Contents

Cohesion .................................................................................................................. 3
GUIs .......................................................................................................................... 7
Model-View-Controller (MVC) ............................................................................... 9
Smalltalk MVC Example ....................................................................................... 11
  Opening a Customer Window .............................................................................. 15
Window Specs ......................................................................................................... 17
  Specifying the Window Spec ............................................................................. 18
  Multiple Specs ..................................................................................................... 19
ValueHolder ............................................................................................................ 23
Hiding Instance Variable Access .......................................................................... 27
AspectAdaptor ........................................................................................................ 33

References

Pattern-Oriented Software Architecture, Buschmann et al., 1996


VisualWorks GUI Developer's Guide, GUIDevGuide.pdf in the docs directory of the VW 5i4 distribution
Design

Design is a series of trade-offs

Often we trade flexibility for complexity
Cohesion

Measure of the interdependence among modules

"Unnecessary object coupling needlessly decreases the reusability of the coupled objects"

"Unnecessary object coupling also increases the chances of system corruption when changes are made to one or more of the coupled objects"
Example
Smalltalk.CS535 defineClass: #Customer
  superclass: #{Core.Object}
  indexedType: #none
  private: false
  instanceVariableNames: 'name phone id '
  classInstanceVariableNames: 'NextFreeID '
  imports: ''
  category: 'Course-GUI-Examples'

Class Methods
newID
  NextFreeID isNil ifTrue:[NextFreeID := 0].
  NextFreeID := NextFreeID + 1.
  ^NextFreeID

name: aString phone: aPhoneString
  ^self new
    setName: aString
    setPhone: aPhoneString
    setID: self newID

Instance Methods
setName: aNameString setPhone: aPhoneString setID: anInteger
  name := aNameString.
  phone := aPhoneString.
  id := anInteger.
How to print out the Customer

Solution 1

Customer>>display
Transcript
  show: 'Customer(';
  print: name;
  show: ',',
  print: phone;
  show: ',',
  print: id;
  show: ')

Simple and direct
Couples Customer to Transcript
Limited usefulness
Solution 2

Customer>>printOn: aStream
   aStream
      print: 'Customer(';
      print: name;
      print: ', ';  
      print: phone;
      print: ', ';  
      print: id;
      print: ')'

Couples Customer to Stream interface (print:)

Can be used to write Customer many different places

test := Customer name: 'Foo' phone: '222-1111'.
Transcript
   show: test printString
GUIs

Domain information

Information about the subject of the program

- Customer records
- Inventory
- Names
- Reports
- Addresses

Application information

Information needed to make visual interface function

- Menus
- Error Messages
- Help information
- Labels
Separation of Domain and Application Information

Keep domain and application information separate

Application information changes faster

Often there is multiple view of domain information
Model-View-Controller (MVC)

Architectural structure for GUI applications

Model

Encapsulates

Domain information

Core data and functionality

Independent of

Specific output representations

Input behavior

View

Display data to the user

Obtains data from the model

Multiple views of the model are possible
Controller

Handles input

  Mouse movements and clicks

  Keyboard events

Each view has its own controller

Programmers commonly don't see controllers
Smalltalk MVC Example

Customer with Window Created with UI Painter

Smalltalk.CS535 defineClass: #Customer
  superclass: #{UI.ApplicationModel}
  indexedType: #none
  private: false
  instanceVariableNames: 'id phone name '
  classInstanceVariableNames: 'NextFreeID '
  imports: "
  category: 'Course-GUI-Examples'

Class Methods

newID
  NextFreeID isNil ifTrue:
    [NextFreeID := 0].
  NextFreeID := NextFreeID + 1.
  ^NextFreeID

name: aString phone: aPhoneString
  ^self new
    setName: aString
    setPhone: aPhoneString
windowSpec
"UIPainter new openOnClass: self andSelector: #windowSpec"

