**Lecture 10: Declarative UI**

Topics
- L1: usability
- L2: user-centered design, user & task analysis
- L3: MVC, observer, view hierarchy
- L4: component, stroke & pixel models, redraw, double-buffering
- L5: perception, cognition, motor, memory, vision
- L6: events, dispatch & propagation, finite state controllers, interactors
- L7: interface styles, direct manipulation, affordances, mapping, visibility, feedback
- L8: Nielsen’s heuristics
- L9: paper prototyping, fidelity, look/feel, depth/breadth, computer prototyping, Wizard of Oz
- L10: automatic layout, layout propagation, constraints, model-based user interfaces

Everything is fair game
- Class discussion, lecture notes, readings, assignments
- Closed book exam, 80 minutes

---

**Today’s Topics**

- Automatic layout
- Constraints
- Model-based UI

---

**Quiz on Monday**

- Topics
  - L1: usability
  - L2: user-centered design, user & task analysis
  - L3: MVC, observer, view hierarchy
  - L4: component, stroke & pixel models, redraw, double-buffering
  - L5: perception, cognition, motor, memory, vision
  - L6: events, dispatch & propagation, finite state controllers, interactors
  - L7: interface styles, direct manipulation, affordances, mapping, visibility, feedback
  - L8: Nielsen’s heuristics
  - L9: paper prototyping, fidelity, look/feel, depth/breadth, computer prototyping, Wizard of Oz
  - L10: automatic layout, layout propagation, constraints, model-based user interfaces

- Everything is fair game
  - Class discussion, lecture notes, readings, assignments
  - Closed book exam, 80 minutes

---

**Declarative vs. Procedural**

- **Declarative programming**
  - Saying *what* you want
- **Procedural programming**
  - Saying *how* to achieve it

**Declarative**
A tower of 3 blocks.

**Procedural**
1. Put down block A.
2. Put block B on block A.
3. Put block C on block B.
Example: Automatic Layout

- Layout = component positions & sizes
  - Sometimes called geometry
- Declarative layout
  - Declare the components
    - Java: component hierarchy
  - Declare their layout relationships
    - Java: layout managers
- Procedural layout
  - Write code to compute positions and sizes

Reasons to Do Automatic Layout

- Higher level programming
  - Shorter, simpler code
- Adapts to change
  - Window size
  - Font size
  - Widget set (or theme or skin)
  - Labels (internationalization)
  - Adding or removing components

Layout Managers

- Layout manager performs automatic layout of a container’s children
  - 1D (BoxLayout, FlowLayout, BorderLayout)
  - 2D (GridLayout, GridBagLayout, TableLayout)
- Advantages
  - Captures most common kinds of layout relationships in reusable form
- Disadvantages
  - Can only relate siblings in component hierarchy

Layout Propagation

computePreferredSize(Container parent)
for each child in parent,
  computePreferredSize(child)
compute parent’s preferred size from children
  e.g., horizontal layout,
  (prefwidth, prefheight) = (sum(children prefwidth),
  max(children prefheight))

layout(Container parent) requires: parent’s size already set
apply layout constraints to allocate space for each child
  child.(width, height) = (parent.width / #children, parent.height)
set positions of children
  child[i].(x, y) = (child[i-1].x + child[i-1].width, 0)
for each child in parent,
  layout(child)
How Child Fills Its Allocated Space

- space allocated to child
- child's actual size & position

Expanding: OK
Padding: OK
Anchoring: northwest, centered

How Child Allocations Grow and Shrink

- some children are fixed-size
- other children grow & shrink with available space in parent

- strut: invisible, fixed-size component used for adding whitespace between child allocations
- glue: invisible, growable component used for right-justification

Using Nested Panels for Layout

Constraints

- Constraint = relationship among variables
  - Automatically maintained by system
- Constraint propagation: When a variable changes, other variables are automatically changed to satisfy constraint
Using Constraints for Layout

```
label1.left = 5
label1.width = textwidth(label1.text, label1.font)
label1.right = textbox.left
label1.left + label1.width = label1.right

Textbox

label2.right = parent.width
```

Using Constraints for Behavior

- **Input**
  - `checker.(x,y) = mouse.(x,y)`
  - `if mouse.button1 && mouse.(x,y) in checker`
- **Output**
  - `checker.dropShadow.visible = mouse.button1 && mouse.(x,y) in checker`
- **Interactions between components**
  - `deleteButton.enabled = (textbox.selection != null)`
- **Connecting view to model**
  - `checker.x = board.find(checker).column * 50`

Constraints Are Declarative UI

```
scrollbar.thumb.y
scrollbar.track.height = scrollbar.thumb.height

scrollbar.thumb.y
scrollbar.track.height = scrollbar.thumb.height
```

Model-Based User Interfaces

- **Programmer writes logical model of UI**
  - State variables (bool, int, string, list)
  - Commands
- **System generates actual presentation**
  - Grouping into windows, tabs, panels
  - Widget selection
  - Layout
- **Same motivation as other declarative UI**
  - Higher-level programming
  - Adapting to change; particularly for devices and users
    - Screen size (watch, phone, PDA, laptop, desktop, wall)
    - Widgets available (phone vs. desktop)
    - Input style (mouse vs. arrow buttons; speech, finger, pen)
    - Output style (speech vs. display)
    - User behavior (uses some components more)
**UIML Approach**

- Programmer writes XML spec for both model and view
  - Model: `<description>`
  - Grouping: `<structure>`
  - Labels: `<data>`
  - Widget selection & layout: `<style>`
  - Behavior: `<events>`
- Separation of concerns allows managing families of interfaces
  - Reuse application parts for multiple devices
  - Reuse device parts for multiple applications

**SUPPLE Approach**

- Application model
  - Elements: state variables and commands
  - Tree structure: grouping
  - Labels for each element
- Device description
  - Widget set
  - Navigation costs (switch, enter, leave)
  - Manipulation costs (changing value)
- User data
  - Trace of actions by a user
- System automatically searches for a presentation
  - Assignment of widgets to model elements that minimizes cost of user trace