Stage 3.2 - Examples from Cocoa

Scene 1: On-Set

Now we'll look at some examples of singletons from Apple's Cocoa frameworks.

NSUserDefaults

NSUserDefaults is a simple mechanism for data persistence. It provides a dictionary interface for storing small amounts of app specific data. Typically, these are things such as configuration preferences, high scores(in the case of a game), or a date stamp of the last execution of the app.

NSUserDefault reads from and writes to the device's filesystem, but hides the complexity associated with file I/O.

NSUserDefaults offers the *+standardUserDefaults* class method for retrieving the singleton instance. Storing values is done through -setObject:forKey:(or type specific a method for simple types). Retrieving values is done with -objectForKey:

<KEYNOTE SLIDE showing NSUserDefaults example>

```
NSNumber *score = @342;
NSUserDefaults *userDefaults = [NSUserDefaults standardUserDefaults];
[userDefaults setObject:score forKey:@"high_score"];
[userDefaults synchronize];
```

This code snippet shows using -setObject:forKey to save an NSNumber(342) with the "high_score" key in the NSUserDefaults persistent store.
The call to -synchronize forces the record to be written immediately. It is usually safe to leave this all out as NSUserDefaults data is written out periodically and also prior to application exit.

```
NSNumber *Oldscore = [[NSUserDefaults  standardUserDefaults]
objectForKey:@"high_score"];
```

This call to -objectForKey can be used to retrieve the stored "high_score" value, which we would expect to be 342.

</KEYNOTE SLIDE>

NSFileManager

NSFileManager is the primary interface for file system interaction in Cocoa. It provides functionality for file-system, reading, writing, directory traversing, creating paths and interacting with iCloud ubiquity containers.
Though it exposes more file system details than NSUserDefaults, it still abstracts away much of the Unix file system complexity.

<KEYNOTE SLIDE showing NSFileManager Methods>

Given an NSString path for some filesystem location: dirPath
This code snippet will test for the existence of a file “somefile.txt”

```
NSFileManager *fm = [NSFileManager defaultManager];
NSString *filePath = [dirPath stringByAppendingPathComponent:@”somefile.txt”];
BOOL exists = [fm fileExistsAtPath:filePath];
```

As is typical, NSfileManager is used as a singleton through the +defaultManager factory method.
The NSFileManager singleton makes sense since there is only one file system for the application to interact with.

</KEYNOTE SLIDE>
tcal
UIApplication

A ubiquitous singleton in iOS (as one is present in every running iOS app) is UIApplication. UIApplication represents the app in the interface between the operating system and a running application.
<KEYNOTE SLIDE showing UIApplicationDelegate Methods>

```
-(BOOL) application:(UIApplication*) willFinishLaunchingWithOptions:(NSDictionary*)
-(BOOL) application:(UIApplication*) didFinishLaunchingWithOptions:(NSDictionary*)

-(void) applicationDidEnterBackground:(UIApplication*)
-(void) applicationWillTerminate:(UIApplication*)
-(void) applicationDidReceiveMemoryWarning:(UIApplication*)
```

The instance reference is available through the +sharedApplication class method, however most interactions occur through the UIApplicationDelegate protocol.

These methods are the some of the first user code to execute in an application, specifically: -application:willFinishLaunchingWithOptions and -application:DidFinishLaunchingWithOptions:

-applicationDidEnterBackground, -applicationWillTerminate and -applicationDidRecieveMemoryWarning are all critical for reacting to important events in the lifecycle for your app.

</KEYNOTE SLIDE>