<resource: #canvas>
^#{UI.FullSpec}
  #window:
  #{UI.WindowSpec}
    #label: 'Customer'
    #bounds: #{Graphics.Rectangle} 512 384 712 584 )
  #component:
  #{UI.SpecCollection}
    #collection: #{
      ###{UI.InputFieldSpec}
        #layout: #{Graphics.Rectangle} 79 30 179 52 )
        #name: #name
        #model: #name
      ###{UI.InputFieldSpec}
        #layout: #{Graphics.Rectangle} 79 75 181 99 )
        #name: #phone
        #model: #phone
        #type: #string
        #formatString: '#@@@-@@-@@@@' )
      ###{UI.InputFieldSpec}
        #layout: #{Graphics.Rectangle} 79 120 179 142 )
        #name: #id
        #model: #id
        #isReadOnly: true
        #type: #number
        #formatString: '0' )
      ###{UI.LabelSpec}
        #layout: #{Core.Point} 24 31 )
        #name: #Label1
        #label: 'Name' )
      ###{UI.LabelSpec}
        #layout: #{Core.Point} 24 76 )
        #name: #Label2
        #label: 'Phone' )
      ###{UI.LabelSpec}
        #layout: #{Core.Point} 24 121 )
        #name: #Label3
        #label: 'ID' ) ) ) ) )}
Instance Methods

initialize
    self initializeID

initializeID
    id isNil ifTrue: [id := self class newID asValue]

setName: aNameString setPhone: aPhoneString
    self name value: aNameString.
    self phone value: aPhoneString.
    self initializeID.

printOn: aStream
    aStream
        print: 'Customer(';
        print: name value;
        print: ', ';
        print: phone value;
        print: ', ';
        print: id value;
        print: ')';
CS535. Customer methodsFor: 'aspects'

id
"This method was generated by UIDefiner."

^id isNil
   ifTrue:
     [id := 0 asValue]
   ifFalse:
     [id]

name
"This method was generated by UIDefiner."

^name isNil
   ifTrue:
     [name := String new asValue]
   ifFalse:
     [name]

phone
"This method was generated by UIDefiner."

^phone isNil
   ifTrue:
     [phone := String new asValue]
   ifFalse:
     [phone]
Opening a Customer Window

A New Customer

Customer open

An Existing Customer

existingCustomer := Customer name: 'Foo' phone: '222-1111'. existingCustomer open
How Does this Work

Things to Explain

Window specs

ValueHolders - asValue
Window Specs

windowSpec
"UIPainter new openOnClass: self andSelector: #windowSpec"

<resource: #canvas>
^#(#{UI.FullSpec}
  #window:
  #(#{UI.WindowSpec}
    #label: 'Customer'
    #bounds: #(#{Graphics.Rectangle} 512 384 712 584 ) )
  #component:
  #(#{UI.SpecCollection})

Window Spec tells how to put a window together

When you "open" a window a builder object constructs the window

Your program can interact with the builder to modify the window as it is being built
Specifying the Window Spec

New Customer

Customer openWithSpec: #windowSpec

Existing Customer

existingCustomer := Customer name: 'Foo' phone: '222-1111'.
existingCustomer openInterface: #windowSpec.
Multiple Specs

