Steps

00-Start

Creating a project, understanding bundle identifiers, class prefixes. The concept of universal apps.

1. Create a new project.
    1. Create a single-view application.
    2. Name it 'Diary'.
    3. Set your organisation name to your name.
    4. Set the bundle identifier to your domain name, but reversed.
    5. Set the class prefix to 'DY'.
    6. Set the device type to 'iPhone'.

We're now going to make the app use the provided icons.

1. Open the Images.xcassets file.
2. Select the AppIcon image set.
3. In the Attributes selector, turn on 'iPad 7.0 and later sizes'.
4. Drag the appropriate images into the image slots.

Creating new classes. View controllers. Using the interface builder. Making connections.

We're going to make DYViewController become the DYNoteViewController.

1. Rename DYViewController to DYNoteViewController. -File Name, Interface/Implementation name and #import name in file.
2. Add a text view. Make it fill the screen.

01-ViewControllers

Next, we'll make the view controller that lists all of the notes.

1. Add a Navigation Controller to the storyboard.
2. Drag the Initial View Controller arrow from the Note View Controller to the Navigation Controller.
3. Select the Table View.
    1. Set its Content to Static Cells.
    2. Select the Table View Section. Set its number of rows to 1.
    3. Select the cell.
    4. Set its Style to Basic.

Later, we'll make it so that there's more than one cell - one for each note you add.

Now, we'll make it so that tapping the cell takes you to the Note View Controller.

1. Control-drag from the cell to the Note View Controller, and create a Push segue

One last step: set the title of the view controller.

1. Select the bar at the top of the table view controller, and change its title to Notes.
2. Select the bar at the top of the DYNoteViewController, and change the Title to Note.

02-Note

Objective-C syntax. Properties. Methods.

1. Make a new Objective-C class named "DYNote". Make it a subclass of NSObject.
2. Add content as per DYNote.h

DYNote.h

```
+#import <Foundation/Foundation.h>
+
+@interface DYNote : NSObject
+
+/// The text stored in the note.
+// 'nonatomic' means that the setter isn't guaranteed to be thread-safe.
+@property (nonatomic, strong) NSString* text;
+
+/// The date and time that the note was created.
+@property (readonly) NSDate* createdDate;
+
+/// The date and time that the note was last modified.
+@property (readonly) NSDate* modifiedDate;
+
+/// Returns the number of words in the note.
+- (int) wordCount;
+
+@end
```

and DYNote.m.

DYNote.m

```
+#import "DYNote.h"
+
+@implementation DYNote
+
+// The init method is called when the object is created.
+- (id)init
+{
+    self = [super init];
+    if (self) {
+
+        // Set the dates. These are declared as 'readonly' in the header,
so doing this won't work:
+        // self.createdDate = ...
+        // Instead, we set the variable directly:
+        // _createdDate = ...
+
+        // Created date is now
+        _createdDate = [NSDate date];
+
+        // Last modified date is also now
+        _modifiedDate = [NSDate date];
+
+    }
+    return self;
+}
+
+// The word count method splits the text up by spaces, and counts the
number of pieces.
+- (int) wordCount {
+
+    NSArray* words = [self.text componentsSeparatedByString:@" "];
+
+    return [words count];
+}
+
+// Override the 'setText:' method, which is called when the 'text' property
is set.
+// We're overriding it because we want the 'last modified' date to be
updated whenever the property is updated.
+- (void)setText:(NSString *)text {
+    _text = text;
+    _modifiedDate = [NSDate date];
+}
+
+@end
```

Key features of the Note object at this point

```
@property (nonatomic, strong) NSString* text;
@property (readonly) NSDate* createdDate;
@property (readonly) NSDate* modifiedDate;
- (int) wordCount;
```

03-NoteCollection

Foundation. Arrays and other container objects. Mutable and immutable objects.

First, we'll create the code for the Notes screen (the table view controller.)

1. Create a new Objective-C object. Call it DYNoteListViewController and make it a subclass of UITableViewController.
2. Open the Storyboard. Select the Table View Controller. Make it use DYNoteListViewController as its class.
3. Provide the code for DYNoteListViewController.m (there's nothing in the header yet.)

