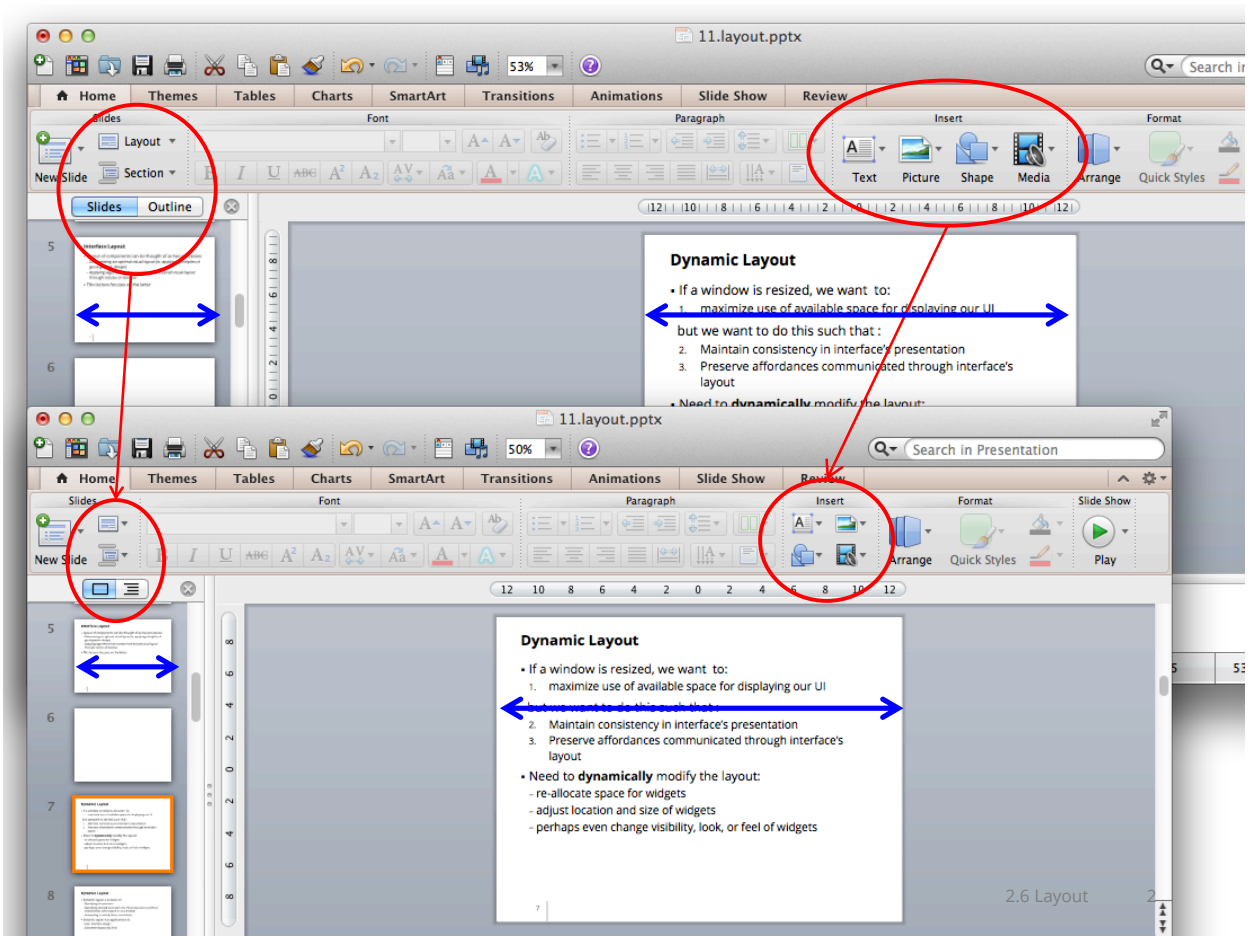
Layout

Dynamic layout

Layout design pattern

Layout strategies



Two Interface Layout Tasks

1. ***Designing* a spatial layout** of widgets in a container
2. ***Adjusting* that spatial layout** when container is resized