A class can have multiple specs to provide different views of the same object

Clerk View
In UIPainter I removed the phone label and input field

I stalled the new window spec on the Customer class as clerkView

```plaintext
Customer class>>clerkView
"UIPainter new openOnClass: self andSelector: #clerkView"
<resource: #canvas>
^#(#{UI.FullSpec}
  #window:
    ^(#{UI.WindowSpec}
      #label: 'Customer'
      #bounds: #(#{Graphics.Rectangle} 460 362 654 478 )
    )
  #component:
    #(#{UI.SpecCollection}
      #collection: #(
        #(#{UI.InputFieldSpec}
          #layout: #(#{Graphics.Rectangle} 79 30 179 52 )
          #name: #name
          #model: #name
        )
        #(#{UI.InputFieldSpec}
          #layout: #(#{Graphics.Rectangle} 81 66 181 88 )
          #name: #id
          #model: #id
          #isReadOnly: true
          #type: #number
          #formatString: '0'
        )
        #(#{UILabelSpec}
          #layout: #(#{Core.Point} 24 31 )
          #name: #Label1
          #label: 'Name'
        )
        #(#{UILabelSpec}
          #layout: #(#{Core.Point} 26 67 )
          #name: #Label3
          #label: 'ID' )
      )
    )
  )
)```

Using the Clerk View

Customer openWithSpec: #clerkView

existingCustomer := Customer name: 'Foo' phone: '222-1111'.
existingCustomer openInterface: #clerkView.
Two Views at Once

existingCustomer := Customer name: 'Foo' phone: '222-1111'.
existingCustomer open.
existingCustomer openInterface: #clerkView.

This will open two windows on the same customer object

Each window shows a different view

The windows open in the same location!
Specifying where to open a Window

| existingCustomer builder window |

existingCustomer := Customer name: 'Foo' phone: '222-1111'.
existingCustomer open.

builder := existingCustomer allButOpenInterface: #clerkView.
window := builder window.
window openIn: (50@50 extent: 200@100).
ValueHolder

The following create a value holders

0 asValue
String new asValue
anyObject asValue

A value holder

Holds a value

Can have observers

The message value: aNewValue

Replaces the value of the ValueHolder

Tells the observers that the value has changed

Observers can then perform the appropriate action
ValueHolders and Windows

`#{UI.InputFieldSpec}
    #layout: #{Graphics.Rectangle} 79 30 179 52
    #name: #name
    #model: #name`

When the customer window is built the builder

Calls the name method on customer object

Gives the name value holder to the name input field

The input field registers as an observer to the name value holder

Keeps a reference to the value holder

When the user changes a value in the window

The widget send value: to the value holder

Your program and other observers know the change

When your program changes the value of a value holder

All observers are informed

Widgets then redisplay the new value
ValueHolders and Model-View Separation

Customer object is the model

Customer objects

Have no direct knowledge of any views

Instance variables are now value holders

All access to instance variables is changed

We have:

```plaintext
printOn: aStream
  aStream
    print: 'Customer(';
    print: name value;

