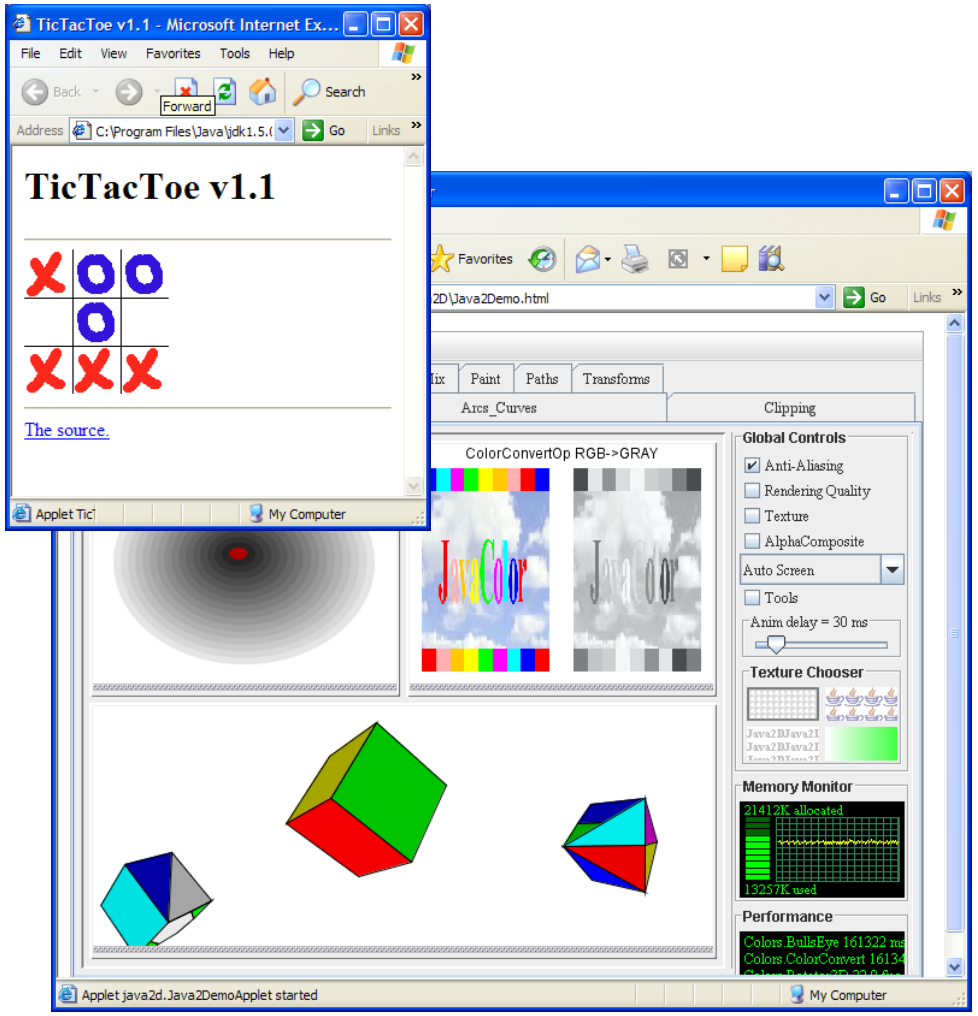
CompSci 230
Software Construction

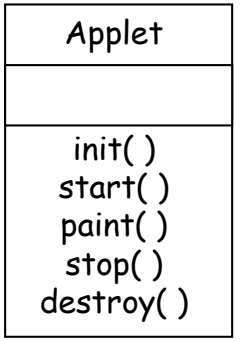
Applets, AWT S1 2015

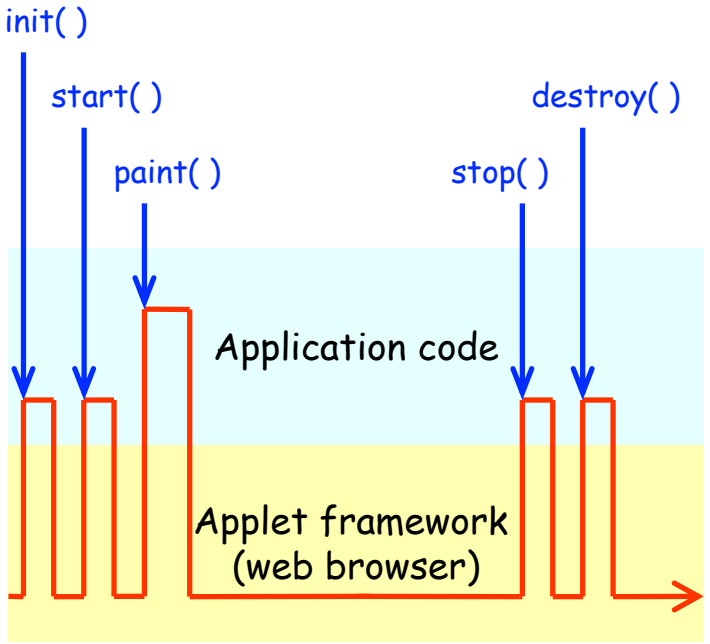
Learning Goals

- You will gain a high-level understanding of two GUI frameworks
  - Applets
  - AWT
  - The details are uninteresting (but necessary if you're implementing)
- Basic GUI terminology:
  - Component, container, panel, window, frame. Layout manager.
- Theory:
  - Inversion of control
  - Composite design pattern
    - Painting and layout are recursive.
  - Classes may have (or "require") interfaces, they don't just implement (or "provide") them.

