Chapter 18
Saving, Loading, and Application States

- Archiving
- Application Sandbox
- NSKeyedArchiver and NSKeyedUnarchiver
- Application States and transitions
- Writing to the filesystem using NSData

Archiving

- The model objects are the entities that hold the user data.
- Saving and loading data is basically the writing and reading information in the model objects to some data store.
- Arching is one of the most common ways of persisting data model objects in iOS.
- Classes whose instances need to be archived and unarchived must conform to the NSCoding protocol

NSCoding Protocol Reference

- NSCoding Protocol Reference:

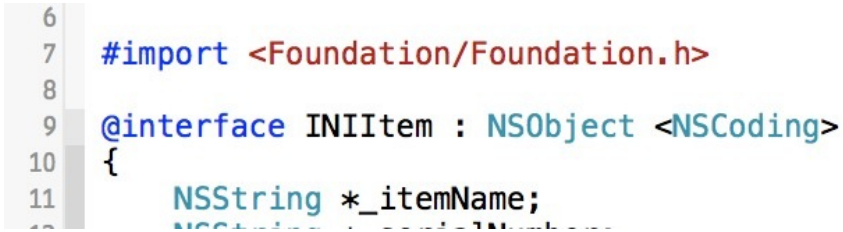
  The NSCoding protocol declares the two methods that a class must implement so that instances of that class can be encoded and decoded. This capability provides the basis for archiving (where objects and other structures are stored on disk) and distribution (where objects are copied to different address spaces).

  Initializing with a Coder

  - -(instancetype)initWithCoder: (NSCoder *)aCoder (required method)

  Encoding with a Coder

  - -(void)encodeWithCoder: (NSCoder *)aCoder (required method)

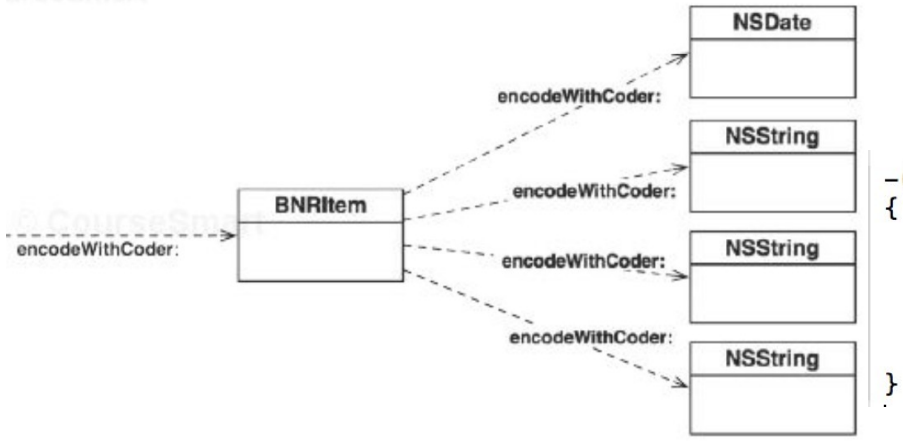


encodeWithCoder:

- When a INIItem is sent the message encodeWithCoder:, it will encode all of its properties into the NSCoder object that is passed as an argument.
- For saving, we will use NSCoder to write out a stream of data that will be stored on the filesystem.
- The stream of data will be organized as key-value pairs.

Encoding Data

- encodeArrayOfObjCType:count:at:
- encodeBool:forKey:
- encodeBycopyObject:
- encodeByrefObject:
- encodeBytes:length:
- encodeBytes:length:forKey:
- encodeConditionalObject:
- encodeConditionalObject:forKey:
- encodeDataObject:
- encodeDouble:forKey:
- encodeFloat:forKey:
- encodeInt:forKey:
- encodeInteger:forKey:
- encodeInt32:forKey:
- encodeInt64:forKey:
- encodeObject:
- encodeObject:forKey:
- encodeRootObject:
- encodeValueOfObjCType:at:
- encodeValuesOfObjCTypes:




```
-(void) encodeWithCoder:(NSCoder *) aCoder
{
    [aCoder encodeObject: self.itemName forKey:@"itemName"];
    [aCoder encodeObject: self.serialNumber forKey:@"serialNumber"];
    [aCoder encodeObject: self.dateCreated forKey:@"dateCreated"];
    [aCoder encodeObject: self.itemKey forKey:@"itemKey"];
    [aCoder encodeInt: self.valueInDollars forKey:@"valueInDollars"];
}
```

initWithCoder:

- This method returns an object initialized from data in a given unarchiver.
- The purpose of the key used when encoding is to retrieve the encoded value when this instance of INIItem is loaded from the filesystem later.
- Objects being loaded from an archive are sent the message initWithCoder:
- This method should grab all of the objects that were encoded in encodeWithCoder: and assign them to the appropriate instance variable.

