ETH Zürich
Chair of Software Engineering

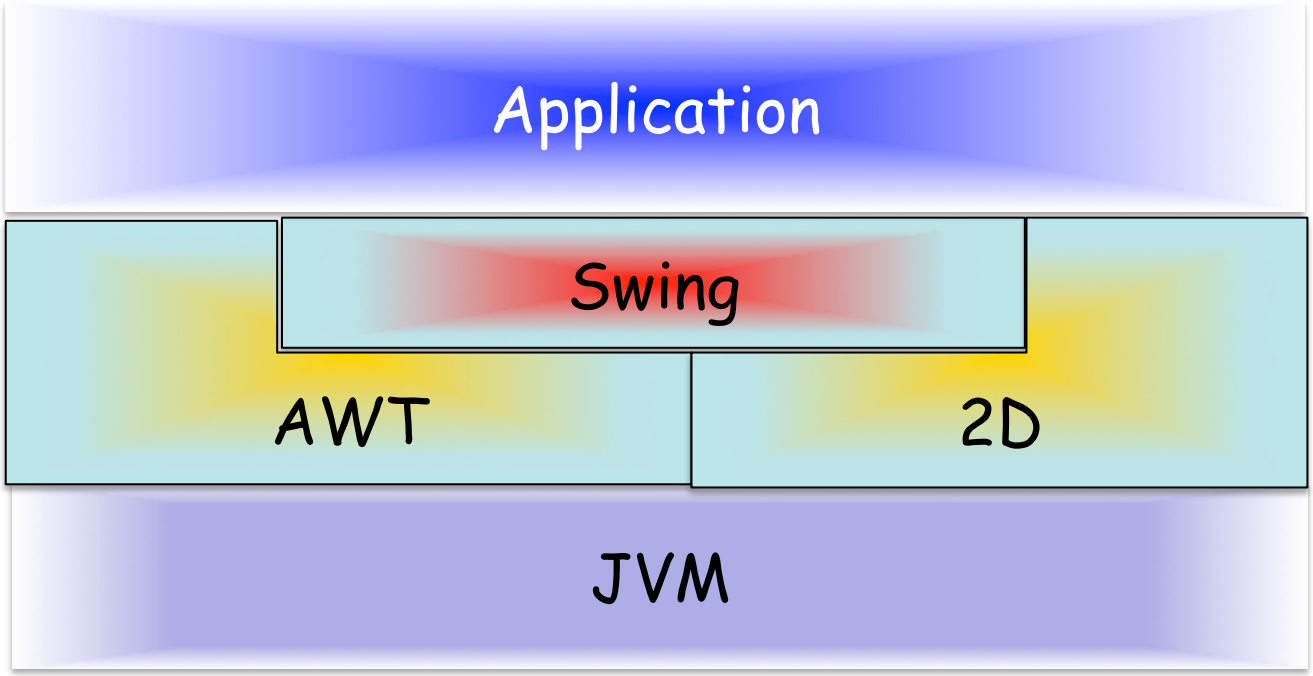
Java and C# in depth

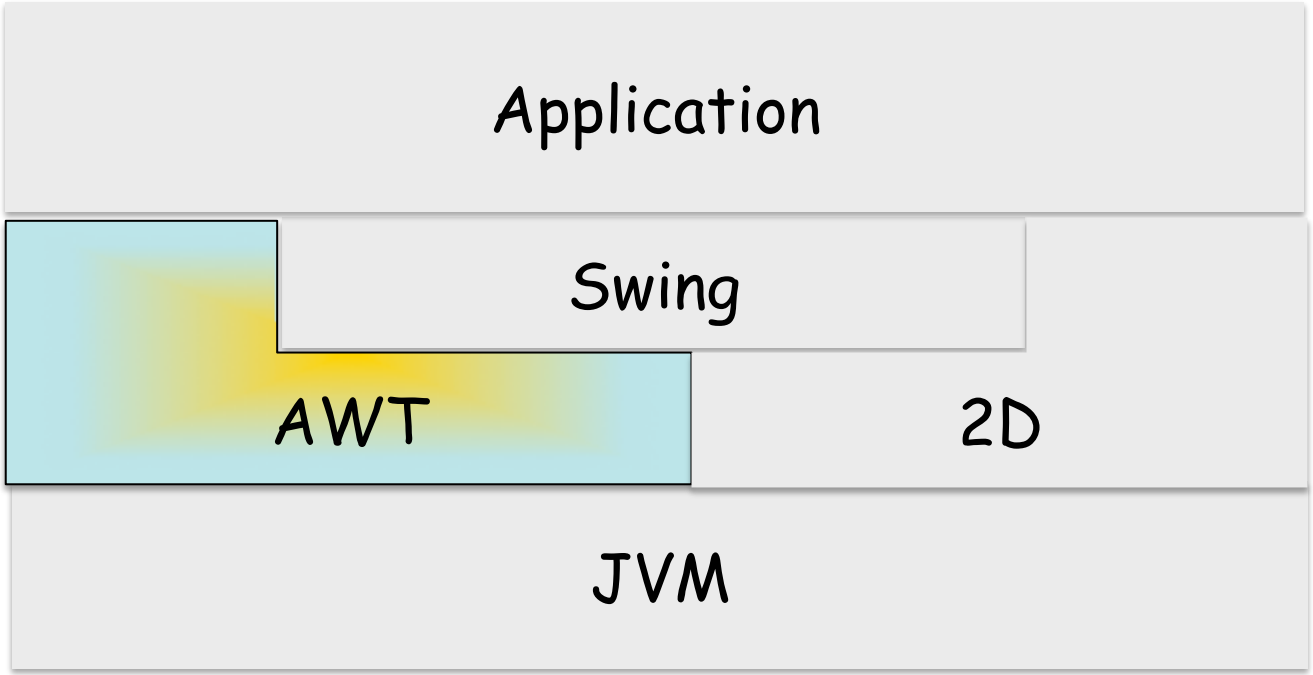
Carlo A. Furia, Marco Piccioni, and Bertrand Meyer

