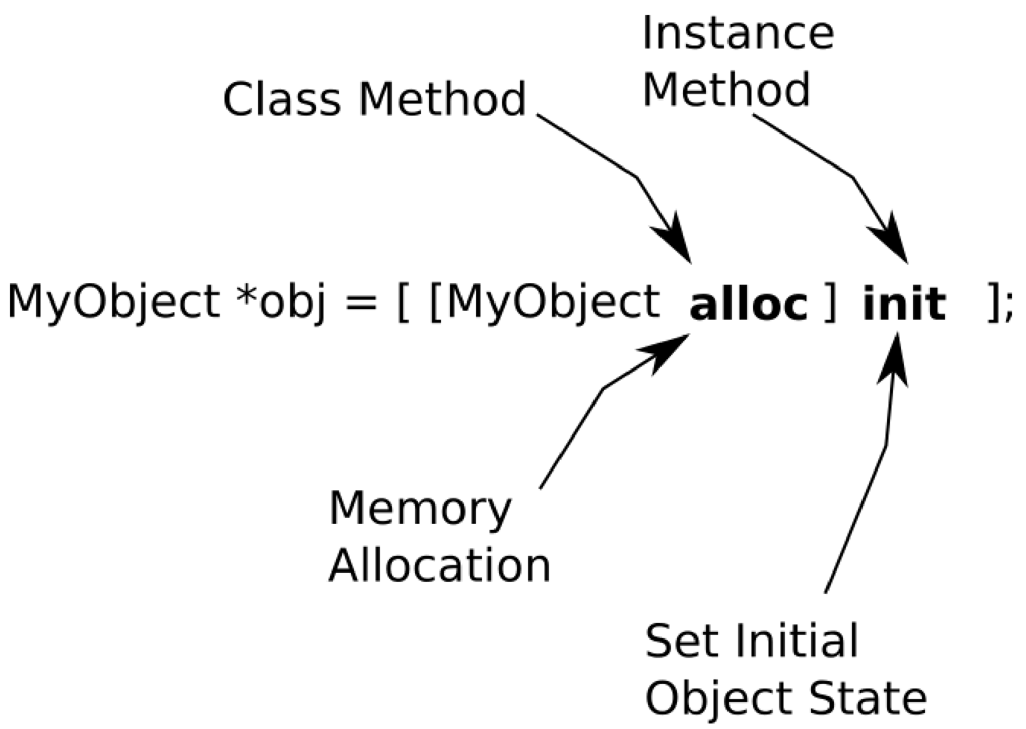
## Stage 2: video 1

### Scene 1: On-Set

In this video we'll examine two-stage object creation using alloc and init methods.

Here's a typical example of Objective-C object creation using a hypothetical class, **MyObject**:

**<MOTION arrows/annotations should appeared around the line of code as I mention them>**



The object, *obj,* is instantiated using the *alloc* class method of the object's type, followed by an *init* instance method called on the *alloc's* return value.

To review a little: an instance method is executed by an existing instance of a class, while a class method is executed by the class itself, and so is not tied to any specific instance.

*Alloc* (as its name suggests) allocates the memory for an instance, does low-level configuration of that *instance*, and connects it to the Objective-C runtime system. It provides a “blank slate” object ready for initialization.

**</MOTION>**

A class method, such as alloc, is necessary for obtaining an instance, but, since it’s a method acting on a class, not on a specific instance, its not ideal for initialization. It’s limitation is that it can’t act on an individual instance without being explicitly passed a reference to that instance. An instance method, on the other hand, has access to *“self”* (which is a reference to the instance), and *“super”* (which is a reference to the instance, cast to it’s parent type).

The use of “self” and “super” references in *init* methods are the most common form of *introspection* in Objective-C.

Thanks to the “self” reference available in instance methods, initialization can be shared across methods within a class and within a class hierarchy without the need to pass parameters.
As we’ll see in the following example, this allows object configuration to be easily customized while maximizing code reuse.