Java applets: a simple GUI framework



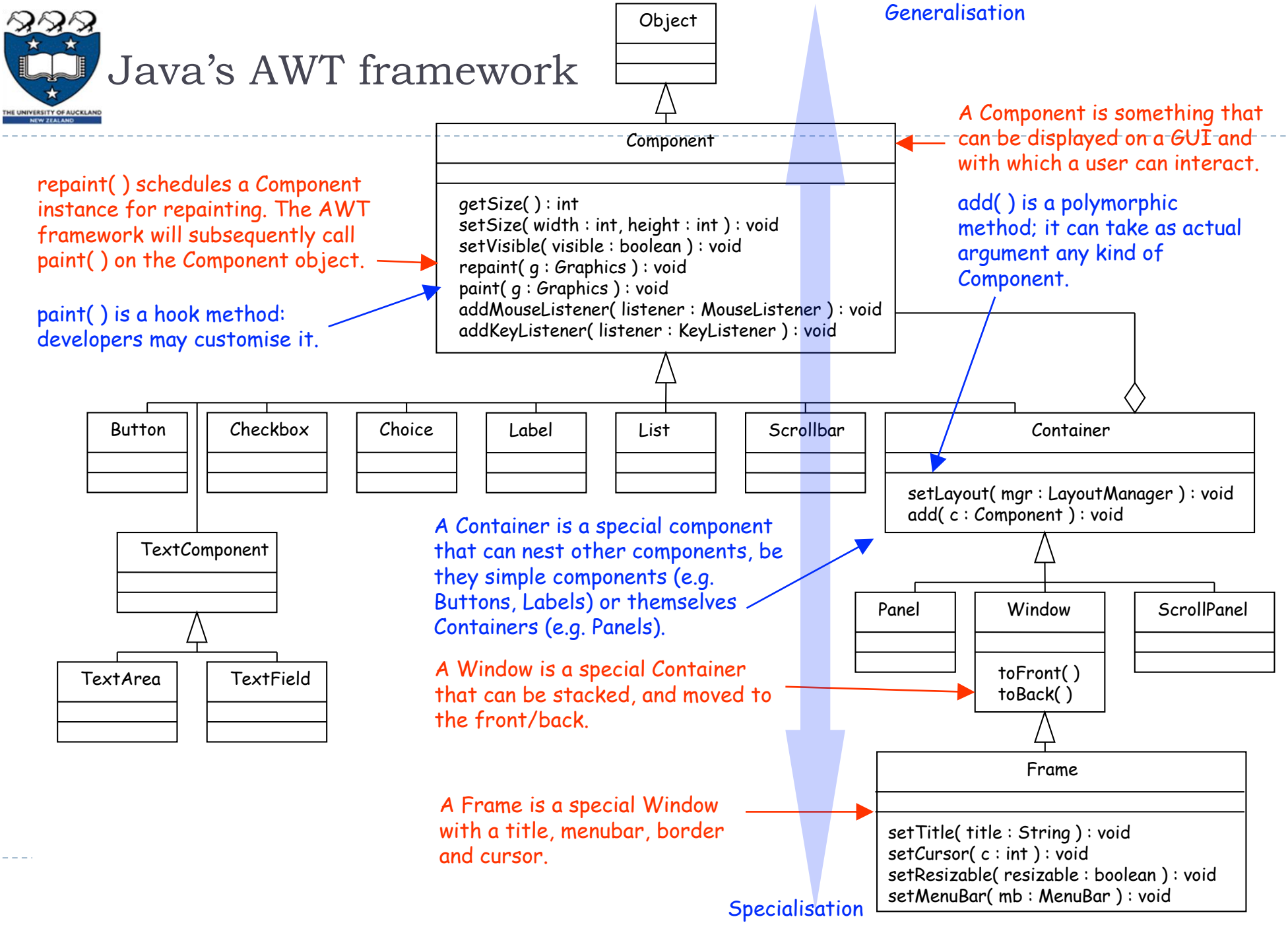


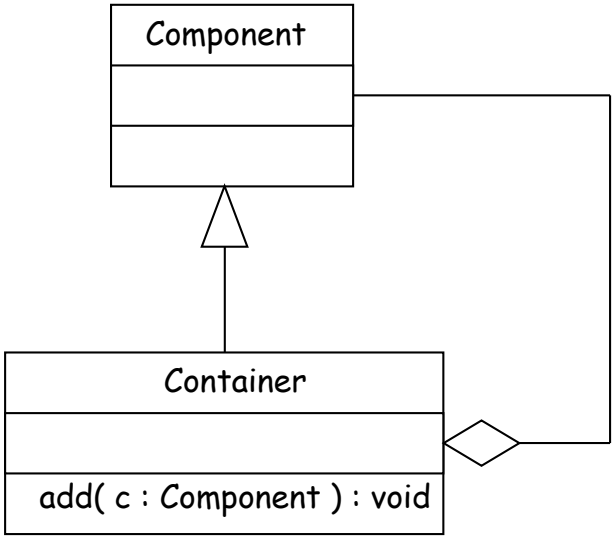


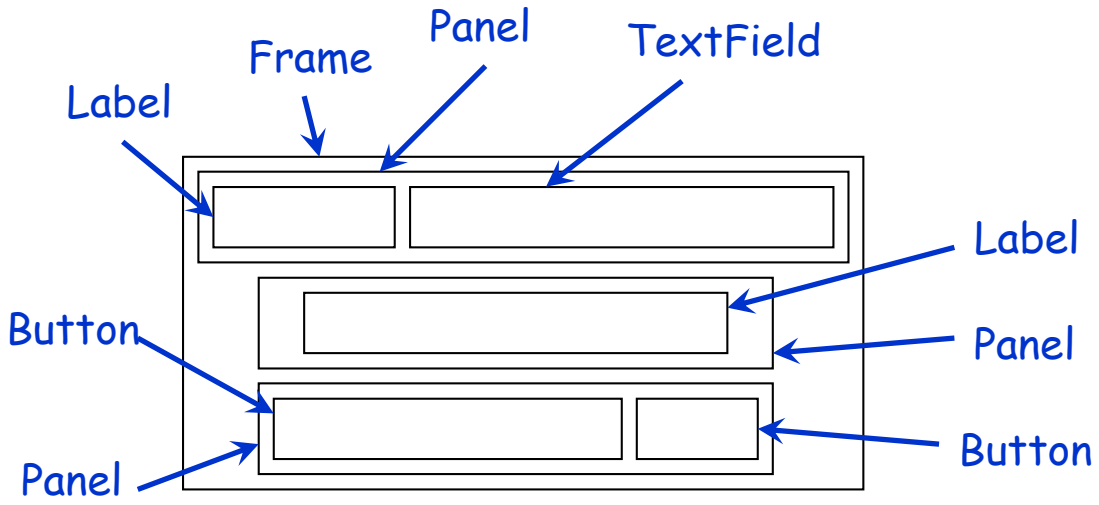
Hello World applet

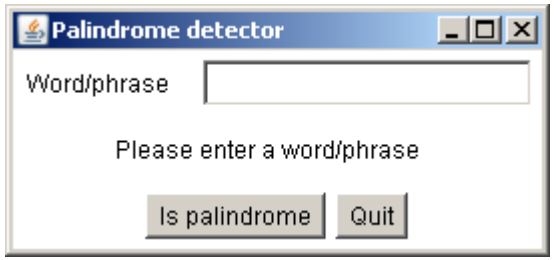
```
import javax.swing.JApplet;
import javax.swing.SwingUtilities;
import javax.swing.JLabel;

public class HelloWorld extends JApplet {