not

printOn: aStream
  aStream
    print: 'Customer(';
    print: name;
```
Dealing with the ValueHolder Instance Variables

In many cases dealing with the value holders is not a problem.

If it is a problem try using:

- Information Hiding
- AspectAdaptor
Hiding Instance Variable Access

Some people claim all accesses of instance variables in a class should be through method access

Smalltalk.CS535 defineClass: #Customer
    instanceVariableNames: 'name '

printOn: aStream
    aStream
        print: 'Customer(';
        print: self name;
        print: ')

name
    ^name

name: aString
    name := aString
WindowSpec and Information Hiding in Class

Define a special method for widgets to get value holders on instance variables

Define setters and getter methods for the instance variables

   Getter returns actual value not value holder

   Setter changes the value of the value holder

Methods in the class use getters and setters to interact with the instance variables
Example with Information Hiding

Smalltalk.CS535 defineClass: #Customer
  superclass: #{UI.ApplicationModel}
  indexedType: #none
  private: false
  instanceVariableNames: 'id phone name '
  classInstanceVariableNames: 'NextFreeID '
  imports: ''
  category: 'Course-GUI-Examples'

Class Methods

newID
  NextFreeID isNil ifTrue:[NextFreeID := 0].
  NextFreeID := NextFreeID + 1.
  ^NextFreeID

name: aString phone: aPhoneString
  ^self new
    setName: aString
    setPhone: aPhoneString
windowSpec
"UIPainter new openOnClass: self andSelector: #windowSpec"

<resource: #canvas>
^#(#{UI.FullSpec}
    #window:
    #(#{UI.WindowSpec}
        #label: 'Customer'
        #bounds: #(#{Graphics.Rectangle} 512 384 712 584 )
    )
    #component:
    #(#{UI.SpecCollection}
        #collection: #(
            #(#{UI.InputFieldSpec}
                #layout: #(#{Graphics.Rectangle} 79 30 179 52 )
                #name: #name
                #model: #nameValueHolder
            )
            #(#{UI.InputFieldSpec}
                #layout: #(#{Graphics.Rectangle} 79 75 181 99 )
                #name: #phone
                #model: #phoneValueHolder
                #type: #string
                #formatString: '#@@@-@@-@@@@'
            )
            #(#{UI.InputFieldSpec}
                #layout: #(#{Graphics.Rectangle} 79 120 179 142 )
                #name: #id
                #model: #idValueHolder
                #isReadOnly: true
                #type: #number
                #formatString: '0'
            )
            #(#{UI.LabelSpec}
                #layout: #(#{Core.Point} 24 31 )
                #name: #Label1
                #label: 'Name'
            )
            #(#{UI.LabelSpec}
                #layout: #(#{Core.Point} 24 76 )
                #name: #Label2
                #label: 'Phone'
            )
            #(#{UI.LabelSpec}
                #layout: #(#{Core.Point} 24 121 )
                #name: #Label3
                #label: 'ID' ) ) ) ) )


Instance Methods

initialize
self initializeID

initializeID
id isNil ifTrue: [id := self class newID asValue]

setName: aNameString setPhone: aPhoneString
    self name: aNameString.
    self phone: aPhoneString.
    self initializeID.

printOn: aStream
    aStream
        print: 'Customer(';
        print: self name;
        print: ', ';
        print: self phone;
        print: ', ';
        print: self id;
        print: ')'

idValueHolder
    ^id isNil ifTrue: [id := 0 asValue] ifFalse: [id]

nameValueHolder
    ^name isNil
        ifTrue: [name := String new asValue]
        ifFalse: [name]
phoneValueHolder
  \^phone isNil
    ifTrue: [phone := String new asValue]
    ifFalse: [phone]

id
  \^self idValueHolder value

id: anInteger
  \^self idValueHolder value: anInteger

name
  \^self nameValueHolder value

name: aString
  \^self nameValueHolder value: aString

phone
  \^self phoneValueHolder value

phone: aString
  \^self phoneValueHolder value: aString
AspectAdaptor

A GUI widget expects to get a ValueHolder from the model

We want the GUI to call accessor methods

Use an adapter

- Acts like a ValueHolder to the GUI widget

- Converts value, value: messages to accessor method on model

- When model changes it must broadcast the change
Example
Smalltalk.CS535 defineClass: #Customer
  superclass: #{UI.ApplicationModel}
  indexedType: none
  private: false
  instanceVariableNames: 'id phone name '
  classInstanceVariableNames: 'NextFreeID '
  imports: "
  category: 'Course-GUI-Examples'

Class Methods

newID
  NextFreeID isNil ifTrue: [NextFreeID := 0].
  NextFreeID := NextFreeID + 1.
  ^NextFreeID

name: aString phone: aPhoneString
  ^self new
    setName: aString
    setPhone: aPhoneString

windowSpec
  <resource: #canvas>
    ^#{UI.FullSpec}
      #window:
        #{UI.WindowSpec}
          #label: 'Customer'
          #bounds: #({Graphics.Rectangle} 512 384 712 584 )
      #component:
        #{UI.SpecCollection}
          #collection: #(
            #{UI.InputFieldSpec}
              #layout: #({Graphics.Rectangle} 79 30 179 52 )
              #name: #name
              #model: #nameValueHolder
              "same as example on page 30) ) ) )
Instance Methods

initialize
  self initializeID

initializeID
  id isNil ifTrue:[id := self class newID]

setName: aNameString setPhone: aPhoneString
  self name: aNameString.
  self phone: aPhoneString.
  self initializeID.

printOn: aStream
  aStream
    print: 'Customer(';
    print: self name;
    print: ', ';
    print: self phone;
    print: ', ';
    print: self id;
    print: ')'

idValueHolder
  | adaptor |
  adaptor := AspectAdaptor forAspect: #id.
  adaptor
    subject: self;
    subjectSendsUpdates: true.
  ^adaptor
nameValueHolder
  | adaptor |
  adaptor := AspectAdaptor forAspect: #name.
  adaptor
    subject: self;
    subjectSendsUpdates: true.
  ^adaptor

phoneValueHolder
  | adaptor |
  adaptor := AspectAdaptor forAspect: #phone.
  adaptor
    subject: self;
    subjectSendsUpdates: true.
  ^adaptor

id
  ^id

id: anInteger
  id := anInteger.
  self changed: #id

name
  ^name

name: aString
  name := aString.
  self changed: #name
phone
  ^phone

phone: aString
  phone := aString.
  self changed: #phone