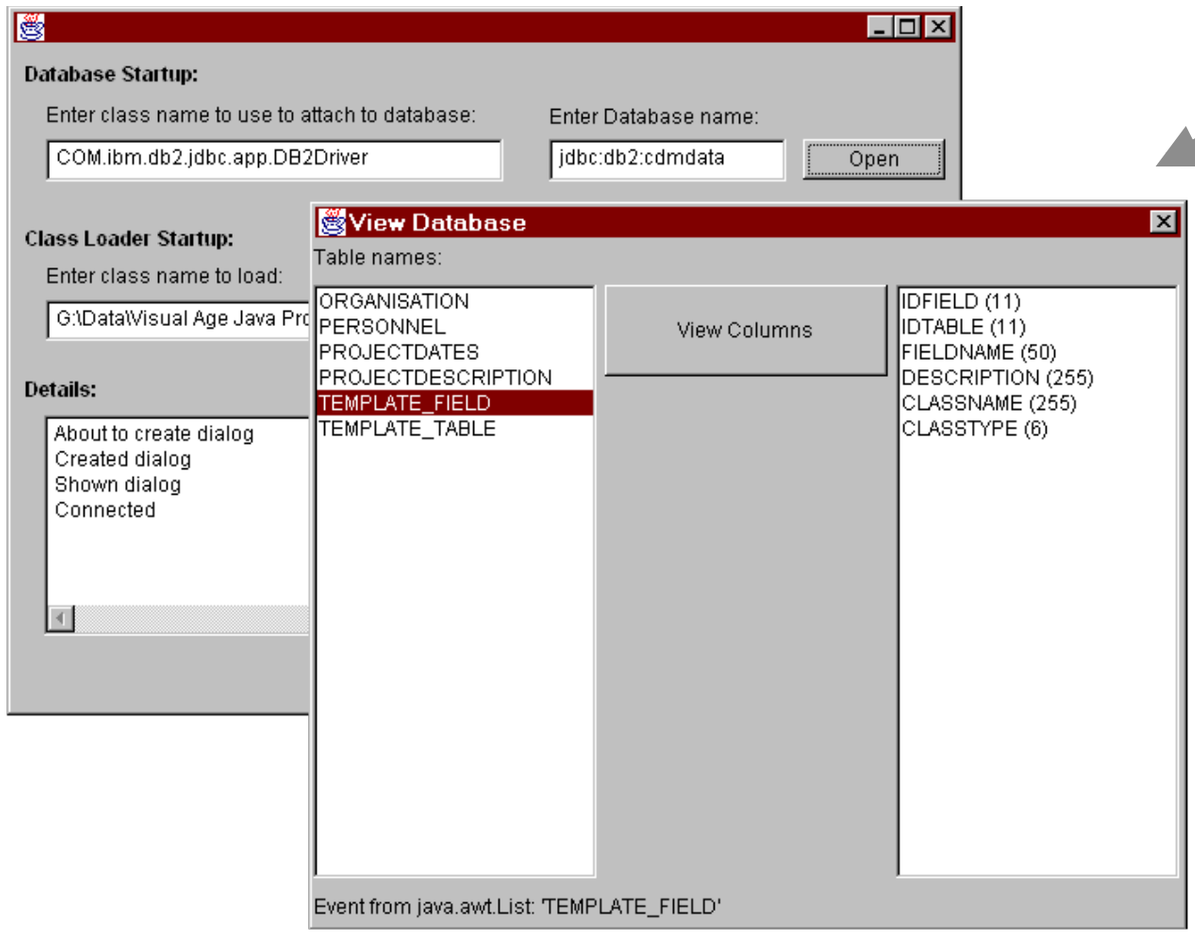
AWT stands for 'Abstract Window Toolkit'.

The underlying philosophy was the following:

- Graphical User Interfaces are usually drawn by the operating system ... so Java should do the same.
- A Java Windows application should look like a Windows application
- A Java Mac OS application should look like a Mac OS application
- ... and so on

The Abstract Window Toolkit (AWT)





renders GUI

The Abstract Window Toolkit (AWT)

Java is supposedly "Write Once Run Anywhere".

