CS 683 Emerging Technologies: Embracing Change
Spring Semester, 2001
Doc 24 Pluggable Lists & TextViews

Contents

Pluggable List Example..............................................................2
Source Code ............................................................................2
Pluggable List Example Explained.................................5
Spreadsheet like Views.................................................................7
Solution 1 Different Models.......................................................8
Solution 2 Same Model.................................................................15

Copyright ©, All rights reserved. 2001 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (http://www.opencontent.org/opl.shtml) license defines the copyright on this document.
Pluggable List Example
Source Code
Object subclass: #PluggableListExample
  instanceVariableNames: 'selectedIndex gradeList '
  classVariableNames: "
  poolDictionaries: "
  category: 'Whitney-Examples'

Comment
An example of using a PluggableListMorph. Display a list of grades in a
list that users can select.

PluggableListMorph needs to know three different methods on its
model. The first is the method to get the list to display. The second
method is to get the currently selected in the list. The third is the method
to call to inform the model that the user has selected a different item in
the list.

Structure:
  selectedIndex <integer> the index in gradeList of currently selected
grade
  gradeList <Array or OrderedCollection of Strings> the grades
displayed

Class Methods
new
  ^super new initialize

openAsMorph
  "PluggableListExample openAsMorph"
  ^self new openAsMorph
Instance Methods

gradeList
   "return a list to be displayed"
   ^gradeList

gradeListIndex
   ^selectedIndex

gradeListIndex: integer
   selectedIndex := integer min: gradeList size.

   "The changed message tells the PluggableListMorph to update itself"
self changed: #gradeList.
Transcript
   show: ' You selected: ', (gradeList at: selectedIndex);
   cr.

initialize
   selectedIndex := 1.
   gradeList := #('A' 'A-' 'B+' 'B' 'B-' 'C+' 'C' 'C-' 'D+' 'D' 'D-' 'F').
openAsMorph
  "PluggableListExample openAsMorph"
  | window list |
  window := SystemWindow labelled: 'List'.
  window model: self.
  list := PluggableListMorph
    on: self
    list: #gradeList
    selected: #gradeListIndex
    changeSelected: #gradeListIndex:. 

  window
    addMorph: list
    frame: (0@0 extent: 1@1).
  window openInWorld.
Pluggable List Example Explained

The List

list := PluggableListMorph
  on: self
  list: #gradeList
  selected: #gradeListIndex
  changeSelected: #gradeListIndex:

The list uses three methods to interact with its model

getListSelector
  #gradeList in the example above

  Sent to the model to get the list to display

GetIndexSelector
  #gradeListIndex in the example above

  Sent to the model to get index of the item to highlight in the list

SetIndexSelector
  #gradeListIndex: in the example above

  Sent to the model to inform the model that the user has selected a new item in the list
The Model

The model can send two different arguments in changed: method

getListSelector

self changed: #gradeList

Informs the PluggableListMorph that the display list has changed

GetIndexSelector

self changed: #gradeListIndex

Informs the PluggableListMorph that selected item has changed

Must be sent to indicate a change, even if the PluggableListMorph notified the model of the change
Spreadsheet like Views

We wish to display a grid of editable text fields (PluggableTextMorphs)

Solution 1
  Give each PluggableTextMorph a different model

Solution 2
  Have one model
  Give each PluggableTextMorph a way to identify itself to the model
Solution 1 Different Models

First we need a model for each different value to display

ValueHolder
  Squeak class

  Holds one value

  contents returns the value

  contents: sets the value & broadcasts the change

We need our model to interact with a PluggableTextMorph

PluggableTextMorph sets/gets values at Text objects

ValueModel
  New class

  Subclass of ValueHolder

  Adds stringContents & stringContents: for interaction with PluggableTextMorph

  Assumes value is a number, so should be called NumberValue

Both ValueHolder & ValueModel could use more work to make them more flexible
ValueModel

