CS193E
Lecture 14

Cocoa Bindings

Agenda

- Questions?
- Personal Timeline IV
- Key Value Coding
- Key Value Observing
- Key Value Binding
- Cocoa Bindings

What are Cocoa Bindings?

- Added in Panther, more mature in Tiger
- Provide infrastructure to help implement MVC Controller “glue code”
- Tight integration with Interface Builder

Example Glue Code

- In Draw, it’s all the code that updates and responds to the inspector UI

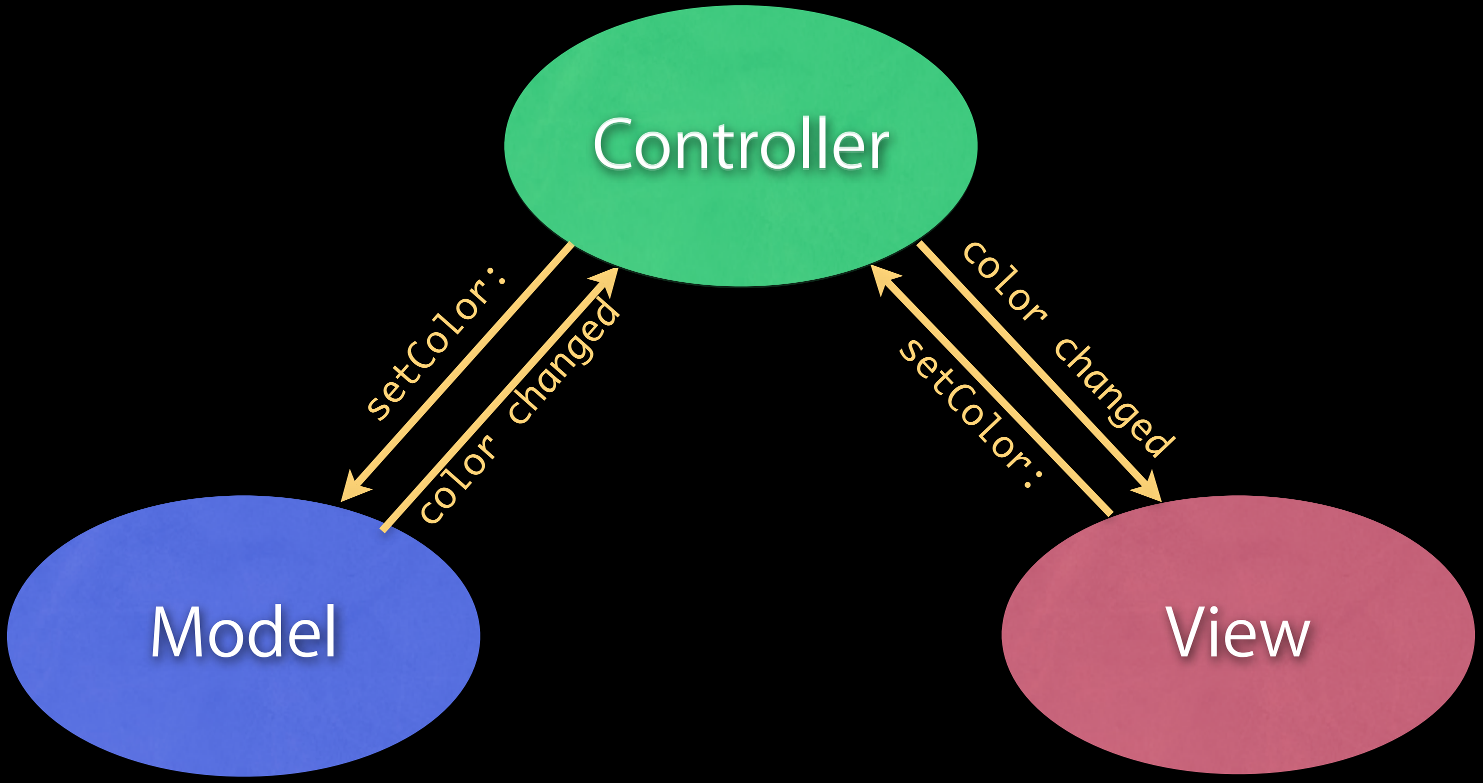
```
- (void)updateUI {
    [drawBorderCheckbox setState:[shape drawsBorder]];
    [fillCheckbox setState:[shape isFilled]];
    [drawBorderCheckbox setState:[shape drawsBorder]];
}

- (void)fillColorChanged:(NSColorWell *)sender {
    [shape setFillColor:[sender color]];
}
```

- Posting of notifications when selection or properties change, etc.

