COM6516
Object Oriented Programming and Software Design

The contents of this module has been developed by Adam Funk, Kirill Bogdanov, Mark Stevenson, Richard Clayton and Heidi Christensen

6. Graphics programming

Aims

Provide an overview of Swing and the AWT, and show how to build simple GUIs in Java (more detail next week)

Objectives

…

6. Graphics programming

Outline

- …

Readings

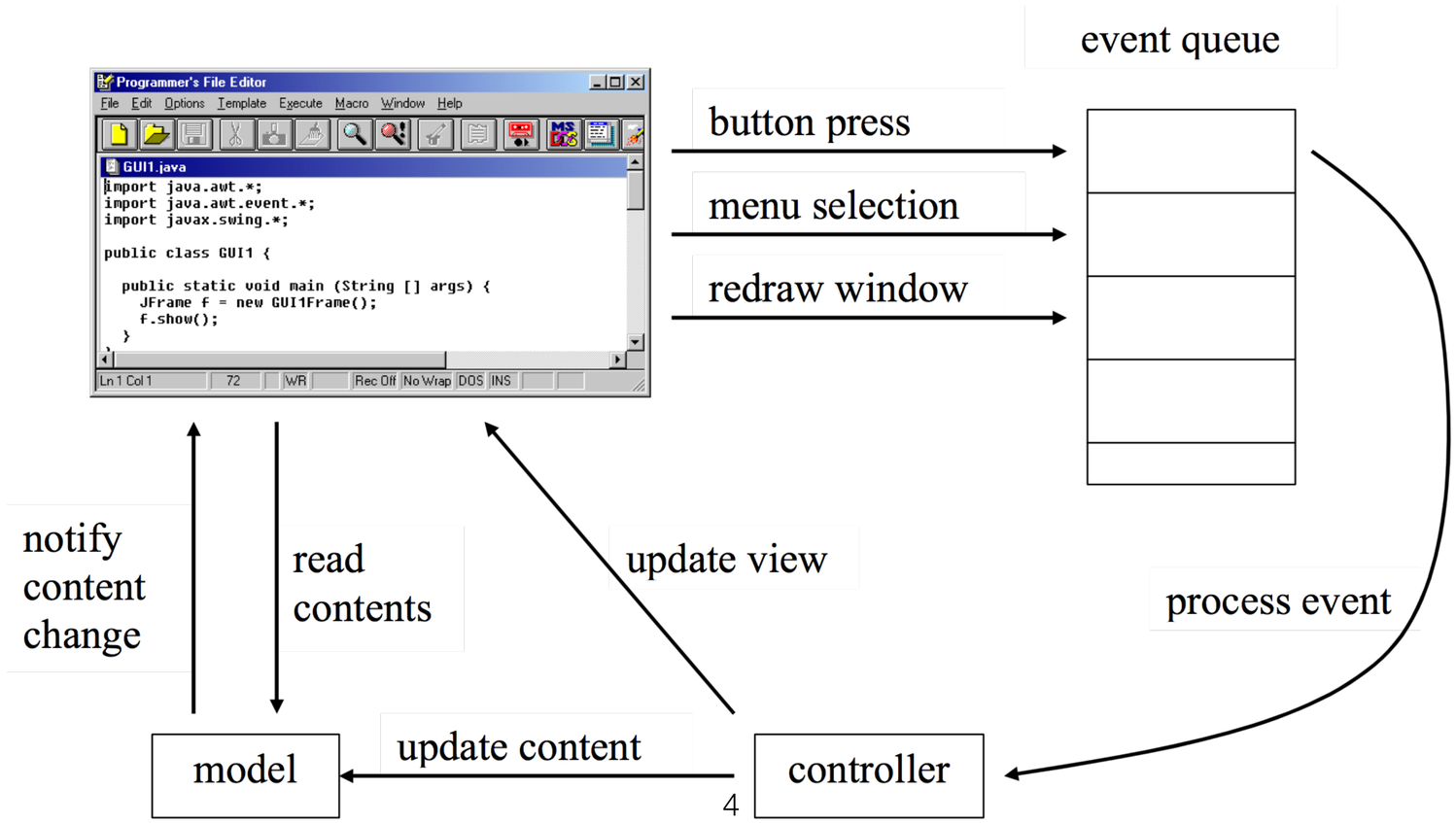
Core Java, vol.1, chapters 7,8, and 9

http://download.oracle.com/javase/tutorial/uiswing/components/index.html

http://download.oracle.com/javase/tutorial/2d/index.html

Introduction

- To create a Graphical User Interface
  - We need to put pieces (***components***) together as part of an interface, e.g. buttons and menus.
  - We need to respond to actions (***events***) initiated by the user or as part of the working of the system.



The Java Abstract Window Toolkit (AWT)

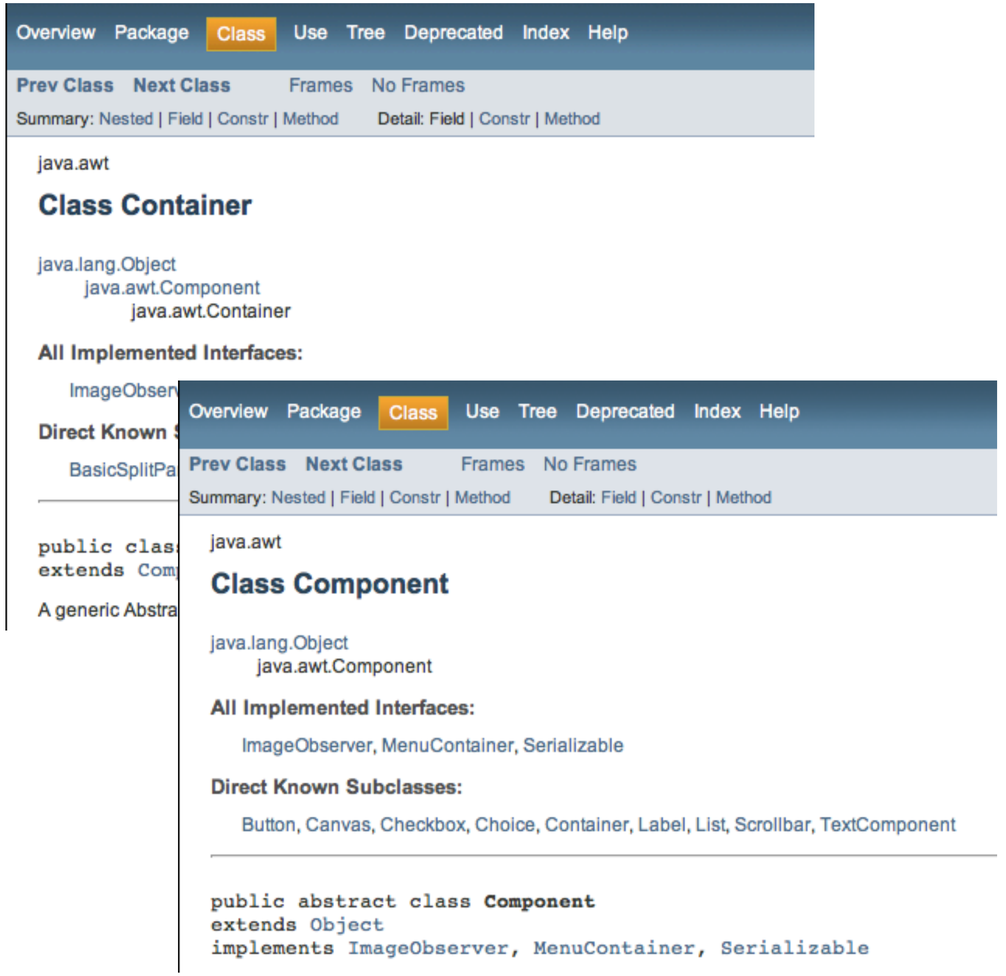
- In Java 1.1, GUIs were constructed using classes in the AWT
- AWT uses ‘native’ (*peer-based*) user-interface elements on the host platform, i.e. uses Windows components on Windows, X11 on Linux, etc.
- The AWT elements are called *heavyweight components*
- Disadvantages:
  - Different peer-based routines on different platforms;
  - Different *look-and-feel*s;
  - Different bugs on different platforms;
  - Lowest common denominator approach to GUI construction.