DYNoteListViewController.h

```
+#import <UIKit/UIKit.h>
+
+@interface DYNoteListViewController : UITableViewController
+
+@end
```

DYNoteListViewController.m

```
+#import "DYNoteListViewController.h"
+#import "DYNote.h"
+
+// This is a 'class extension', which lets you add methods, properties and
variables
+// to a class without having to put them in the header file, which other
classes can see.
+// Anything you put in the class extension can only be accessed by this
class.
+@interface DYNoteListViewController () {
+    NSMutableArray* _notes;
+}
+
+@end
+
+@implementation DYNoteListViewController
+
+- (void)viewDidLoad {
+
+    // Create the array that stores the notes
+    _notes = [NSMutableArray array];
+
+    // Create a test note
+    DYNote* note = [[DYNote alloc] init];
+    note.text = @"Hello!";
+
+    // Add the note
+    [_notes addObject:note];
+}
+
+@end
```

04-NoteList

Table views. Delegates. Configuring table view cells.

1. Open the storyboard.
2. Change the Table View's Content from Static Cells to Dynamic Prototypes.
3. Select the table view cell.
4. Set its Identifier to NoteCell.
5. Add the `numberOfSectionsInTableView:`, `tableView: numberOfRowsInSection:` and `tableView: cellForRowAtIndexPath:` methods to

DYNoteListViewController.m.

```
#import "DYNoteListViewController.h"
+#import "DYNote.h"
```

```
+// This is a 'class extension', which lets you add methods, properties and
variables
+// to a class without having to put them in the header file, which other
classes can see.
+// Anything you put in the class extension can only be accessed by this
class.
@interface DYNoteListViewController () {
  NSMutableArray* _notes;
}

@end

@implementation DYNoteListViewController

- (void)viewDidLoad {
+
+    // Create the array that stores the notes
     _notes = [NSMutableArray array];
+
+    // Create 100 test notes
+
+    for (int i = 1; i <= 100; i++) {
+        // Make a note
+        DYNote* note = [[DYNote alloc] init];
+
+        // Create the note's text
+        NSString* text = [NSString stringWithFormat:@"Note %i", i];
+        note.text = text;
+
+        // Add the note to the list
+        [_notes addObject:note];
+    }
+}
+
+// Returns the number of sections (groups of cells) in the table
+- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
+    // There is only one section in this table.
+    return 1;
+}
+
+
+// Returns the number of rows (cells) in the given section.
+- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:
(NSInteger)section {
+
+    // There's only ever one section, so we don't need to worry about
checking the value of 'section'.
+
+    // The number of rows is equal to the number of notes we have.
+    return [_notes count];
```

```
+}
+
+// Returns a table view cell for use, which shows the data we want to
display.
+- (UITableViewCell*) tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath {
+
+    // Get a cell to use.
+    UITableViewCell* cell = [tableView
dequeueReusableCellWithIdentifier:@"NoteCell"];
+
+    // Work out which note should be shown in this cell.
+    DYNote* note = _notes[indexPath.row];
+
+    // Give the note's text to the cell.
+    cell.textLabel.text = note.text;
+
+    // Return the cell to the table view, which will then show it.
+    return cell;
+
}

@end
```

05-NoteEditing

Outlets. Segues.

First, set up DYNoteViewController by adding a property to store the note and connecting the text view to an outlet.

1. Go to the Note View Controller in the storyboard.
2. Connect the text field to the view controller - put the outlet in the class extension in DYNoteViewController.m.

DYNoteViewController.m +@property (weak, nonatomic) IBOutlet UITextView *noteTextView;

1. Add a DYNote* property called 'note' in DYNoteViewController.h.

DYNoteViewController.h

```
+import "DYNote.h"
...
+@property (strong) DYNote* note;
```

1. Implement the `viewWillAppear:` and `viewWillDisappear:` methods in DYNoteViewController.m.

DYNoteViewController.m.

```
+// Called when the view controller is about to appear.
+- (void)viewWillAppear:(BOOL)animated {
+    // Make the note text view use the text that's in the note.
+    self.noteTextView.text = self.note.text;
+}
+
+// Called just before the view controlled is about to go away.
+- (void)viewWillDisappear:(BOOL)animated {
+    // Store the text that's in the note text view into the note itself.
+    self.note.text = self.noteTextView.text;
+}
+
```

1. Import DYNoteViewController.m in DYNoteListViewController.m.
2. Implement the `prepareForSegue:` and `viewWillAppear:` in DYNoteListViewController.