- can be done by hand (i.e. graphic design)
  or automatically (i.e. with algorithms).
- (spatial layout is one component of visual design …)

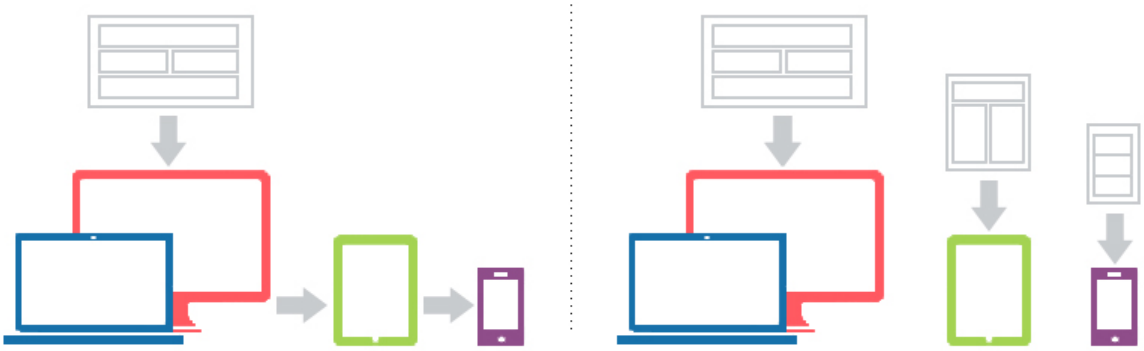
Dynamic Layout

- If a window is resized, we want to:
  1. maximize use of available space for displaying widgets

  but we want to do this such that:
  2. maintain consistency with spatial layout
  3. preserve visual quality of spatial layout
- Need to **dynamically** modify the layout:
  - re-allocate space for widgets
  - adjust location and size of widgets
  - perhaps even change visibility, look, and/or feel of widgets

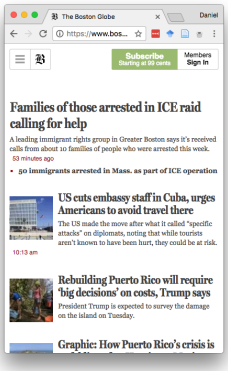
Responsive vs. Adaptive

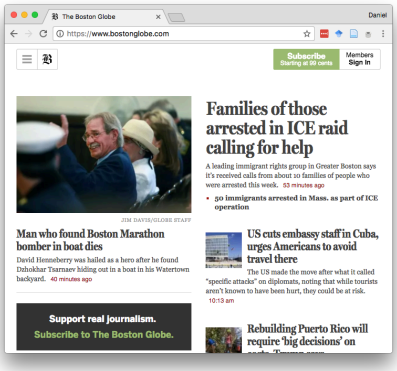
- **Responsive**: universal design reflows spatial layout to fit width
- **Adaptive**: switch between optimized spatial layouts to fit devices
- In practice, the two approaches can be combined