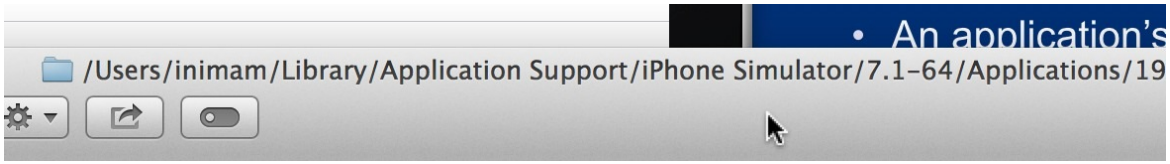
Decoding Data

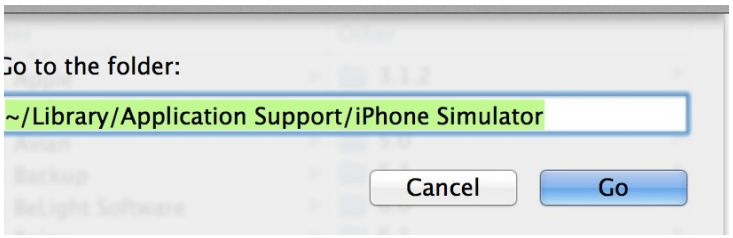
- decodeArrayOfObjCType:count:at:
- decodeBoolForKey:
- decodeBytesForKey:returnedLength:
- decodeBytesWithReturnedLength:
- decodeDataObject
- decodeDoubleForKey:
- decodeFloatForKey:
- decodeIntForKey:
- decodeIntegerForKey:
- decodeInt32ForKey:
- decodeInt64ForKey:
- decodeObject
- decodeObjectForKey:
- decodeValueOfObjCType:at:
- decodeValuesOfObjCTypes:
- decodeObjectOfClass:forKey:
- decodeObjectOfClasses:forKey:
- decodePropertyListForKey:

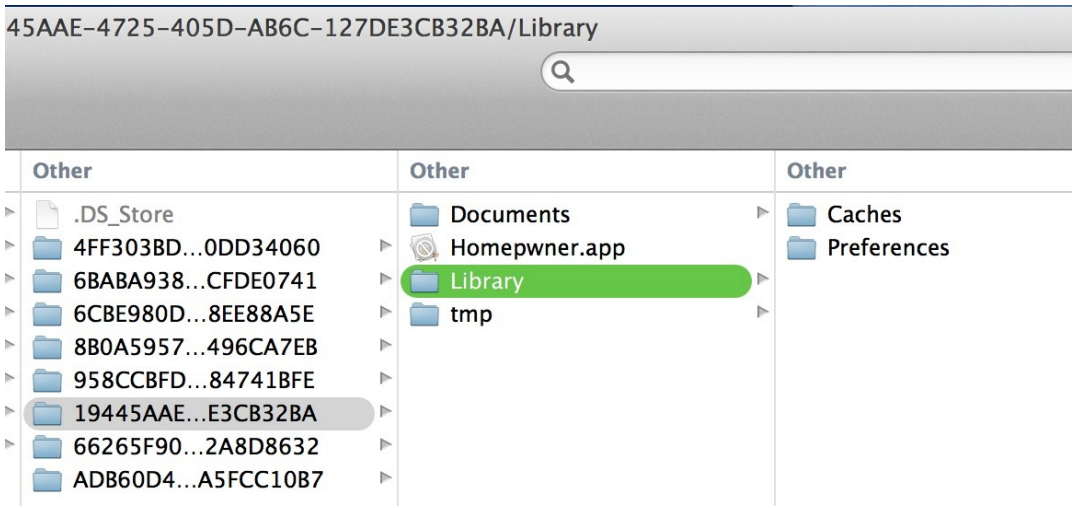
```
-(instancetype) initWithCoder:(NSCoder *) aDecoder
{
    self = [super init];
    if (self) {
        _itemName = [aDecoder decodeObjectForKey:@"itemName"];
        _serialNumber = [aDecoder decodeObjectForKey:@"serialNumber"];
        _dateCreated = [aDecoder decodeObjectForKey:@"dateCreated"];
        _itemKey = [aDecoder decodeObjectForKey:@"itemKey"];
        _valueInDollars = [aDecoder decodeIntForKey:@"valueInDollars"];
    }
    return self;
}
```

Application Sandbox

- Every iOS application has its own application sandbox.
- An application sandbox is a directory on the filesystem that is barricaded from the rest of the filesystem.
- Every application must stay in its sandbox, and no other application can access your sandbox.
- An application's sandbox contains:
  - The application bundle (read only)
  - Library/Caches
  - Library/Preferences
  - Documents (backed up, Use to write user persistent data)
  - tmp (not backed up)







