Lecture 10:

UI Implementation Concepts: Callbacks, Resources, Widget Hierarchies, Geometry Management

05-431/631 Software Structures for User Interfaces (SSUI)

Fall, 2022

Widgets as objects

- Many features of programming languages were added to support UI Toolkits
  - Even Object-Oriented itself – from Smalltalk – (previous lecture)
- In Unix: Motif and Tk each widget is at least one window
  - Since windows already have mechanisms for mouse enter/leave, etc.
    - But high overhead
  - In most other toolkits, widgets are not windows
- Decorative lines, labels and boxes also are "widgets“

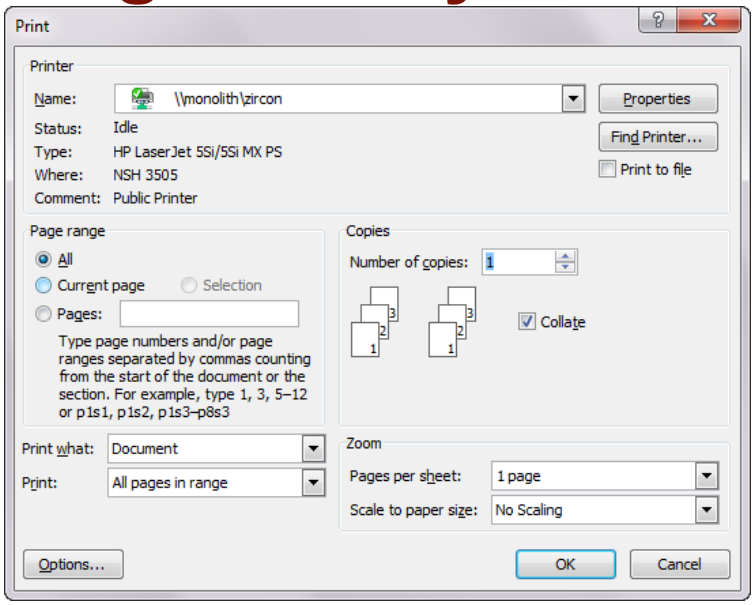
Intrinsics

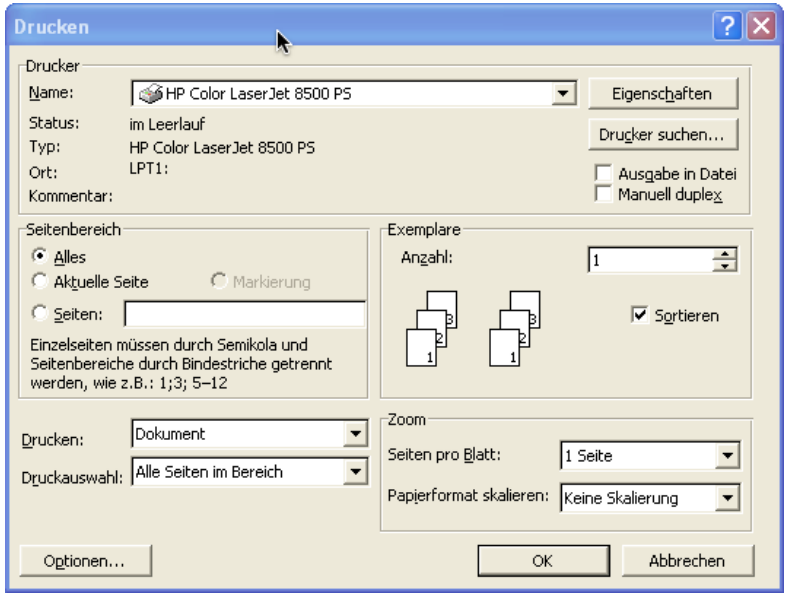
- How the widgets are implemented
- Original Mac toolbox (1984) - call `GetNewMBar` to create a menu, and then `SetMenuBar` to fill its items, followed by `DrawMenuBar` to actually display it
- When “C” was the key language, people still wanted OO
  - C++ (1982) had limitations and wasn’t available for free
  - Unix Motif -- made a "fake" object system out of C, called Xt intrinsics
    - Same in Andrew toolkit
  - Tk -- Tcl language, and descendants
  - Amulet – we created a new Prototype-instance object system, constraints, Opal graphics model, Interactors input model, command objects
- Java (1996) – OO to fix some flaws in C++
  - Make more reliable, also to be better for UIs
- Raw html – some widgets built-in
  - `<input …>`
  - Buttons, menus, checkboxes, radio buttons, text input fields, …
- JavaScript object model
  - Very dynamic
  - Became more like Java with introduction of `Class` in 2015

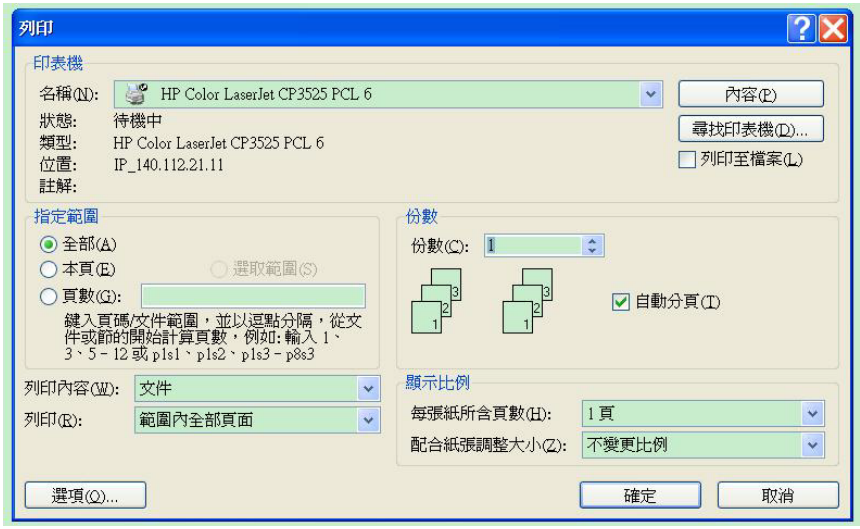
"Resources"