- Once you have written a program it should run on any machine with any operating system that has Java

So the AWT had to abstract for every GUI component on every platform.

- java.awt.Button for buttons
- java.awt.Frame for windows ...

But not all types of GUI components exist on all platforms.

Some work in different ways on some platforms to others.

- e.g. menus on Windows compared to Mac

Layout issues – slight differences on different platforms.

Swing – the successor to the AWT

The Swing library attempts to fix some of the problems with the AWT

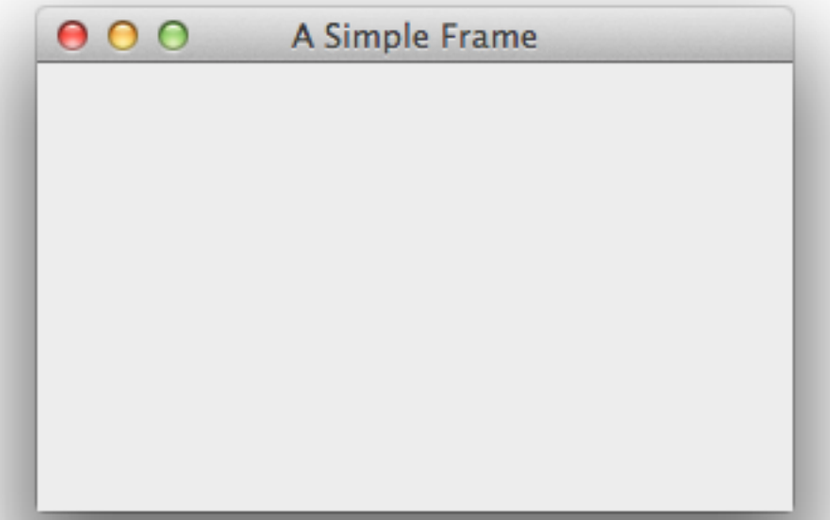
The operating system is responsible for generating a window, but Swing draws the GUI elements

- Consistent look and feel across platforms
- AWT components live in the java.awt package.
- Swing components live in the javax.swing package.
- Swing components start with a "J", which differentiates them from their AWT counterparts.

Getting Started

A top level window is called a ‘frame’ in Java. The Swing class is called JFrame.

```
import javax.swing.*;

public class SimpleFrame extends JFrame {

    public SimpleFrame() {
        setTitle("A Simple Frame");
        setSize(300, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }

    public static void main(String[] args) {
        new SimpleFrame();
    }
}
```



Constructing a JFrame does not display it. We need to call setVisible(true)

This causes the program to terminate when the window is closed

Positioning a JFrame

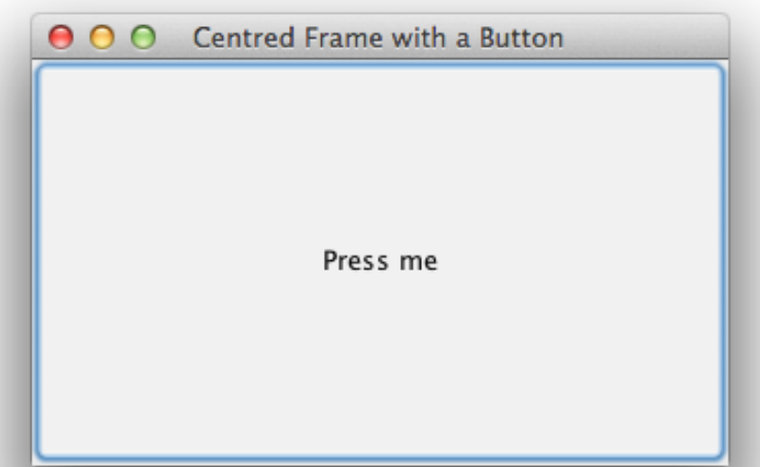
```
import java.awt.Dimension;
import java.awt.Point;
import java.awt.Toolkit;
import javax.swing.JFrame;

public class CentredFrame extends JFrame {

    public CentredFrame() {
        setTitle("A Centred Frame");

        // A Toolkit lets us retrieve system information.
        // Find out more about it in the JavaDocs.
        Toolkit toolkit = Toolkit.getDefaultToolkit();

        Dimension screenDimensions = toolkit.getScreenSize();
        setSize(screenDimensions.width/2, screenDimensions.height/2);
        setLocation(new Point(screenDimensions.width/4, screenDimensions.height/4));

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {
        CentredFrame centredFrame = new CentredFrame();
        centredFrame.setVisible(true);
    }
}
```