Constructing a file path

- The instances of INIItem from Homeowner will be saved to a single file in the Documents directory.
- INIItemStore will handle writing to and reading from that file.
- INIItemStore needs to construct a path to this file.

```
-(NSString *) itemArchivePath
{
    //Make sure that the first argument is NSDocumentDirectory and not NSDocumentationDirectory

    NSArray *documentDirectories = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);

    //Get the one document directory from that list

    NSString *documentDirectory = [documentDirectories firstObject];
    return [documentDirectory stringByAppendingPathComponent:@"items.archive"];
}
```

```
Declaration  NSArray * NSSearchPathForDirectoriesInDomains (
                 NSSearchPathDirectory directory,
                 NSSearchPathDomainMask domainMask,
                 BOOL expandTilde
             );
```

```
43      // Here is the real (secret) initializer
44  - (instancetype) initPrivate
45  {
46      self = [super init];
47      if ( self) {
48          _privateItems = [[NSMutableArray alloc] init];
49      }
50      NSLog(@"%@", [self itemArchivePath]);
51      return self;
52  }
```

```
:/iPhone Simulator/7.1-64/Applications/19445AAE-4725-405D-AB6C-127DE3CB32BA/Documents/items.archive
```

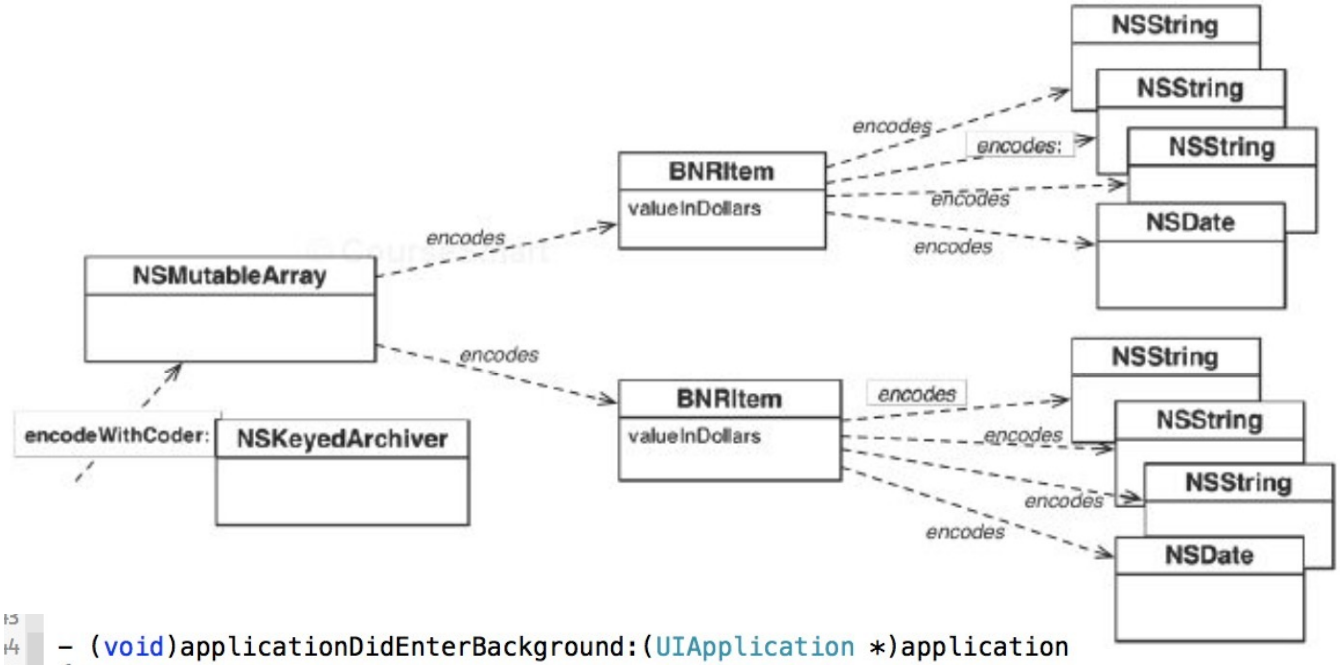
NSKeyedArchiver and NSKeyedUnarchiver

- To save INIItems we need to start NSKeyedArchiver when the application exits, in iOS this is when the user pushes the home button and the application enters the background state.
- We need to create a method in INIItemStore that saves the items (will call it saveChanges)
- We need to modify HomeownerAppDelegate and implement applicationDidEnterBackground: so that we can kick of saving the changes.