DYNoteListViewController.m

```
+#import "DYNoteViewController.h"
...
+// Called when the view controller is about to move to another view
controller.
+- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
+
+    // If this is the "showNote" segue, then we're about to segue to the
Note View Controller because the user tapped on a table view cell.
+    if ([segue.identifier isEqualToString:@"showNote"]) {
+
+        // Get the note view controller we're about to move to.
+        DYNoteViewController* noteViewController =
segue.destinationViewController;
+
+        // Get the cell that was tapped on.
+        UITableViewCell* cell = sender;
+
+        // Work out which row this cell was.
+        NSIndexPath* indexPath = [self.tableView indexPathForCell:cell];
+
+        // Use that to get the appropriate note.
+        DYNote* note = _notes[indexPath.row];
+
+        noteViewController.note = note;
+    }
+}
+
+// Called when the view controller is about to appear.
+- (void)viewWillAppear:(BOOL)animated {
+
+    // Because the notes might have been changed, make the table view
reload all data.
+    [self.tableView reloadData];
+}
+
```

06-NoteAddingAndRemoval

1. Modify the `viewDidLoad` method in DYNoteListViewController to add the Edit button.

DYNoteListViewController.m

```
-(void)viewDidLoad{
...
+// Get the table view's "Edit" button, which will put the table into Edit
mode when tapped
+self.navigationItem.leftBarButtonItem = self.editButtonItem;
}
```

1. Implement the `tableView: commitEditingStyle: forRowAtIndexPath:` method.

DYNoteListViewController.m

```
+// Called when the user has tapped the Delete button.
+- (void)tableView:(UITableView *)tableView commitEditingStyle:
(UITableViewCellEditingStyle)editingStyle forRowAtIndexPath:(NSIndexPath
*)indexPath {
+
+    // We only want to take action if the edit that was made is a deletion.
+    if (editingStyle == UITableViewCellEditingStyleDelete) {
+
+        // Find the note that we're talking about
+        DYNote* note = _notes[indexPath.row];
+
+        // Remove it from the list of notes
+        [_notes removeObject:note];
+
+        // Finally, remove the cell.
+        [tableView deleteRowsAtIndexPaths:@[indexPath]
withRowAnimation:UITableViewRowAnimationLeft];
+
+    }
+}
```

Next, we'll add a + button to the top-right - when tapped, it'll add a new note.

1. Remove the code that creates the demo notes.

DYNoteListViewController.m

```
-(void)viewDidLoad{
...
-    // Create 100 test notes
-
-    for (int i = 1; i <= 100; i++) {
-        // Make a note
-        DYNote* note = [[DYNote alloc] init];
-
-        // Create the note's text
-        NSString* text = [NSString stringWithFormat:@"Note %i", i];
-        note.text = text;
-
-        // Add the note to the list
-        [_notes addObject:note];
```

1. Open the storyboard.
2. Add a bar button item to the top-right of the navigation bar in the Notes List View Controller.
3. Change the new button's identifier to "Add".
4. Connect the button to an action method, called "addNote:".
5. Implement the `addNote:` method.

DYNoteListViewController.m

```
+// Called when the user taps the Add button.
 +- (IBAction)addNote:(id)sender {
 +
 +    // Create a new, empty note
 +    DYNote* note = [[DYNote alloc] init];
 +
 +    note.text = @"New note";
 +
 +    // Insert the new note at the start of the array
 +    [_notes insertObject:note atIndex:0];
 +
 +    // Tell the table view to add a new cell for this note:
 +
 +    // First, create an index path that describes the position of the cell
 +
 +    NSIndexPath* indexPath = [NSIndexPath indexPathForRow:0 inSection:0];
 +
 +    // Now add the row
 +    [self.tableView insertRowsAtIndexPaths:@[indexPath]
withRowAnimation:UITableViewRowAnimationTop];
 +
 +}
```

07-CoreData

1. Create a new Managed Object Model. Call it "Diary".
2. Open the new model. Add a new Entity. Call it "Note".
3. Add an attribute called "text", which is a String.
4. Add two attributes: "createdDate" and "modifiedDate", both Dates.
5. Set the Class of the entity to `DYNote`.
6. Update DYNote.h and DYNote.m to reflect the latest changes. Basically, almost all of the code in these files are changed.