Adding GUI Components

Creates a button with a simple label

```
import java.awt.Container;
import javax.swing.JButton;

public class OneButtonFrame extends CentredFrame {

    public OneButtonFrame() {
        setTitle("Centred Frame with a Button");

        // instantiate a button
        JButton button = new JButton("Press me");

        // add it to the JFrame
        Container contentPane = getContentPane();
        contentPane.add(button);
    }

    public static void main(String[] args) {
        OneButtonFrame oneButtonFrame = new OneButtonFrame();
        oneButtonFrame.setVisible(true);
    }
}
```

GUI components are laid onto containers. To add the button we need to get JFrame's container – called its "content pane"



Laying out components

As soon as we want to add multiple components to a container, we are confronted with the problem of layout.

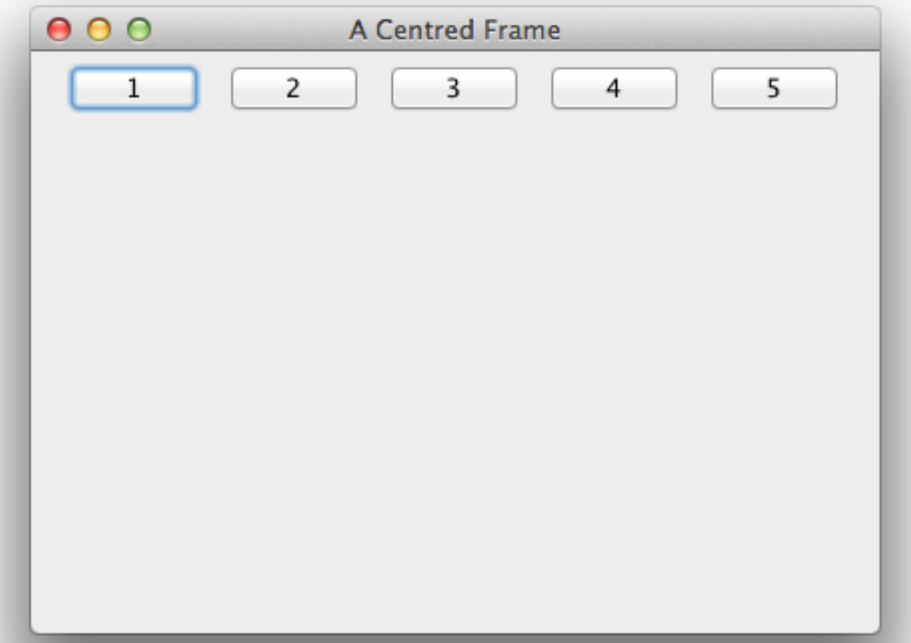
We need to use a layout manager, which is attached to a container and controls how the GUI components are positioned in the container.

A typical interface will have several layout managers, each responsible for a different portion of the overall interface.