ValueHolder subclass: #ValueModel
  instanceVariableNames: ""
  classVariableNames: ""
  poolDictionaries: ""
  category: 'Whitney-Examples'

Class Methods

contents: anObject
  ^super new
    contents: anObject;
    yourself

Instance Methods

stringContents
  ^self contents asString.

stringContents: aText
  "PluggableTextMorph sets results as a Text object
  convert to number"
  super contents: aText asString asNumber.
  self changed: #stringContents.
  ^true

Note rather than assume the ValueModel holds a number and provide stringContents, stringContents: methods it would be better to provide an adapter between ValueHolder and PluggableTextMorph
ArrayedTextExample

Maintains a N by M grid of numbers

Each number is in an ValueModel

All ValueModels are in a Array2D object

Each number is displayed in a different PluggableTextMorph

To open a window:
(ArrayedTextExample
  width: 4
  height: 3) openAsMorph
Object subclass: #ArrayedTextExample
  instanceVariableNames: 'text2DArray'
  classVariableNames: ''
  poolDictionaries: ''
  category: 'Whitney-Examples'

Class Methods

width: x height: y
  ^super new
    setArrayWidth: x
    height: y

Instance Methods
initialize

setArrayWidth: x height: y
  | count value |
  text2DArray := Array2D
    width: x
    height: y.
  count := 1.
  text2DArray rowsAndColumnsDo:
    [:row :column |
      value := ValueModel contents: count.
      text2DArray
        at: column
        at: row
        put: value.
      count := count + 1.|]
accessing

at: aPoint
    "Unwrap content from ValueModel"
    | holder |
    holder := text2DArray
    at: aPoint x
    at: aPoint y.
    ^holder contents

at: aPoint put: anObject
    "Wrap value in ValueModel"
    | holder |
    holder := text2DArray
    at: aPoint x
    at: aPoint y.
    ^holder contents: anObject
openAsMorph
| window textView testViewRectangle |

window := SystemWindow labelled: 'Text Array'.
window model: self.
text2DArray rowsAndColumnsDo:
    [:row :column |
        textView := self textViewOnRow: row
column: column.
        testViewRectangle := self rectangleForRow: row
column: column.
        window
            addMorph: textView
            frame: testViewRectangle].
window openInWorld.
Private

rectangleForRow: row column: column
    | width height extent left top |
    width := 1/text2DArray width.
    height := 1/text2DArray height.
    extent := width @ height.

    left := width * ( column - 1).
    top := height * (row -1).
    ^left@ top extent: extent.

textViewOnRow: row column: column
    | model textView |
    model := text2DArray
        at: column
        at: row.
    textView := PluggableTextMorph
        on: model
        text: #stringContents
        accept: #stringContents:.
    textView hideScrollBarIndefinitely.
    ^textView
Solution 2 Same Model

This solution requires more work

PluggableTextMorph must be modified to let the model which view is sending information to the model

PluggableTextMorph subclass: #PluggableArrayedTextMorph
  instanceVariableNames: 'index '
  classVariableNames: "
  poolDictionaries: "
  category: 'Whitney-Examples'

index is used to store an identifier for the view. It will be an index into the Array2D that holds the models. The name should be changed

Class Methods

on: anObject text: getTextSel accept: setTextSel index:
anIntegerOrPoint
| textMorph |
textMorph :=self new index: anIntegerOrPoint.
textMorph
  on: anObject
  text: getTextSel
  accept: setTextSel
  readSelection: nil
  menu: nil.
textMorph hideScrollBarIndefinitely.
^textMorph
Instance Methods

accessing

index: integerOrPoint
  index := integerOrPoint

updating

update: aSymbol with: aParameter
  aParameter = index
  ifTrue:\[self update: aSymbol\]

Overridden Methods

The following methods from PluggableTextMorph are modified to inform the one model which morph it is.

getText
  "Retrieve the current model text"

| newText |
  getTextSelector == nil ifTrue: [^ Text new].
  newText _ model
    perform: getTextSelector
    with: index.
  newText ifNil: [^Text new].
  ^ newText shallowCopy
accept
"Inform the model of text to be accepted, and return true if OK."