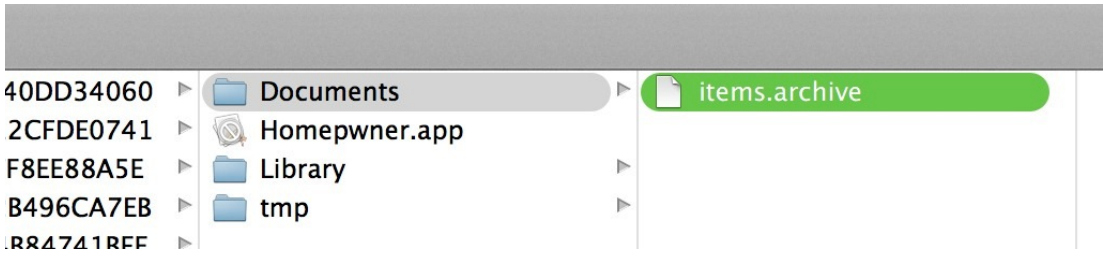
```
- (BOOL) saveChanges
{
    NSString *path = [self itemArchivePath];
    // Returns YES on success
    return [NSKeyedArchiver archiveRootObject: self.privateItems toFile: path];
}
```

```
- (void)applicationDidEnterBackground:(UIApplication *)application
{
    BOOL success = [[BNRItemStore sharedStore] saveChanges];
    if (success)
    {
        NSLog(@"Saved all of the BNRitems");
    }
    else
    {
        NSLog(@"Could not save any of the BNRItems");
    }
}
```

Archiving the privateItems array




```
- (void)applicationDidEnterBackground:(UIApplication *)application
{
    BOOL success = [[INIItemStore sharedStore] saveChanges];
    if (success) {
        NSLog(@"Saved all of the BNRItems");
    } else {
        NSLog(@"Could not save any of the BNRItems");
    }
}
```

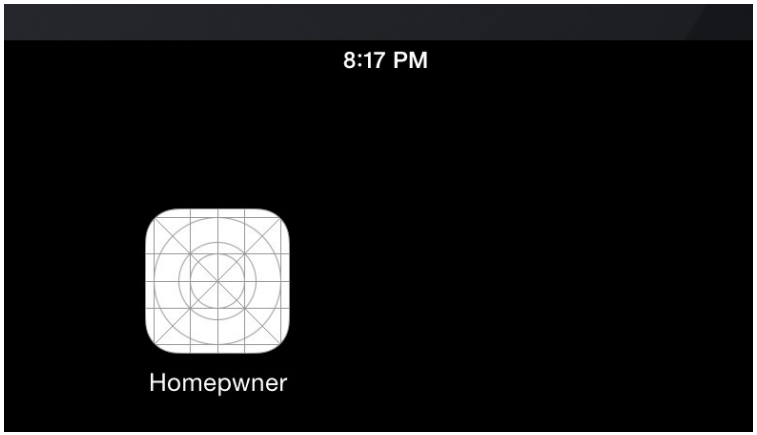


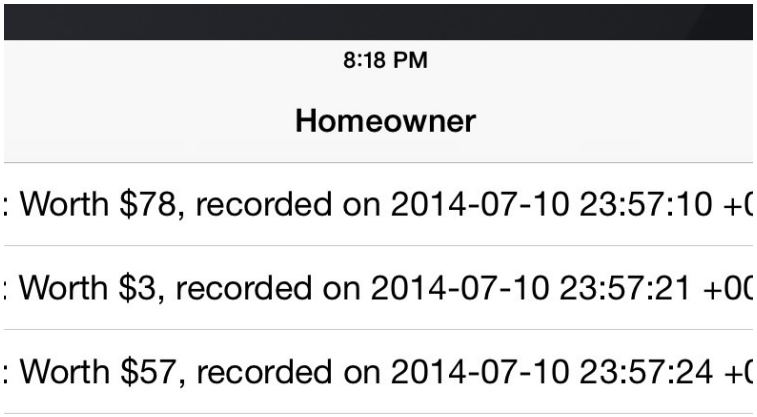

```
bplist00.......OPX$versionX$o
bjectsY$archiverT$top........
......")*/678;<CDEHIU$null...
..ZNS.objectsV$class.........
................\serialNumber
[dateCreatedXitemName^valueIn
DollarsWitemKey.......N....[R
usty SporkU3K0Y4... !WNS.time
#A.o_.|.....#$%&Z$classnameX$
classesVNSDate.'(VNSDateXNSOb
ject_.$F732C3FA-8070-4773-9AE
A-09A87B13AD55.#$+,WINIItem.-
.WINIItemXNSObject.......0123
.5............[Shiny SporkU0X
2Q7...9!#A.o_.WP..._.$E3D88C6
B-3784-42C1-A576-552247CBA270
.......=>?@.B.......9....[Rus
ty SporkU0S3R9...F!#A.o_..v..
._.$12EEFB26-6768-4E54-B5E9-0
CFA49E79140.#$JK^NSMutableArr
ay.LMN^NSMutableArrayWNSArray
XNSObject_..NSKeyedArchiver.Q
RTroot.........#.-.2.7.N.T.Y.
d.k.o.q.s.u.w................
.............................
....&.M.R.Z.].e.n.{.}........
.............................
..........2.7.F.J.Y.a.j.|....
................S.............
...
```

Loading items from items.archive

- To load instances of INIItem when the application launches, we will use the class NSKeyedUnarchiver when the INIItemStore is created.
- This means that we have to modify initPrivate.