DYNote.h

```
#import <Foundation/Foundation.h>

+@import CoreData;
+
+// By making the object an NSManagedObject, it will know how to exist in a
database.
+@interface DYNote : NSManagedObject

/// The text stored in the note.
// 'nonatomic' means that the setter isn't guaranteed to be thread-safe.
@property (nonatomic, strong) NSString* text;

/// The date and time that the note was created.
@property (readonly) NSDate* createdDate;

/// The date and time that the note was last modified.
@property (readonly) NSDate* modifiedDate;

/// Returns the number of words in the note.
- (int) wordCount;

@end
```

DYNote.m

```
#import "DYNote.h"

+// By re-declaring these properties as read-write, this class can assign
values to the properties
+// while still keeping everyone else from being able to do so.
+@interface DYNote ()
+
+@property (readwrite) NSDate* createdDate;
+@property (readwrite) NSDate* modifiedDate;
+
```

```
+@end
+
@implementation DYNote

+// '@dynamic' tells the compiler to not create variables for these
properties - instead,
+// this storage will be handled by Core Data.
+@dynamic text;
+@dynamic createdDate;
+@dynamic modifiedDate;
+
+- (void)awakeFromInsert {
+    // Because these properties are readwrite, they can be assigned to.

+    // Created date is now
+    self.createdDate = [NSDate date];
+
+    // Last modified date is also now
+    self.modifiedDate = [NSDate date];
+
+}
+
+// Called when the object is about to be saved.
+-(void)willSave {
+
+    // When we're saved, update the last modified date.
+
+    // If the last modified date is more than 1 second ago, update it.
+    // (This prevents 'willSave' from infinitely recursing, because setting
+    // a property makes Core Data attempt to save the object again.)
+    NSDate* now = [NSDate date];
+
+    if ([now timeIntervalSinceDate:self.modifiedDate] > 1.0) {
+        self.modifiedDate = now;
}

// The word count method splits the text up by spaces, and counts the number
of pieces.
- (int) wordCount {

  NSArray* words = [self.text componentsSeparatedByString:@" "];

  return [words count];
}

@end
```

1. Add DYNoteStorage.

DYNoteStorage.h

```
+
+#import <Foundation/Foundation.h>
+
+// Using the '@import' keyword makes Xcode load the Core Data framework.
+@import CoreData;
+
+#import "DYNote.h"
+
+@interface DYNoteStorage : NSObject {
+
+
+}
+
+- (NSFetchedResultsController*) createFetchedResultsController;
+
++ (DYNoteStorage*) sharedStorage;
+
+- (DYNote*) createNote;
+- (void) deleteNote:(DYNote*)note;
+
+- (DYNote*) noteWithURL:(NSURL*)url;
+
+@end
```