Java: Graphical User Interfaces (GUI)

With material from Christoph Angerer

The essence of the Java Graphics API

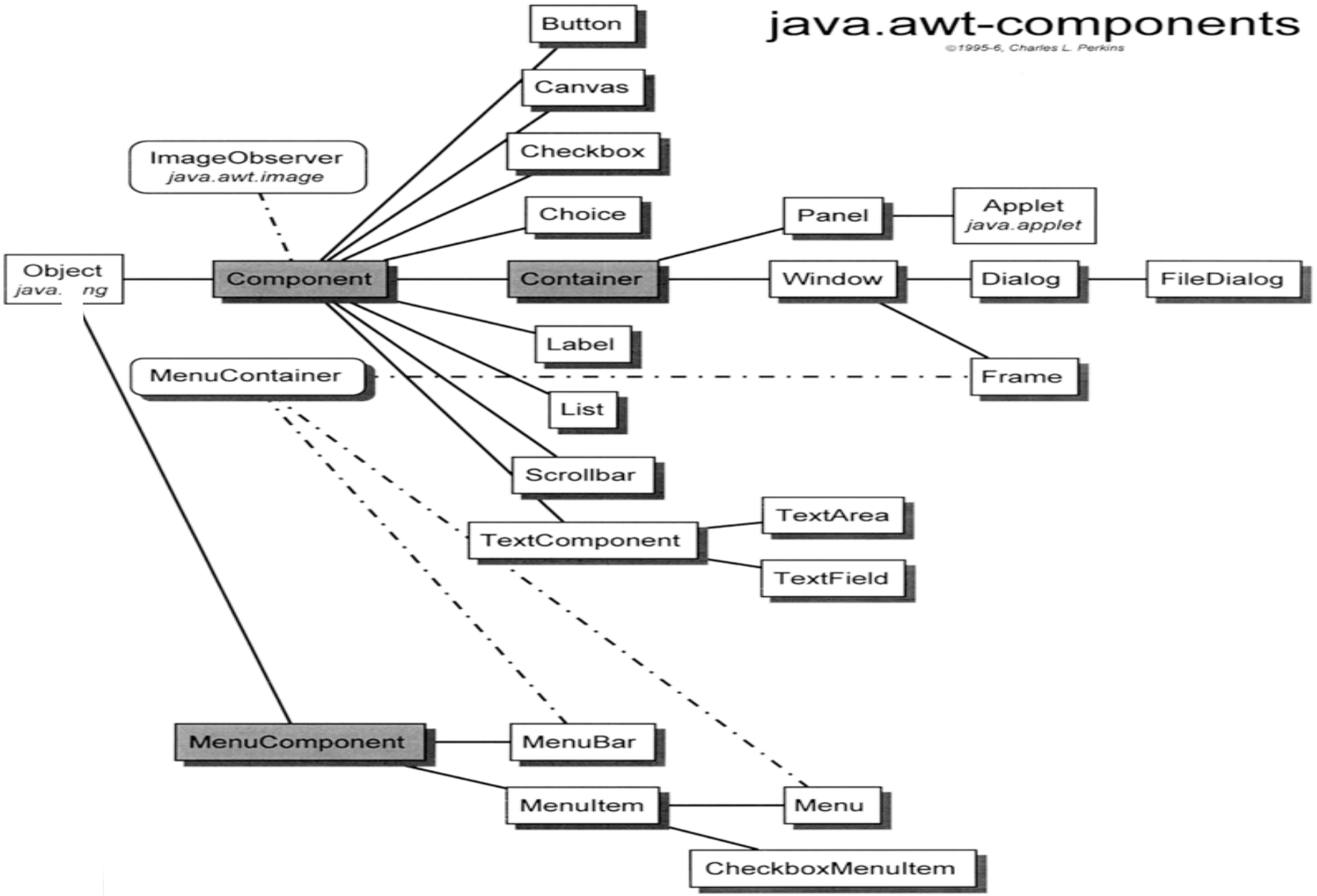


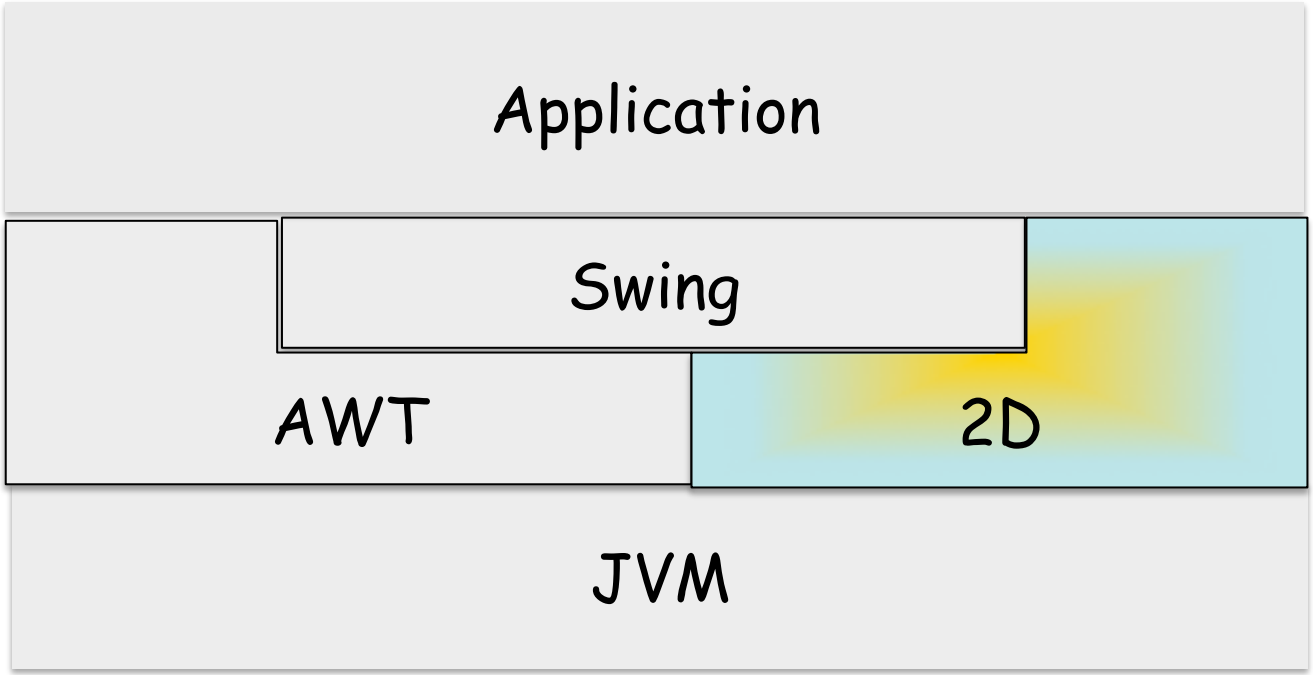

Application
Swing
AWT
2D
JVM

Abstract Windowing Toolkit (AWT)

- The first available API for Java GUIs
- Platform independent (if there are JVMs)
- It does not look the same on all platforms
- Calls native libraries on the user system
- Handles events, cut and paste, drag and drop, keyboard focus, user input
- Still used directly for top-level containers (frames, applets, dialogs)