Layout Managers

FlowLayout lays the components out in a centred row

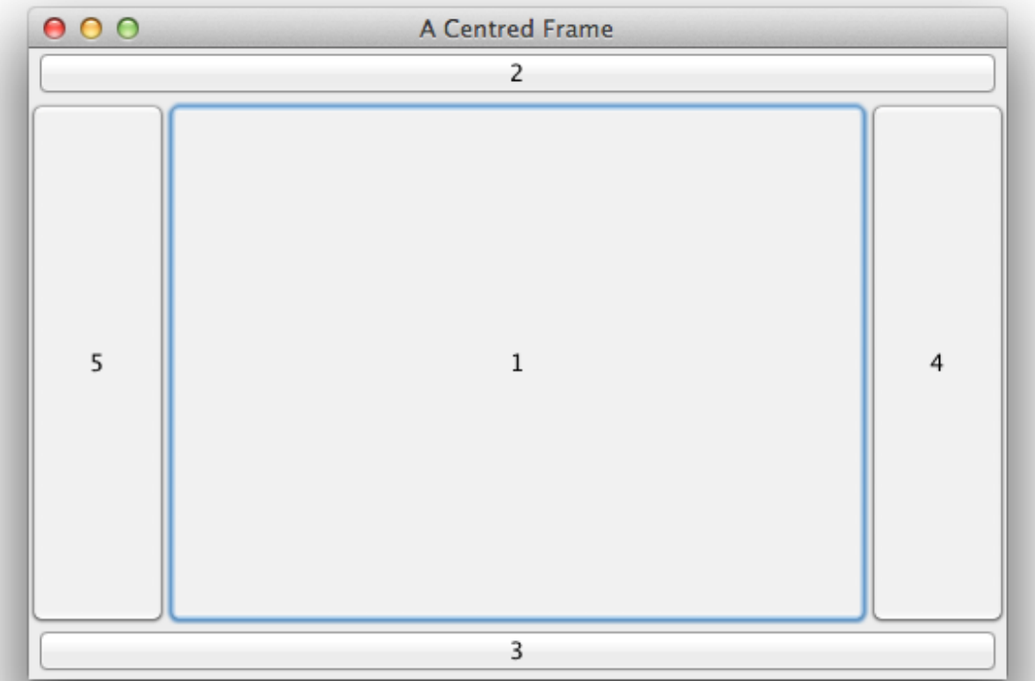
```
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());

contentPane.add(new JButton("1"));
contentPane.add(new JButton("2"));
contentPane.add(new JButton("3"));
contentPane.add(new JButton("4"));
contentPane.add(new JButton("5"));
```



Layout Managers

BorderLayout divides up the container into five zones - ‘north’, ‘south’, ‘east’, ‘west’ and ‘center’