Glue Code

- Conceptually you’ve got UI elements that are associated with properties in model objects
- All the code that:
  - keeps the UI elements up to date
  - pushes changes from the UI into the model objects

Glue Code



Removing the glue lets you focus on the view and model, less code to maintain!

Bindings Fundamentals

- Leverages on a few underlying technologies
- Key Value Coding: generic mechanism for accessing properties of objects by key
- Key Value Observing: generic mechanism for objects to know when properties change
- Key Value Binding: generic mechanism for associating a property of one object with a property of another object

Key Value Coding

Accessing properties using keys

Key Value Coding (KVC)

- Generic way for properties of objects to be accessed (by key)
- Instead of:

  `[shape fillColor]` and `[shape setFillColor:color]`
- One could do:

  `[shape valueForKey:@”fillColor”]` and

  `[shape setValue:newColor forKey:@”fillColor”];`
- Conceptually every object becomes a dictionary!

Properties

- KVC allows access to all object “properties”
- Properties are:
  - Attributes: Simple, immutable values like BOOLs, ints, floats, strings… (scalar data types)
  - Relationships: references to other objects which have properties of their own
    - to-one: single object (e.g. outlet in IB, NSWindow’s contentView)
    - to-many: one or more objects (e.g. an array of objects, NSView’s subviews)

Getting values via KVC

- Given a key, you get a value back (or nil)

```
- (id)valueForKey:(NSString *)key
```

- Scalar types such as BOOL and int are “boxed” automatically in NSNumbers
- Structs such as NSRect are boxed in NSValues.
- Example:

```
NSColor *fillColor = [shape valueForKey:@”fillColor”];

NSNumber *showBorder = [shape valueForKey:@”showBorder”];
```

Setting values via KVC

- Given a value, you set it on an object using

```
    - (void)setValue:(id)value forKey:(NSString *)key
```

- Scalar types such as BOOL and int are “unboxed” automatically
- Example:

```
  [shape setValue:[NSColor redColor] forKey:@”fillColor”];

  [shape setValue:[NSNumber numberWithBool:YES]
           forKey:@”showBorder”];
```

From Keys to Values

- NSObject’s implementation of valueForKey: will
  - Search for a public accessor method based on “key”. For example, `[shape valueForKey:@”fillColor”]` will try to find `[shape fillColor]` or `[shape getFillColor]`
  - Search for a private accessor method (with an underscore), `[shape _fillColor]` or `[shape _getFillColor]`
  - Search for an instance variable based on “key”. For example, `_fillColor` or `fillColor`
- If none of the above are found, exception is thrown

From Keys to Values

- Setting values works the same (mostly)
  - Search for a set<Key>: method, `[shape setValue:color forKey:@”fillColor”]` will try to find `[shape setFillColor:color]`
  - Try to find corresponding instance variable with name _<key> or <key>. For example, `_fillColor` or `fillColor`.
- If none of the above are found, exception is thrown

To-many Relationships

- For immutable to-many relationships, can be accessed the same way as attributes:

```
NSArray *shapes = [canvas valueForKey:@”shapes”];
```

- For mutable to-many relationships you have to request them differently:

```
NSMutableArray *shapes;

 shapes = [canvas mutableArrayValueForKey:@”shapes”];
 [shapes addObject:newShape];
```

- Returns a “proxy” mutable array for an underlying mutable to-many relationship

NSDictionary KVC

- NSDictionary has a custom implementation of KVC that attempts to match keys against keys in the dictionary.
- Useful for doing rapid prototyping where you don’t have to create custom classes or need extra custom logic
- For example, our canvas could probably just be a dictionary with a “shapes” property

Key Paths

- Keys can be chained together to access nested object properties
- For example, if document has a selectedShape property we could get the fill color by doing:

```
NSColor *color;
color = [document valueForKeyPath:@”selectedShape.fillColor”];
```

- Equivalent to:

```
color = [[document selectedShape] fillColor];
```

- Corresponding setter methods:

```
[document setValue:color
        forKeyPath:@”selectedShape.fillColor”];
```

Pros and Cons

- Allows, dynamic, generic access to properties without even caring what the class of an object is
- Loses all type specification because values are always typed (id)
  - Compiler can’t help with type checking
  - Compiler can’t guard against mistyped keys
- Sometimes can be a bit “too magic” and can be difficult to debug
  - This is as close to operator overloading as ObjC gets!

