Stage 2: video 2

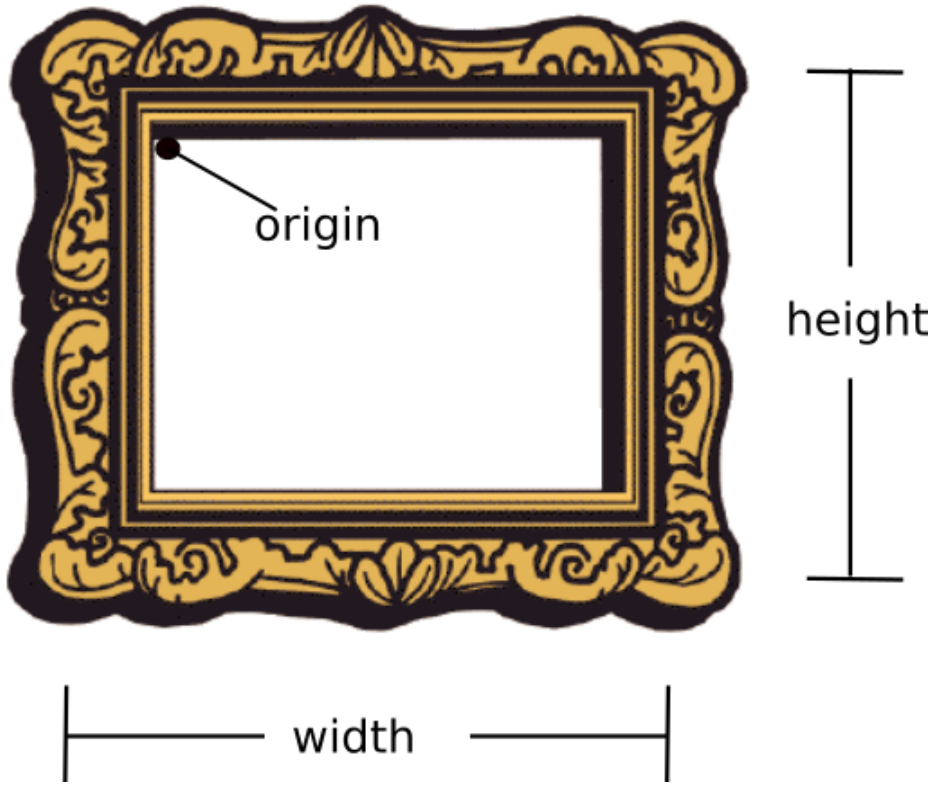
Scene 1: On-Set

Now we'll look at some examples of two-stage creation from the Cocoa and Cocoa-Touch frameworks:

In a previous video, we talked about designated initializers as if they were unique. This is usually the case, however, for some classes where there are multiple, distinct mechanisms for creating instances, there can be multiple designated initializers. This is true of UIViews which may be generated programmatically or instantiated from an XML representation stored in a NIB.

We'll begin by discussing the programmatic way.

<MOTION>



The placement of any UIView requires a position and a size. Adding a UIView as a subview is analogous to hanging a picture frame on a wall. If you have a finished frame (a size), and you know where you want it (a position), you can hang the frame even if you don't have the artwork yet.

</MOTION>

It makes sense then that the designated initializer for the programmatic creation of a UIView is *initWithFrame*.

<SLIDE>

```
struct CGRect {
        CGPoint origin;
        CGSize size;
};

struct CGPoint {
        CGFloat x;
        CGFloat y;
};

struct CGSize {
        CGFloat width;
        CGFloat height;
};
```

The *"frame"* for a UIView is concisely specified using the the *CGRect* C structure: composed of a *CGPoint* structure for position and a *CGSize* structure specifying the width and height.

</SLIDE>

When a UIView is instantiated from a nib (that is, a xib or a storyboard) rather than a CGRect, a different initialization method is required.

<SLIDE>

When a developer lays out a view in interface builder, size and position information is encoded as XML. So, rather than using -initWithFrame, nibs use the designated initializer -initWithCoder for creating UIViews.

```
-(id) initWithCoder:(NSCoder*)decoder
```

</SLIDE>

The NSCoder parameter is a reference to an un-archiver used by the *init* method to decode an XML representation of a view.

Next, we'll look at an example from Foundation, NSString:

Scene 2: On-Set

Since there are a variety of ways a programmer may wish to generate a new string, NSString provides a number of different init methods.
<SLIDE>

Some examples:

- **-(instancetype) initWithBytes:length:encoding** - allows you to create an NSString object from a raw C data-buffer of a given size and a known encoding
- **-(instancetype) initWithFormat:** - lets you initialized a new string from a string specified by a format (similar to how NSLog() works)
- **-(instanceType) initWithUTF8String:** generates an NSString from a standard C char array.
- **-(instancetype) initWithString:** creates an NSString object that is a distinct copy of a string. This method creates a new NSString object reference.

</SLIDE>

NSString also provides a number of class *factory-methods.*

A "factory method" is a class method which wraps complex alloc/init invocations to provide a more convenient way to create objects. An example is +stringWithString, which copies an NSString, but is a little shorter and clearer than the alloc/init way of doing it.

We'll look at factory methods more in the next video.

Questions:

1. True or False: UIView has two designated initializers (ans. true)
2. True or False: UIView's programmatic designated initializer is -initWithSquare: (ans. false)
3. True or False: -initWithCoder: is for views created in interface builder. (ans. true)
4. True or False: Interface builder views are encoded in JSON. (ans false)
5. True or False: C structures are never used in Objective-C (ans. false)