Bringing it all together: implementing different interaction techniques

Week 7 Lecture
20.02.2008

Lyn Bartram
lyn@sfu.ca

SCHOOL OF INTERACTIVE ARTS + TECHNOLOGY

Today's agenda

- Correct assignment information (and an OOPS)
- Review of interaction techniques and the event model
- Interaction techniques in the wild : does size matter?
- ….
- And if it does, how do we code for it?
- The Model-View-Controller design pattern

SCHOOL OF INTERACTIVE ARTS + TECHNOLOGY

Administrivia

- Assignment 2 is DUE tonight not march 3
- Assignment 3 comes out later today
  - Due a week from Friday
  - We'll go over the details in the tutorial on Friday
- If you have trouble with the assignments
  - Make an appointment or come in office hours
  - Email at the last minute is difficult to debug
- You may not learn exactly what to do from me
  - You will learn how to find out what to do

Recap: interaction techniques

- A method for carrying out a specific interactive task
  - Example: enter a number in a range
    - Could use a (simulated) slider
    - (simulated) knob
    - Enter text in a text edit box
    - Select a number from a menu
  - Each is a different interaction technique

- We’ve seen how there are a number of java interactors that we can use for these techniques
- All based on event model
  - Is that all we need?

- How many different interactors do we need?

Logical device approach: recap

- Fixed set of categories
  - Core graphics standard
  - keyboard, locator, valuator, button, ….
  - Pick, stroke basic operations

- Simulated in software
  - Valuator -- > slider
  - 3D locator --> 3 knobs/buttons

Logical device approach

- Useful abstraction
- But sometimes abstracts away too many details
  - Some are important
  - Example: mouse vs. pen on PDA
  - Both are locators
  - What’s the big difference?

Logical device approach has limitations

- But still useful to think in terms of “what information is returned”
- Catgorisation of devices
  - Two broad classes emerged
  - **Event** devices
  - **Sampled** devices

Categorising devices

- Event devices
  - Time of input is determined by the user
  - Best example: Button
  - When activated, creates an “event record”)
    - Record of significant action

Categorising devices

- Event devices

- Sampled devices
  - Time of input is determined by the program (more specifically, the computational environment)
  - Best example: valuator or locator
  - Value is constantly updated (considered **continuous**)
  - Program asks for current value when it needs it

Categorising devices

- Event model manages both
  - Treats everything as an event
  - Sampled devices are handled with “incremental change” events
  - Each measurable change or defined input produces an event containing the new value
  - Program keeps track of the current value if it wants to sample

  - Issues?

Simulated sampling can cause problems

- Lots of little events

- Can fall behind if each event requires substantial computation (for example, a redraw)

- Sampling provided a built-in throttle

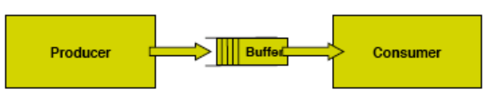
- Events can “get ahead of us”

The producer-consumer model

- User and system are inherently acting in parallel
  - Asynchronously
  - Leads to callback/listener model you use in Swing
- This is a producer-consumer problem
  - User or device produces events
  - System consumes them

Synchronisation

- Need to synchronise activity between producers and consumers
- Need to make sure we don't lose events
- Solution: use a queue or buffer



- As long as buffer does not overflow, producer doesn't need to block
- Consumer operates on events when it can

Implications

- We are really operating on events from the past
  - Hopefully the recent past!
- But sampled input is from the present
  - Mixing them can cause problems
  - E.g., inaccurate position at end of a drag operation
  - A particularly important problem for sensing devices!!
- More generally, very sensitive to type of device and interaction

Case study: rubberbanding

- Suppose we want to implement an interaction for specifying a line
- We could just specify 2 endpoints
  - Click .. Click
  - Not great: no affordances, no feedback
- Better feedback is to use "rubberbanding"
  - Stretch out the line as you drag
  - At all times, show where you would end up if you let go.
  - Can we improve affordance?
    - Changing cursor shape is about all we have to work with

SCHOOL OF INTERACTIVE ARTS + TECHNOLOGY

Implementing rubberbanding

```
Accept the press for endpoint p1;
P2 = P1;
Draw line P1-P2;
Repeat
    Erase line P1-P2;
    P2 = current_position();
    Draw line P1-P2;
Until release event;
Act on line input;
```

- Need to get around this loop MINIMUM 5 times/sec
  - 10 is better
  - More even better
  - (note we need to draw and undraw here)
- What's wrong with this code?