Key Value Observing

Was that a tree that just fell?

Key Value Observing (KVO)

- Allows objects to express interest in knowing when a property of an object changes
- Any time the underlying property is changed, all observers are notified
- Similar (conceptually) to notifications, but more specific and lightweight
  - Object to object, no “center” in the middle
- Like KVC, properties are identified by key
- Built into NSObject (all objects are observable!)

Observing Properties

- Object that wants to hear about changes calls:

```
-(void)addObserver:(id)observer forKeyPath:(NSString *)keyPath
         options:(NSKeyValueObservingOptions)options
         context:(void *)context;
```

- Observer must then implement:

```
 - (void)observeValueForKeyPath:(NSString *)keyPath
          ofObject:(NSObject *)observedObject
          change:(NSDictionary *)change context:(void *)context;
```

  which will be called any time the value changes

- For example, CanvasView might do:

```
[document addObserver:self
          forKeyPath:@”selectedShape.fillColor”
          options:NULL context:NULL];
```

  to hear about any changes to fillColor of selected shape

Observing Properties

- Like NSNotifications, make sure to unregister when you no longer need to hear about changes:

```
[document removeObserver:self
             forKeyPath:@”selectedShape.fillColor”];
```

- Failing to do this will lead to crashes!

What Do You Have To Do?

- You just write your regular setter method:

```
- (void)setFillColor:(NSColor *)color {
      if (color != i_color) {
          [i_color release];
          i_color = [color retain];
      }
 }
```

- When this method is called directly or indirectly via KVC, observers will be notified — but how?
- The ObjC runtime is automatically altered
- As soon as someone registers as an observer on a shape’s fillColor attribute, the setFillColor method is replaced with a “notifying” wrapper

So How Does It Work?

- Effectively your method is transformed from:

```
- (void)setFillColor:(NSColor *)color {
     if (color != i_color) {
         [i_color release];
         i_color = [color retain];
     }
}
```

to this:

```
- (void)setFillColor:(NSColor *)color {
      [self willChangeValueForKey:@”fillColor”];
      if (color != i_color) {
          [i_color release];
          i_color = [color retain];
      }
      [self didChangeValueForKey:@”fillColor”];
 }
```

Major Caveat

- In order for KVO to work reliably, all access to properties must be done using “KVC Compliant” means
- Changing values out from underneath KVC lets observers get out of sync which is bad (ie, leads to exceptions and/or crashes)
- Fortunately you can adopt KVC incrementally so it’s not as bad as it sounds
- Can be difficult to debug

Key Value Binding

The glue that binds it all together

Key Value Binding (KVB)

- Ties together KVC and KVO
- Allows a property of one object to be bound to the property of another object
- You can think of it as an alternative to “outlets” and “actions” for connecting objects
  - But it’s much, much more!
- Easily configured in IB using the Bindings Inspector
  - Can also be configured programmatically

Finding the Selection

- The trouble in the inspector is how to identify, by a key or key path, the selected node
- The inspector should inspect what's selected in the main window
- The application keeps track of the main window
- Selected node is owned by the document
- The main window can access the document through its window controller
- Is this enough to fish out the selected node?

Finding the Selection

- Starting from the global shared application, NSApp, we can find the selected node of the document displayed in the main window

```
[[[[NSApp mainWindow] windowController] document] selectedNode]
```

- Key path looks something like this:

```
mainWindow.windowController.document.selectedNode
```

- Since the shared application object is available in any nib, we can bind UI to the document’s selectedNode!

KVO and Key Paths

- When using KVO with a key path, the observer gets notified when any component of the key path changes
  - Provided the change is done in a KVC compliant manner
  - This is the most common problem that trips people up with KVO!
- This is very powerful!

```
mainWindow.windowController.document.selectedNode
```

↑

Any time the main window changes, inspector will update

Recap

- Key Value Coding
  - Get/set properties using keys or key paths
- Key Value Observing
  - Notifications about changes to keys or key paths
- Bindings
  - Alternative to IB outlets and actions
  - Uses KVC to get/set values, and KVO to know when to update

Intersted?

- Bindings can save a significant amount of code
- There’s definitely a learning curve to using them
- They can be frustrating to debug
- Can you use them in your final project?
  - Maybe, but we want to approve any usage first to make sure that it’s an appropriate use and that it won’t cause more headaches than it’s worth

Questions?