AWT Components




Application
Swing
AWT
2D
JVM

Java 2D

Introduced in 1.2

Basic and advanced drawing operations

Image manipulation

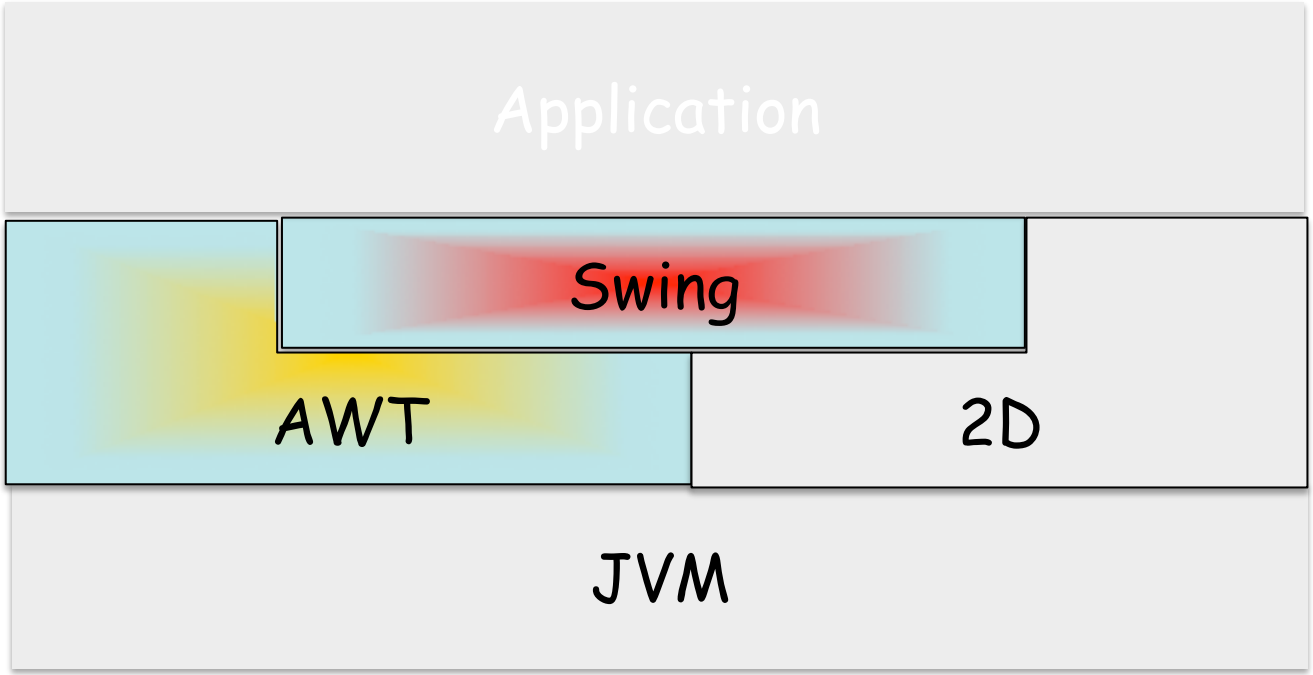
Text

Printing

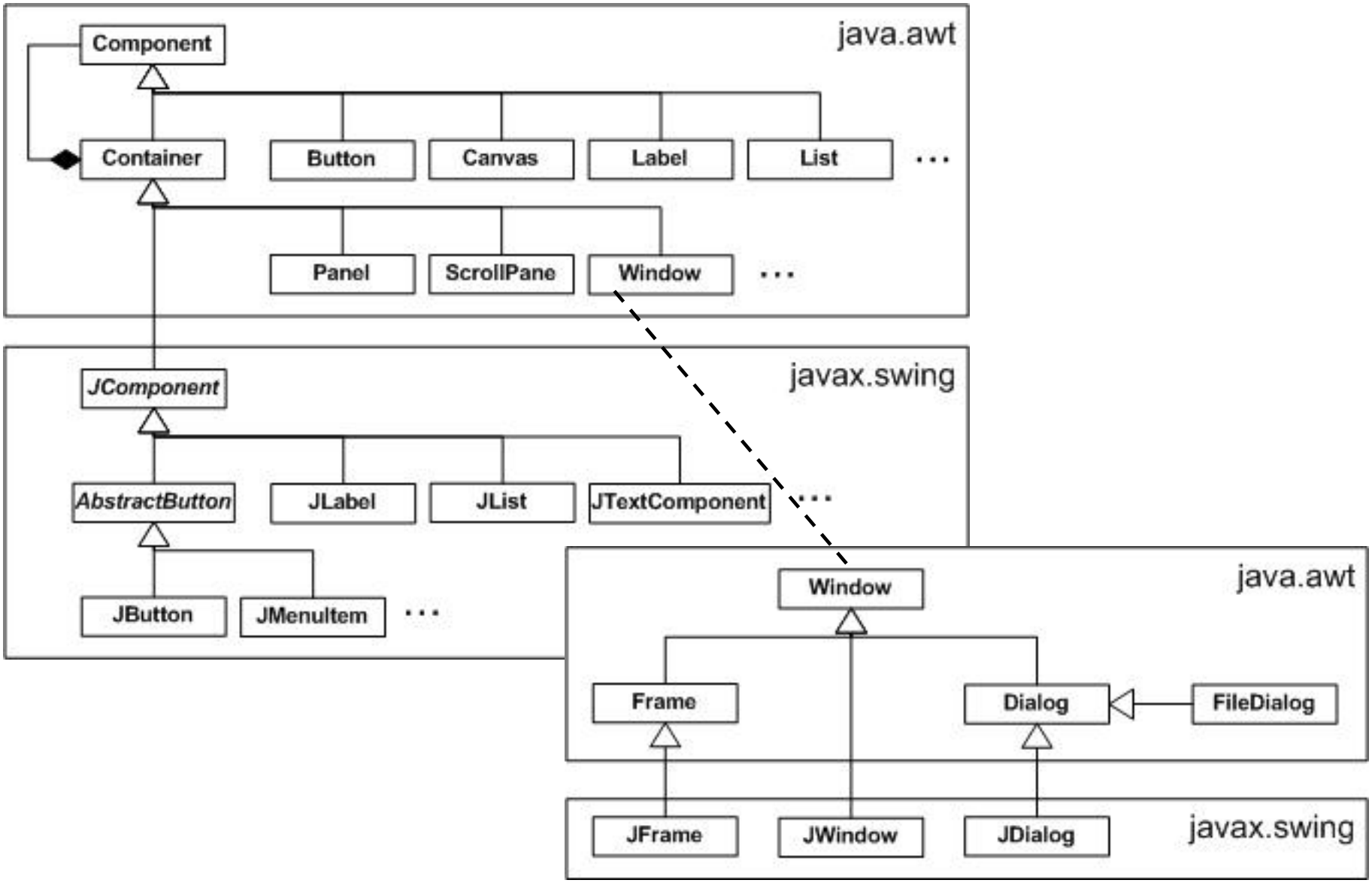
Can be used directly, or indirectly via Swing

Handles Swing's rendering operations (e.g. GUI component background and border)

The essence of the Java Graphics API



Swing and AWT



Java Swing

Introduced in 1.2

Main Java GUI library for desktop apps

Lightweight

- relies on 2D for graphics
- relies on AWT for top-level containers and application behavior via event management