    //Called when this applet is loaded into the browser.
    public void init() {
        //Execute a job on the event-dispatching thread; creating this applet's GUI.
        try {
            SwingUtilities.invokeAndWait(new Runnable() {
                public void run() {
                    JLabel lbl = new JLabel("Hello World");
                    add(lbl);
                }
            });
        } catch (Exception e) {
            System.err.println("createGUI didn't complete successfully");
        }
    }
}
```

Java's AWT framework





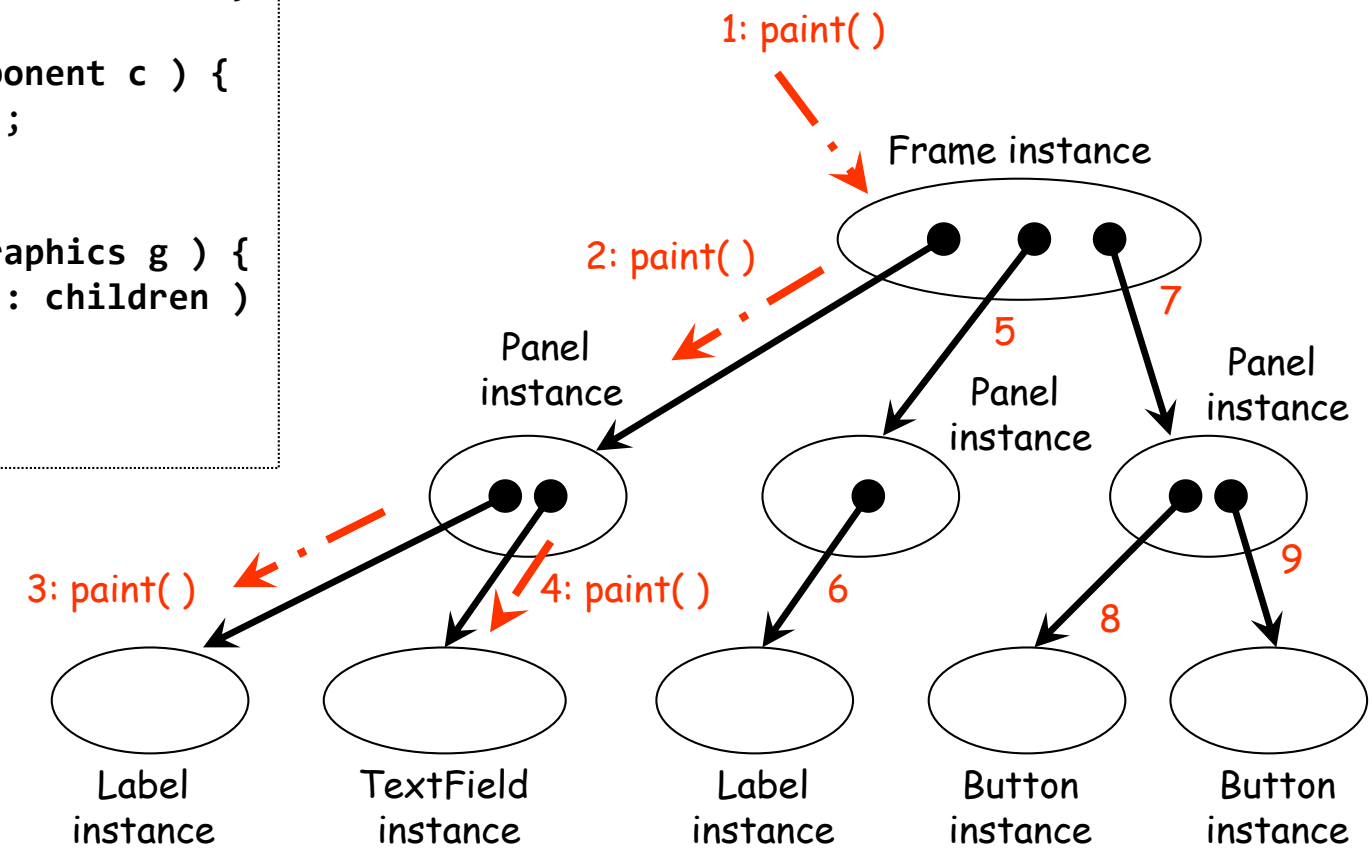





```
class Container extends Component {
   private List<Component> children;
   ...
   public void add( Component c ) {
         children.add( c );
         }
   }
```