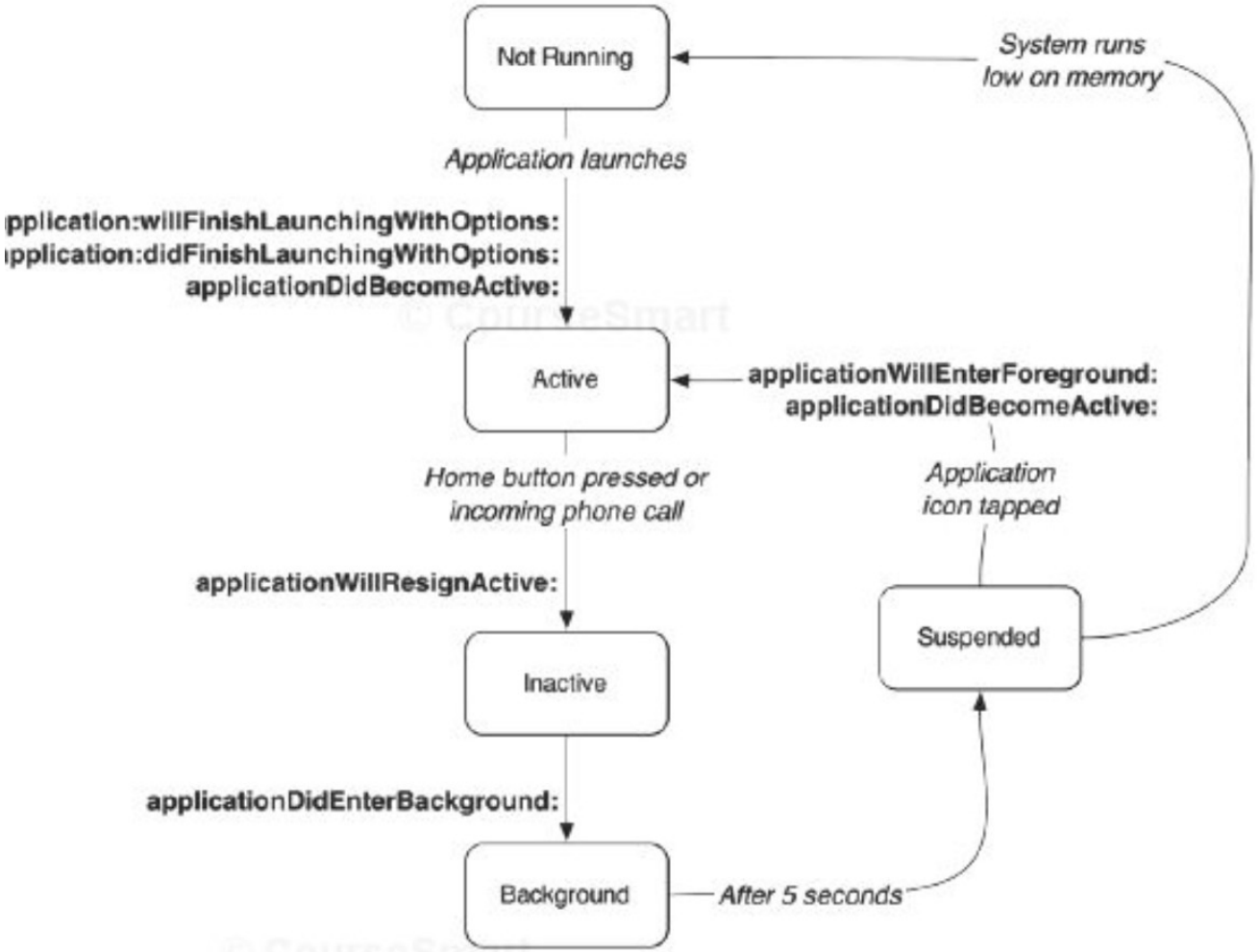
```
    // Here is the real (secret) initializer
- (instancetype) initPrivate
{
    self = [super init];
    if ( self) {
        NSString *path = [self itemArchivePath];
        _privateItems = [NSKeyedUnarchiver unarchiveObjectWithFile: path];

        // If the array hadn't been saved previously, create a new empty one
        if (!_privateItems) {
            _privateItems = [[NSMutableArray alloc] init];
        }
    }
    return self;
}
```