Swing Concepts

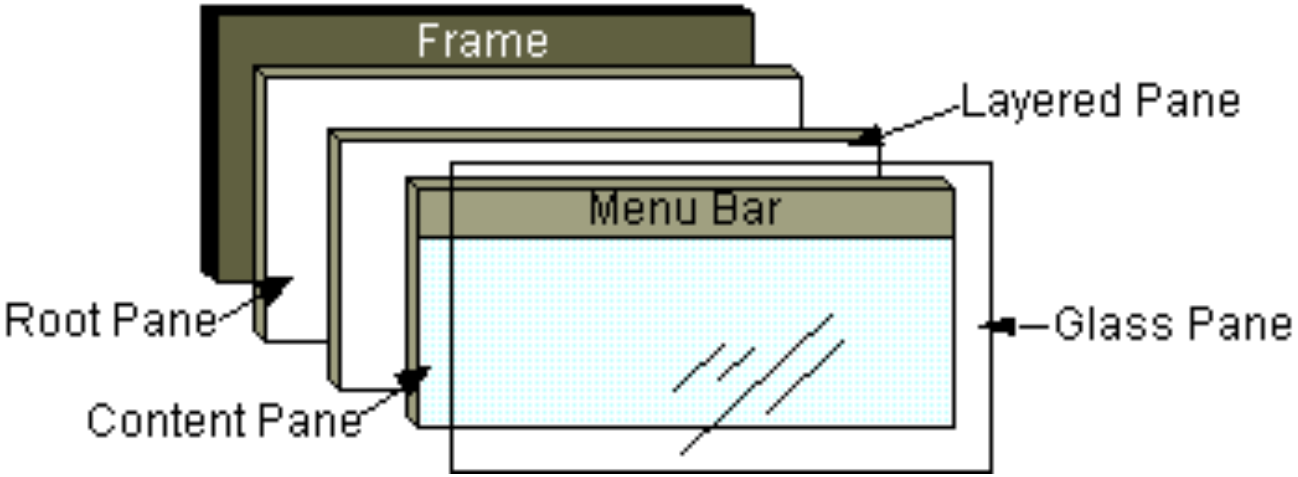
UI structure

- Top-level containers (need native support): JFrame, JDialog, JApplet.
- N-level containers (implemented in Java): JPanel, JSplitPane, JTabbedPane, ...
- Leaf components: JButton, JTextField, JTable, JList, JProgressBar, JScrollBar, ...

UI design via layout managers

Top-level containers structure: JRootPane

Top-level containers always contain a JRootPane

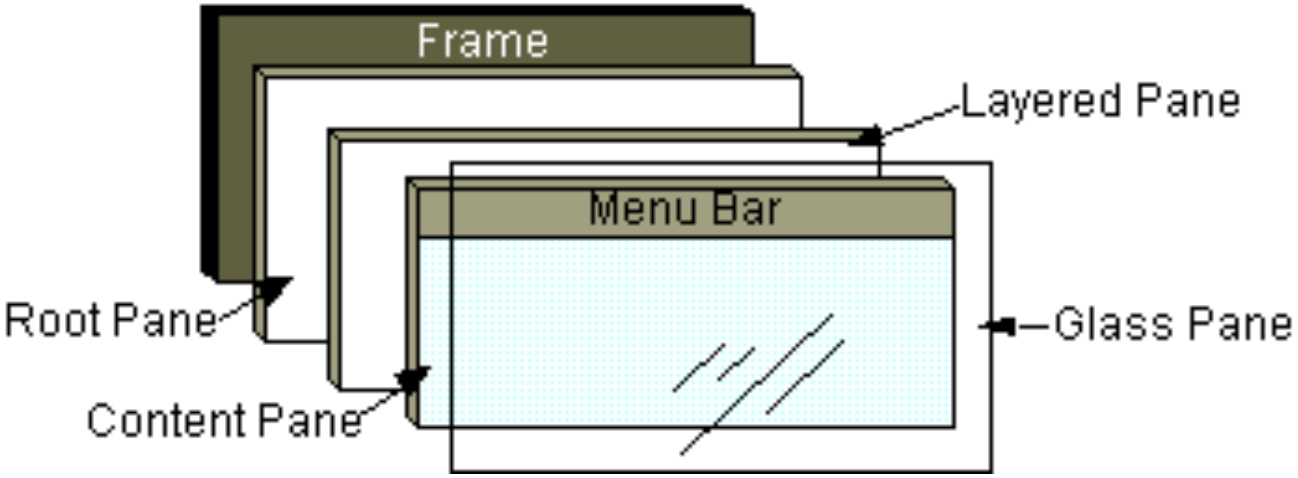


JRootPane contains JLayeredPane, a content pane, JMenuBar, and a glass pane

Can be used to intercept mouse clicks and paint over multiple components

Top-level containers structure: JLayeredPane

JRootPane contains a JLayeredPane

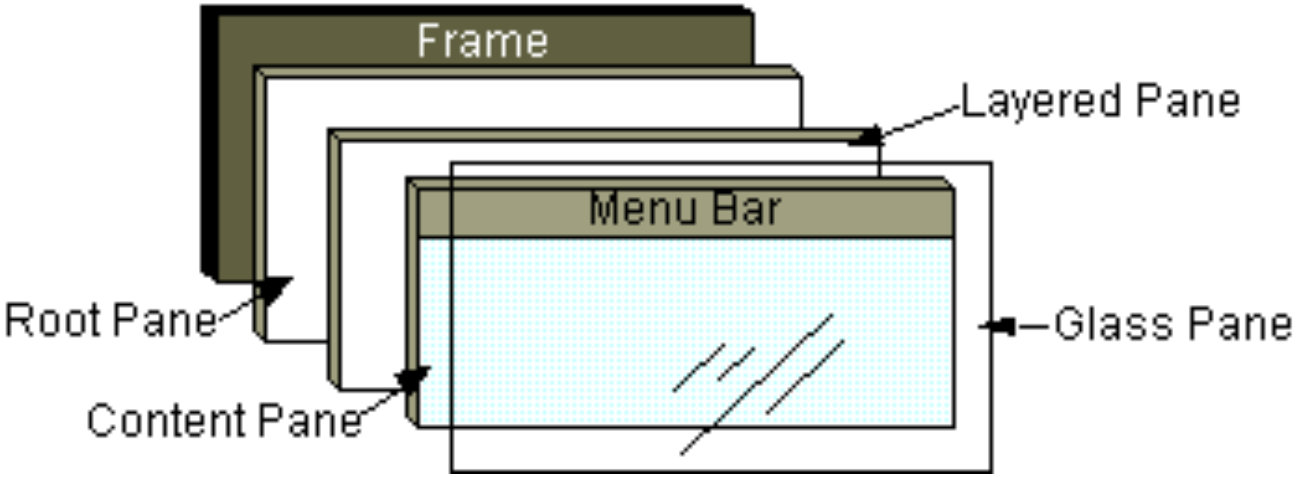


JLayeredPane contains and positions a content pane, an optional menu bar, and possibly dialogs and toolbars