Traversing the nested structure

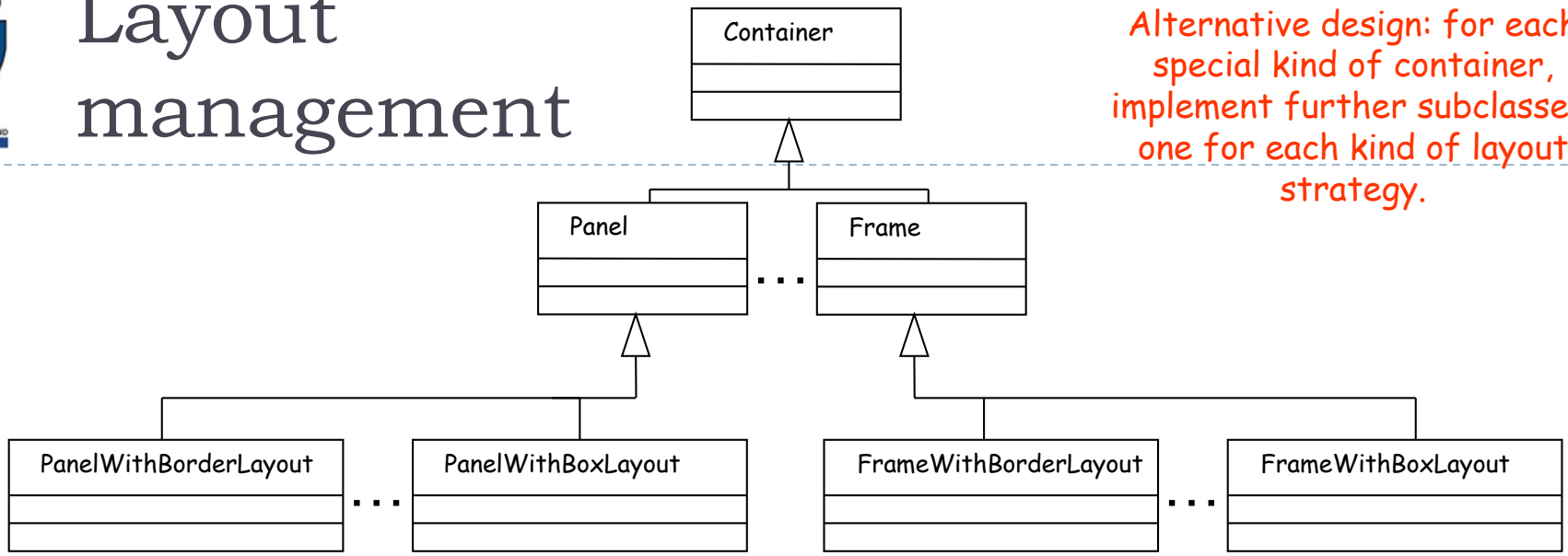
```
class Container extends Component {
    private List<Component> children;
    …
    public void add( Component c ) {
        children.add( c );
    }

    public void paint( Graphics g ) {
        for( Component c : children )
            c.paint( );
    }
}
```



Layout management

Why is this design for layout management so effective?