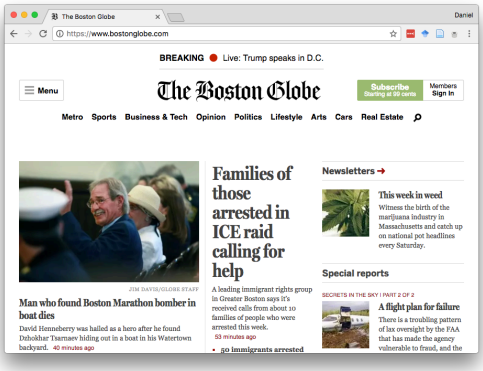


Article and Demo

- https://css-tricks.com/the-difference-between-responsive-and-adaptive-design/



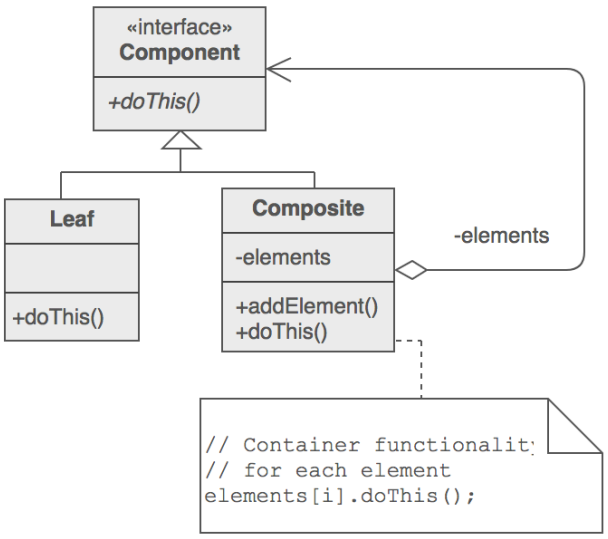




https://www.bostonglobe.com/

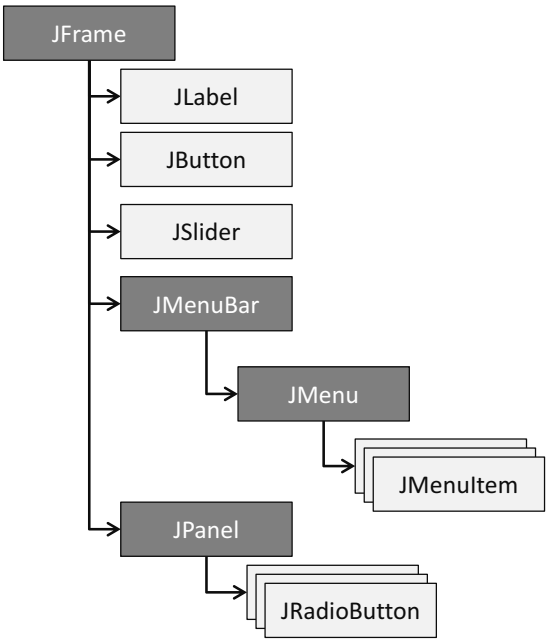
Layout uses Composite Design Pattern

- Treat leaf objects and compositions of objects uniformly
- Creates a tree data structure



In Swing, a “leaf” is a simple widget like a button and a “composite” is a container widget like a JPanel

Composite Pattern with Swing



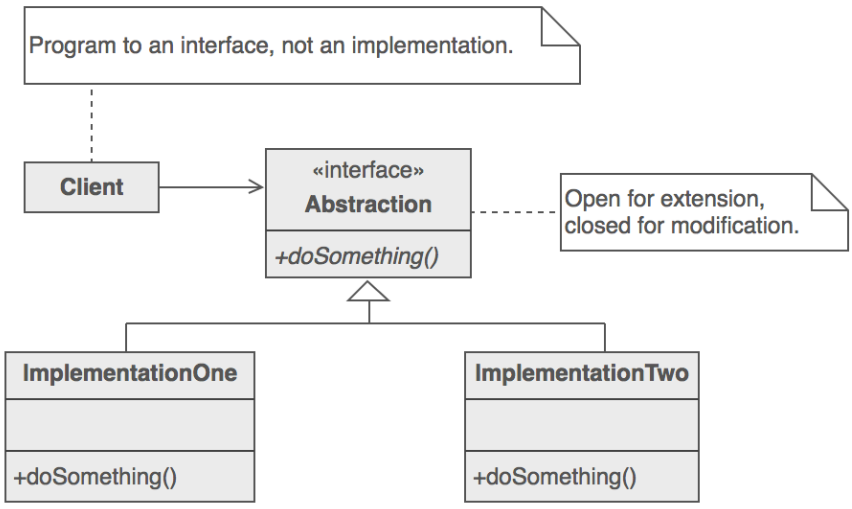
Widget Size

- To make a layout dynamic, widgets need to be “flexible”
  - x,y position may be changed
  - width and height may be changed
- Widgets give the layout algorithm a range of sizes as “hints”
- Containers and leaves have size hints

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

LayoutManager is a Strategy Design Pattern

- Factors out an algorithm into a separate object, allowing a client to dynamically switch algorithms