Implementing rubberbanding


```
Accept the press for endpoint p1;
P2 = P1;
Draw line P1-P2;
Repeat
    Erase line P1-P2;
    P2 = current_position();
    Draw line P1-P2;
Until release event;
Act on line input;
```


- Not event-driven!
- Not in the basic event-redraw cycle form
  - Hard to mix event and sampled
  - Can't ignore events for arbitrary lengths of time in some applications
- How do we do this?

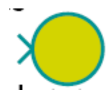
Simulate control flow

- Chop up actions in the code and redistribute/remodel them as events
- “event driven control flow”
- Key point: you need to maintain STATE (where you are) between events and start up again in the state where you left off

Finite state machine controllers

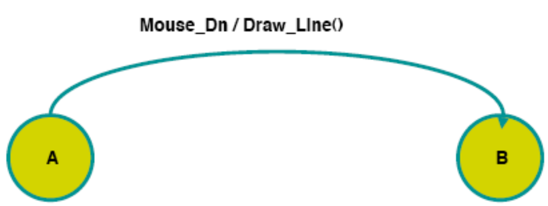
- One good way to maintain state is to use a “state machine”
- FSM - the finite state machine
- Simple model of state-transition-new state

FSM notation

- Circles represent states 
- Arrow for start state 
- Double circle for final state
  - Notion of final state is a little weird for Uis (don't ever "end")
  - Still we use this for complete actions
  - Reset to start state

FSM notation

- Arcs represent transitions

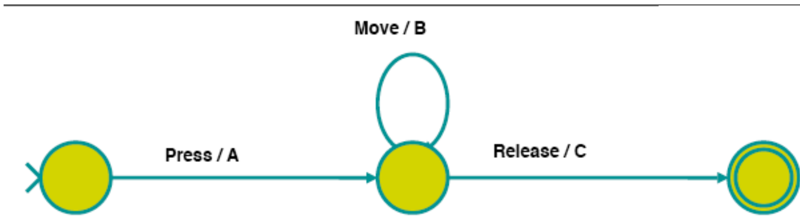


- When you are in state A and see a mouse down, do the action (draw_line) and go to state B

Rubberbanding again

Accept the press for endpoint p1;
A: P2 = P1;
   Draw line P1-P2;
Repeat
B: Erase line P1-P2;
   P2 = current_position();
   Draw line P1-P2;
Until release event;
C: Act on line input;

SCHOOL OF INTERACTIVE ARTS + TECHNOLOGY




```
A: P2 = P1;
   Draw line P1-P2;
B: Erase line P1-P2;
   P2 = current_position();
   Draw line P1-P2;
C:  Act on line input;
```


SCHOOL OF INTERACTIVE ARTS + TECHNOLOGY

From the specific to the general

- Different interactors demand different ways of managing input
- The essential paradigm of drawing a line by specifying the line parameters remains the same
  - Line start, end
- (there's a lesson here)
- Different interaction contexts demand
  - Different interactors
  - Different views

Interaction techniques:

- Does size matter?
- Interaction in the small and in the large: a quick survey

Interaction in the small

- The diversity of small devices is proliferating
- All of them are web-enabled
- There are (about a billion) a lot of them ….

SCHOOL OF INTERACTIVE ARTS + TECHNOLOGY

Small display issues

- Ergonomics
  - Fine motor control
  - Limited Viewing Area
  - Contexts of use
- Tasks
  - Information summaries
  - Mobile access
- Interface Design
  - Re-apply desktop metaphors?
  - Shrink existing interface components ?
  - What needs to be re-thought

SCHOOL OF INTERACTIVE ARTS + TECHNOLOGY

It's not just display size …

- **input devices**
  stylus? phone keypad? jog dial wheel? speech? gesture?
- **context of use**
  people moving around into all sorts of surprising places
- **activity**
  people use small-screen devices for different activities than desktops; don't assume you understand these activities already
- **attention**
  you often don't have people's full attention; they're doing something else while using your device

Different use model

- Attention is a scarce resource
- Competing sources of information in the mix

- Designs demand too much direct attention

Should we export the desktop?