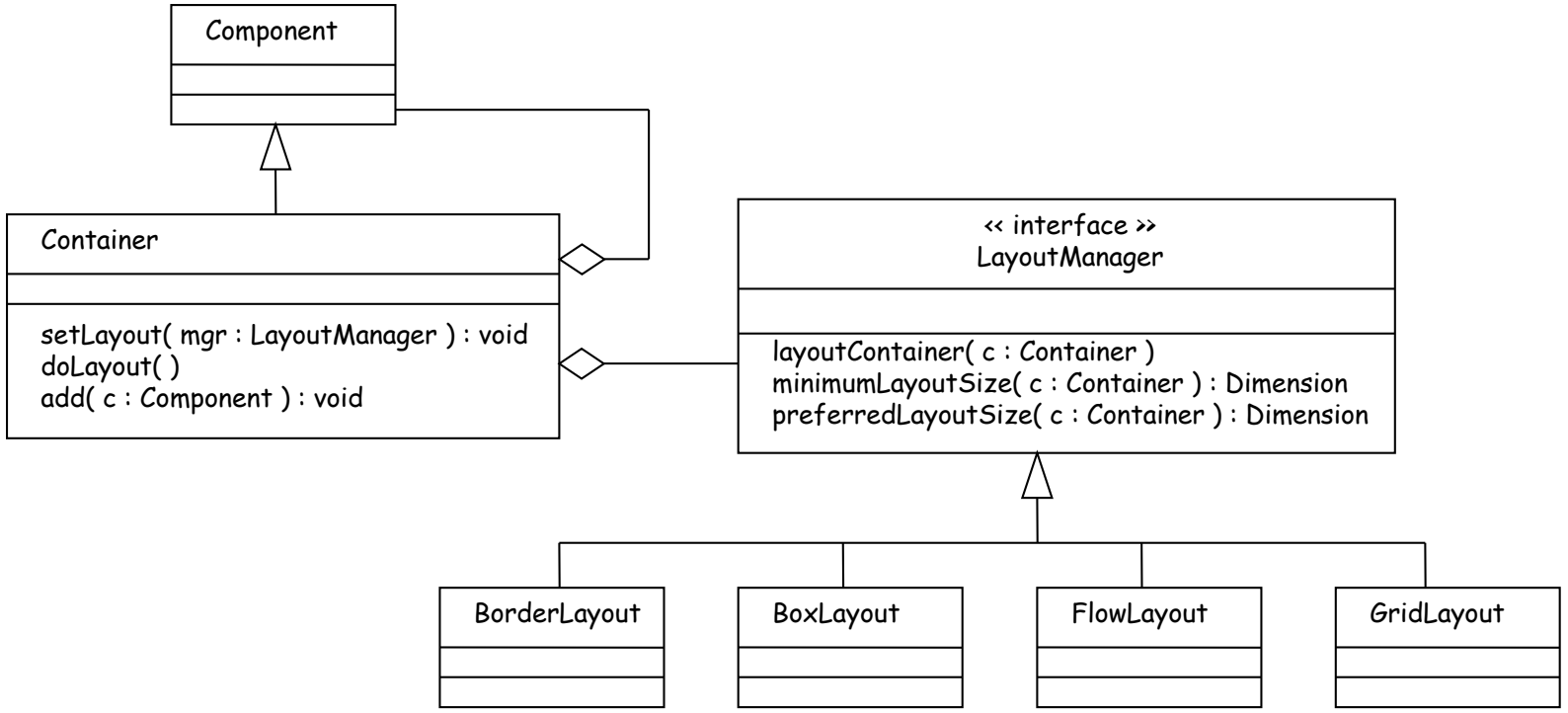
Layout management



Alternative design: for each special kind of container, implement further subclasses, one for each kind of layout strategy.