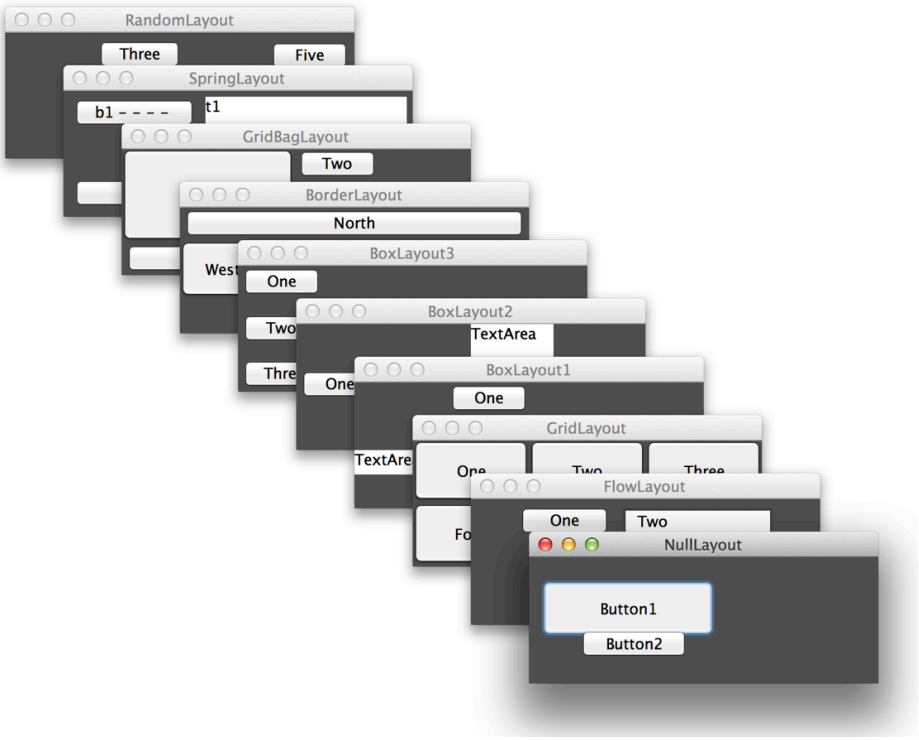


Java LayoutManager

- Container widgets can use different LayoutManagers
  - a LayoutManager is an “strategy” object that factors out the layout algorithm to size and position child widgets
- Example:

```
container.setLayout(new GridLayout(2, 3));
```


Code Demo: LayoutDemo.java

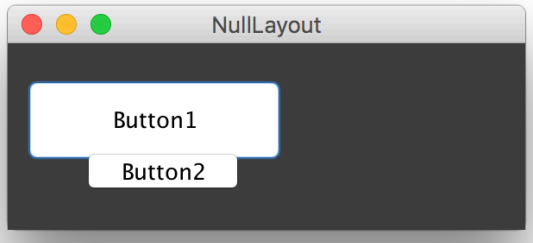


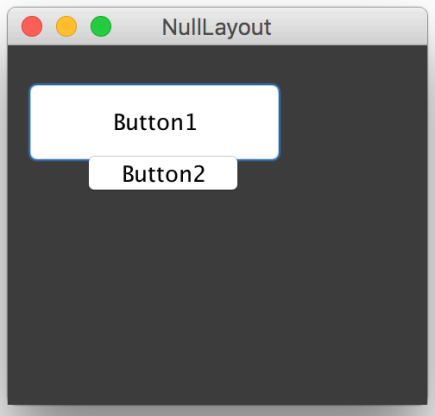
General Layout Strategies

- Fixed layout
- Intrinsic size
- Variable intrinsic size
- Struts and springs
- Constraints

Fixed Layout

- Widgets have a fixed size, fixed position
- In Java, achieved by setting LayoutManager to null
- Where/when is this practical?
- How can it break down even when windows aren't resized?