Swing

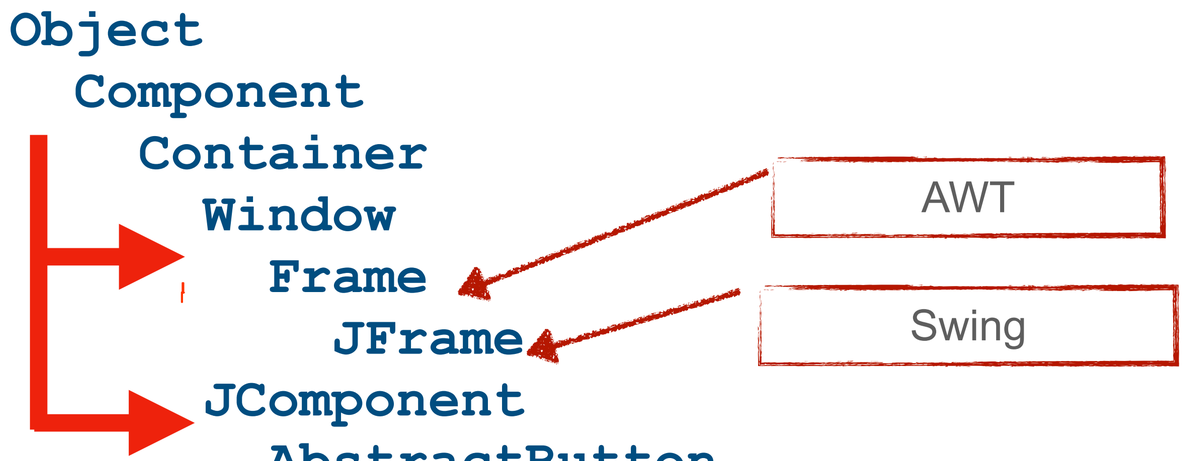
- From Java 1.2, the **Swing** classes can be used for GUI construction.
- Only rely on two platform specific routines:
  - Display a window;
  - Paint on a window.
- Components, e.g. buttons and menus, are *painted* onto blank windows.
- Called *lightweight components* because they are written completely in Java.
- Choice of look-and-feel: Windows, Motif and Metal.
- Benefits:
  - A better set of user interfaces;
  - Much less platform-dependence;
  - Consistent look-and-feel across platforms.
  - Disadvantage is that drawing a GUI is slower as Java does more work (so native code does less).

Components and containers

- See API documentation for details: http://download.oracle.com/javase/8/docs/api/
- AWT-based GUIs are built from **containers** such as `Frame` and **components** such as `Button`s
- Swing-based GUIs are built from ***containers*** such as `JFrame` and ***components*** such as `JButtons`.
- Using `JComponent`s means we can nest components, which gives greater flexibility when arranging interface components.
- The simplest GUI that we can build is to just display a single component on the screen

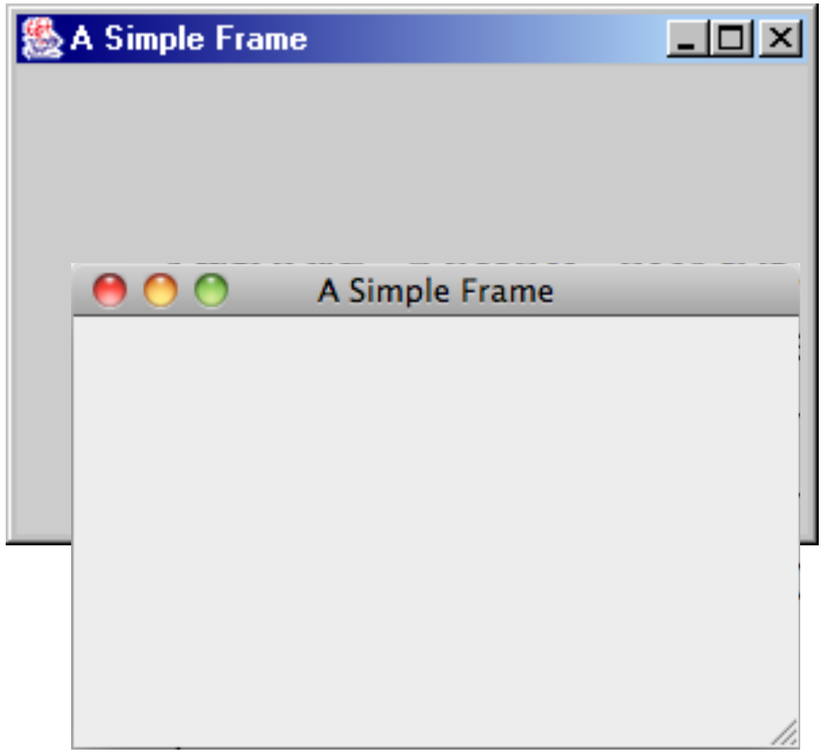


Components and containers



Example – SimpleJFrame.java

SimpleJFrame.java



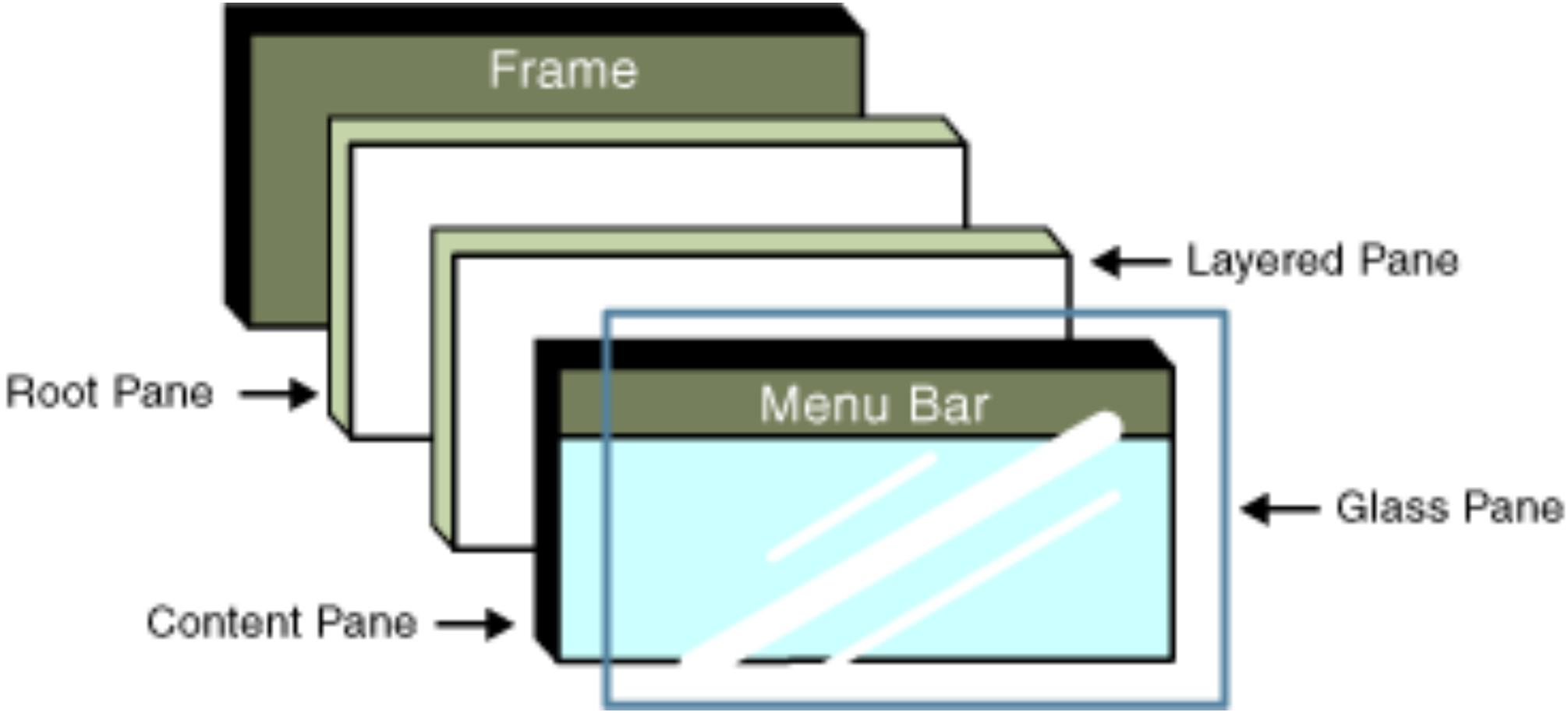

```
import javax.swing.*;
public class SimpleJFrame extends JFrame {
   public SimpleJFrame() {
      setTitle("A Simple Frame");
      setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
   }
   public static final int DEFAULT_WIDTH = 300;
   public static final int DEFAULT_HEIGHT = 200;

   public static void main (String[] args) {
      JFrame frm = new SimpleJFrame();
      frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frm.setVisible(true);
   }
}
```

Notes on SimpleJFrame

- A main method is embedded to enable class testing.
- A top-level window is called a *frame* in Java.
- In the AWT the frame class is called `Frame`; in Swing it is called `JFrame` (which extends the AWT classes `Window` and `Frame`).
- setVisible(…) must be called before the frame is displayed.
- setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE) causes the program to terminate when the window is closed.
- A `JFrame` is a container into which we can put other components, so can contain buttons, images, pull-down menus, etc.
- JFrame is an extension (via Window and Frame) of the generic (but not abstract) AWT superclass Container
- SimpleJFrame is a subclass of JFrame which is given a size and a title.
- JFrame inherits setTitle(…), setSize(…), and setVisible(…) (as well as other methods and fields) from its superclasses.