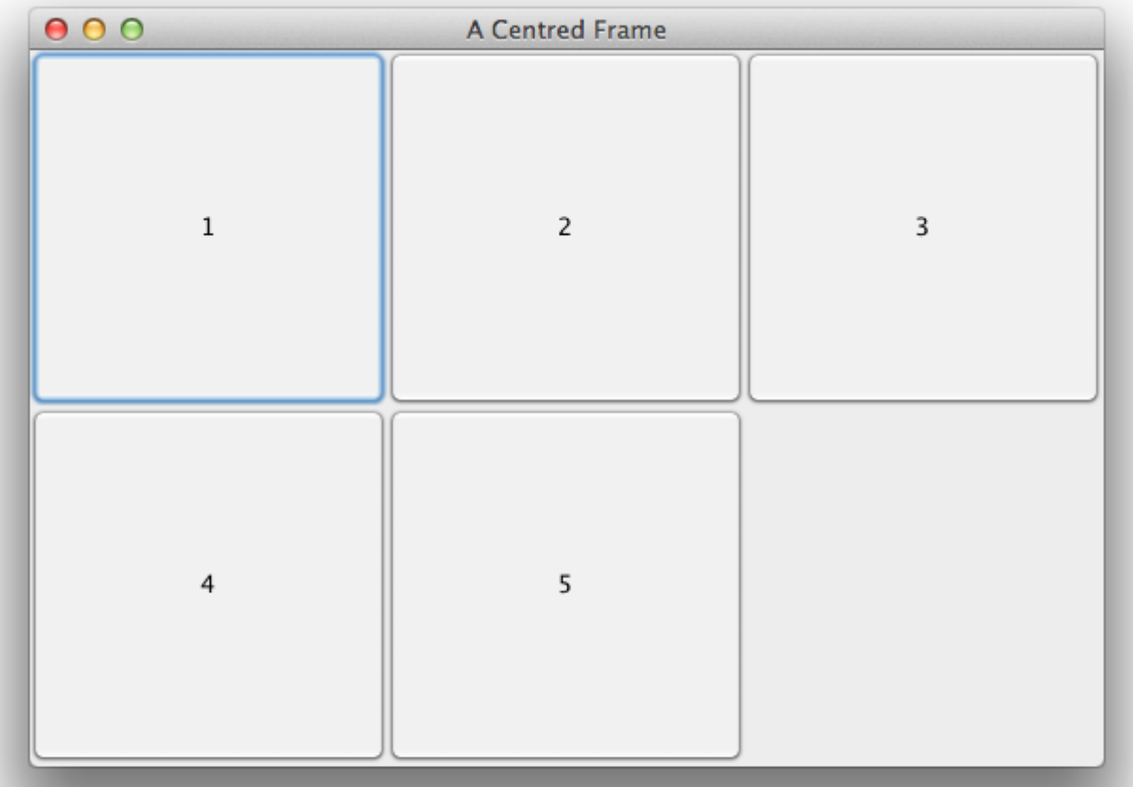
```
Container contentPane = getContentPane();
contentPane.setLayout(new BorderLayout());

contentPane.add(new JButton("1"), BorderLayout.CENTER);
contentPane.add(new JButton("2"), BorderLayout.NORTH);
contentPane.add(new JButton("3"), BorderLayout.SOUTH);
contentPane.add(new JButton("4"), BorderLayout.EAST);
contentPane.add(new JButton("5"), BorderLayout.WEST);
```



Layout Managers

GridLayout lays out components in a 2D grid

```
Container contentPane = getContentPane();
contentPane.setLayout(new GridLayout(0, 3));

contentPane.add(new JButton("1"));
contentPane.add(new JButton("2"));
contentPane.add(new JButton("3"));
contentPane.add(new JButton("4"));
contentPane.add(new JButton("5"));
```



The constructor takes two integers: the number of rows, and the number of columns. If one dimension is left as zero, it is inferred according to the number of elements added. So here we constrain the grid to 3 columns, and a flexible number of rows.

Other Layout Managers

BoxLayout - Single row or column. More flexible over uneven size and spacing than GridLayout

GridBagLayout - Flexible form of GridLayout. Complex. Uses notion of constraints.

SpringLayout - Works by defining relationships between component edges.

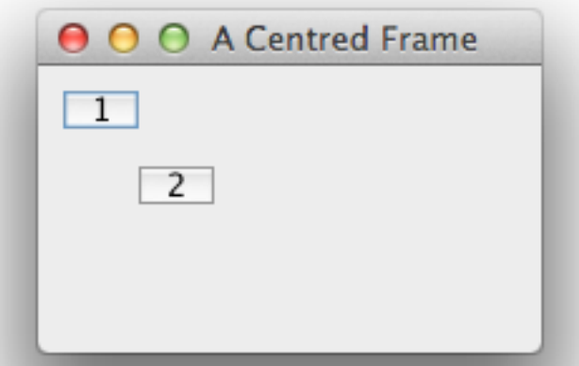
CardLayout - Area contains different components at different times (like tabs).

Check them out in the JavaDocs.

Layout Managers

Or we can specify the null layout, where we have to code absolute positions. However, the layout does not adjust to the container size, as with a proper layout manager.

```
Container contentPane = getContentPane();
contentPane.setLayout(null);

JButton jbutton1 = new JButton("1");
JButton jbutton2 = new JButton("2");

contentPane.add(jbutton1);
contentPane.add(jbutton2);

jbutton1.setBounds(10, 10, 30, 15);
jbutton2.setBounds(40, 40, 30, 15);
```