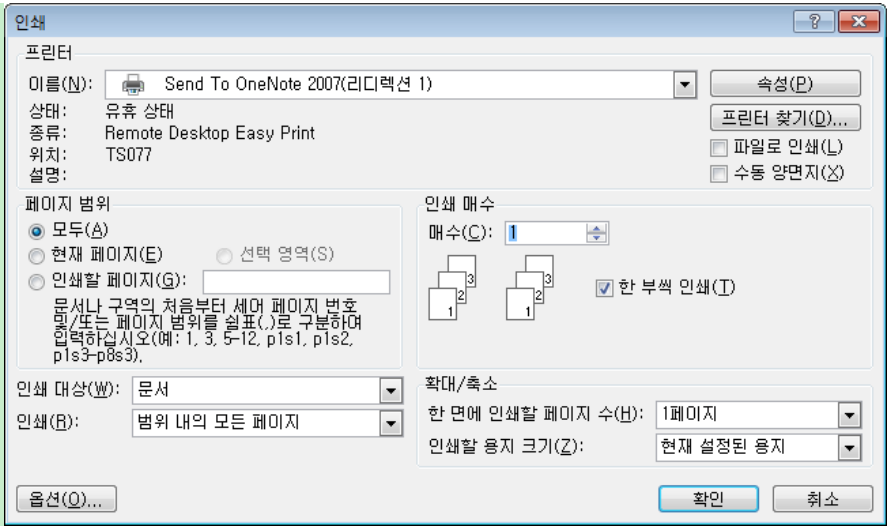
- Starting from original Macintosh as "resource fork" – for language independence (& better memory management)
  - All natural language strings and their positions in a separate file
- Every parameter of widgets in Motif
  - Passed as a parameter to the create routine, set afterwards, or read from a configuration file
- Called "options" by Tk
- Each resource has a default value defined by the class
- In an X file =
  appl.widget1.resource: value
  appl.widget1.widget2.resource: value
  *.resource: value
- Many now use XML variants
  - XAML language from Microsoft for .Net resources
    - Extensible Application Markup Language

