Stage 4.1 - KVC

Scene 1: On-Set

KVC or Key-Value-Coding is a generic way of accessing an object's properties and variables by using a dictionary style interface. It might just seem to be a slightly more cumbersome way of accessing instance data, but KVC actually provides a flexible, adaptive and useful means for talking to objects.

<KEYNOTE SLIDE>

The method for accessing a variable or property using KVC is -valueForKey. In this example, we ask for the value of the "someKey" property and store it in "val":

id val = [objectInstance valueForKey:@"someKey"]

Setting a value is done using -setValue:ForKey:
Here we're storing the NSNumber 19 using the key "someOtherKey"

[objectInstance setValue:@19 forKey:@"someOtherKey"]

</KEYNOTE SLIDE>

The mechanism of KVC is actually adaptive in it's storing and retrieving of values, and operates using an "order of precedence" similar to arithmetic operators.

<KEYNOTE SLIDE>
When accessing instance data, the following sequence of priority applies:

1. **accessor methods: -<key>, -get<Key>**
2. **methods: -_<key>, -_get<Key>, -_set<Key>**
3. **instance variables: <key> or _<key>**
4. **-valueForUndefinedKey:, setValue:forUndefinedKey:**

For example, if we call -valueForKey using a hypothetical key, "someVar", KVC will first attempt to retrieve a value using the standard property accessor method, Failing that, it will look for methods -_someVar and -_getSomeVar, then it will try instance variables. Finally, if no source for "someVar" is found, KVC will call -valueForUndefinedKey

</KEYNOTE SLIDE>

To make this process more clear, we'll examine how this works using a code example:

Scene 2: Screencast

We'll create a new project called KVC1.

// create project

Once again, since we're working at the Cocoa Foundation level and we want to keep things simple and focused on the topic, we'll use the command-line project template. The project will be called KVC1.

// create KVC1 project

We'll create an NSObject subclass called KVCobject, that will have both an empty interface and empty implementation;

// create KVCobject

In main, we'll add an allocation of a KVCobject and a -valueForKey call for @"theString"

Running the code, our app terminates with an uncaught exception indicating that our class is not key value-coding compliant for @"theString". This is understandable since we haven't implemented any sources for the value, "theString".

First we'll implement the method -valueForUndefinedKey with a log message for the undefined key and we'll return the value @"Default Value"

// add -valueForUndefinedKey

Building and running the code again, we see our log message and the "Default Value" is the value returned for "theString". We now have a "net" for catching any undefined keys.

To demonstrate this we can add a log statement that requests a value for a new key, "theNumber".

// add "theNumber"

If we build and run again, we see the same default value returned for both requests, and log messages telling us that -valueForUndefinedKey has caught requests for both "theString" and "theNumber".

First Value
Now we'll add a "theString" instance variable(or ivar) and implement an init method that initializes the ivar to the string "First Value".

// add ivar
Building and running again, we see that the new ivar is providing the value for the "theString" key and preventing -valueForUndefinedKey from executing. Again, -valueForUndefinedKey catches the request for "theNumber".

Second Value
Next we'll add the implementation of a " _theString" method that will return the string "Second Value".

// add _theString

Building and running again we see that even though the ivar still exists and is still set to "First Value" the method has trumped the ivar and is now providing the value for the key "theString".

Third Value
We then implement a method, "-_getTheString" and have it return the value "Third Value".

// add _getTheString

Doing a build and running now shows us the result of "-valueForKey:theString" to be the one provided by our latest method, indicating that of the three sources for the value "theString" the _getTheString method has the highest precedence so far.

Fourth Value
Finally, we add a property, "theString" and set its value to "FourthValue" in the init method.

// add property

Building and running one last time we see that the property (or rather its auto-synthesized getter method) has the highest precedence for returning a value for the key "theString".

"theNumber" key request continues to be handled by -valueForUndefinedKey.

In this last case, XCode is good enough to warn us that the property getter will trump the ivar, but it doesn't appear to notice the other sources for the same value.

Questions:

1. True or False. KVC can only be used to retrieve values. (ans. false)

2. What does KVC stand for? (ans. Key-Value-Coding)
3. True or False. Standard property accessor methods have the highest precedence in KVC (ans. true)
4. What is the method called when -valueForKey: is called for a key for which no value is stored? (ans. -valueForUndefinedKey:)
5. What method is called when -setValue:ForKey: is called for a non-existent key? (ans. -setValue:forUndefinedKey:)