absolute positioning

Hierarchical Layouts

Most GUIs have containers within containers, each with a different layout, in order to achieve the desired effect.

```
JLabel picture = new JLabel(image);
picture.setPreferredSize(new Dimension(800, 600));

JPanel buttonPanel = new JPanel();
buttonPanel.setLayout(new GridLayout(0, 1));
buttonPanel.add(new JButton("Bird"));
buttonPanel.add(new JButton("Cat"));
buttonPanel.add(new JButton("Dog"));
buttonPanel.add(new JButton("Alligator"));

Container contentPane = getContentPane();
contentPane.setLayout(new BorderLayout());
contentPane.add(buttonPanel, BorderLayout.WEST);
contentPane.add(picture, BorderLayout.CENTER);
```

JLabels are a GUI component that we can use to display images or text

JPanels are containers that we can use to achieve hierarchical layouts. We can also draw onto panels - more later

GridLayout

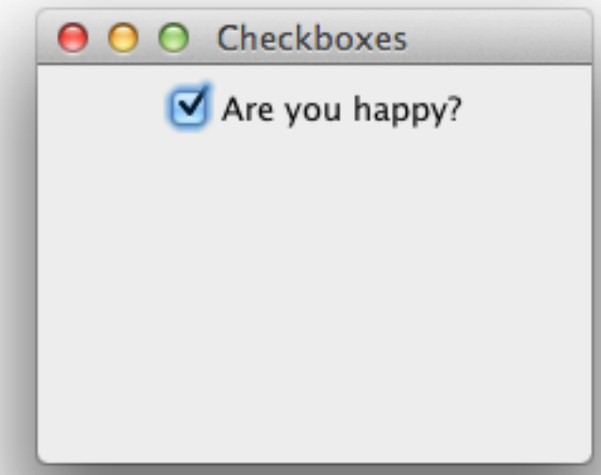
BorderLayout

Further GUI Components

Swing offers a variety of different GUI components

e.g. Checkboxes

```
JCheckBox cb = new JCheckBox("Are you happy?");
cb.setSelected(true);
container.add(cb);
```

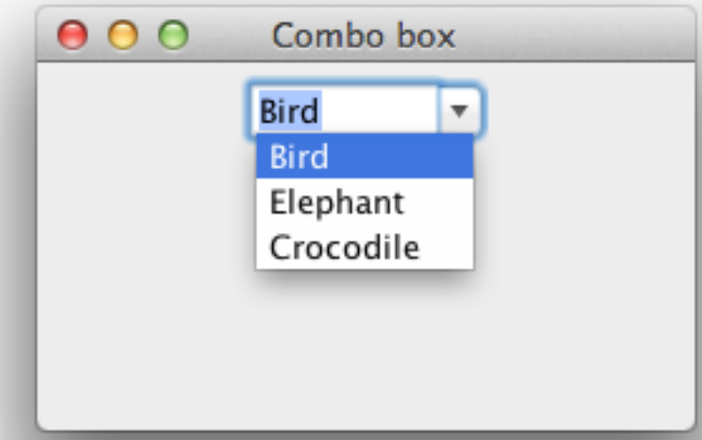


Radio buttons

```
ButtonGroup group = new ButtonGroup();

JRadioButton r1 = new JRadioButton("1");
group.add(r1);

JRadioButton r2 = new JRadioButton("2");
group.add(r2);

JRadioButton r3 = new JRadioButton("3");
group.add(r3);

container.add(r1);
container.add(r2);
container.add(r3);
```