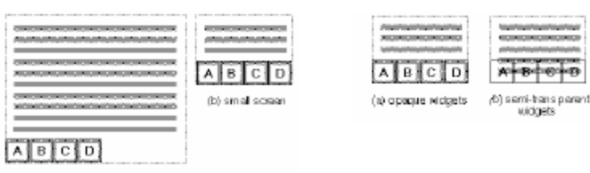
- We have WIMP habits

windows on a Palm | page transformations animated transitions | history / breadcrumbs | accordion summarization

- Use similar direct interactions to select (stroking a word)

Small screens mean …

- Proportionate demands of UI controls are higher
- Interaction is inherently modal



Adpative display and interaction techniques

- Processors in some cases (PDAs) are getting fast enough for adaptive interfaces
- Cell phones are not there yet (and memory limitations are an issue!)

- Techniques from information visualilzation
- Lenses
- Summarisation
- Focus+context
- Translucency
- Zooming and multiscale

Possible solutions

- How can we partition information across multiple screens?
- Can we gain insights from information visualization strategies?
- Can we design the (hardware) device better?
- Desktop metaphors?
- Get implicit information (i.e. contextual information or easy gesture information?
- Utilize other modalities?

Some research in the area …

- Patrick Baudisch and Ken Kinckley at Microsoft Research
- http://www.patrickbaudisch.com/publications/2005-Baudisch-SmallScreens.ppt
- http://research.microsoft.com/users/kenh/

Interface technology
IAT 351

Interaction in the large ….

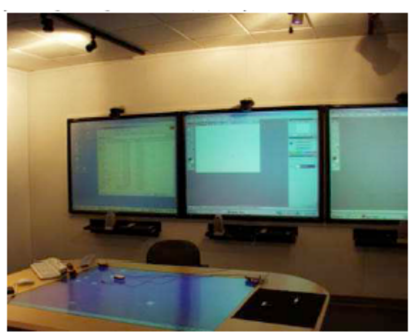
SCHOOL OF INTERACTIVE ARTS + TECHNOLOGY

XeroX LiveBoard (1992)

- Elrod et al. (Liveboard: a large interactive display supporting group meetings, presentations, and remote collaboration. CHI 1992.

Stanford iRoom (1999)

- Brad Johanson, Armando Fox, Terry Winograd. The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms. This is the best overall overview of the project. IEEE Pervasive Computing Magazine 1(2), April-June 2002.



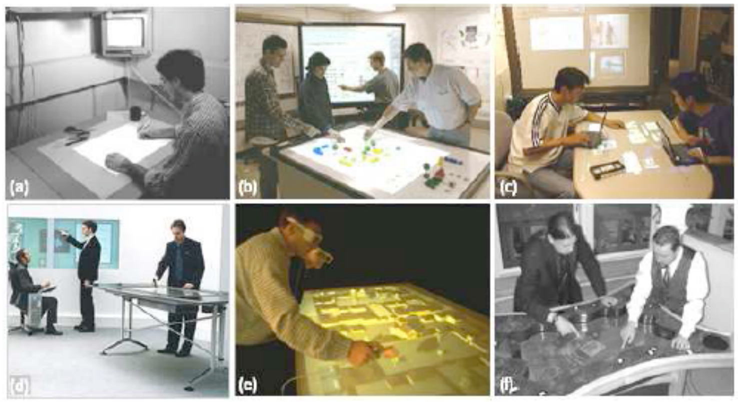
DynaWall (1999)

- Norbert A. Streitz , Jörg Geißler , Torsten Holmer , Shin'ichi Konomi, Christian Müller-Tomfelde , Wolfgang Reischl , Petra Rexroth ,Peter Seitz , Ralf Steinmetz, i-LAND: an interactive landscape for creativity and innovation. CHI 1999.

Roomware (2002)

- Norbert Streitz, Thorsten Prante, Christian Müller-Tomfelde, Peter Tandler, & Carsten Magerkurth. Roomware©: the second generation. CHI 2002

Tabletop Displays



Examples of six tabletop systems: (a) the DigitalDesk (from Wellner, 1993); (b) Envisionment and Discovery Collaboratory (from Arias et al., 1999); (c) the Augmented Surfaces tabletop (from Rekimoto and Saitoh, 1999); (d) the InteracTable (from Streitz et al., 2002); (e) the Responsive Workbench (from Agrawala et al., 1997); and (f) the Pond (from Ståhl et al., 2002).

Large Display experiences

- "Single" or chained wall-size display surface
  - Viewer only/mainly
  - 2-foot and the 10-foot interface
- Large desktop configurations
  - Multiple monitors
  - Panoramic displays

Large Display Issues

- Input
  - Direct vs. Indirect
  - Absolute vs. Relative
  - Proximity
  - Device Ergonomics
  - Number of simultaneous inputs
- Size
  - As size increases, it "becomes qualitatively different" (Swaminathan & Sato, 1997).
  - Large displays more effective for spatial tasks (Tan et al. 2003).
  - However, potential for information to be missed ….