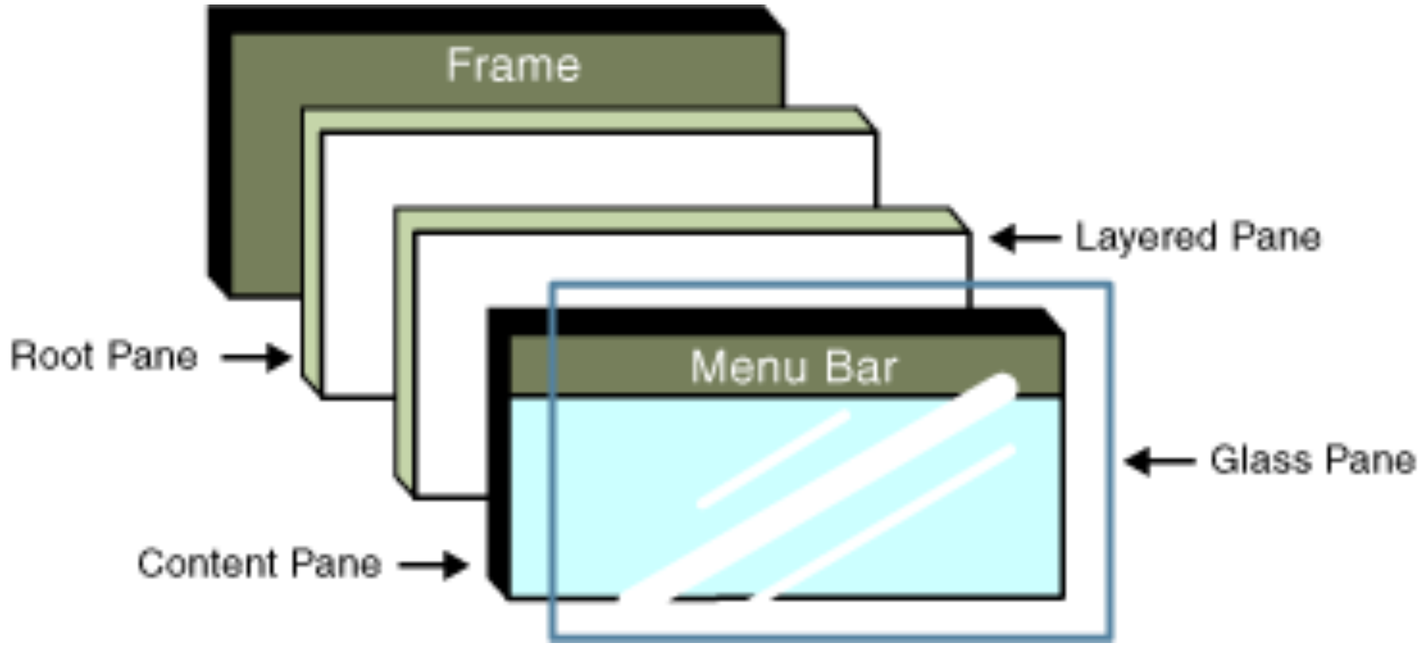
Content of a JFrame

- A JFrame has several layers: most are rather low level and used to implement the look and feel (menu bar, border, etc.).
- These layers are called panes. Content Pane is the most important.



Content of a JFrame

- A `JFrame` is intended to be a ***container*** for other objects including text, graphics (images, lines), and GUI controls (buttons, menus etc.)
- Usual approach is to draw on a panel (`JPanel`), which is then added to the `ContentPane` of the `JFrame`.



See http://download.oracle.com/javase/tutorial/uiswing/components/toplevel.html

Content of a JFrame

- Panels are containers that extend `JPanel` (in the same way that `SimpleFrame` extends `JFrame`).
- The `ContentPane` is returned by the `getContentPane()` method of `JFrame`:

```
Container contentPane = myFrame.getContentPane();
JComponent c = . . .
contentPane.add(c);
```

- Container is part of the AWT, so code using Container needs to import java.awt.* as well as javax.swing.*

SimpleJFrame2

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  public class SimpleJFrame2 extends JFrame {
5      public SimpleJFrame2() {
6          setTitle("A simple JFrame with content");
7          setSize(400, 150);
8          Container contentPane = this.getContentPane();
9
10         // create a drawing that is a MyPanel object
11         myDrawing = new MyPanel();
12         contentPane.add(myDrawing);
13     }
14
15     private MyPanel myDrawing;
16
17     public static void main (String[] args) {
18         JFrame frm = new SimpleJFrame2();
19         frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
20         frm.setVisible(true);
21     }
22
23 }
```

MyPanel extends JPanel

A simple panel

```
public class MyPanel extends JPanel {
   public MyPanel() {
      setBackground(Color.white);
   }

   public void paintComponent(Graphics g) {
      super.paintComponent(g);
      g.drawLine(20,20, 120,70);

   }
}
```

- To draw anything on a `Panel` we need to create a class that extends `JPanel`, and we need to provide a `paintComponent` method
- paintComponent is automatically called every time the window needs to be redrawn.
- super.paintComponent is in JComponent.

See http://download.oracle.com/javase/8/docs/api/java/awt/Graphics.html for other methods including drawImage(), drawOval(), drawPolygon() etc.

A simple panel

```
public class MyPanel extends JPanel {
   public MyPanel() {
      setBackground(Color.white);
   }

   public void paintComponent(Graphics g) {
      super.paintComponent(g);
      g.drawLine(20,20, 120,70);

   }
}
```

The Graphics object g that is passed to paintComponent is an object that is part of the AWT (java.awt.Graphics).

It is automatically received, and provides a collection of attributes that allow us to display items on our JPanel.

See http://download.oracle.com/javase/8/docs/api/java/awt/Graphics.html for other methods including drawImage(), drawOval(), drawPolygon() etc.

Graphics2D

Graphics2D extends Graphics to “provide more sophisticated control over geometry, coordinate transformations, color management, and text layout” A Graphics2D object can be easily obtained by casting the Graphics object g that is part of JPanel.

```
public void paintComponent(Graphics g) {
    //super.paintComponent(g);

    Graphics2D g2 = (Graphics2D) g;
```

Graphics2D

```
public void paintComponent(Graphics g) {
    //super.paintComponent(g);

    Graphics2D g2 = (Graphics2D) g;
    g2.setStroke(new BasicStroke(5));

    // use red to render text
    g2.setPaint(Color.green);

    // set the Font attribute
    g2.setFont(new Font("Papyrus", Font.PLAIN, 48));

    // make text and shapes appear
    g2.setRenderingHint(
            RenderingHints.KEY_ANTIALIASING,          // smoother
            RenderingHints.VALUE_ANTIALIAS_ON );