<table>
<thead>
<tr>
<th>textToAccept ok</th>
</tr>
</thead>
<tbody>
<tr>
<td>self canDiscardEdits ifTrue: [^ self flash].</td>
</tr>
<tr>
<td>self hasEditingConflicts ifTrue:</td>
</tr>
<tr>
<td>[(self confirm:</td>
</tr>
<tr>
<td>'Caution This method may have been</td>
</tr>
<tr>
<td>changed elsewhere since you started</td>
</tr>
<tr>
<td>editing it here. Accept anyway?') ifFalse: [^ self flash]].</td>
</tr>
<tr>
<td>textToAccept _ textMorph asText.</td>
</tr>
<tr>
<td>ok _ (setTextSelector == nil) or:</td>
</tr>
<tr>
<td>[model</td>
</tr>
<tr>
<td>perform: setTextSelector</td>
</tr>
<tr>
<td>with: index</td>
</tr>
<tr>
<td>with: textToAccept asString].</td>
</tr>
<tr>
<td>ok ifTrue:</td>
</tr>
<tr>
<td>[self setText: self getText.</td>
</tr>
<tr>
<td>self hasUnacceptedEdits: false.</td>
</tr>
<tr>
<td>(model dependents detect: [:dep</td>
</tr>
<tr>
<td>ifNone: [nil]) doIfNotNil:</td>
</tr>
</tbody>
</table>
|     [:aPane | model changed: #annotation]]
ArrayedTextExample

Start with:
(ArrayedTextExample
  width: 4
  height: 3) openAsMorph

Object subclass: #ArrayedTextExample
  instanceVariableNames: 'text2DArray'
  classVariableNames: ''
  poolDictionaries: ''
  category: 'Whitney-Examples'

Class Methods

width: x height: y
  ^super new
    setArrayWidth: x
    height: y
Instance Methods
Initialize

setArrayWidth: x height: y
| sampleText |
text2DArray := Array2D
  width: x
  height: y.
sampleText := 1.
text2DArray rowsAndColumnsDo:
  [:row :column |
  text2DArray
    at: column
    at: row
    put: sampleText asString.
sampleText := sampleText + 1.]

at: aPoint
^text2DArray
  at: aPoint x
  at: aPoint y
Do not allow any changes on the first row

This is done just to show how it is done

at: aPoint put: anObject
    "To accept the changes return true, to reject return false"
    aPoint x = 1
    ifTrue:
        [self
            changed: #at:
            with: aPoint.
            ^false].
    text2DArray
        at: aPoint x
        at: aPoint y
        put: anObject.
    self
        changed: #at:
        with: aPoint.
    ^true
openAsMorph
    | window textView testViewRectangle |

    window := SystemWindow labelled: 'Text Array'.
    window model: self.

    text2DArray rowsAndColumnsDo:
        [:row :column |
            textView := self textViewOnRow: row
                column: column.
            testViewRectangle := self rectangleForRow: row
                column: column.
            window
                addMorph: textView
                frame: testViewRectangle].
    window openInWorld.
Private

rectangleForRow: row column: column
| width height extent left top |
  width := 1/text2DArray width.
  height := 1/text2DArray height.
  extent := width @ height.

  left := width * (column - 1).
  top := height * (row - 1).
  ^left@ top extent: extent.

textViewOnRow: row column: column
  ^PluggableArrayedTextMorph
    on: self
    text: #at:
    accept: #at:put:
    index: (column @ row).
Object

The above example needs the following method in Object

changed: aSymbol with: aParameter
"Receiver changed. The change is denoted by the argument
aParameter. Usually the argument is a Symbol that is part of the
dependent's change protocol. Inform all of the dependents."

self dependents do:
  [:aDependent |
    aDependent
    update: aSymbol
    with: aParameter]