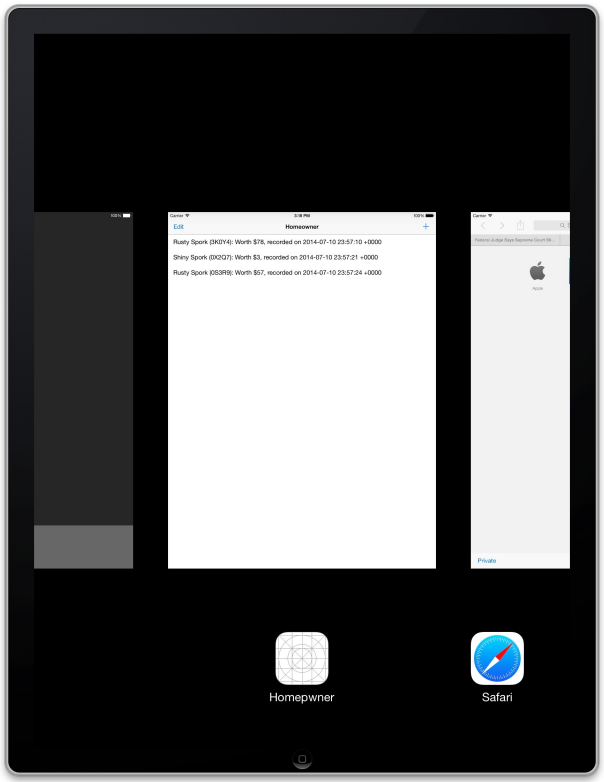





```
- (INIItem *) createItem
{
    // INIItem *item = [INIItem randomItem];
    INIItem *item = [[INIItem alloc] init];
    [self.privateItems addObject: item];
    return item;
}
```

The states of the application






```
(BOOL)application:(UIApplication *)app
          didFinishLaunchingWithOptions:(NSDictionary *)options
(void)applicationDidBecomeActive:(UIApplication *)app;
(void)applicationWillResignActive:(UIApplication *)app;
(void)applicationDidEnterBackground:(UIApplication *)app;
(void)applicationWillEnterForeground:(UIApplication *)app;
```

State	Visible	Receives Events	Executes Code
Not Running	No	No	No
Active	Yes	Yes	Yes
Inactive	Mostly	No	Yes
Background	No	No	Yes
Suspended	No	No	No

Writing to the filesystem using NSData

- The current version of INIImageStore stores images in a dictionary that is kept in the cache as long as the application is active.
- To be able to save and retrieve images based on the itemKey we need to write them and reload them to the documents subfolder once they are associated with an item via the itemKey
- Changes that need to be made to INIImageStore:
  - Modify setImage:forKey to save the image to the file system
  - Modify deleteImage:forKey to delete the image from the file system
  - Modify imageForKey: so that it will load the image from the file system if it is already in the dictionary

```
18  -(NSString *)imagePathForKey:(NSString *)key;
19
20  @end
```

Composing the path name

```
-(NSString *)imagePathForKey:(NSString *)key
{
    NSArray *documentDirectories = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);
    NSString *documentDirectory = [documentDirectories firstObject];
    return [documentDirectory stringByAppendingPathComponent: key];
}
```