Dialog Box Layouts: Print

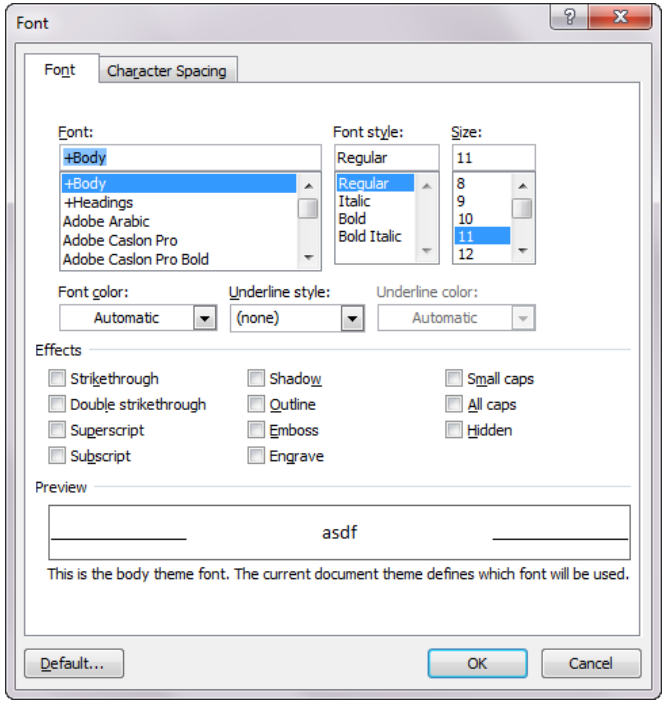


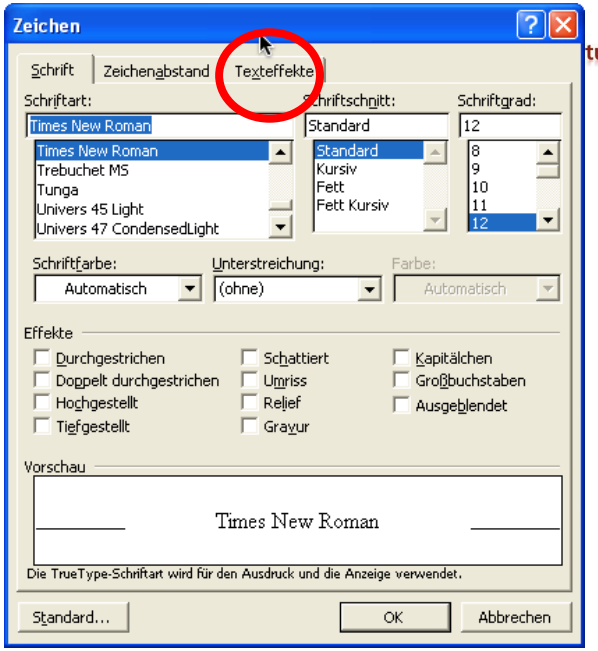


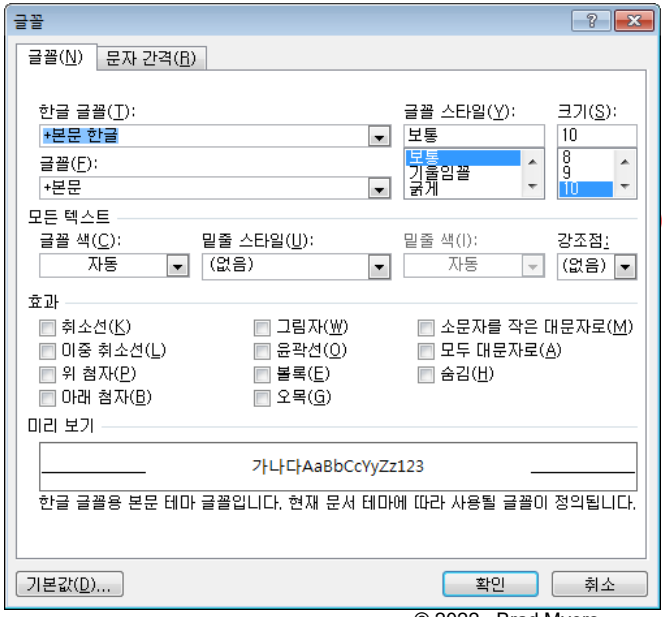


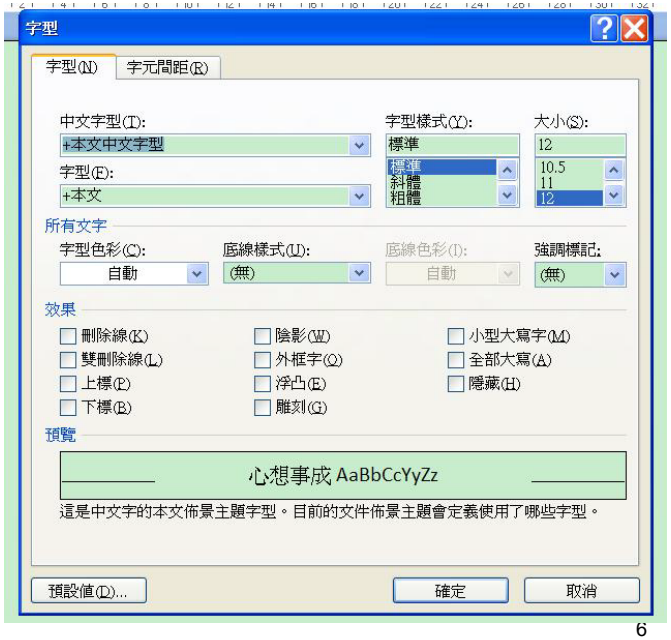


Dialog Box Layouts: Fonts

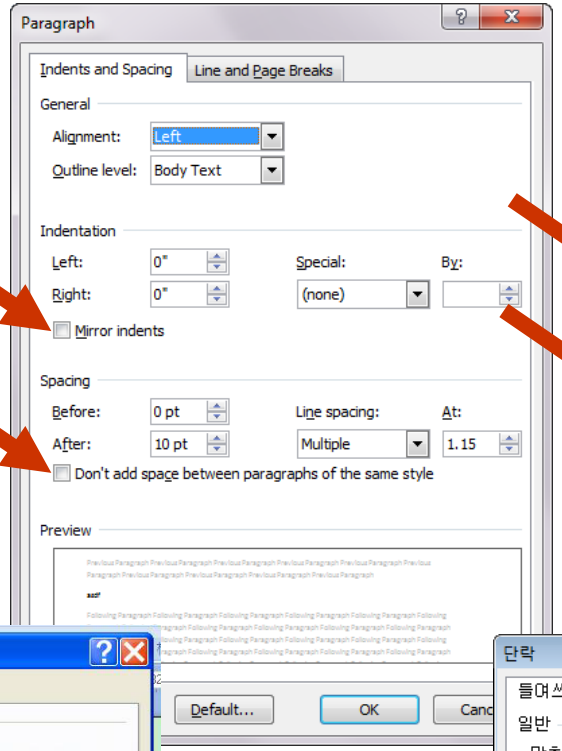


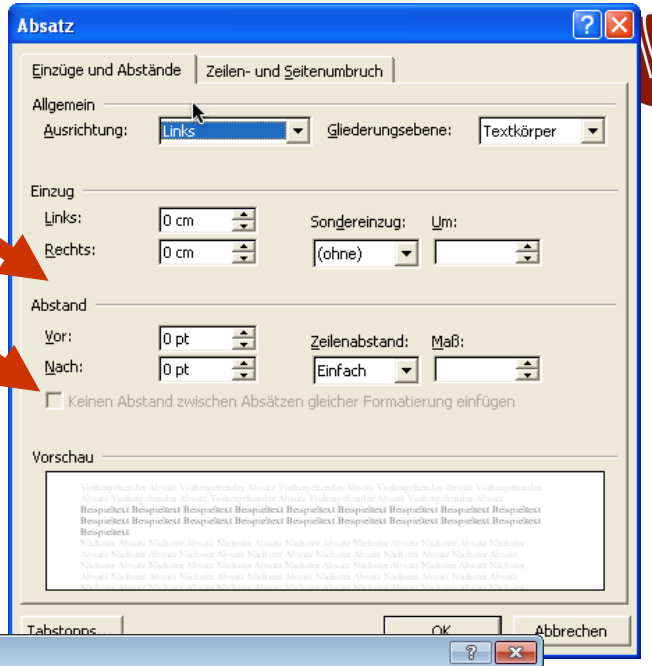


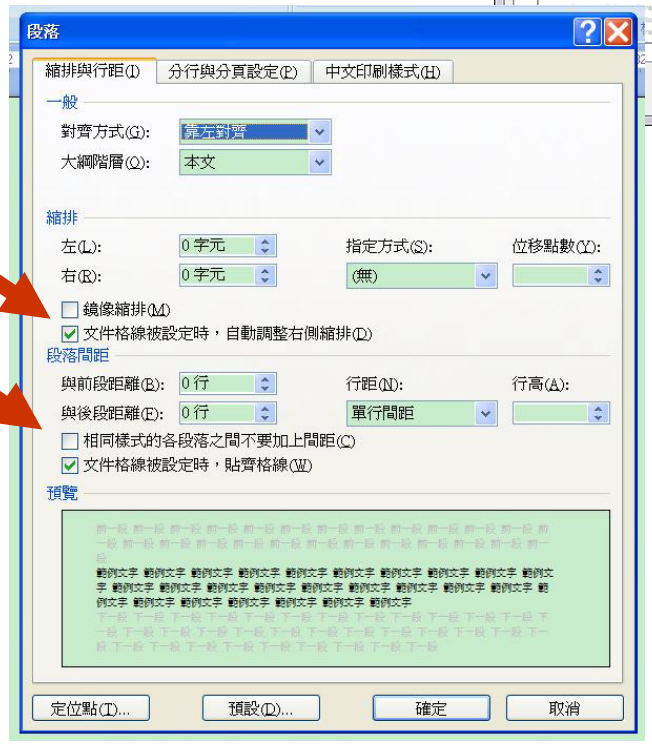


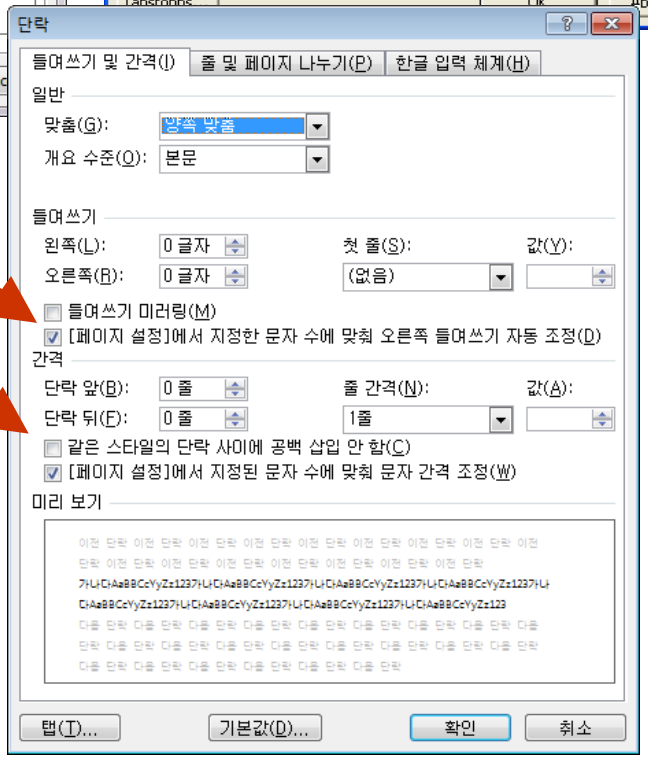


Dialog Box Layouts: Paragraph









Resources, cont.

- Advantages
  - Easier to change languages
  - UI designer can do layout and wording
  - Can give appropriate names to the IDs
  - Easier for interactive editors to save to separate resource file
- Disadvantages
  - Not always a good separation
    - E.g., when need whole new commands
    - Right-to-left languages require code to be different
  - Code is harder to write
    - Have to keep looking up the IDs to use
  - Code is harder to debug and understand
    - Names don't appear in the code, so hard to search for

Callbacks

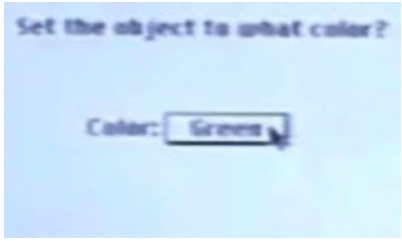
- In JavaScript – event handler functions
- In Motif, associate C procedures with widgets
  - Many different callbacks for the same widget
    - create, start, abort, finish, destroy, ...
  - Registered (set) at widget creation time, invoked at run time
  - Are "resources"
  - There are also "actions" which are internal to the widget and called by events
- In VB, “event handlers”
  - Button: click, focus-in/out, change, etc.
- In Amulet, invoke Command Objects on "interactors" or widget finish, and call-back is the DO method.