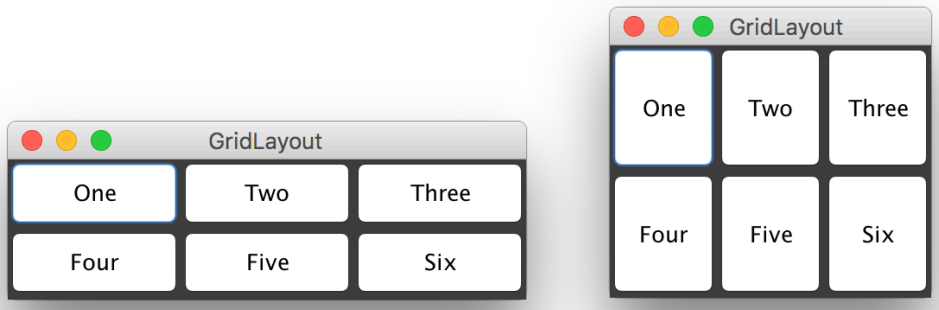


Intrinsic Size Layout

- Query each item for its preferred size
- Grow the widget to perfectly contain each item
- A bottom-up approach where top-level widget's size completely dependent on its contained widgets
- Example LayoutManagers:
  `BoxLayout, FlowLayout`
- Examples of use in interface design?
- How to handle when too big?

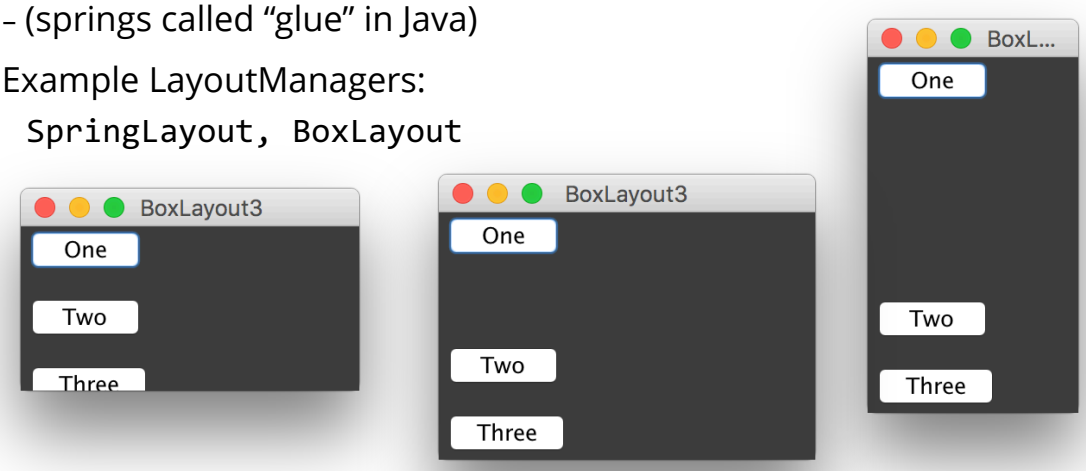
Variable Intrinsic Size Layout

- Layout determined in two-passes (bottom-up, top-down)
  1. Get each child widget's preferred size (includes recursively asking all of its children for their preferred size…)
  2. Decide on a layout that satisfies everyone's preferences, then iterate through each child, and set it's layout (size/position)
- Example LayoutManagers:
  `GridBagLayout, BorderLayout`



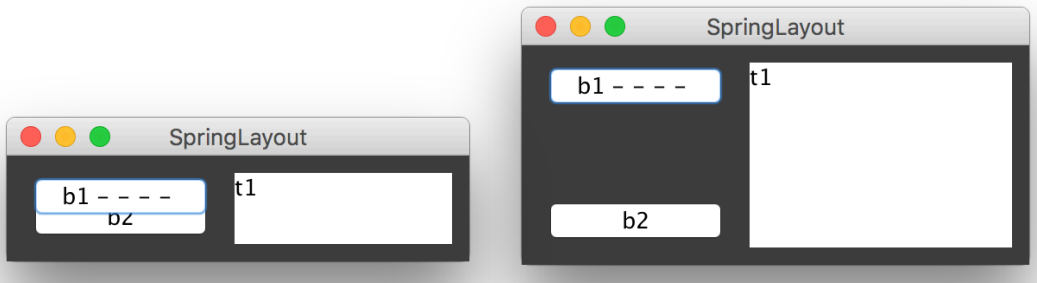
Struts and Springs Layout

- Layout specified by marking space as fixed or “stretchable”
- Strut is a fixed space (width/height)
  - Specifies invariant relationships in a layout
- Spring “stretches” to fill space (or expand widget size)
  - Specifies variable relationships
  - (springs called “glue” in Java)
- Example LayoutManagers:
  `SpringLayout, BoxLayout`