    // Prepare graphics object to render
    String text = "Hello COM6516!";
    g2.drawString(text, 40, 80);
}
```

Graphics2D provides many, many methods to have fun with, see http://docs.oracle.com/javase/tutorial/2d/index.html

-- http://download.oracle.com/javase/8/docs/api/java/awt/Graphics2D.html

Repainting

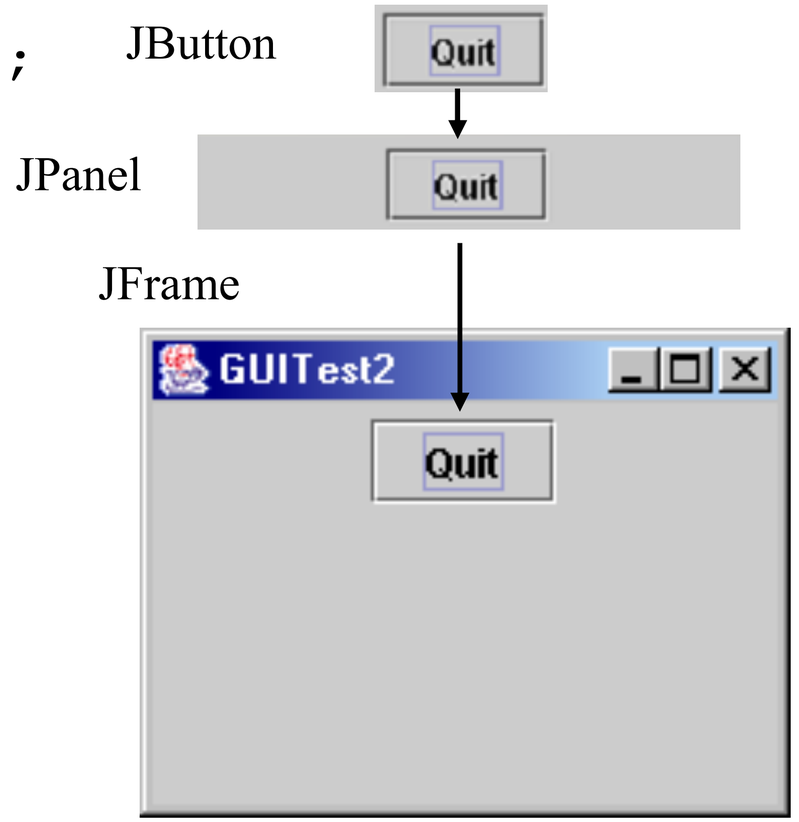
- When you would like to change what is being displayed:
  - change a model and request the view to refresh, or
  - invalidate the content of the frame. This will eventually cause a repaint method to be called.
- Your *paint* method (or *paintComponent*) can be called at any time if your window is covered by another one and subsequently made visible again.

Adding interface components to JPanel

In Swing, user interface components are added to the `JFrame`'s *content pane* just like a drawing.

JPanels act as containers for other components and are added to a JFrame

```
JButton quitButton = new JButton("Quit");
...
Container contentPane = frame.getContentPane();
JPanel p = new JPanel();
p.add(quitButton);
contentPane.add(p);
```



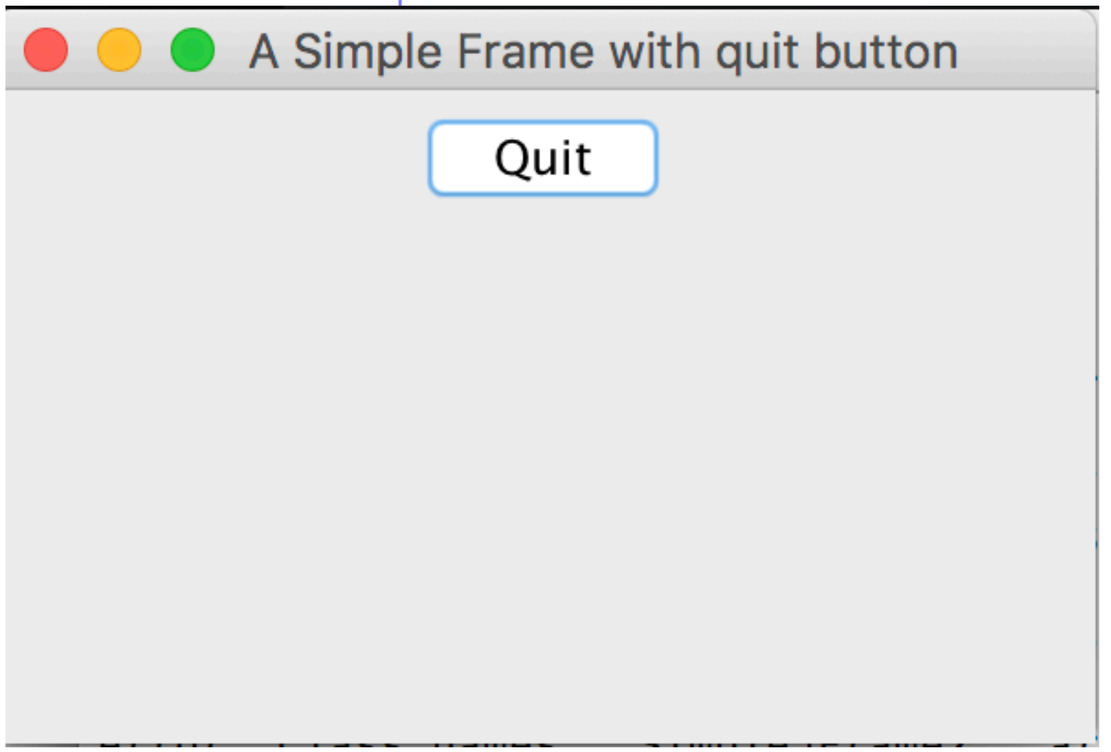
Panels are one of the most useful interface components, since you can draw on them and they can act as containers for other components.

Getting interfaces to do something (SimpleFrameWithQuitButton)

```
public class SimpleFrameWithQuitButton extends JFrame implements ActionListener {
    public SimpleFrameWithQuitButton() {
        setTitle("A Simple Frame with quit button");
        setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
        Container contentPane = this.getContentPane();
        JPanel p = new JPanel();
        quitButton = new JButton("Quit");
        quitButton.addActionListener(this);
        p.add(quitButton);
        contentPane.add(p);
    }