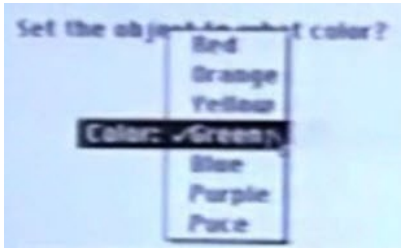
HTML call-back model

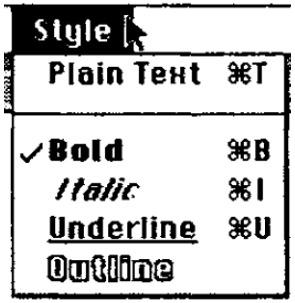
- Form controls (buttons, text field) have various events or “actions”
  - Built-in actions like “submit”
  - Attach script code to run when used
    ```
    <button type="submit" formaction "myfunction">
    ```
- ReactJS – like regular JavaScript
  - Has “state” variables that are shared (“reactive”)
- AngularJS supports “data bindings”
  - Like Amulet constraints
  - Connects JavaScript variable to html element
  -
  - Can be two-way

Widget Hierarchies

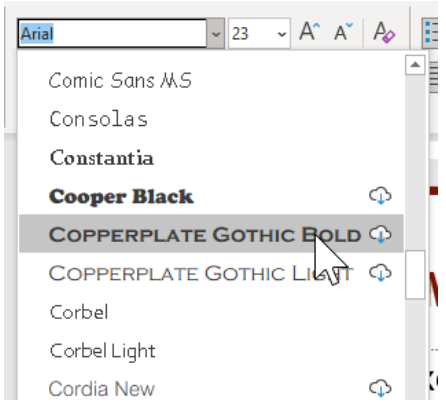
- Inheritance to give the right methods to widgets
  - Menu in option button should look and work the same as a pull-down menu
- Not used in JavaScript
- Functions down the parent or class hierarchy
- Java swing hierarchy:
  - https://docs.oracle.com/javase/10/docs/api/index.html?javax/swing/package-tree.html
  - (similar for Java v.18, but reorganized)
  - 6002 interfaces, classes, subclasses; up to 8 levels deep!
  - 68 different classes and interfaces with the word “menu” as part of the name
    - Rarely use the simple one (2 with the name “menu”)
- Separate hierarchies for internal look-and-feel classes
  - Visible when debugging







Geometry Management



- Simplest – fixed values for location and size of each widget
  - Some widgets have *dynamically*-determined content
    - E.g., fonts menu, width=widest font name
  - Doesn't work for changing languages; changing window size
- Widgets don't set their own location.
  - Widgets put into special group objects called "geometry managers" that perform the layout by setting the component's positions and size
- Each widget negotiates with parent for more room when resize

JavaScript Layout

- Html/CSS placements:
  - Default – flows with text – next position or next line
  - Also, fixed location, percent of the way across page, offset relative to previous item
- CSS Layouts
  - CSS framework (library), like Bootstrap
  - CSS float property – move to sides
  - CSS grid container – rows and columns
  - CSS flexbox container – deals with changing window size

Independent Constraints

- Programmer defines a set of **constraints**
  - Mathematical relationships among the objects that compute the position, size, etc.
  - E.g.: width of menu = width of its widest item
    width of ribbon = width of window – 4
    OK-Cancel buttons are centered at the bottom of the window

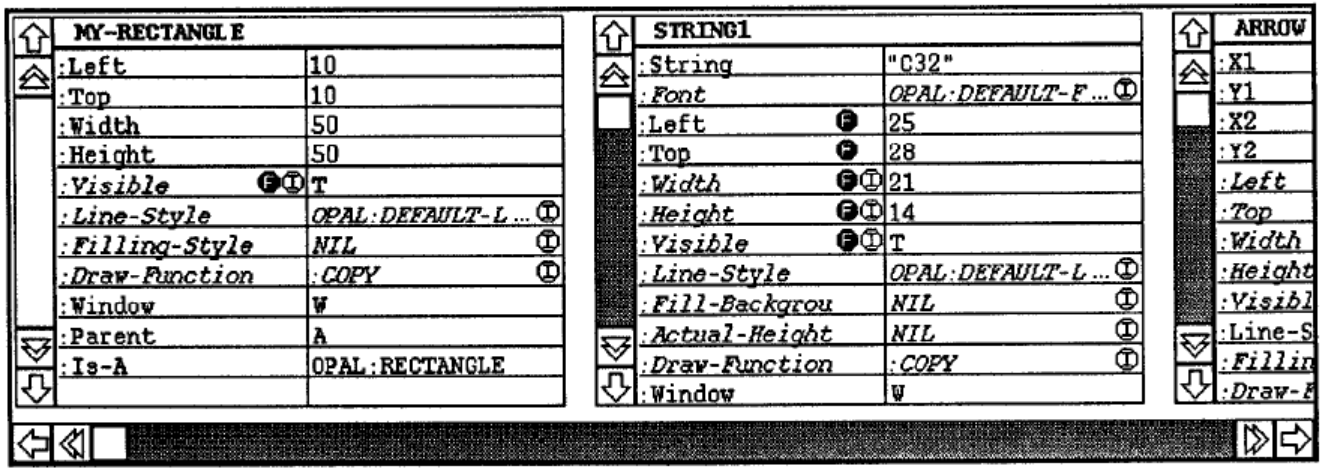
- Idea dates back to Sketchpad research system – 1963!
- Covered in depth in Lectures 16-18

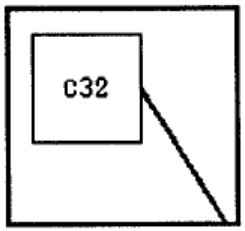
Amulet geometry management

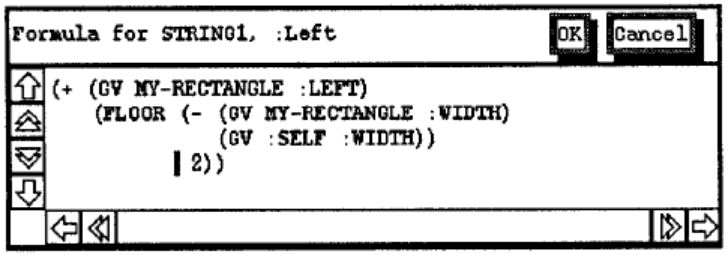
- Combination of arbitrary constraints and containers
- Group can have the Am_LAYOUT slot set with a constraint that depends on other slots
  - Sets positions of parts by side effect
  - Default layout routines: Horizontal and Vertical layout, for lists or tables.
- Rest done by arbitrary constraints