Relative Layout

- Relative position constraints too
  - e.g. widget must be EAST of another widget
- Example LayoutManagers in Java
  `SpringLayout`

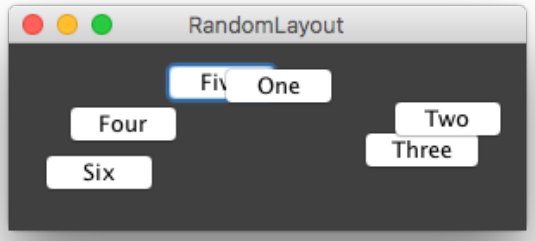


Custom Layout

- Implement the LayoutManager Interface

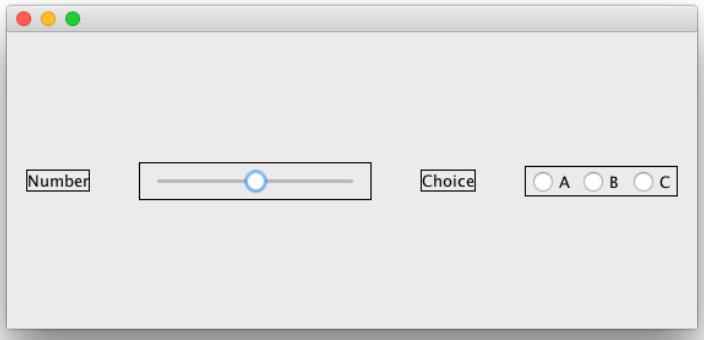
```
void addLayoutComponent(String, Component)
void removeLayoutComponent(Component)
Dimension preferredLayoutSize(Container)
Dimension minimumLayoutSize(Container)
void layoutContainer(Container)
```

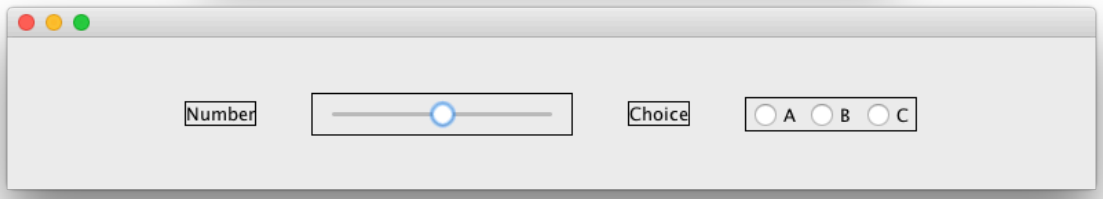
- Also a LayoutManager2 Interface
  - adds five more methods



Custom Layout Example: AlignLayoutDemo.java

- layout components in horizontal row equally spaced
- row of components is centred in window





AlignLayoutDemo.java

```
class AlignLayout implements LayoutManager {

  public AlignLayout(int minSpacing, int preferredSpacing) {
     ...
  }

  public Dimension preferredLayoutSize(Container parent) {
     ...
  }

  public Dimension minimumLayoutSize(Container parent) {
     ...
  }

  public void layoutContainer(Container parent) {
     ...
  }

  private Dimension calculateSpace(Container parent,
                                   boolean isPreferred) {
     ...
  }
```