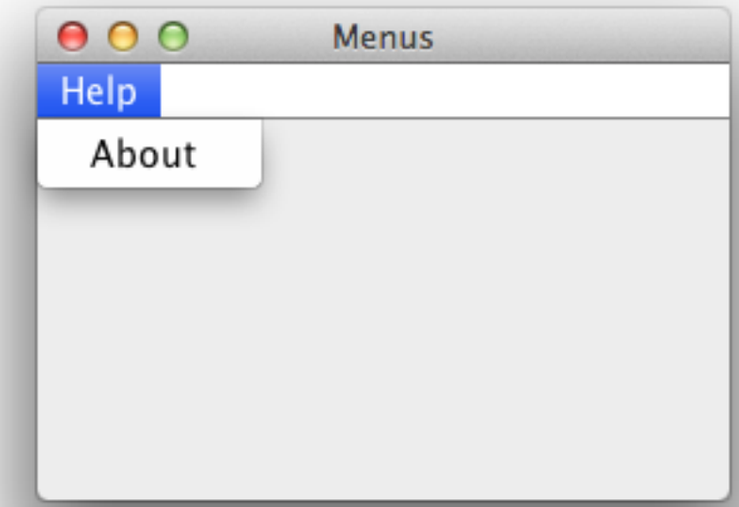
Combo boxes (drop-down lists)

```
JComboBox comboBox = new JComboBox();
comboBox.addItem("Bird");
comboBox.addItem("Elephant");
comboBox.addItem("Crocodile");

// can set it so that the box may be edited
comboBox.setEditable(true);

container.add(comboBox);
```



Menus

```
JMenuBar menuBar = new JMenuBar();
this.setJMenuBar(menuBar);

JMenu menu = new JMenu("Help");
menuBar.add(menu);

JMenuItem menuItem = new JMenuItem("About");
menu.add(menuItem);
```



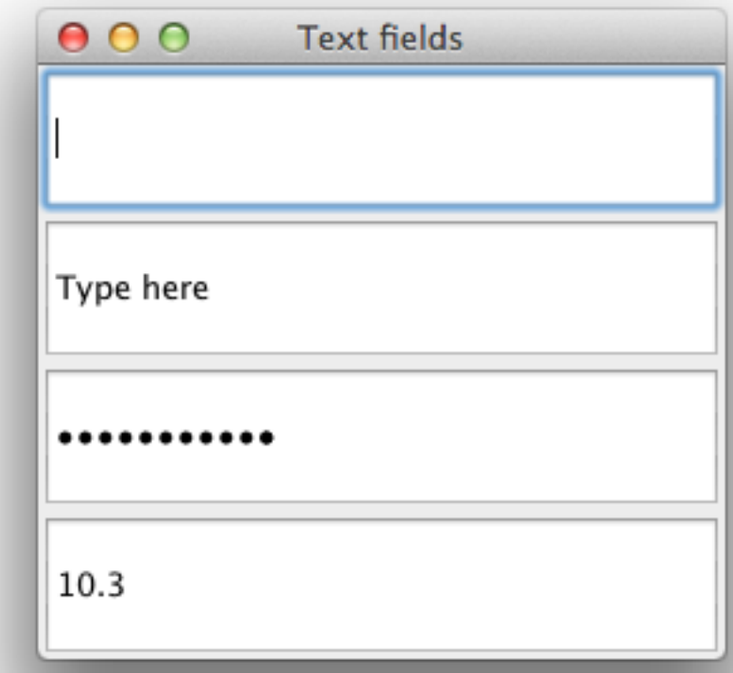
Adds the menu bar to a JFrame instance (this)

Labels

```
container.add(new JLabel("The first label of the set"));
container.add(new JLabel("Right-aligned label", JLabel.RIGHT));
container.add(new JLabel("<html><b>HTML</b> <i>label</i></html>"));
```

Text Fields

```
JTextField tf1 = new JTextField(20);

JTextField tf2 = new JTextField(20);
tf2.setText("Type here");

JPasswordField pf = new JPasswordField(20);
pf.setText("No peeking!");

JFormattedTextField ftf = new JFormattedTextField();
ftf.setValue(new Double(10.3));

container.add(tf1);
container.add(tf2);
container.add(pf);
container.add(ftf);
```