Our old research on layout: C32

- Brad A. Myers. "Graphical Techniques in a Spreadsheet for Specifying User Interfaces," Proceedings SIGCHI'91: Human Factors in Computing Systems. New Orleans, LA. April 28-May 2, 1991. pp. 243-249. ACM ref. See also YouTube video or video

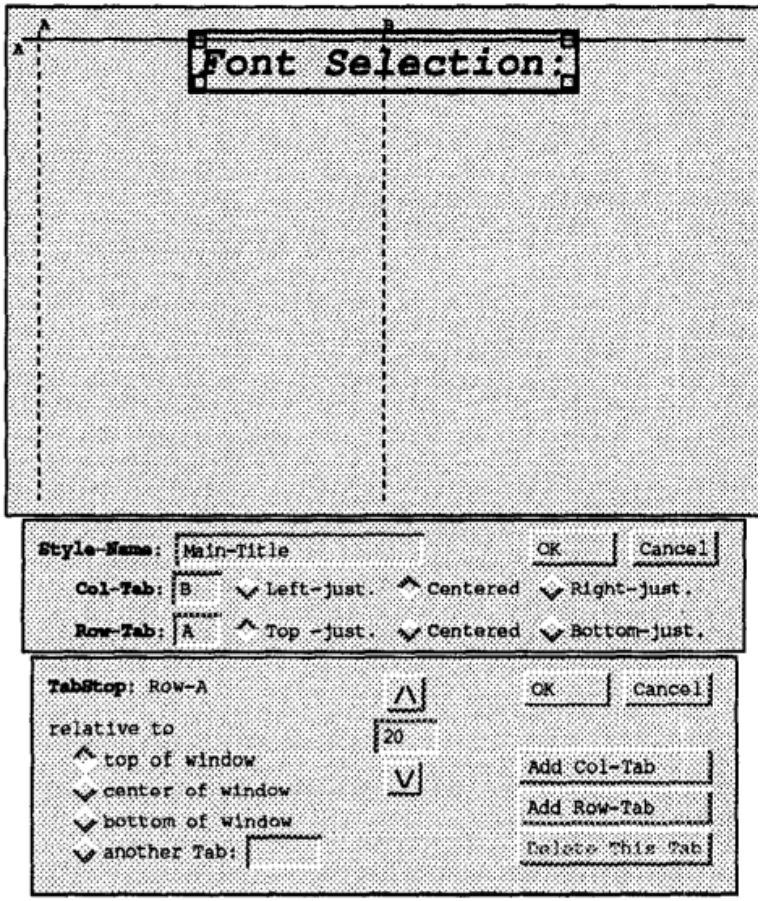


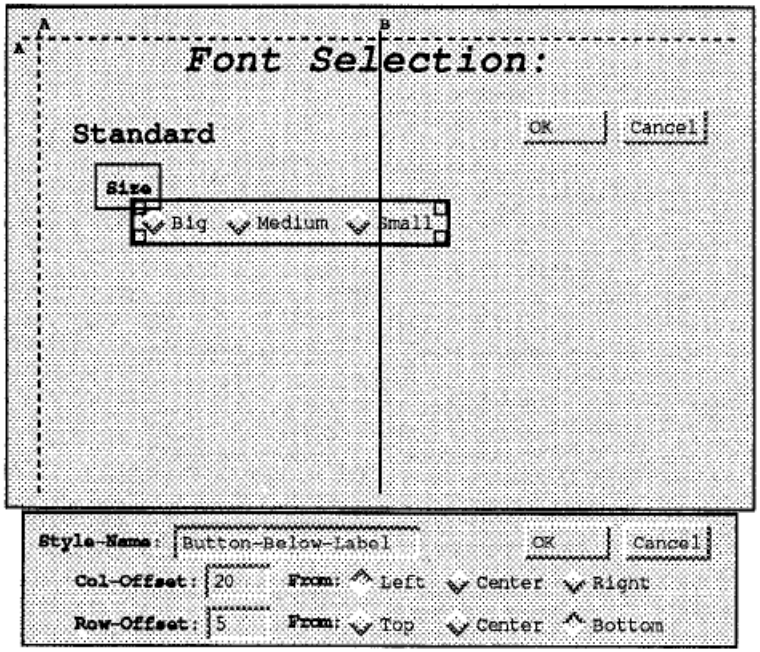




Our old research on layout: Gilt

- Osamu Hashimoto and Brad A. Myers. "Graphical Styles For Building User Interfaces by Demonstration," *ACM Symposium on User Interface Software and Technology: UIST'92*, Monterey, CA, Nov. 16-18, 1992. pp. 117-124. pdf.