AlignLayoutDemo.java

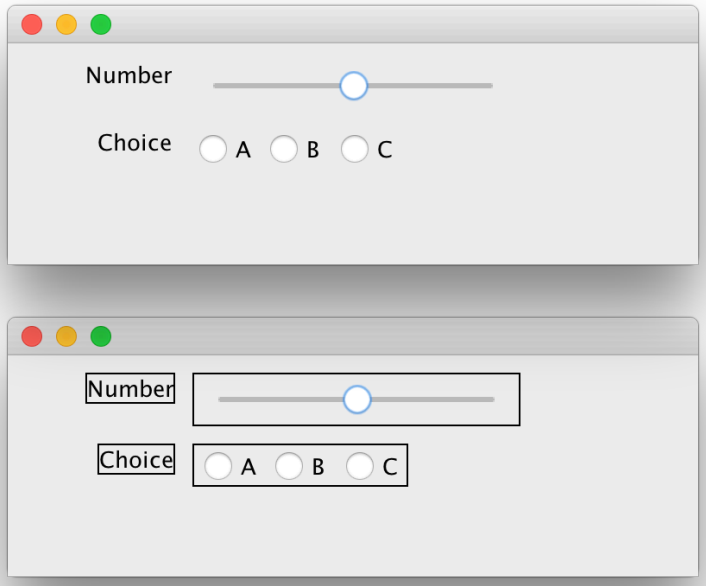
```
  Dimension calculateSpace(Container parent, boolean isPreferred) {

     Dimension result = new Dimension(0,0);

     int nComponents = parent.getComponentCount();
     for (int i = 0; i < nComponents; i++) {

        Dimension d;
        if (isPreferred) {
               d = parent.getComponent(i).getPreferredSize();
        } else {
               d = parent.getComponent(i).getMinimumSize();
        }
        // update the total width and height required
        result.width += d.width;
        result.height = Math.max(result.height, d.height);
     }

     // add spacing in between components
     if (isPreferred) {
        result.width += (nComponents - 1) * preferredSpacing;
     } else {
        result.width += (nComponents - 1) * minimumSpacing;
     }

     return result;
  }
```

AlignLayoutDemo.java

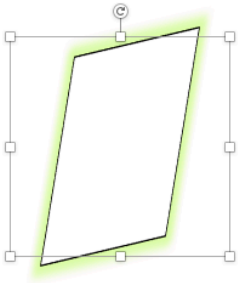
```
public void layoutContainer(Container parent) {

        Dimension space = calculateSpace(parent, true);

        // this container's padding
        Insets insets = parent.getInsets();

        // get actual space available in parent
        int w = parent.getWidth() - insets.left - insets.right;
        int h = parent.getHeight() - insets.top - insets.bottom;

        // vertical centre line to layout component
        int y = h / 2;

        // starting x is left side of all components to lay out
        int x = (w - space.width) / 2;

        int nComponents = parent.getComponentCount();
        for (int i = 0; i < nComponents; i++) {
                Component c = parent.getComponent(i);
                Dimension d = c.getPreferredSize();
                c.setBounds(x, y - d.height / 2, d.width, d.height);


                x += d.width + preferredSpacing;
        }

}
```

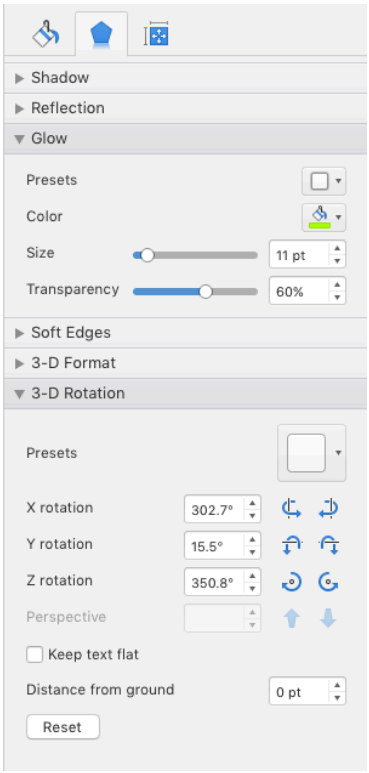

(Extra) FormLayout Custom Layout Manager

- widgets organized in two columns
- order widget added determines column

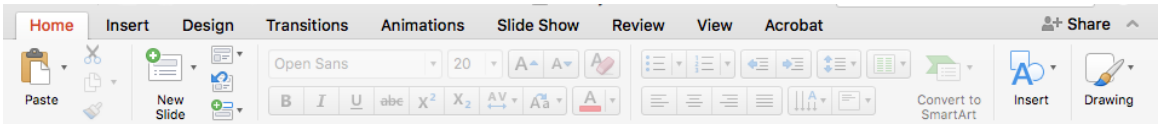


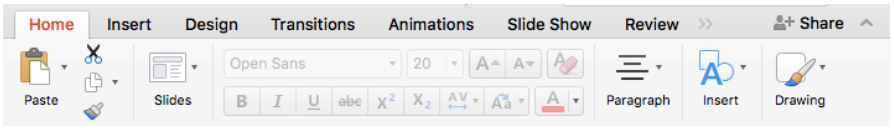
How to implement an “Accordion” LayoutManager?