    public void actionPerformed(ActionEvent event) {
        Object source = event.getSource();
        // return source object of the event
        if (source == quitButton) {
            System.exit(0);
        }
    }
```

Components that do something (e.g. a Button), must be linked to an action.

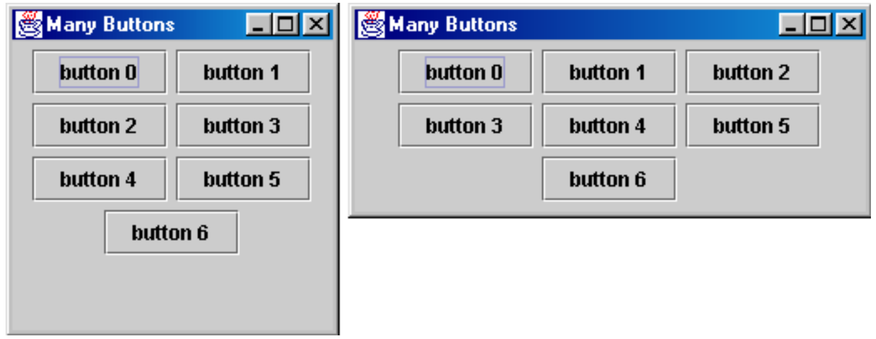
More on this next week.



Layout Managers

Layout Managers

- There are also IDE tools that can do layout management, but you need to know how to do it by writing Java code.
- A *layout manager* controls arrangement of `JComponent`s within a `Container`.
- Layout manager classes conform to the `LayoutManager` interface.
- The default layout manager for a `JPanel` is `FlowLayout`, with the `JComponent`s aligned in a row in the centre of the `JPanel`.
- If the frame is resized, or more buttons are added, the buttons stay centred – it is the *layout manager* that controls this dynamic behaviour.
- Container.setLayout(LayoutManager mgr); is used to change the layout manager.



Other layout managers

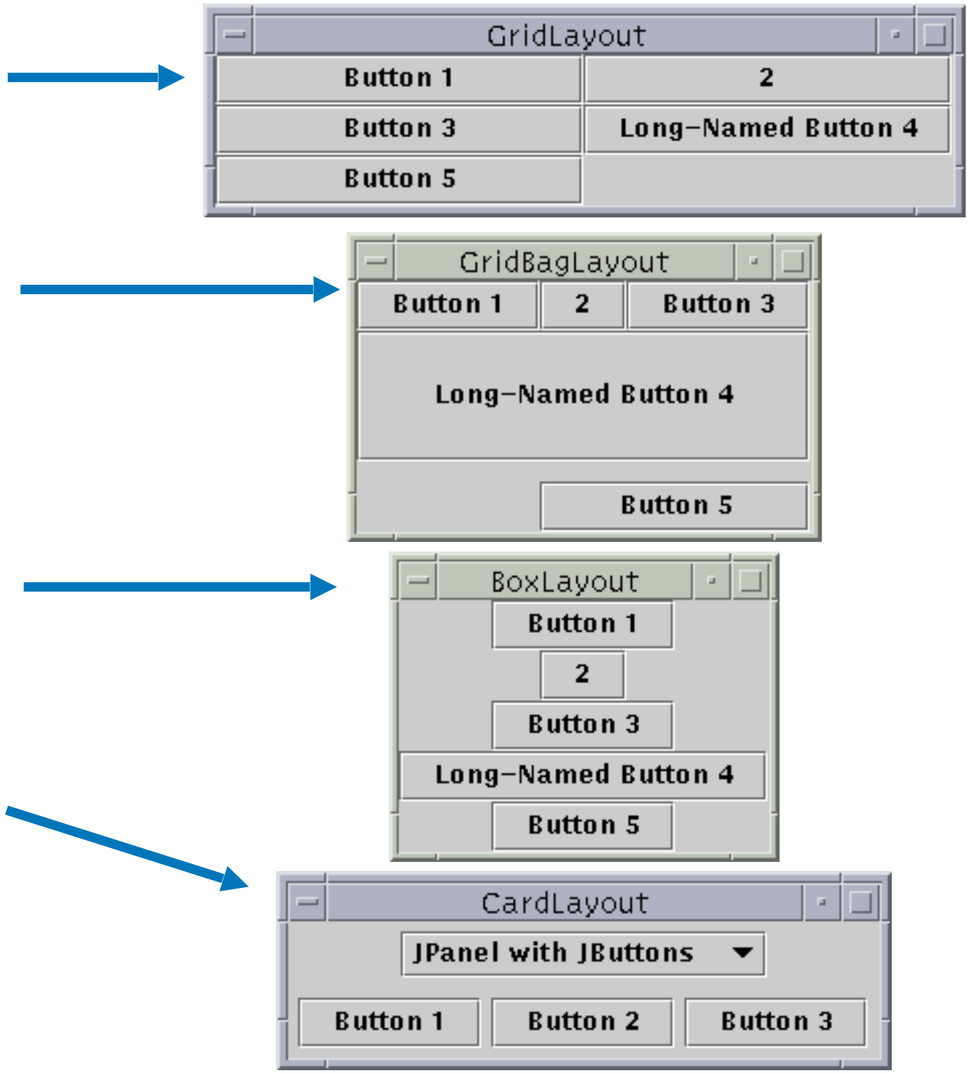
`BorderLayout` – see later

`GridLayout` – aligns components in a rectangular grid of equal-sized spaces.

`GridBagLayout` – aligns components vertically and horizontally, without requiring that the components be of the same size.

`BoxLayout` – puts components in a single row or column.

`CardLayout` – lets you implement an area that contains different components at different times.



Other layout managers

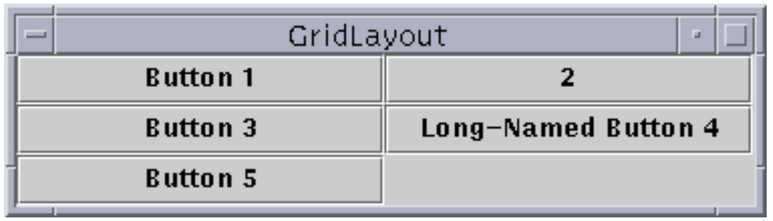
`BorderLayout` – see later

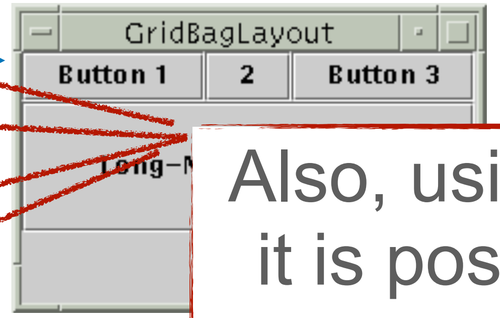
`GridLayout` – aligns components in a rectangular grid of equal-sized spaces.

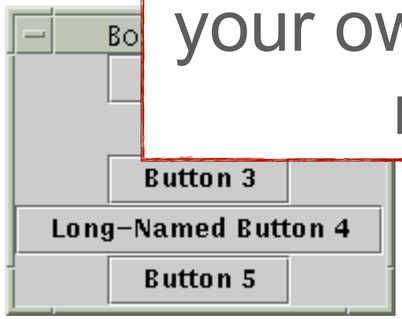
`GridBagLayout` – aligns components vertically and horizontally, without requiring that the components be of the same size.

`BoxLayout` – puts components in a single row or column.

`CardLayout` – lets you implement an area that contains different components at different times.

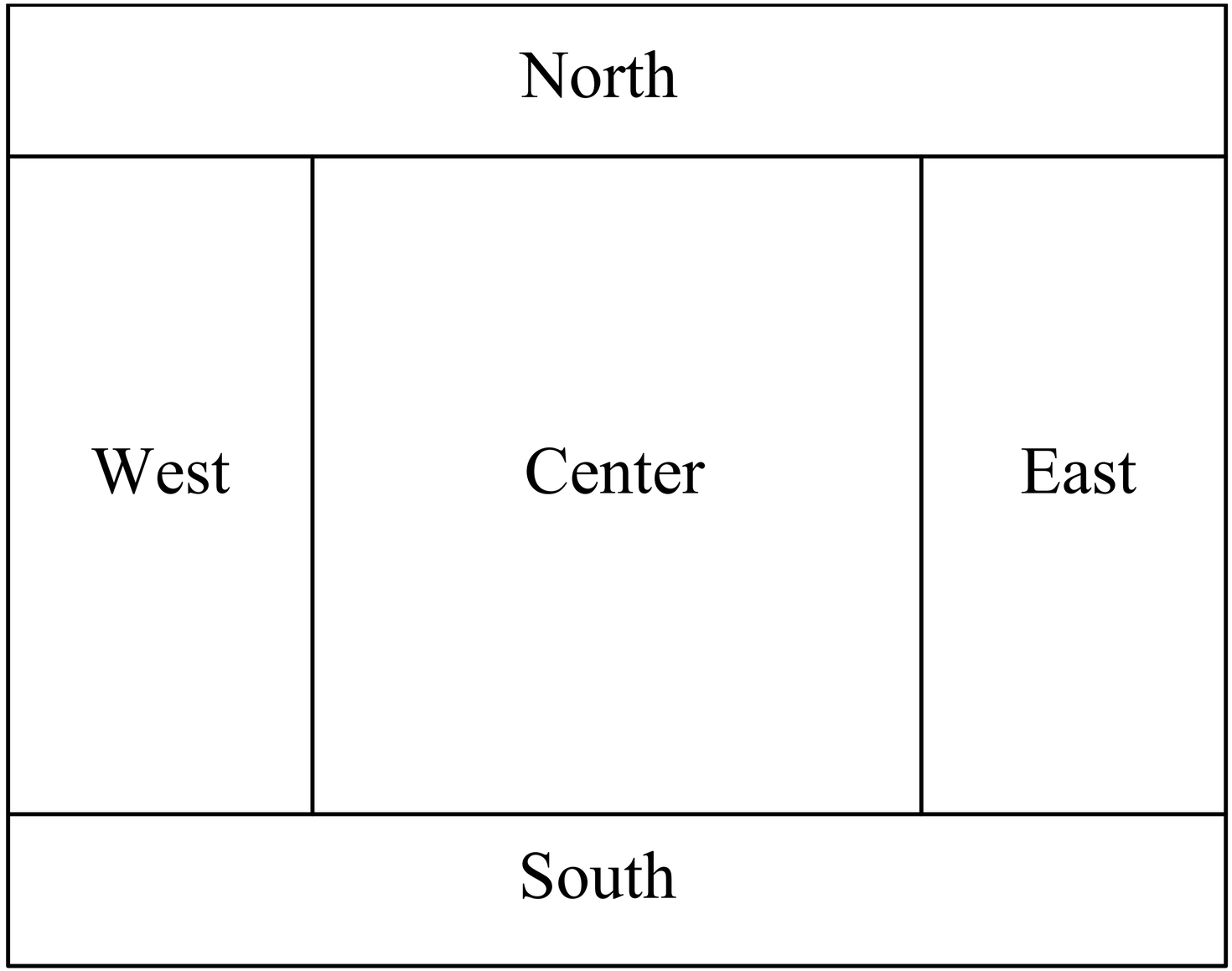






Also, using inheritance, it is possible to define your own custom layout managers

BorderLayout



BorderLayout

`BorderLayout` is the default layout manager for the content pane of a `JFrame`.

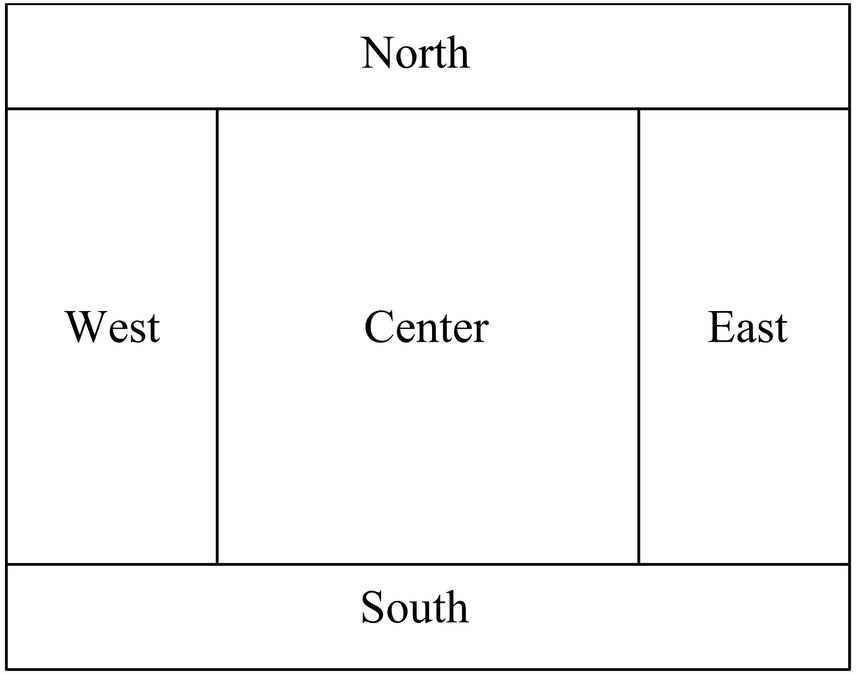
Position to add components can be controlled:

```
Container contentPane = getContentPane();
JPanel p = new JPanel();
   ...
contentPane.add(p, BorderLayout.SOUTH);
```

Default position for adding components is “Center” or `BorderLayout.CENTER`

Edge components are laid out first, then “Center”
Components grow to fill up available space.

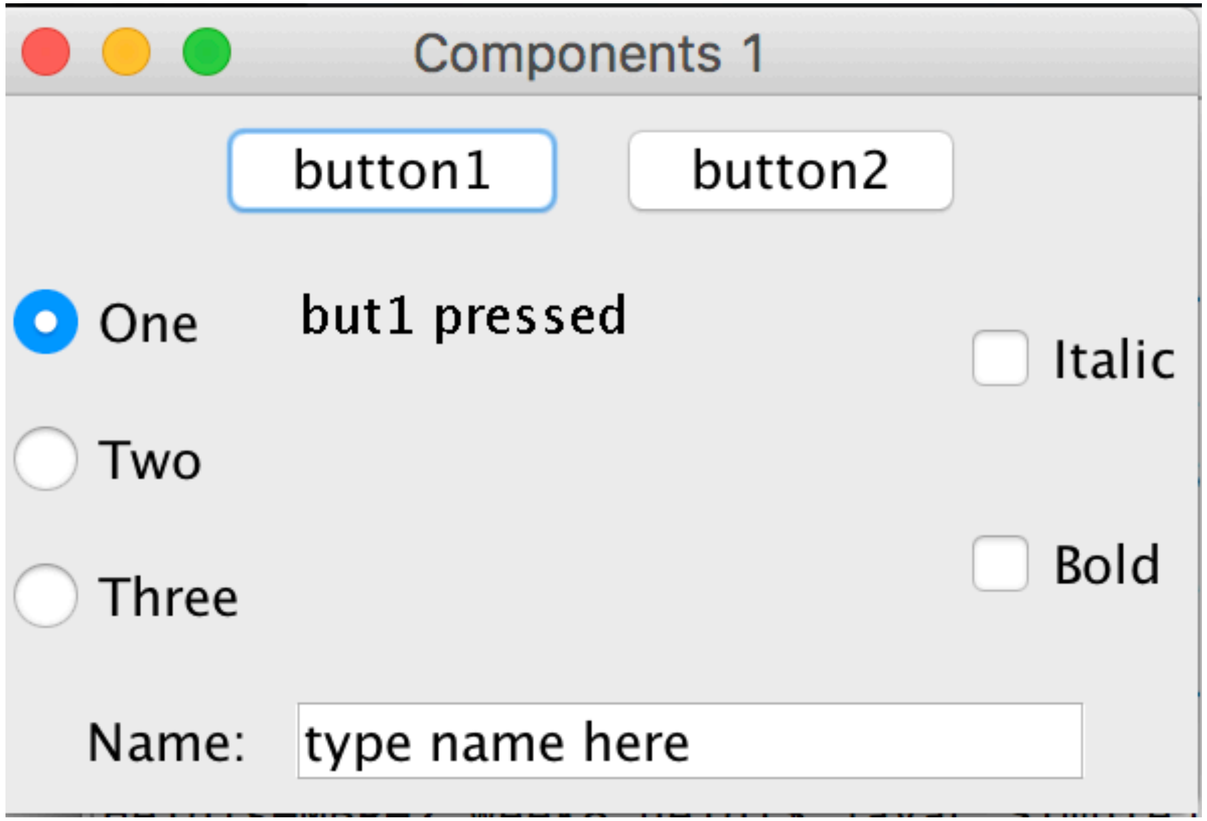
To add multiple buttons to the “South” of the frame, we would first add them to a panel using `FlowLayout` and then add the panel to the “South” of the frame.



Creating components and adding to a `JPanel`

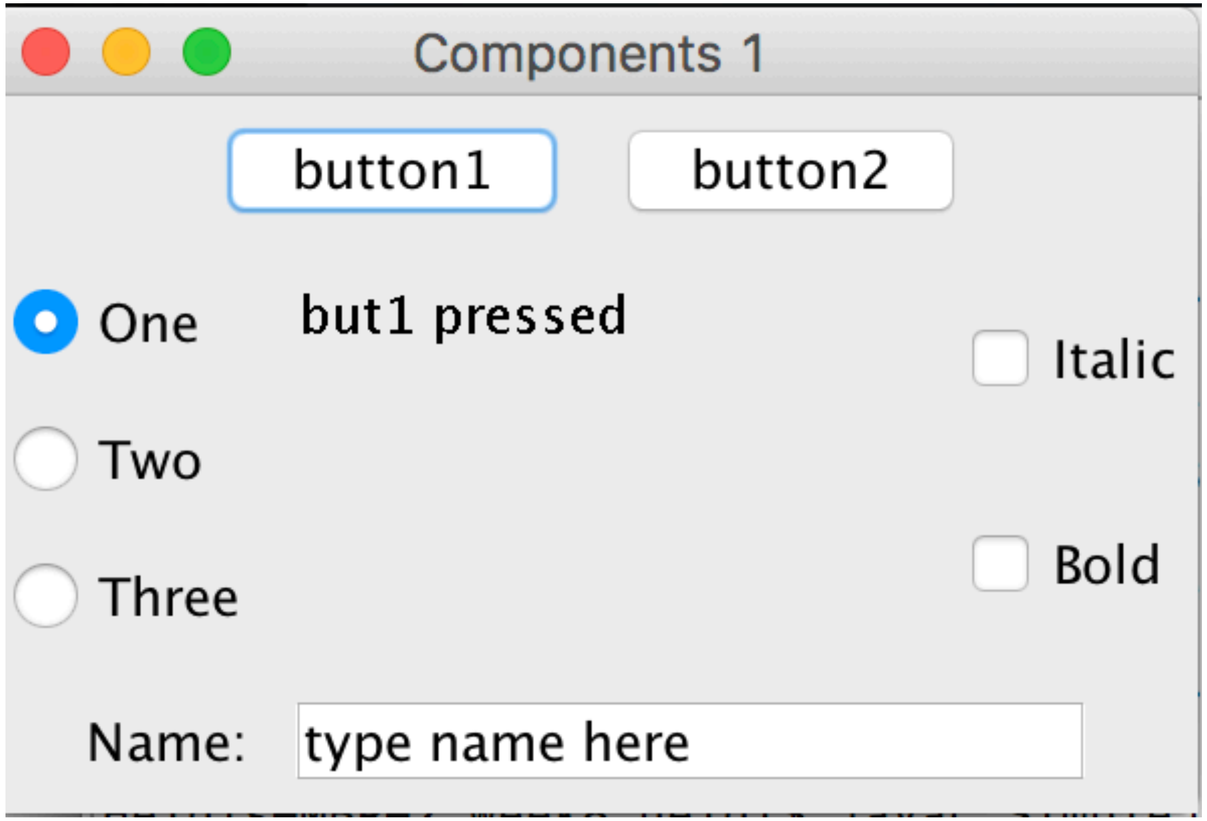
Example components include:

- `JLabel` – a line of text or/and an icon
- `JTextField` – input a line of text
- `JCheckBox` – for yes/no (on/off) choices
- `JRadioButton` – a set of alternatives, only one of which may be on
- `JMenu` – pull-down hierarchical menus
- ….others – see the Java API



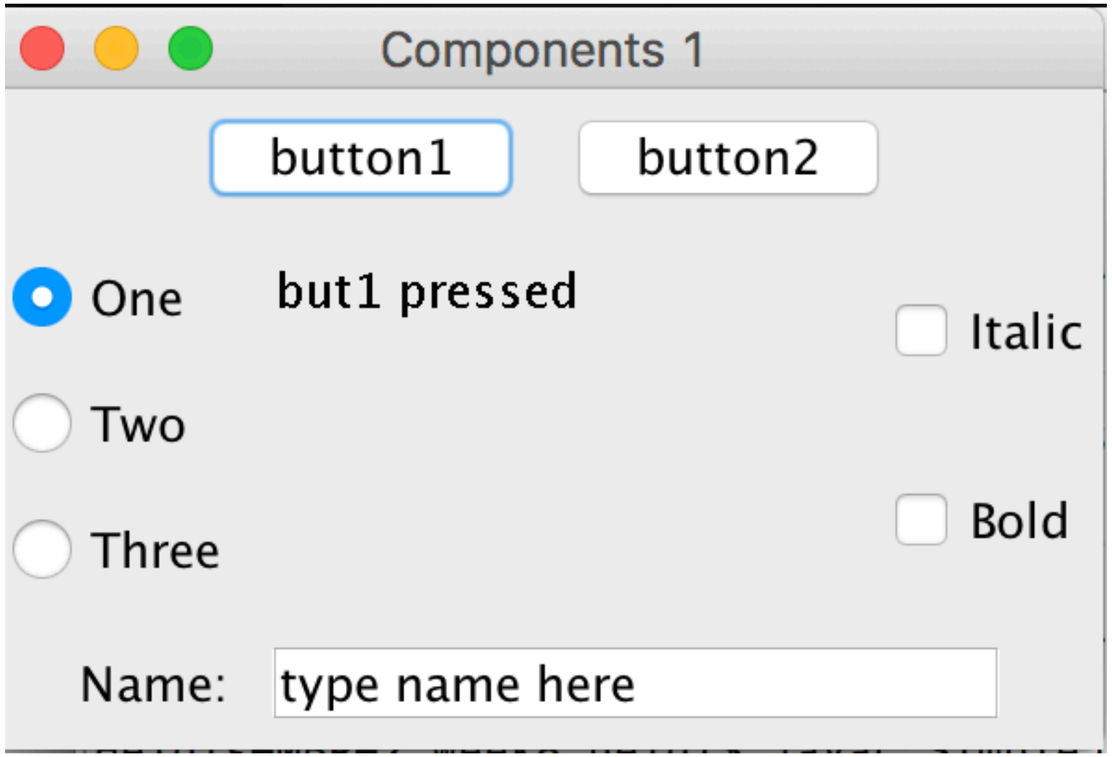
Creating components and adding to a `JPanel`