## Scene 2: Screencast

We’ll use the XCode “Command Line Tool” template for simplicity since we’re working with Foundation classes below the level of the GUI. The Cocoa in this example will work equally well on MacOS and iOS (or Cocoa-Touch).

We’ll call our new project: Alloc_Init
*// create project*
*// open main.m*

This is going to be a small amount of code, so we’ll be doing all our work in the main module. This will let us see everything on one screen.

Our project will simply create a Circle object which, for the moment, will only have a diameter.

First we'll add a class called Circle which is a subclass of NSObject and has a single float property, diameter. As I mentioned, we're doing everything in main(), so the interface and implementation blocks for Circle will be here.

// create Circle interface block

The class should have an initialization method that allows the property to be set at creation time, so we'll add one. <start typing in -initWithDiameter> The return type for our method could be either *id* (which is the type for a generic objective-C reference that can point to an instance of any NSObject subclass) or *instancetype*. *Instancetype* is more constrained than *id* and is limited to members of the Circle class' lineage. That is to say, it can refer to any instance of a subclass of Circle, but not just any NSObject subclass.

// create implementation block & -initWithDiameter + self = [super init];

We set *self* equal to the return value of *[super init]* rather than assuming initialization of the current self reference, because, in some rare cases, an init method may change our alloc'd reference. This is also part of the reason why we do the call to the parent class' init first.

*// check for nil self*

Then we verify that the returned *self* reference is not *nil* (which might happen if there is a failure in initialization)*,*

*// set diameter*

then initialize our diameter property to the value passed into the method as 'd'.

Since our new init method provides the most thorough initialization (that is, it initializes all the object's properties), it makes sense to make it the *designated initializer.*

The “designated initializer” is the central init method called by all other initializer methods and is usually the one that takes the most parameters.

Designated initializers are important for consistent configuration of new objects. They funnel execution up to the parent initialization methods on behalf of the other class init methods. This ensures a standardized initialization process.

Since we have a designated initializer, we should make sure it’s always called. So, we’ll need to override the standard, “vanilla”, initialization method, *init*.

The new *init* calls initWithDiameter, so we have the opportunity to set the default value for newly created Circle objects. We’ll choose a default value of 3. From now on, any Circle created with an unspecified value for the diameter will have that property set to 3.

We’ll step through the initialization process for our Circle class with the debugger so we can observe exactly how it works

1. I’ll set an execution breakpoint in XCode by clicking on the line number where we want to begin stepping through execution.
2. After building and running we’re stopped at our breakpoint.
3. We follow execution into Circle’s init method using the XCode *step-into* button.
4. *stepping-in* again we find ourselves in the designated initializer.
5. Using the “p” command on the LLDB command line, we see that the designated initializer was called with the default value of 3 as the *d* parameter.
6. Using the “*step-over”* button allows us to step forward in the thread’s execution, while remaining at the same level within the call stack.
7. Following the execution of initWithDiameter, by clicking *“step-over”* again, brings us into the conditional block that sets the value of diameter for this instance.
8. At this point in execution, *self* refers to the instance that will be returned to main() and become the Circle *c*.

9. In addition to the simple “p”, LLDB has a “po” or “print object” command used to examine the state of objective-C objects. “po self” displays our Circle instance with the pre-initialized value for diameter. If we step once more, (using *up-arrow* to repeat the command) we see the value of diameter set to our chosen default value.

## Questions:

1. True or False: *Alloc* is a class method. (ans. true)
2. True or False: *init* is an instance method (ans. true)
3. In an instance method “super” is? (ans. c)
    a. A reference to the class of the instance.
    b. Exactly the same as “self”
    c. “self” cast to the type of the instance’s superclass.
4. True or False: When implementing an init method, the parent class’ init method should always be called last. (ans. false)
5. True or False: A class has a designated initializer to ensure consistent initialization. (ans. true)
6. True or False: Foundation Cocoa is the same across MacOS and iOS. (ans. true)