Composition and interfaces	Inheritance
This solution requires only one class for each special type of container, and one class for each layout strategy. Any kind of container instance can be configured with any kind of strategy object.	**Subclass explosion!** To support all container (c) and layout strategy (s) combinations, c * s classes would be required.
Composition offers run-time flexibility. A Frame can change how it lays out its children simply by swapping its current layout strategy (e.g. BorderLayout) for another (e.g. GridLayout).	**Overly rigid structure**. Once a PanelWithBorderLayout object, for example, has been created, the panel will always be laid out according to the border strategy.

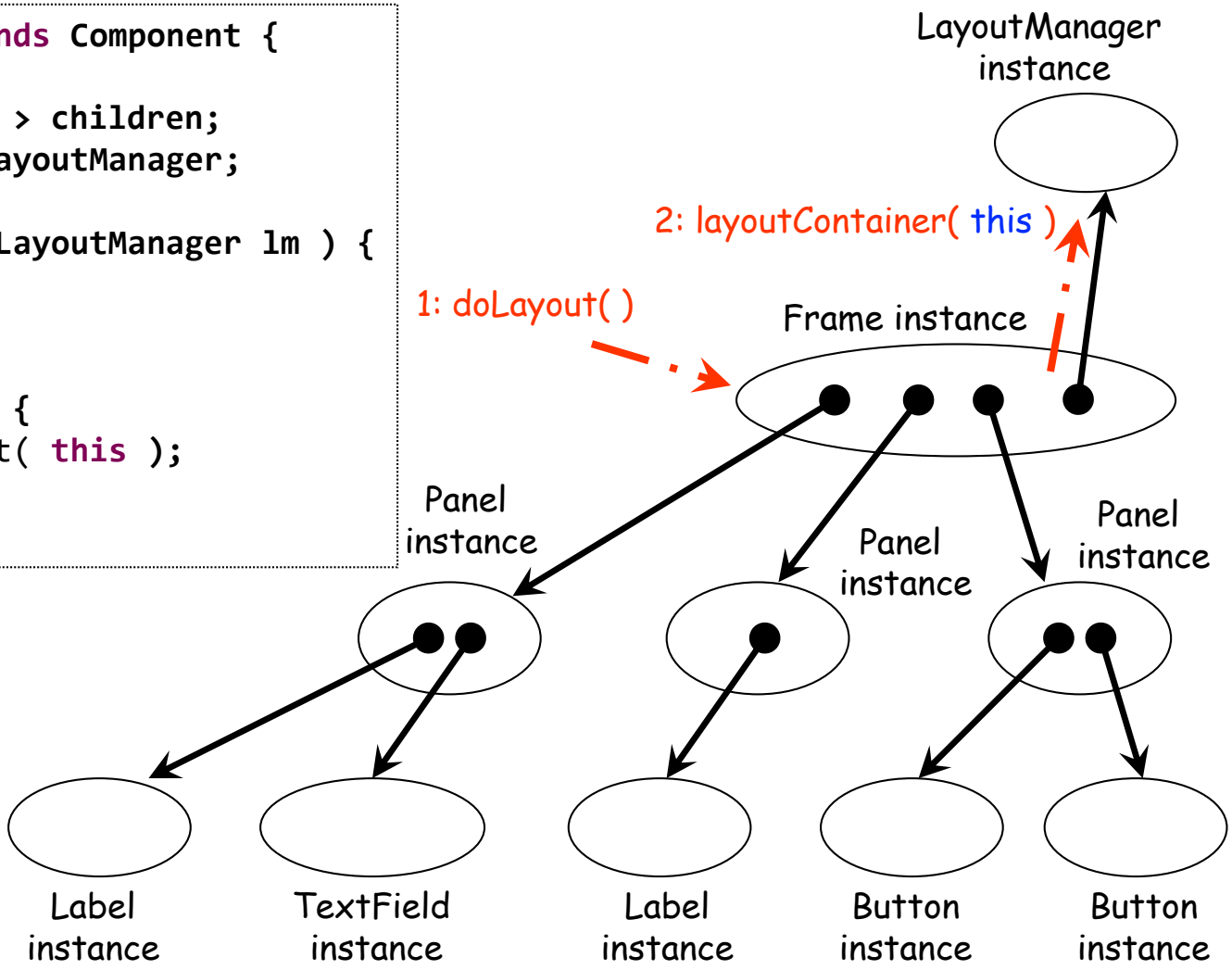
Provided and Required Interfaces



- A `Container` requires a `LayoutManager`
- A `BorderLayout` provides a `LayoutManager`

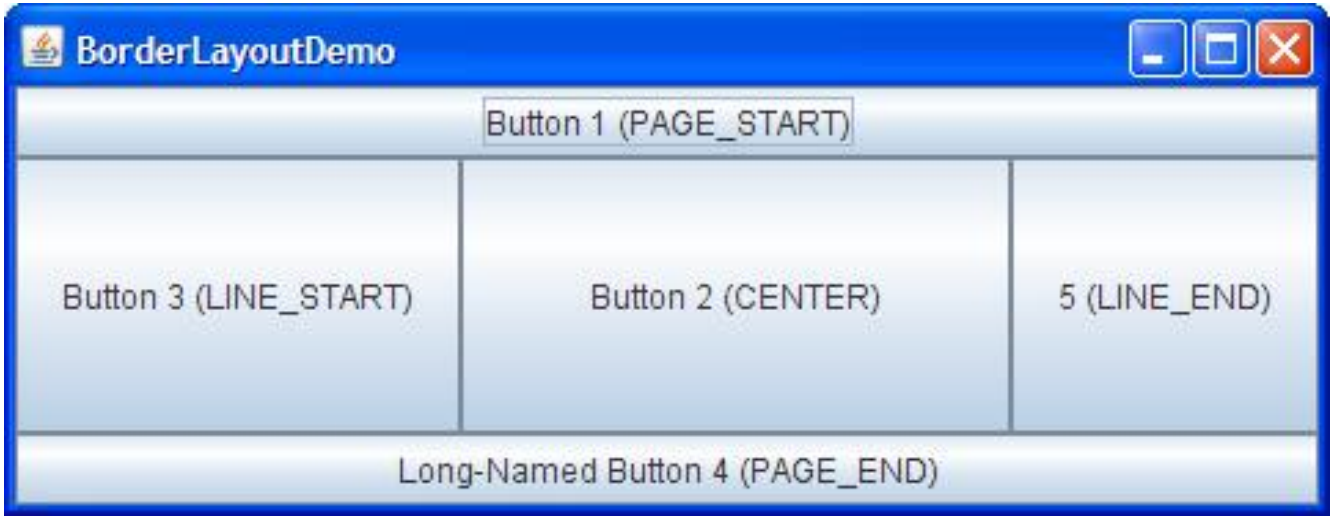
Layout management

```
public class Container extends Component {
    …
    private List< Component > children;
    private LayoutManager layoutManager;

    public void setLayout( LayoutManager lm ) {
        layoutManager = lm;
    }

    public void doLayout( ) {
        layoutManager.layout( this );
    }
}
```



Layout Managers

- BorderLayout:
  - This scheme defines five areas for the component.
  - All extra space is placed in the center area.
- FlowLayout:
  - Simplest, just one row
- BoxLayout, GridLayout, …



Learning Goals: Review

- You will gain a high-level understanding of two GUI frameworks
  - Applets
  - AWT
  - The details are uninteresting (but necessary if you're implementing)
- Basic GUI terminology:
  - Component, container, panel, window, frame. Layout manager.
- Theory:
  - Inversion of control
  - Composite design pattern
    - Painting and layout are recursive.
  - Classes may have (or "require") interfaces, they don't just implement (or "provide") them.