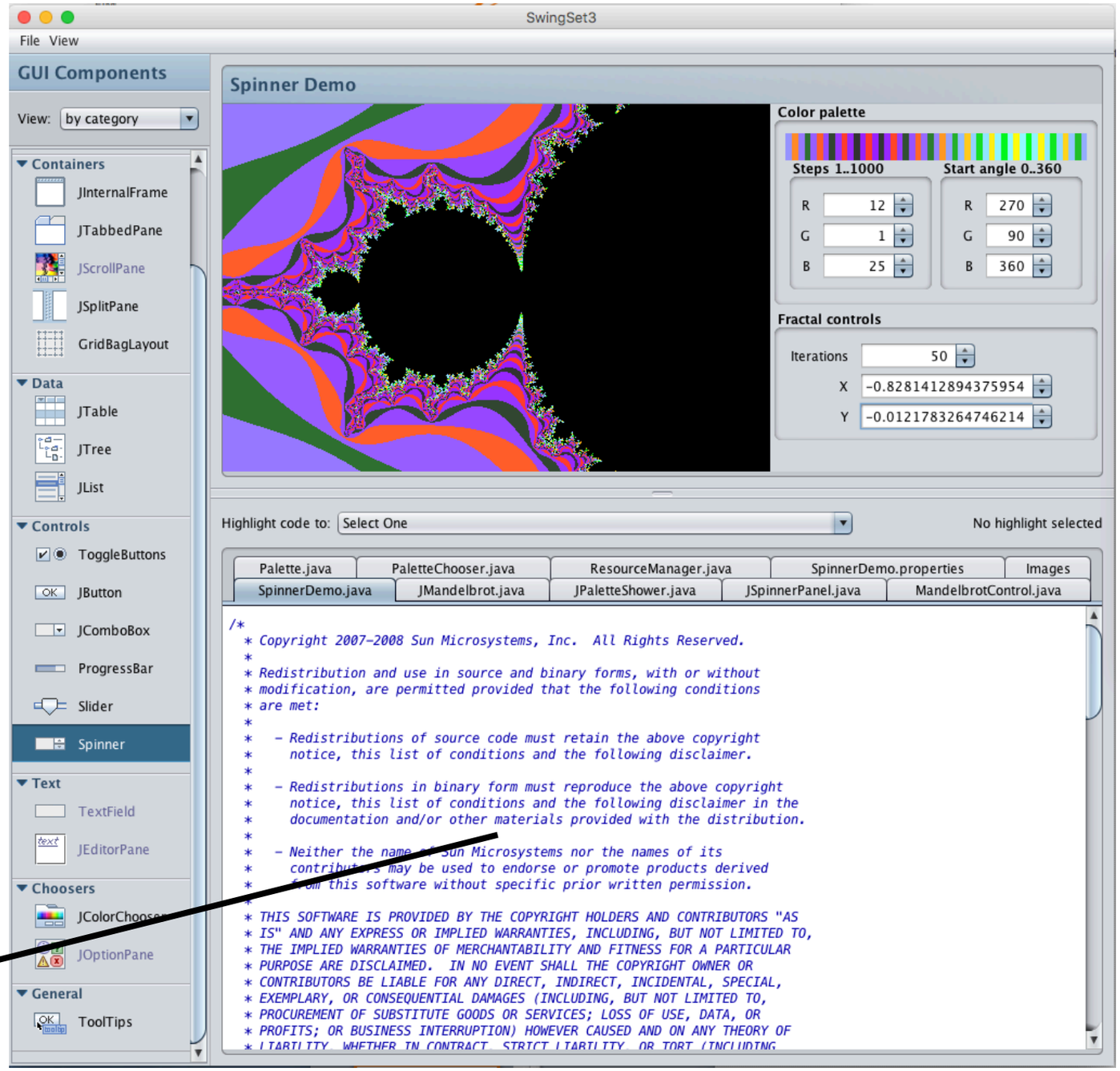


Will only allow values corresponding to the type set here, i.e. double values

The swingset3 demo

- Sliders, tooltips, trees
- Different types of pane – scrolling, tabbed, split
- Dialogs, file choosers, progress bars
- Internal frames
- See the SwingSet3 demo
- See the JavaDocs

Shows you the source code so that you can see how the component you need works



https://java.net/projects/swingset3/

http://mlapshin.com/index.php/projects/swingset3/