Large Display Issues (2)

- Ergonomics
  - Object placement
  - Proximity
  - Viewing Area : contiguous or distributed
- Tasks
  - Magnify vs. Detail
  - Activity & Social Awareness
  - Individual vs. Collaborative
- Interface Design
- Number of Displays
  - Seamless vs. bezels

Contexts of use

- Personal : Lots and LOTS of work space
  - 20% users now running multiple monitors
- Presentation
  - Remote control
  - Presenter (controller), viewer
- Collaborative
  - Several users
  - Personal and shared space

Major usability issues

- Losing the cursor
  - Users accelerate mouse movements to compensate for screen size
  - Mouse “jumps” and speeding cursorns
- Bezel problems
  - Perceptual frame effect
  - Visual distortion and interaction distortion
- Distal information access problems
  - Difficult and time intensive to access icons, windows, menus
- Window management
  - Windows and menus pop up in unexpected places
  - Don’t’ want windows to cross bezels
  - “Parking” issues

Major usability issues (2)

- Task management
  - We do more because we have more open windows and
  - We have more open windows because we CAN
  - More complex multitasking
  - And more window management …

- Configuration complexity
  - Hard to add and remove monitors
  - Lose windows when a monitor is removed
  - Ghost apps
  - Poor support for hetereogeneity
  - This is a real issue for laptops and mobile users

Some research in the area …

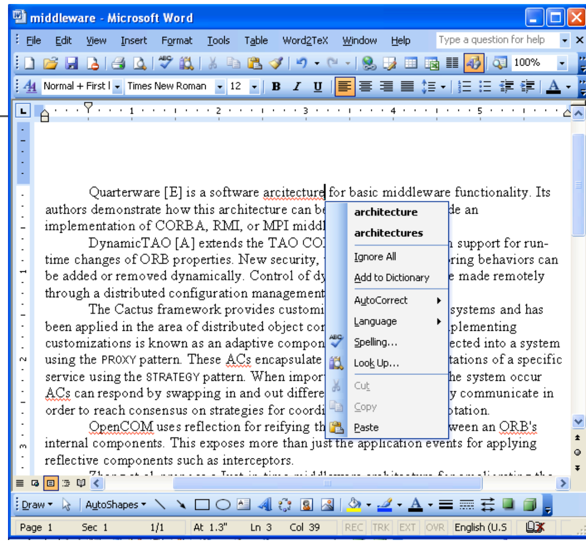
- The VIBE group at Microsoft Research
- http://www.patrickbaudisch.com/publications/2004-Baudisch-LargeDisplays.ppt
- http://research.microsoft.com/vibe/

Problems with GUI programs

- User interfaces are especially prone to changes in requirements
  - New types of input
    - Keyboard
    - Mouse
    - Pen
    - Remote
  - New types of output
    - Porting to different “look-and-feel”
    - Information visualization: charts, graphs, plots
    - Output device heterogeneity: PDAs, applets, Javascript, HTML, Swing

Problems with GUI programs

- New features require user-interface changes
  - User must have a way to access the feature
  - ex. Add in-text spellcheck to word processor



Example: Changes Required for SpellCheck Feature

- Changes to input: Need to input corrections and update the underlying document model
- Changes to underlying document **model**: *None*

The world needs multiple access:

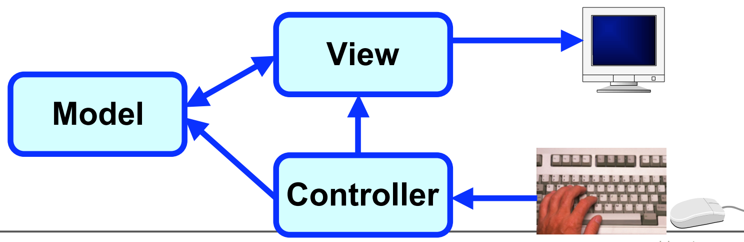
- Many different ways to present and interact with the same underlying information
- Presentation and interaction needs to be only loosely coupled to the underlying computational information abstraction
- How do we implement this?
- Model-View-Controller design pattern

MVC: Model-View-Controller

- Model-View-Controller
  - Architectural pattern for building systems
  - Divide system responsibilities into three parts
    - Model
      - Contains all program data and logic
    - View
      - Visual representation of model
    - Controller
      - Defines system behavior by sending user input to model
  - Step by step
    - User uses controller to change data in model
    - Model then informs view of change
    - View changes visual presentation to reflect change