```
class ComponentFrame1 extends JFrame implements ActionListener {

    private JButton but1, but2;
    private JLabel label;
    private JTextField text;
    private JCheckBox checkItalic, checkBold;
    private JRadioButton orOne, orTwo, orThree;
    private PicturePanel picPanel;

    public ComponentFrame1() {
        setTitle("Components 1");
        setSize(300, 200);
```

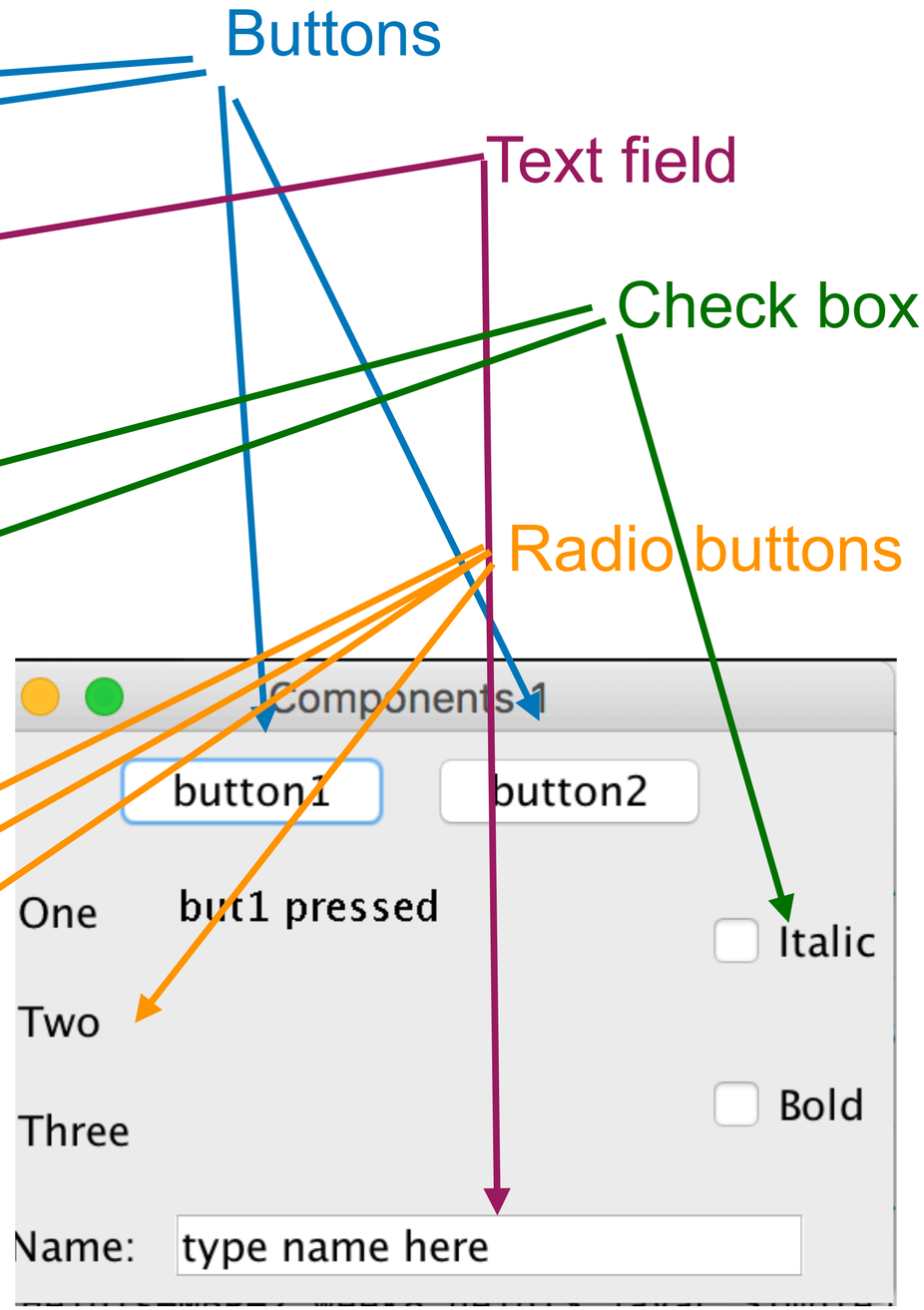


Adding JButtons

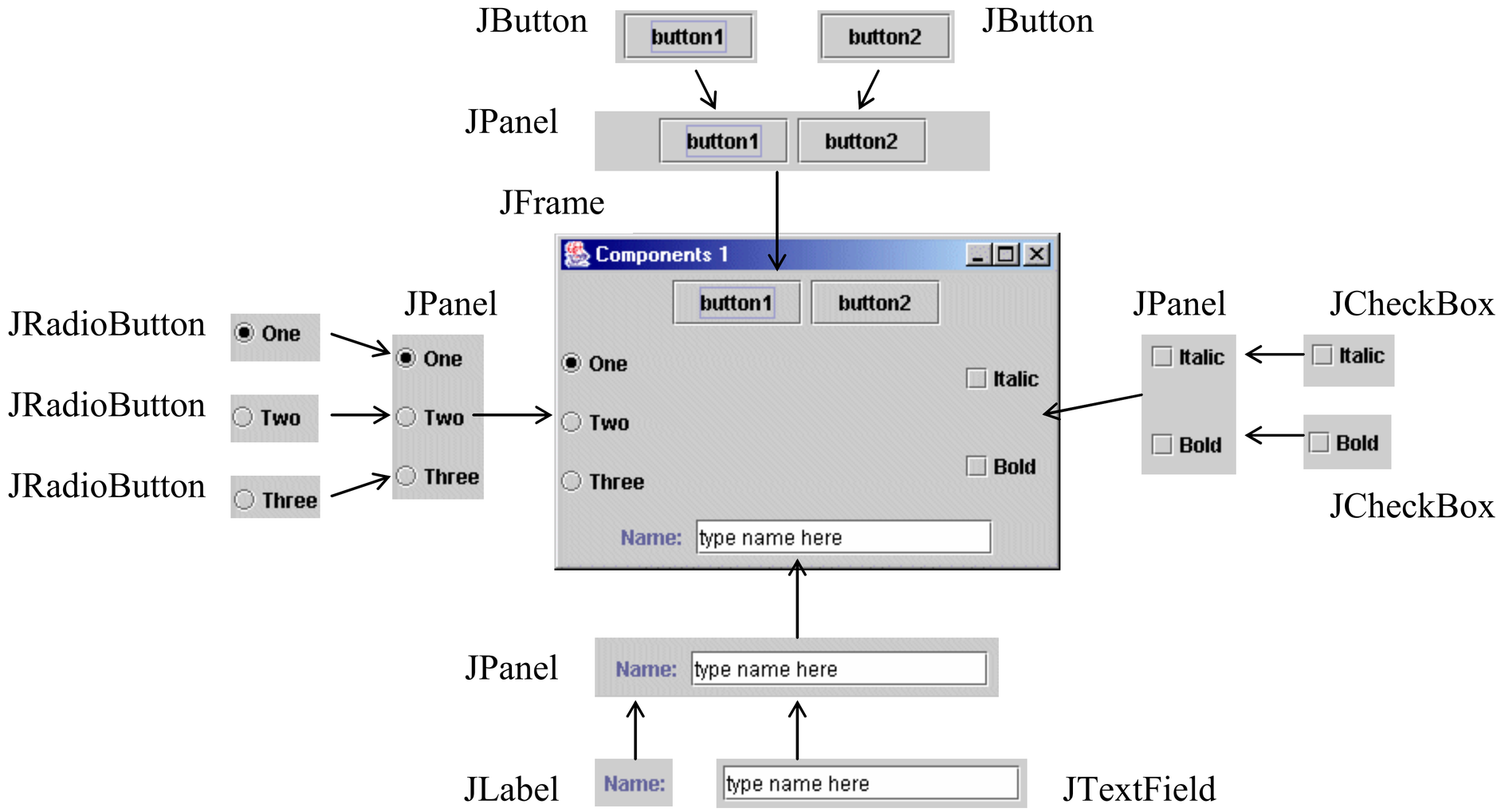
```
class ComponentFrame1 extends JFrame implements ActionListener {