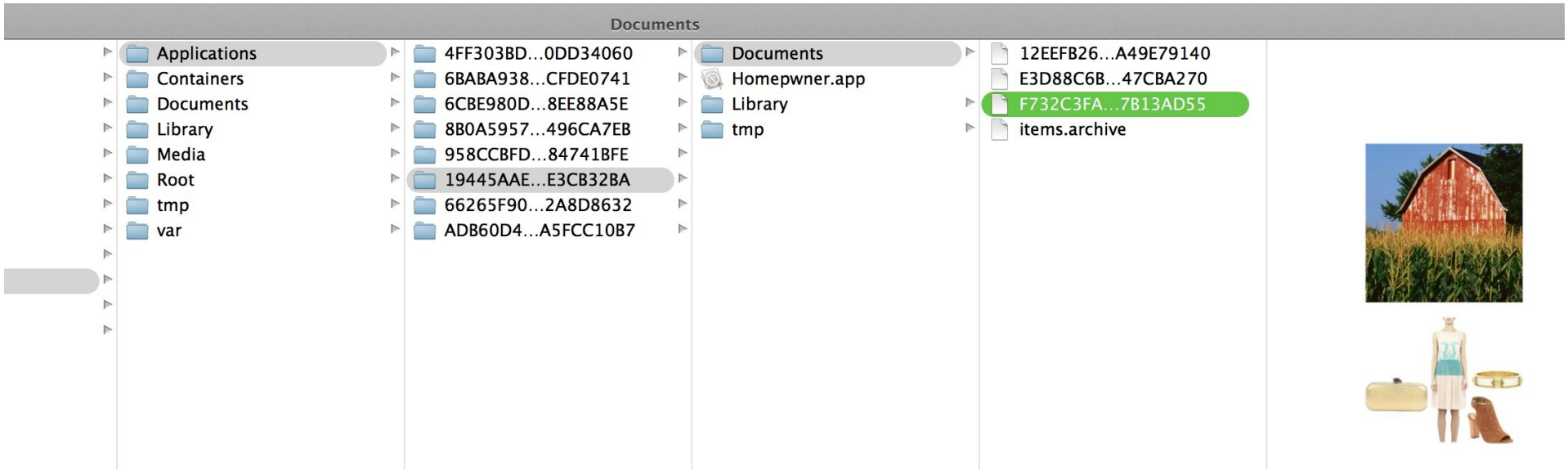
Modifying INIImageStore

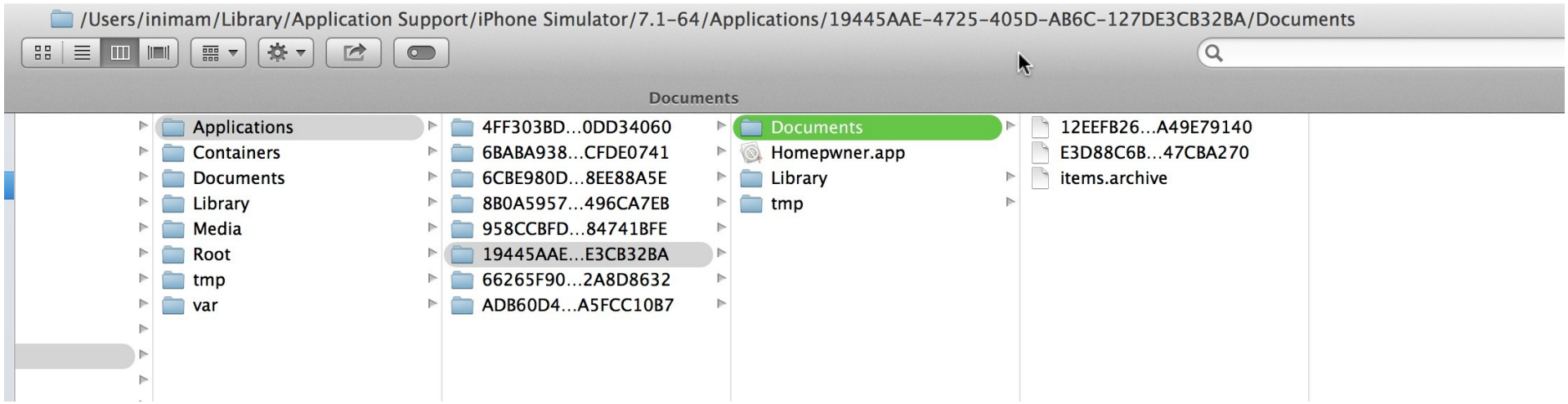
```objc
-(void)setImage:(UIImage *)image forKey:(NSString *)key
{
    self.dictionary[key] = image;

    //Create full path for image
    NSString *imagePath = [self imagePathForKey: key];

    //Turn image into JPEG data
    NSData *data = UIImageJPEGRepresentation(image, 0.5);

    //Write it to full path
    [data writeToFile: imagePath atomically: YES];
}
```

```objc
-(void)deleteImageForKey:(NSString *)key
{
    if (! key){
        return;
    }
    [self.dictionary removeObjectForKey: key];
    NSString *imagePath = [self imagePathForKey: key];
    [[NSFileManager defaultManager]
    removeItemAtPath: imagePath error: nil];
}
```

```objc
-(UIImage *)imageForKey:(NSString *)key
{
   //If possible, get it from the dictionary

    UIImage *result = self.dictionary[key];

    if (!result){
        NSString *imagePath = [self imagePathForKey: key];

        //Create UIImage object from file
        result = [UIImage imageWithContentsOfFile: imagePath];

        //If we found an image on the file system, place it into the cache
        if (result){
            self.dictionary[key] = result;
        } else {
            NSLog(@"Error: unable to find %@", imagePath);
        }
    }
    return result;
}
```