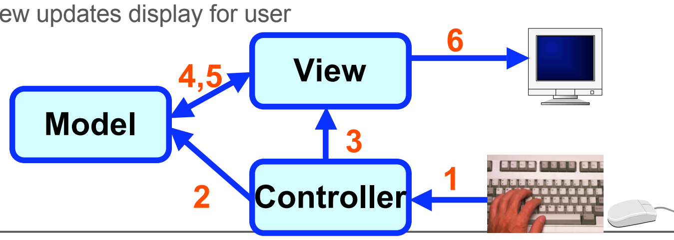
Model-View-Controller (MVC) Pattern

- Developed at Xerox PARC in 1978
- Separates GUI into 3 components
  - Model ⇒ application data
  - View ⇒ visual interface
  - Controller ⇒ user interaction



MVC Interaction Order

1 User performs action, controller is notified
2 Controller may request changes to model
3 Controller may tell view to update
4 Model may notify view if it has been modified
5 View may need to query model for current data
6 View updates display for user

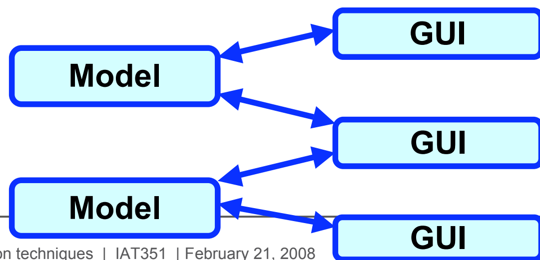


Implementing interaction techniques | IAT351 | February 21, 2008

SCHOOL OF INTERACTIVE ARTS + TECHNOLOGY

MVC Pattern – Advantages

- Separates data from its appearance
  - More robust
  - Easier to maintain
- Provides control over interface
- Easy to support multiple displays for same data



MVC Pattern – Model

- Contains application & its data
- Provide methods to access & update data
- Interface defines allowed interactions
- Fixed interface enable both model & GUIs to be easily pulled out and replaced
- Examples
  - Text documents
  - Spreadsheets
    - provides a number of services to manipulate the data e.g., recalculate, save
    - computation and persistence issues
    - Web browser
  - Video games

MVC Pattern – View

- Provides visual representation of model
- Multiple views can display model at same time
  - Example: data represented as table and graph
- When model is updated, all its views are informed & given chance to update themselves

MVC Pattern – Controller

- Users interact with the controller
- Interprets mouse movement, keystrokes, etc.
- Communicates those activities to the model
- Interaction with model indirectly causes view(s) to update

Principles of GUI Design

- Model
  - Should perform actual work
  - Should be independent of the GUI
    - But can provide access methods
- Controller
  - Lets user control what work the program is doing
  - Design of controller depends on model
- View
  - Lets user see what the program is doing
  - Should not display what controller thinks is happening (base display on model, not controller)

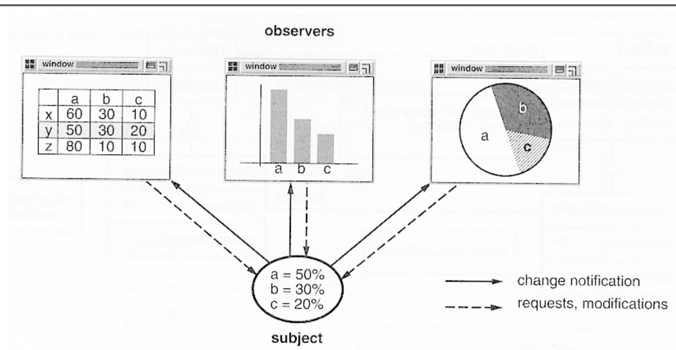
Principles of GUI Design

- Model is separate
  - Never mix model code with GUI code
  - View should represent model as it really is
    - Not some remembered status
- In GUI's, user code for view and controller tend to mingle
  - Especially in small programs
  - Lot of the view is in system code

Do you have a good model?