DYNoteStorage.m

```
+#import "DYNoteStorage.h"
+
+// This variable stores the single, shared instance of this class.
+static DYNoteStorage* _sharedStorage;
+
+// We're using a class extension to store some private properties.
+@interface DYNoteStorage ()
+
+// The following properties are marked as 'nonatomic' because we're
overriding their getters.
+
+// The persistent store coordinator manages access to the actual database
file itself.
+@property (nonatomic) NSPersistentStoreCoordinator*
persistentStoreCoordinator;
+
+// The managed object model describes what kinds of objects live in the
database.
+@property (nonatomic) NSManagedObjectModel* managedObjectModel;
+
+// The managed object context is the 'bag' in which all objects exist.
+@property (nonatomic) NSManagedObjectContext* managedObjectContext;
+
```

```
+@end
+
+@implementation DYNoteStorage {
+}
+
+// Returns the single shared instance of the DYNoteStorage class.
++ (DYNoteStorage*) sharedStorage {
+
+    // The first time this method is called, the shared instance won't
exist.
+    // To ensure it's created (and only ever created once), use
dispatch_once.
+    static dispatch_once_t onceToken;
+    dispatch_once(&onceToken, ^{
+        _sharedStorage = [[DYNoteStorage alloc] init];
+    });
+
+    return _sharedStorage;
+}
+
+// Returns the location of the folder where the app can store documents.
+- (NSURL *)applicationDocumentsDirectory
+{
+    return [[[NSFileManager defaultManager]
URLsForDirectory:NSDocumentDirectory inDomains:NSUserDomainMask]
lastObject];
+}
+
+// Returns the managed object model.
+- (NSManagedObjectModel*) managedObjectModel {
+
+    // If the managed object model has already been created, then just
return it.
+    if (_managedObjectModel != nil)
+        return _managedObjectModel;
+
+    NSURL* modelURL = [[NSBundle mainBundle] URLForResource:@"Diary"
withExtension:@"momd"];
+
+    // ..and then load it.
+    _managedObjectModel = [[NSManagedObjectModel alloc]
initWithContentsOfURL:modelURL];
+
+    return _managedObjectModel;
+
+}
+
+// Returns the managed object context.
+- (NSManagedObjectContext *)managedObjectContext {
+
+    // If the context has already been created, return it.
```

```
+    if (_managedObjectContext != nil)
+        return _managedObjectContext;
+
+    // If it hasn't been created yet, do so now.
+    _managedObjectContext = [[NSManagedObjectContext alloc] init];
+
+    // The managed object context needs to have a persistent store
coordinator to work, so do that now.
+    // (Note that by calling self.persistentStoreCoordinator, it'll cause
one to be created if it didn't already.)
+    [_managedObjectContext
setPersistentStoreCoordinator:self.persistentStoreCoordinator];
+
+    return _managedObjectContext;
+}
+
+// Returns the app's persistent store coordinator.
+- (NSPersistentStoreCoordinator*) persistentStoreCoordinator {
+
+    // Return the coordinator if it's already been created.
+    if (_persistentStoreCoordinator != nil)
+        return _persistentStoreCoordinator;
+
+    // Create the persistent store coordinator. We need to give it the
managed object model in order to do so.
+    _persistentStoreCoordinator = [[NSPersistentStoreCoordinator alloc]
initWithManagedObjectModel:self.managedObjectModel];
+
+    // Work out the location of the database file, by getting the app's
document directory and adding
+    // 'Diary.sqlite' to the path.
+    NSURL* storeURL = [[self applicationDocumentsDirectory]
URLByAppendingPathComponent:@"Diary.sqlite"];
+
+    // Try to add the persistent store.
+    NSError* error = nil;
+
+    [_persistentStoreCoordinator
addPersistentStoreWithType:NSSQLiteStoreType configuration:nil URL:storeURL
options:nil error:&error];
+
+    if (error != nil) {
+        NSLog(@"Failed to open the store! Error: %@", error);
+        _persistentStoreCoordinator = nil;
+        return nil;
+    }
+
+    return _persistentStoreCoordinator;
+
+}
+
```

```
+// Creates a new note, and adds it to the database.
+- (DYNote *)createNote {
+
+    // Create the new note, and insert it into the managed object context.
+    DYNote* newNote = [NSEntityDescription
insertNewObjectForEntityForName:@"Note"
inManagedObjectContext:self.managedObjectContext];
+
+    // Try to save the database, which ensures that the note is stored.
+    NSError* error = nil;
+
+    [self.managedObjectContext save:&error];
+
+    if (error != nil) {
+        NSLog(@"Couldn't save the context: %@", error);
+        return nil;
+    }
+
+    // Return the new note.
+    return newNote;
+}
+
+// Deletes a note from the database.
+- (void)deleteNote:(DYNote *)note {
+
+    // Delete the note...
+    [self.managedObjectContext deleteObject:note];
+
+    // ..and save the database.
+    NSError* error = nil;
+
+    [self.managedObjectContext save:&error];
+
+    if (error != nil) {
+        NSLog(@"Couldn't save the context: %@", error);
+    }
+}
+
+// Creates and prepares a fetched results controller which provides info on
Note entities.
+- (NSFetchedResultsController *)createFetchedResultsController {
+
+    // Create a fetch request, which describes what kind of entity we're
looking for.
+    NSFetchRequest* fetchRequest = [NSFetchRequest
fetchRequestWithEntityName:@"Note"];
+
+    fetchRequest.fetchBatchSize = 20;
+
+    // Sort the results in order of most recently edited.
+    NSSortDescriptor* sortDescriptor = [NSSortDescriptor
```

```
sortDescriptorWithKey:@"modifiedDate" ascending:NO];
+
+    fetchRequest.sortDescriptors = @[sortDescriptor];
+
+    // Create the fetched results controller itself.
+    NSFetchedResultsController* newFetchedResultsController =
+        [[NSFetchedResultsController alloc]
initWithFetchRequest:fetchRequest // What to look for
+
managedObjectContext:self.managedObjectContext // Where to find it
+                                                sectionNameKeyPath:nil // How
to group them (nil = no sections)
+                                                         cacheName:nil]; //
Where to cache them (nil = no caching)
+
+    return newFetchedResultsController;
+}
+
+@end
```