Images Files



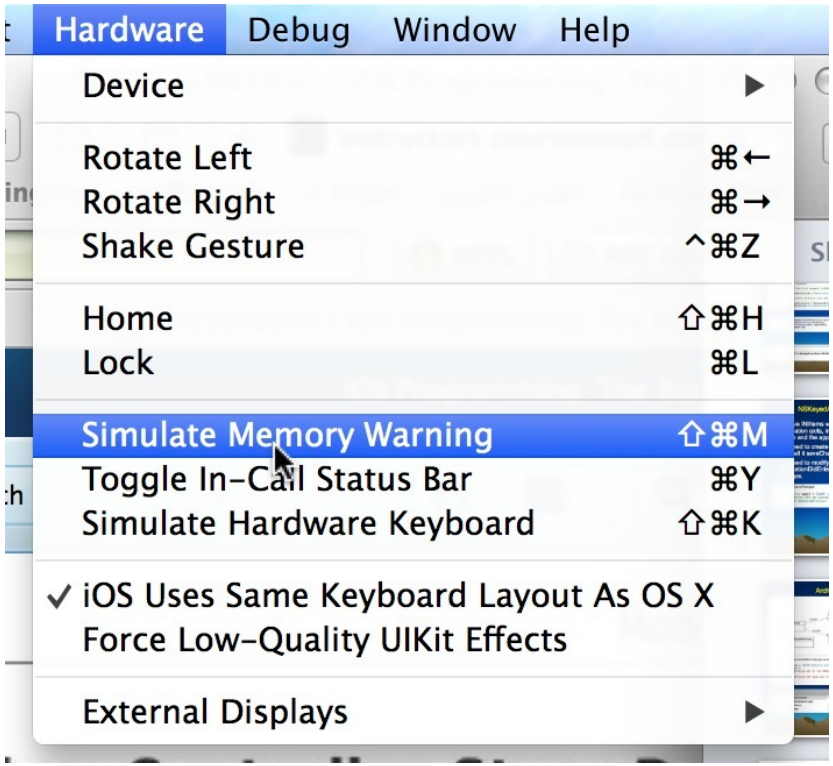


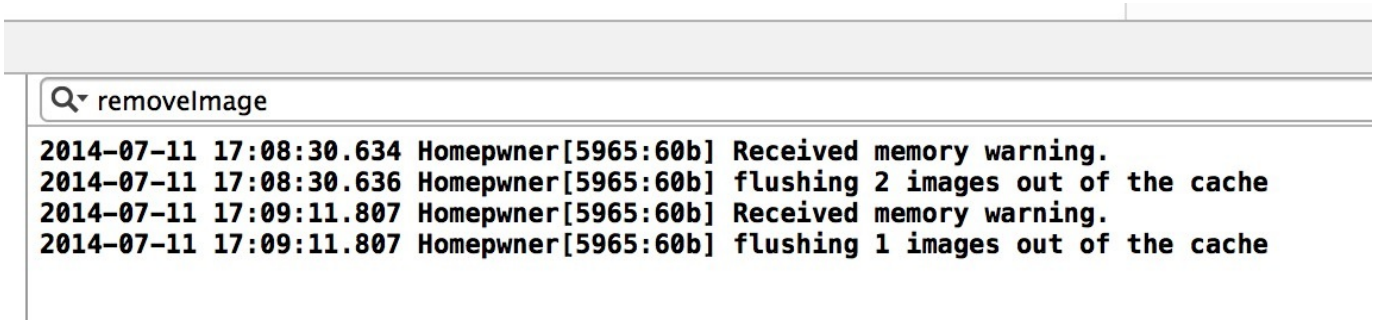
NSNotificationCenter and Low- Memory Warnings

- When the system is running low on RAM, it issues a low memory warning to the running application.
- The application responds by freeing up any resources that it does not need at the moment and can easily recreate.
- View controllers, during a low memory warning, are sent the message didReceiveMemoryWarning.
- Objects other than view controllers that have data that they are not using and can recreate later should register to receive low memory notification and act by clearing all memory they do not need by relinquishing ownership of the data occupying memory.
- In our case the INIImageStore is such object.

```objc
// Secret designated initializer
-(instancetype)initPrivate
{
    self = [super init];
    if (self){
        _dictionary = [[NSMutableDictionary alloc] init];

        NSNotificationCenter *nc = [NSNotificationCenter defaultCenter];
        [nc addObserver: self
               selector: @selector(clearCache:)
                   name: UIApplicationDidReceiveMemoryWarningNotification
                 object: nil];
    }
    return self;
}
```

```objc
-(void)clearCache:(NSNotification *)note
{
    NSLog(@"flushing %lu images out of the cache", (unsigned long)[self.dictionary count]);
    [self.dictionary removeAllObjects];
}
```


Hardware
Debug
Window
Help
Device
Rotate Left ⌘←
Rotate Right ⌘→
Shake Gesture ^⌘Z
Home ⇧⌘H
Lock ⌘L
Simulate Memory Warning ⇧⌘M
Toggle In-Call Status Bar ⌘Y
Simulate Hardware Keyboard ⇧⌘K
iOS Uses Same Keyboard Layout As OS X
Force Low-Quality UIKit Effects
External Displays


removeImage
2014-07-11 17:08:30.634 Homepwner[5965:60b] Received memory warning.
2014-07-11 17:08:30.636 Homepwner[5965:60b] flushing 2 images out of the cache
2014-07-11 17:09:11.807 Homepwner[5965:60b] Received memory warning.
2014-07-11 17:09:11.807 Homepwner[5965:60b] flushing 1 images out of the cache