Enables Z-ordering of the contained components

Top-level containers structure: content pane

In the content pane go your visible GUI components



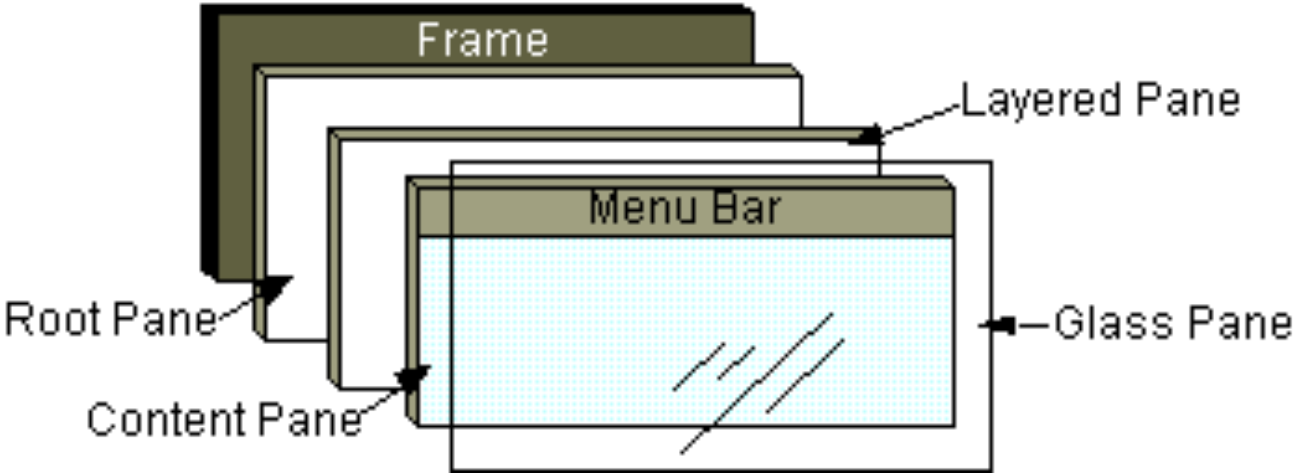
JPanel is the typical content pane

First create a JPanel, then add components to it, and then:

```
myTopLevelContainer.setContentPane(myJPanel)
```

Top-level containers structure: glass pane

The glass pane sits on top of everything, fills the entire view, and it's by default invisible.



Used to intercept mouse and keyboard clicks, drag & drop, and to draw over the entire UI, especially if there are already other components

Creating a JFrame window

```
public class MyFrame extends JFrame {
private JPanel contentPane;
public MyFrame(){
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  setBounds(100, 100, 450, 300);
  contentPane = new JPanel();
  setContentPane(contentPane);
}
}
```

Displaying a JFrame window

```
public class MyFrame extends JFrame {
public static void main(String[] args){
  EventQueue.invokeLater(new Runnable(){
     public void run(){
          try{
               MyFrame frame = new MyFrame();
               frame.setVisible(true)
          } catch (Exception e) {...}
     }});
}...}
```