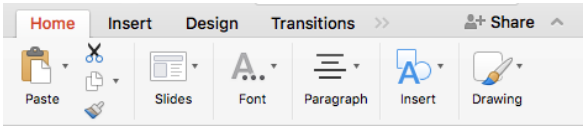




How to implement a “Ribbon” LayoutManager?

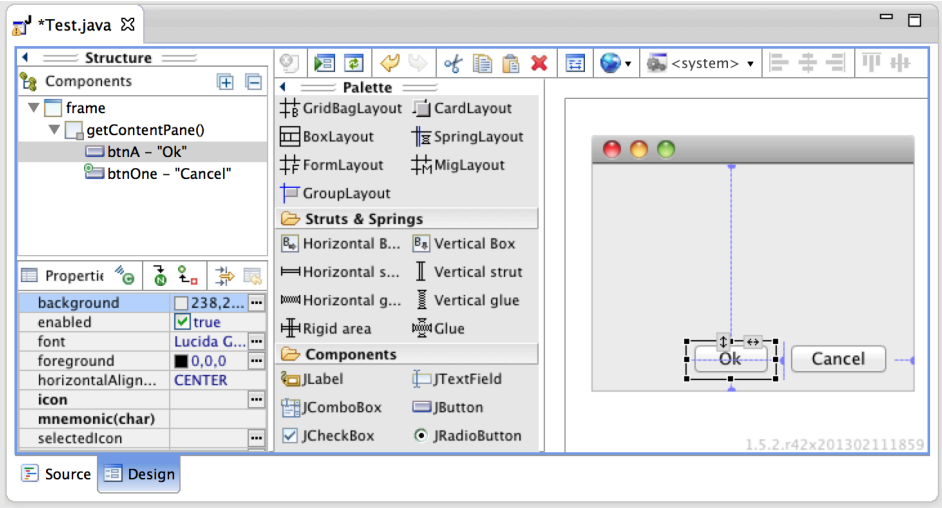






Struts and Springs in GUI Design Tools

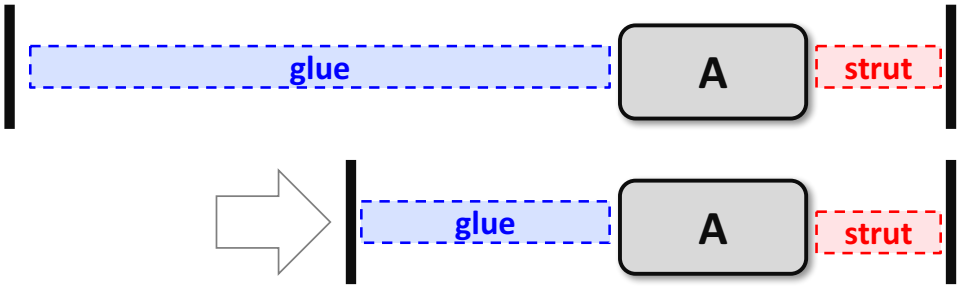
- Very common, especially in Interactive GUI design tools
  - Can be more difficult to do in code
- Good metaphors for people performing layout



Google WindowBuilder Eclipse Plug-in

Struts and Springs ("Glue") in Java

- javax.swing.Box has useful widgets for any layout manager
  - Glue to expand/contract to fill space (i.e. "Springs")
    `Box.createHorizontalGlue(), Box.createVerticalGlue()`
  - Rigid Areas and Struts to occupy space
    `Box.createHorizontalStrut(...),`
    `Box.createVerticalStrut(...)`
    `Box.createRigidArea(...)`



Tips and Strategies

- Break up the UI recursively with panels that contain panels.
- Cluster components into panels based on layout needs
- Provide a layout manager for each panel