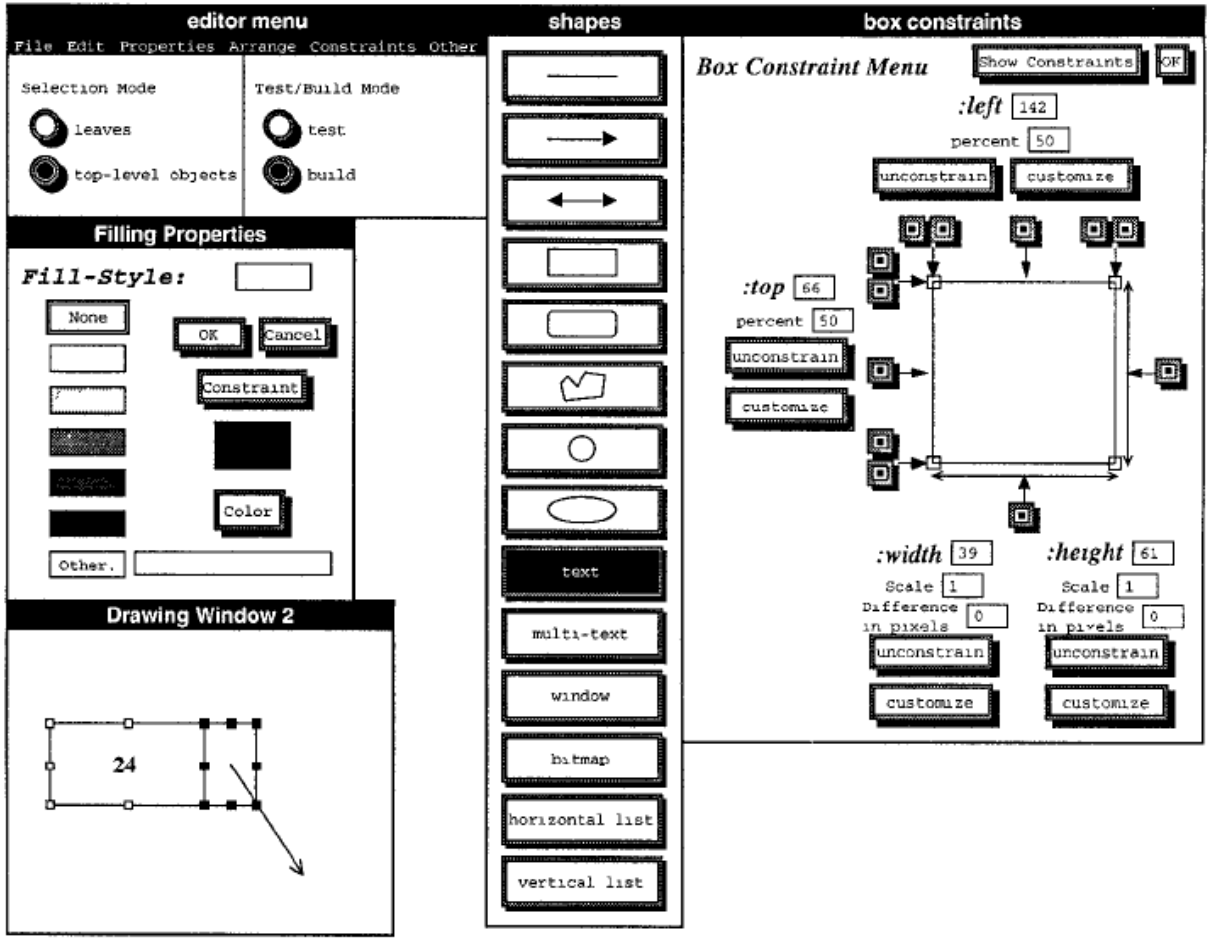




YouTube video. (*9:01*)

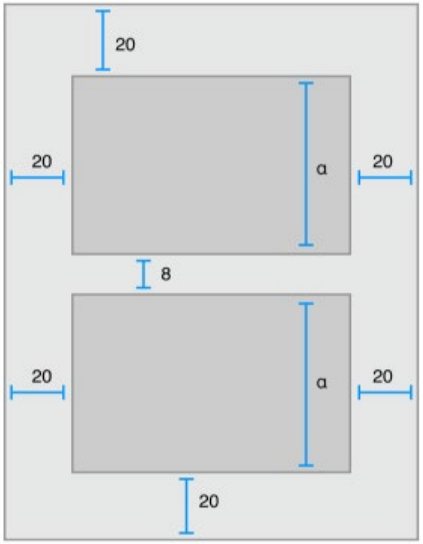
Our old research on layout: Lapidary

- Brad Vander Zanden and Brad A. Myers. "Demonstrational and Constraint-Based Techniques for Pictorially Specifying Application Objects and Behaviors," *ACM Transactions on Computer-Human Interaction*. Vol. 2, no. 4, Dec, 1995. pp. 308-356.



iOS

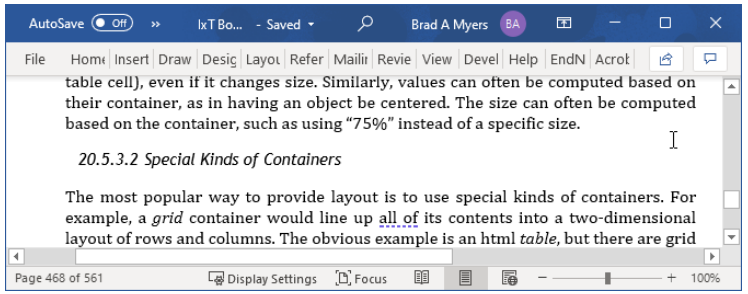
- Can manually place elements of UI
- "Auto Layout"
  - Constraint-based layout system
  - Helps with adaptive UI
- Can set up constraints in the interface builder or programmatically