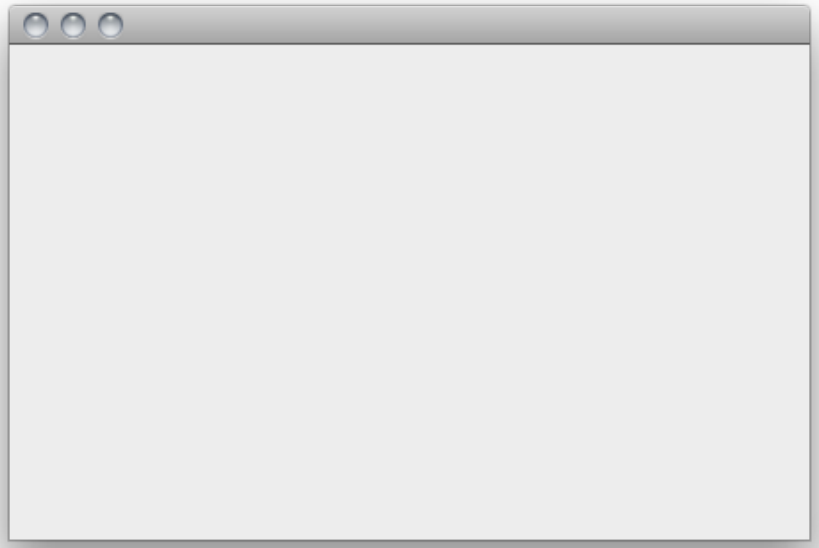
Setting up a glass pane

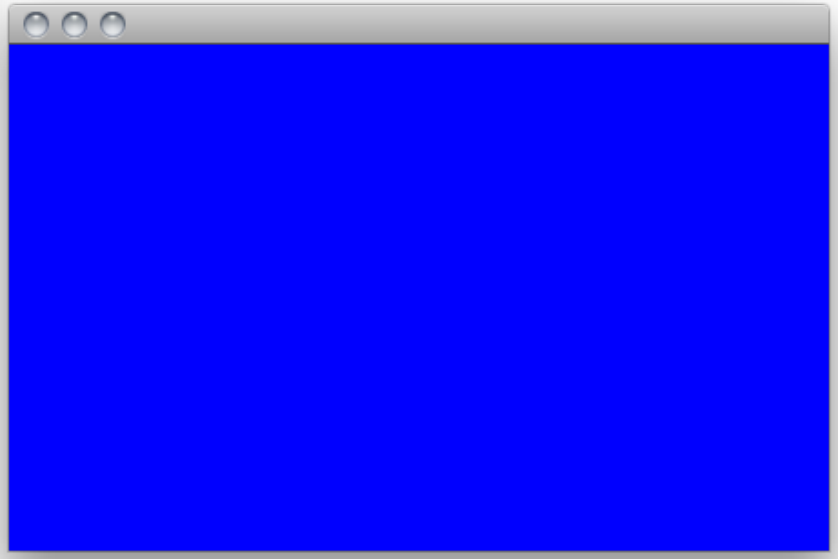
```
public class MyGlassPane extends JComponent {

@Override

protected void paintComponent(Graphics g){

  Rectangle clip = g.getClipBounds();

  g.setColor(Color.BLUE);

  g.fillRect(clip.x, clip.y, clip.width,
clip.height);

}

}
```

Adding a glass pane on top of a JFrame

```
public class MyFrame extends JFrame {
private JPanel contentPane;
public MyFrame(){
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  setBounds(100, 100, 450, 300);
  contentPane = new JPanel();
  setContentPane(contentPane);
  setGlassPane(new MyGlassPane());
  getGlassPane().setVisible(true);
}...}
```

Before and after applying the glass pane





Threading Concepts

Threads running in a Swing application

- Main thread: runs the application's main method
- Toolkit thread: captures system events (mouse, keyboard, …)
- Event Dispatching Thread (EDT): responsible for executing any method that modifies a component's state
  - dispatches the events captured by the toolkit thread to the appropriate components
  - the recommended way to interact with Swing

Caveats

Swing is not a thread-safe API. It is a single-threaded API, and that thread is the EDT

Swing should be used only on the EDT (see previous example)

Hint: don’t perform long-lasting computations or I/O accesses in a method executed by the EDT

Where do you perform these computations then?

Handling heavy computations and I/O

```
SwingUtilities.invokeLater(new Runnable(){
      public void run(){
                  // lengthy computation
      }});
```

This posts a new task on the EDT by invoking `EventQueue.invokeLater(...)`

From Java 6 you can use **`SwingWorker`**, a utility class to run a task on a background thread, and post intermediate or final results on the EDT

SwingWorker usage

