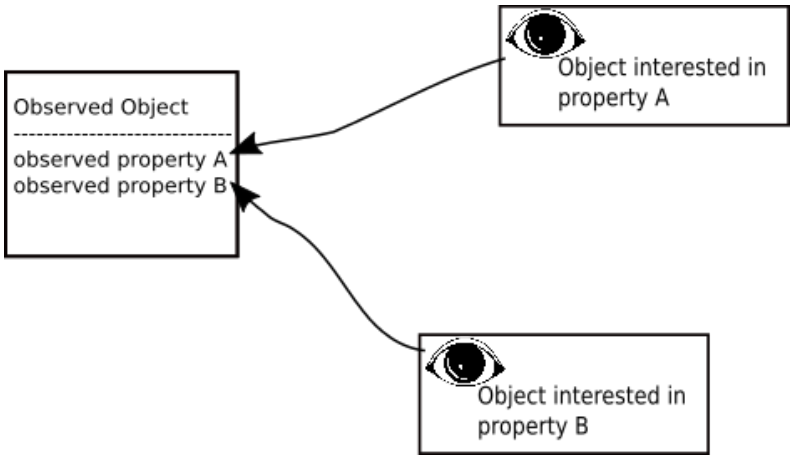
Stage 4.3 - Key-Value-Observing

Scene 1: On-Stage

Key-Value-Observing or KVO is perhaps the most powerful application of Key-Value-Coding.
It allows an object instance to respond to changes to a specific property within any other object.
<MOTION>



Here we have three object instances. One is observed, and two are observing. The observed object has two properties: A and B. Of the observing objects, one has an interest in what goes on with A. The other is interested in B. KVO allows one object to watch out for changes in A only, while the second object watches only for changes in B.
</MOTION>

The power of KVO lies in it's ability to create a selective, focused relationship between an object and a specific property in another object with minimal coupling.

<KEYNOTE SLIDE bold type on slide>

Setting up KVO between two objects is done using -AddObserver:forKeyPath
This example adds an observer: watcher1 to "propertyA" of objectX"

[objectX addObserver:watcher1 forKeyPath:@"propertyA" options:opts context:NULL];

The -AddObjserver call is sent to objectX. The observer is identified by the first argument. The second argument is the Keypath of the property watcher1 is interested in observing. The third argument is a bitmask of options for specifying what information should be delivered when watcher1 gets a callback. The fourth argument, *context* is an optional argument for uniquely identifying this observation request.

</KEYNOTE>
<KEYNOTE SLIDE bold on slide>

The callback notifying watcher1 of a change occurs as a call to -observeValueForKeyPath:

```
-(void) observeValueForKeyPath:(NSString *)keyPath
                      ofObject:(id)object
                      change:(NSDictionary *)change
                      context:(void *)context
```

Since all KVO callbacks are sent to observers as calls to -observeValueForKeyPath, the keyPath is sometimes necessary for identifying the observed property. For example, an observer might be watching multiple properties in one or more objects.

The “ofObject” argument (which identifies the object in which the change was observed) and/or the context identifier may also be used to identify the callback.

The change dictionary provides information about how the property has changed.
</KEYNOTE>

Since Key-Value-Observing support is provided by NSObject, you’re free to use KVO with any key-value-coding compliant property you wish.

Now, we’ll work through an example which should make the KVO mechanism more clear:

Scene 2: Screencast

Here we’ve created a project, consisting only of a main module, containing a class **ExampleObj,** that has two properties: an NSString **ident** and an integer called **someValue.**

The implementation block of ExampleObj contains an implementation of -observeValueForKeyPath that displays the identity of the object being notified and the associated change dictionary.

We’re going to implement a simple example that demonstrates how Key-Value-Observing works using two instances of ExampleObj.

(talk through adding the code in main())

Scene 3: On-Set

KVO and inheritance:

Inheritance adds additional complexity to KVO. While NSObject does not implement -observeValueForKeyPath, any other superclass may, and may require KVO callbacks to function correctly.
Implementing -observeValueForKeypath in a subclass will override the parent implementation and all KVO will be received only by the child class.

This is a situation where the **context** parameter is very useful for uniquely identifying an observer. Any -observe callbacks with an unrecognized context can then be forwarded to the parent class using [super observeValueForKeypath].

context is a void pointer, which is C's generic pointer type (meaning it can point to any sort of object). However, as a KVO identifier it is used simply for its numeric value. So, it can be set to any value as long as it is unique.

One foolproof method of defining context identify is with:

void *contextIdent = &contextIdent;

The context identifier is set to the value of its own address, a value that is guaranteed to be unique.

Questions:

1. True or False. A keypath can be used to specify a value in a nested dictionary. (ans. true).
2. True or False. The -observeValueForKey callback does not identify the object in which the value has changed. (ans. false)
3. The keypath collection operator which calculates an arithmetic average is? (ans. @avg)
4. True or False. The "options" argument in -addObserver:forKeyPath:options:context allows the observer to receive information about the change that has occurred. (ans. true)
5. True or False. The "context" argument in -observeValueForKeyPath:ofObject:change:context refers to a core-data managed object context. (ans. false)