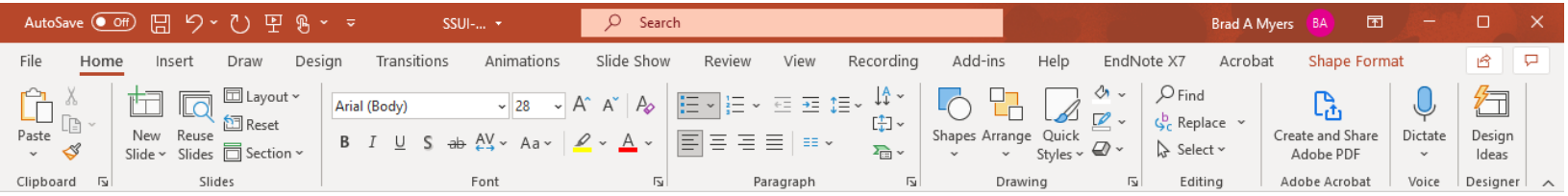


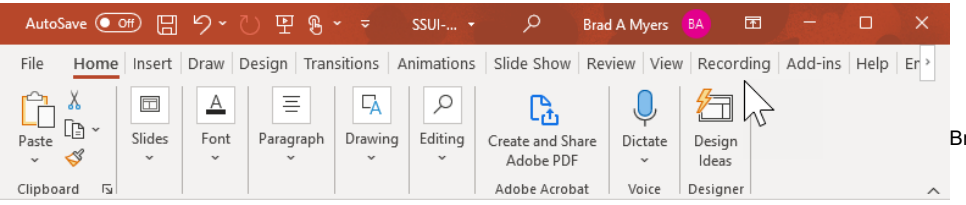
cite

Special Containers

- Put widgets in a container that will place them appropriately
- Simple: Horizontal row or vertical column
  - OK for menus, but little else
- Grid (table) – doesn't cover much
- Java GridBagLayout – more general but too hard to use
- Doesn't support multiple-pass calculations
  - E.g., items centered in a menu depend on menu width which depends on widest item
- Allowed size might be different than desired size
  - E.g., Microsoft ribbon changes the widgets based on *allowed* size







Motif Geometry Management

- Motif (the “winning” X toolkit)
  - RowColumn - add widgets and it lays them out
  - Treats all children the same, so not for ScrollBars
- Form - generic constrained layout
  - Put extra resources on the children widgets
  - "For details, see the Motif Reference Manual, because the complete behavior of Form is quite complicated."
  - Each edge can be constrained
    - at a position or offset from an edge of the Form
    - at an offset from an edge of another widget
    - percent of the way across the Form (edge, not center)
    - a percent calculated based on the initial position
  - If wrong, widgets are on top of each other

Java Widget Layout

- "Using Layout Managers"
- BorderLayout – layout around the edges, center gets extra space
- BoxLayout – vertical or horizontal columns
- CardLayout – overlapping JPanels
- FlowLayout – fills row, then goes to next row
- GridLayout -- components in a grid of cells. Resizes children to fill cell
- GridBagLayout – "one of the most flexible — and complex — layout managers the Java platform provides…. uses a grid of rows and columns, allowing specified components to span multiple rows or columns
  - Funny GridBagLayout video "Totally Gridbag" by Matt Quail (July 2004) https://www.youtube.com/watch?v=UuLaxbFKAcc (*2:42*)
- GroupLayout – new, designed for use by IBs
- SpringLayout -- also new for IBs, constraints for layouts

Funny video

Note: This lesson covers writing layout code by hand, which can be challenging. If you are not interested in learning all the details of layout management, you might prefer to use the `GroupLayout` layout manager combined with a builder tool to lay out your GUI. One such builder tool is the NetBeans IDE. Otherwise, if you want to code by hand and do not want to use `GroupLayout`, then `GridBagLayout` is recommended as the next most flexible and powerful layout manager.

Android

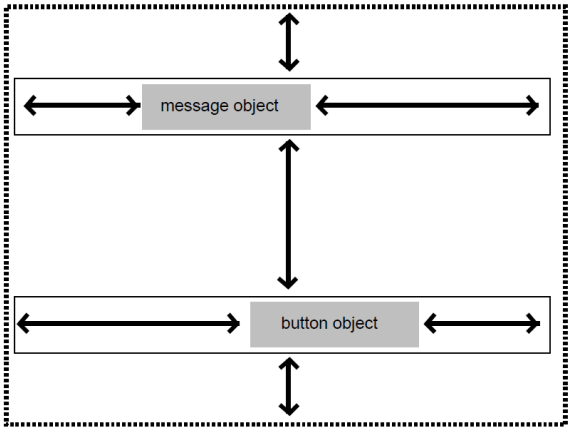
- Similar to Java Swing
- Layout: Different kinds of containers that lay out children
  - Linear, grid, etc.
  - Attach layouts to “view”s

Spacer model

- History: TeX layout model
- Typesetting system designed and mostly written by Donald Knuth starting in 1978
- Boxes (of text) connected by “glue”
  - \vspace also between characters, etc.
- Can control the “stretchiness” of the glue

History: Interviews layout model

- "Interviews" – one of the first C++ toolkits
  - Linton, M.A., Vlissides, J.M., and Calder, P.R., "Composing user interfaces with InterViews." *IEEE Computer*, Feb, 1989. 22(2): pp. 8-22.
- Adopted the TeX boxes and glue metaphor
- hbox tiles its components horizontally
  - hglue
- vbox tiles them vertically
  - Vglue
- Controls have a "natural" size
- Different glues and controls have different stretchiness



"Struts and Springs" layout

- For stretchy or rigid constraints
- Graphical interface layouts
- NeXTStep (1989), MacOS & Galaxy (~1992)