For running long-running tasks in a different thread so as to prevent the GUI from being unresponsive

For updating GUI with the results produced by the long-running task at the end of the task through method `done()`

For updating GUI from time to time with the intermediate results produced and published by the task through methods `publish()` and `process()`

Handling events

Observer design pattern

- Components maintain a list of objects implementing listener interfaces (listeners)
- You can register/unregister a listener XYZ on a component c:

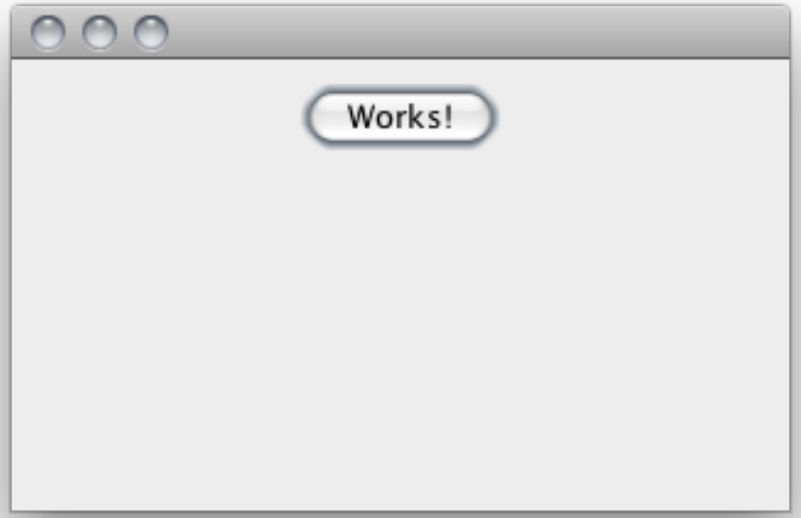
`c.addXYZListener` or `c.removeXYZListener`

- When the component fires an event, all registered listeners are notified using a callback
- The reaction code is typically in the (anonymous inner) class implementing the listener interface

Adding a button and an associated action

```
public class MyFrame extends JFrame {
private JButton myButton;
public MyFrame(){...
  myButton = new JButton("Push me!");
  contentPane.setLayout(new FlowLayout());
  myButton.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
      myButton.setText("Works!");
      }});
  contentPane.add(myButton);
  setContentPane(contentPane);
}...}
```

Firing an ActionEvent



Some Listeners and related Components

Event Listener	Listener methods	Register on
ActionListener	actionPerformed()	JButton, JComboBox, JFileChooser, MenuItem, JTextField, …
FocusListener	focusGained(), focusLost()	Component
MouseListener	mouseClicked(), mouseEntered(), mouseExited(), mousePressed(), mouseReleased()	Component
MouseMotionListener	mouseDragged(), mouseMoved()	Component

Some more Listeners and Components

Event Listener	Listener methods	Register on
KeyListener	keyPressed(), keyReleased(), KeyTyped()	Component
TextListener	textValueChanged()	TextComponent
CaretListener	caretUpdate()	JTextComponent
MenuListener	menuCanceled(), menuDeselected(), menuSelected()	JMenu