- Could you reuse the model if you wanted to port the application to:
  - a command-line textual interface
  - an interface for the blind
  - an iPod
  - a web application, run on the web server, accessed via a web browser

Dependencies



Implementing interaction techniques | IAT351 | February 21, 2008

SCHOOL OF INTERACTIVE ARTS + TECHNOLOGY

Dependencies

- Issues
  - need to maintain consistency in the views (or observers)
  - need to update multiple views of the common data model (or subject)
  - need clear, separate responsibilities for presentation (look), interaction (feel), computation, persistence

Fundamental principle

- Separation:
  - you can modify or create views without affecting the underlying model
  - the model should not need to know about all the kinds of views and interaction styles available for it
- How do we do this in Swing?

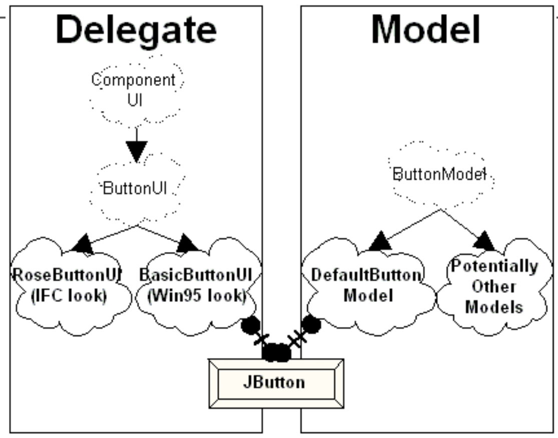
Model/View/Controller

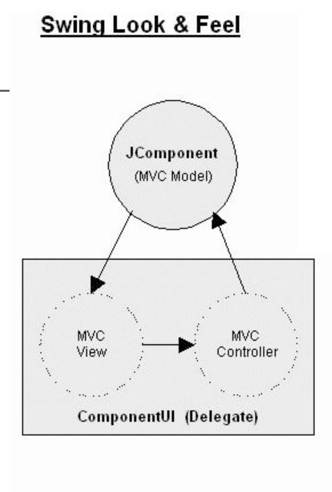
- Java and Swing:
  - concept is still valid to help structure interactive applications
    e.g., use a framework that supports MVC
  - Swing internally uses a variant of MVC for its pluggable look-and-feel capability …

Swing and MVC

- Swing uses MVC variation
  - View/Controller combined into *delegate*
  - View/Controller communication typically complex; delegate simplifies
- Example: Checkbox
  - Has state true/false in Model
  - Screen corresponds to Delegate-View
  - Mouse clicks are handled by Delegate-Controller, sending input to Model

Delegate / Model View





JTree

- Data Model - TreeModel
  - default: DefaultTreeModel
  - getChild, getChildCount, getIndexOfChild, getRoot, isLeaf
- Selection Model - TreeSelectionModel
- View - TreeCellRenderer
  - getTreeCellRendererComponent
- Node - DefaultMutableTreeNode

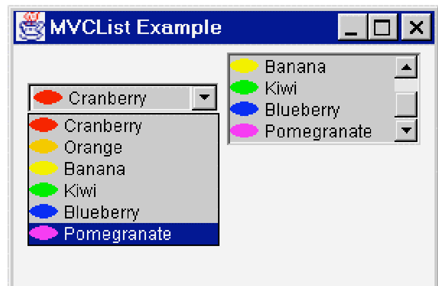
JList

- No longer just text
- Can display Icon
- Can change display line when selected
- Data Model - ListModel
  - default: DefaultListModel
  - getSize / getElementAt (position)
- View - ListCellRenderer
  - getListCellRendererComponent()

SCHOOL OF INTERACTIVE ARTS + TECHNOLOGY

JComboBox

- Data Model - ComboBoxModel
  - Extends ListModel
  - get/set SelectedItem
- Same cell renderer as JList



JTable

- Can just create JTable from data[][] and columnName[] and not worry about anything else
- But …. The model gives you the services of:
- Data Model - TableDataModel
  - default: DefaultTableModel
  - getRowCount, getValueAt, setValueAt, getColumnCount, getColumnName, ...
- View - JTable
  - Contains JTableColumns

JTable Output

Employee ID	First Name	Last Name	Department
0181	Bill	Cat	Political Candid...
0915	Opus	Penguin	Lost and Found
1912	Milo	Bloom	Reporter
3182	Steve	Dallas	Legal
4104	Hodge	Podge	Style
5476	Billy	Boinger	Entertainment
6289	Oliver	Jones	Science
7268	Cutter	John	Travel

Model's for JTables and JList's

- JTable represents a two dimensional array
- JList represents a one dimensional array
- You can create a JTable or JList by just passing an array to the constructor
- Or you can create a model
  - easy and more powerful
  - can handle edits
  - easier to update