

CS 349 Layout

Byron Weber Becker
Winter 2012



Jan 23 Announcements

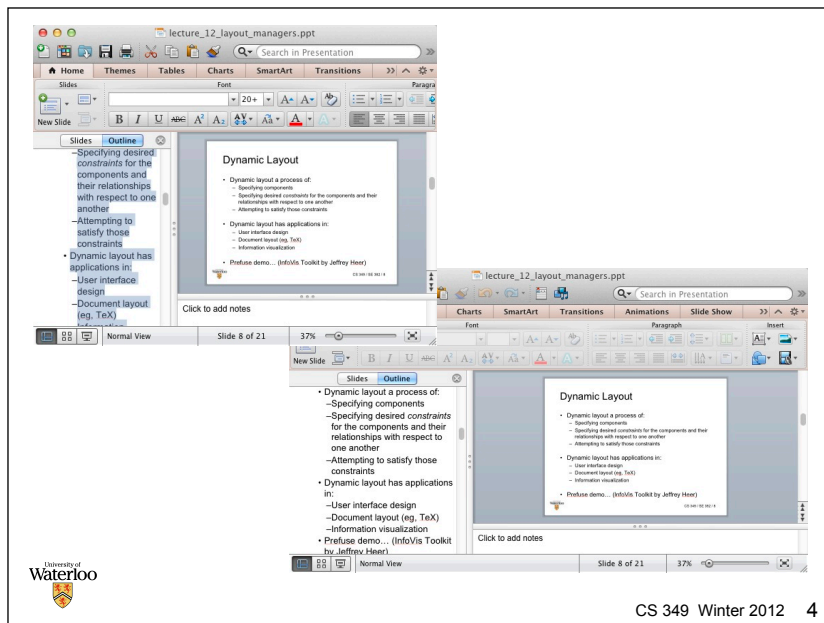
- A02 is ready to go.



Interface Layout

- Layout of components can be thought of as two processes:
 - Determining an optimal visual layout (ie, applying principles of good graphic design)
 - Applying algorithms that maintain that desired visual layout through resizes of window
- This lecture focuses on the latter





Dynamic Layout

- Windows are dynamic, can be resized
- Through any resize, we wish to:
 - Maintain *consistency* in interface's presentation
 - Preserve *affordances* communicated through interface's layout
- Need to dynamically modify allocation of space, locations of objects in interface

Dynamic Layout

- Dynamic layout a process of:
 - Specifying components
 - Specifying desired *constraints* for the components and their relationships with respect to one another
 - Attempting to satisfy those constraints
- Dynamic layout has applications in:
 - User interface design
 - Document layout (eg, TeX)

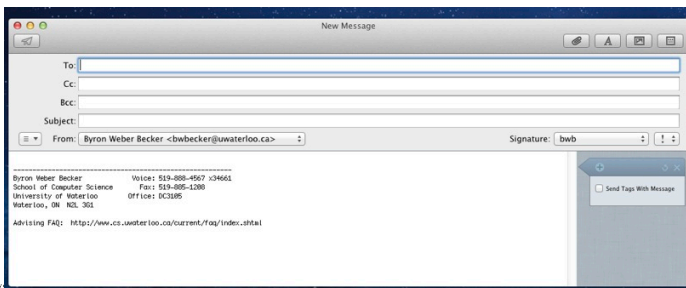
Layout in Java

- *Containers* maintain collection of components
- Container (eg, JPanel) can utilize a *LayoutManager*
`myPanel.setLayout(new GridLayout(2,3))`
- *LayoutManager* a *strategy pattern* that factors out process of positioning, sizing components within that container
- Can vary *LayoutManager* independently of container, components



Java Layout Demo

- LayoutDemo.java
- Available on CS349 Resources page.
- Think about: How would you use these tools to layout



General Layout Strategies

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



Fixed Layout

- Components are of a fixed size, 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

- Query each item for its preferred size
- Grow the component to perfectly contain each item
- A bottom-up approach where top-level component's size completely dependent on its contained components
- Example `LayoutManagers` in Java that use this strategy
 - `BoxLayout`, `FlowLayout`
- Examples of use in interface design?
- Special needs?

Variable Intrinsic Size

- Layout determined in bottom-up and top-down phases
- Bottom-up phase:
 - Container asks each child for its preferred, minimum, maximum sizes
 - Values used to partition the container's space
- Top-down phase:
 - Children are sized and told to lay themselves out in space specified
- Example `LayoutManagers` in Java
 - `GridBagLayout`
 - `BorderLayout`

Struts and Springs

- Layout specified by marking aspects of components that are fixed vs. those that can “stretch”
- Strut defines a fixed length (width/height)
 - Specifies invariant relationships in a layout
- Spring defines a space that “pushes” on nearby edges
 - Specifies variable relationships
 - Called “Glue” in Java
- Example LayoutManagers in Java
 - SpringLayout



Struts and Springs Uses

- One of the most common strategies, especially in user interface builders
- Provides easily accessible metaphors for people performing layout
- Difficult to layout by hand



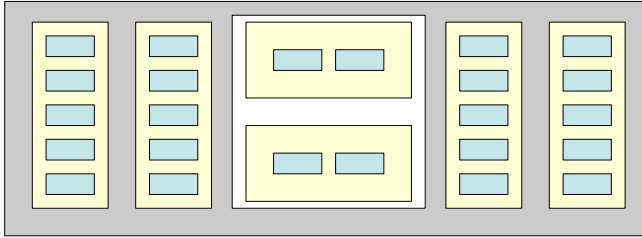
Tips and Strategies

- `javax.swing.Box` has number of useful items that can be used in *any* layout manager
 - “Glue”
 - `Box.createHorizontalGlue()`
 - `Box.createVerticalGlue()`
 - Similar to notion of “springs”: Expands to fill space
 - Struts
 - `Box.createHorizontalStrut()`
 - `Box.createVerticalStrut()`
 - Rigid areas
 - `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



Tips and Strategies

- Define your own layout manager if necessary

```
public interface LayoutManager
{
    void addLayoutComponent(String name, Component comp);
    void removeLayoutComponent(Component comp);
    Dimension preferredLayoutSize(Container parent);
    Dimension minimumLayoutSize(Container parent);
    void layoutContainer(Container parent);
}
// LayoutManager2 has methods for specifying constraints
```

Constraints

- Specify the mathematical relationships between components of the interface.
 - All of the layout managers have constraints to some degree.
 - This is meant to be more general.
- Prefuse takes it to a new level
 - Demo
 - AggregateDemo
 - GraphView
 - Fisheye Menu
 - TreeView
 - See prefuse.org for downloads, videos, etc.
 - Importing into Eclipse is particularly straight-forward if you follow the instructions!