Layout Managers

Used to harmonize component placement

They typically decide the component size

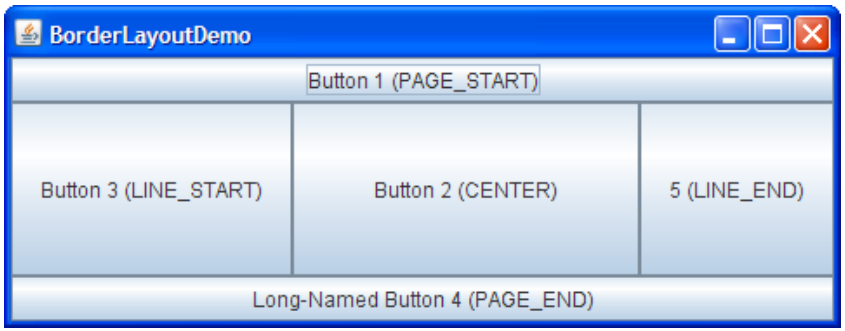
Can be composed with one another

React in a ‘predictable’ way when adding/removing components and resizing the application window

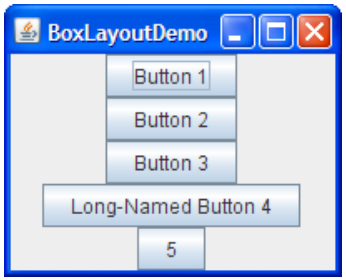
Absolute positioning (x, y, size) is still possible:

```
contentPane.setLayout(null);
```

Some Layout Managers

Border Layout: 

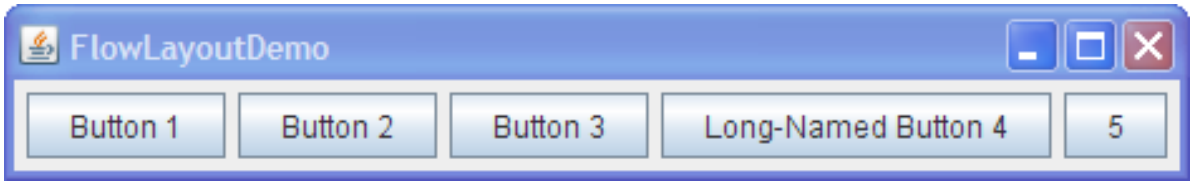
5 areas

BoxLayout: 

single row/column

GridLayout: 

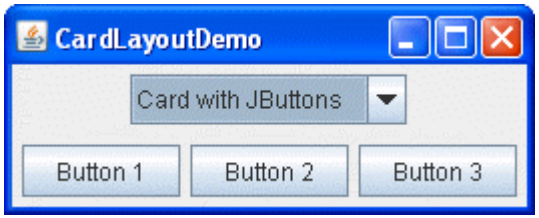
cells are same size

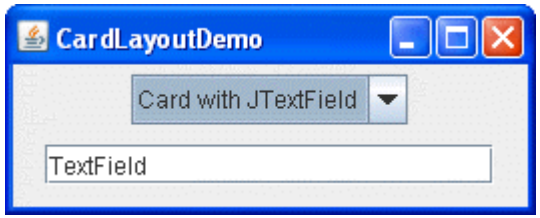
FlowLayout 

http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html

Some more Layout Managers

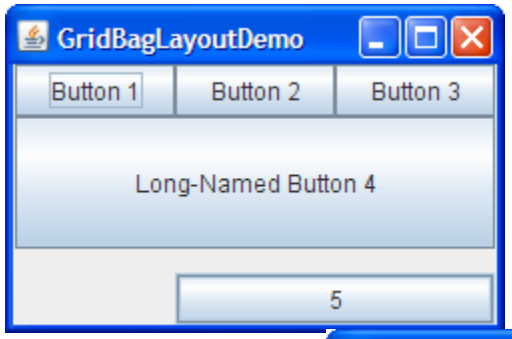
CardLayout: different components at different times





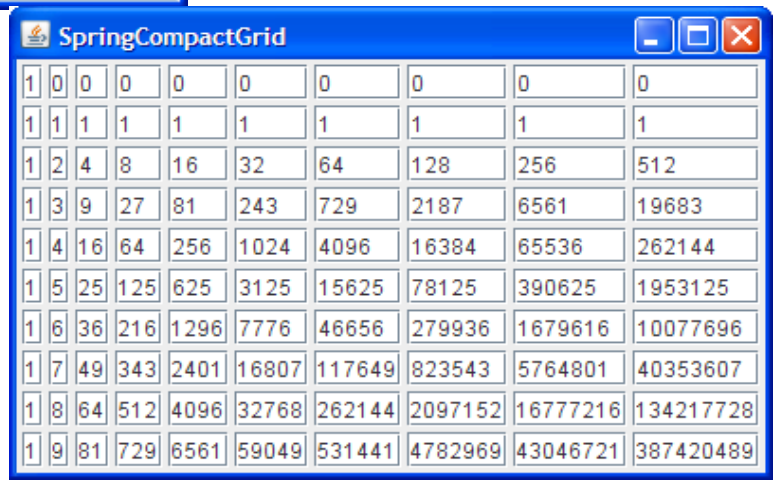
GridBagLayout:

cells of different size



SpringLayout:

fine-grained control



http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html

Do it yourself

You can experiment with the Java GUI APIs trying the RPN calculator assignment

A nice GUI designer tool that produces clean GUI code in the background and you may want to have a look at is Google's Window Builder (Eclipse plug-in):

https://developers.google.com/java-dev-tools/wbpro/