    private JButton but1, but2;

    Container contentPane = getContentPane();

    // add the components to contentPane using BorderLayout
    JPanel p = new JPanel();
    but1 = addJButton(p, "button1", this);
    but2 = addJButton(p, "button2", this);
    contentPane.add(p, "North");
```



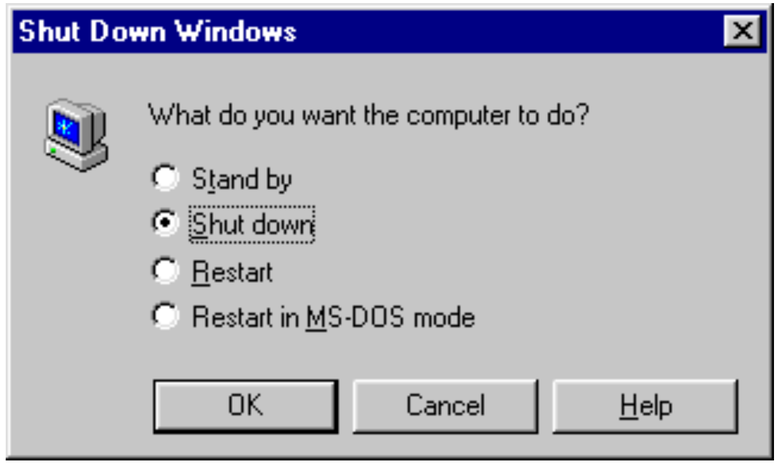
```
    Container contentPane = getContentPane();

    // add the components to contentPane using BorderLayout
    JPanel p = new JPanel();
    but1 = addJButton(p, "button1", this);
    but2 = addJButton(p, "button2", this);
    contentPane.add(p, "North");

    p = new JPanel();
    label = new JLabel("Name: ");
    p.add(label);
    text = new JTextField("type name here", 16);
    p.add(text);
    text.addActionListener(this);
    contentPane.add(p, "South");

    p = new JPanel(new GridLayout(2,1));
    checkItalic = addJCheckBox(p, "Italic", this);
    checkBold = addJCheckBox(p, "Bold", this);
    contentPane.add(p, "East");

    p = new JPanel(new GridLayout(3,1));
    ButtonGroup group = new ButtonGroup();
    orOne =   addJRadioButton(p, "One", true, group, this);
    orTwo =   addJRadioButton(p, "Two", false, group, this);
    orThree = addJRadioButton(p, "Three", false, group, this);
    contentPane.add(p, "West");

    // picPanel is used to display actions associated with components
    picPanel = new PicturePanel();
    contentPane.add(picPanel, "Center");
}
```



Example: GUIComponents1.java



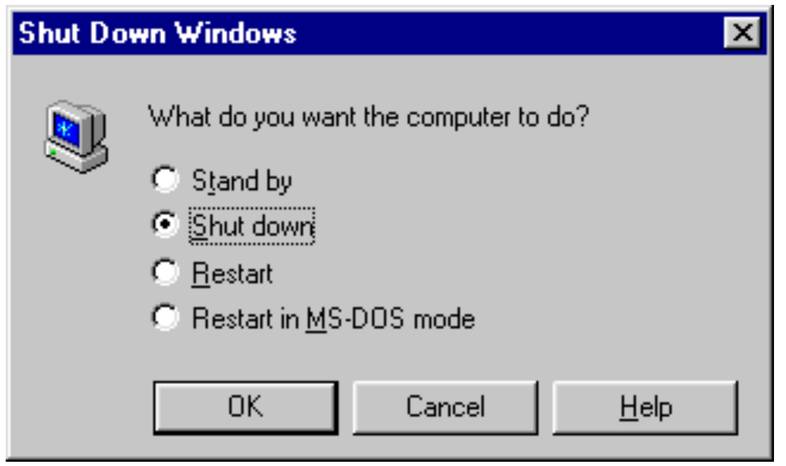
“Radio buttons”?

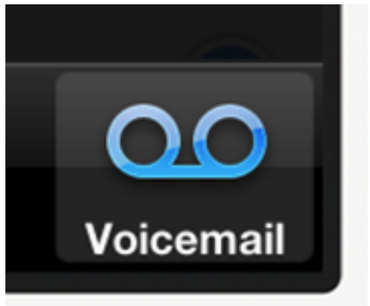


- `JRadioButton` – a set of alternatives, only one of which may be on

“Radio buttons”?

- If you like that, look up “computer icons that don’t make sense any more”





Summary

AWT is limited to a few primitive elements supported by all native user interfaces. Swing minimises the use of native methods → portable, rich GUI.

The `JFrame`'s content pane is the container for other interface components

A `JPanel` is a component that can act as a container, and it can be drawn on.

LayoutManagers control where components are placed in a container.

There are many kinds of components …