1. Update DYNoteListViewController.m to use DYNoteStorage.

DYNoteListViewController.m

```
#import "DYNoteListViewController.h"
#import "DYNote.h"
#import "DYNoteViewController.h"
+#import "DYNoteStorage.h"

// This is a 'class extension', which lets you add methods, properties and
variables
// to a class without having to put them in the header file, which other
classes can see.
// Anything you put in the class extension can only be accessed by this
class.
-@interface DYNoteListViewController () {
-    NSMutableArray* _notes;
+
+// By adding <NSFetchedResultsControllerDelegate> after the parentheses,
+// we're telling the compiler that this object can work as a delegate for
+// an NSFetchedResultsController.
+@interface DYNoteListViewController () <NSFetchedResultsControllerDelegate>
{
}

+// The fetched results controller is the way we get info about the notes in
the database.
+@property (strong) NSFetchedResultsController* fetchedResultsController;
+
@end
```

```
@implementation DYNoteListViewController

- (void)viewDidLoad {

-    // Create the array that stores the notes
-    _notes = [NSMutableArray array];
-
  // Get the table view's "Edit" button, which will put the table into Edit
mode when tapped
  self.navigationItem.leftBarButtonItem = self.editButtonItem;
+
+    // Get the note storage system to give us a fetched results controller,
which we can use
+    // to get the notes themselves.
+    self.fetchedResultsController = [[DYNoteStorage sharedStorage]
createFetchedResultsController];
+
+    // Tell the fetched results controller to let us know when data
changes.
+    self.fetchedResultsController.delegate = self;
+
+    // Finally, tell the controller to start getting objects, and watching
for changes.
+    NSError* error = nil;
+    [self.fetchedResultsController performFetch:&error];
+
+    if (error != nil) {
+        NSLog(@"Problem fetching results! %@", error);
+    }
+
}

// Returns the number of sections (groups of cells) in the table
  - (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
-    // There is only one section in this table.
-    return 1;
+
+    // Ask the fetched results controller to tell us how many sections
there are.
+    return [[self.fetchedResultsController sections] count];
}


// Returns the number of rows (cells) in the given section.
  - (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:
(NSInteger)section {

-    // There's only ever one section, so we don't need to worry about
checking the value of 'section'.
-
```

```
-    // The number of rows is equal to the number of notes we have.
-    return [_notes count];
+    // Ask the fetched results controller to tell us about how many rows
are in the section.
+    id <NSFetchedResultsSectionInfo> sectionInfo =
[self.fetchedResultsController sections][section];
+    return [sectionInfo numberOfObjects];
}

// Returns a table view cell for use, which shows the data we want to
display.
@@ -52,17 +72,24 @@ - (UITableViewCell*) tableView:(UITableView *)tableView
cellForRowAtIndexPath:(N
  // Get a cell to use.
  UITableViewCell* cell = [tableView
dequeueReusableCellWithIdentifier:@"NoteCell"];

-    // Work out which note should be shown in this cell.
-    DYNote* note = _notes[indexPath.row];
-
-    // Give the note's text to the cell.
-    cell.textLabel.text = note.text;
+    [self configureCell:cell atIndexPath:indexPath];

  // Return the cell to the table view, which will then show it.
  return cell;

}

+// Called by either tableView:cellForRowAtIndexPath: or by
+// controller:didChangeObject:atIndexPath:forChangeType:newIndexPath:.
+- (void)configureCell:(UITableViewCell *)cell atIndexPath:(NSIndexPath
*)indexPath {
+
+    // Work out which note should be shown in this cell.
+    DYNote* note = [self.fetchedResultsController
objectAtIndexPath:indexPath];
+
+    // Give the note's text to the cell.
+    cell.textLabel.text = note.text;
+}
+
// Called when the view controller is about to move to another view
controller.
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {

@@ -79,7 +106,7 @@ - (void)prepareForSegue:(UIStoryboardSegue *)segue
sender:(id)sender {
      NSIndexPath* indexPath = [self.tableView indexPathForCell:cell];

      // Use that to get the appropriate note.
```

```
-        DYNote* note = _notes[indexPath.row];
+        DYNote* note = [self.fetchedResultsController
objectAtIndexPath:indexPath];

      noteViewController.note = note;
  }
@@ -99,36 +126,64 @@ - (void)tableView:(UITableView *)tableView
commitEditingStyle:(UITableViewCellEd
  if (editingStyle == UITableViewCellEditingStyleDelete) {

      // Find the note that we're talking about
-        DYNote* note = _notes[indexPath.row];
-
-        // Remove it from the list of notes
-        [_notes removeObject:note];
+        DYNote* note = [self.fetchedResultsController
objectAtIndexPath:indexPath];

-        // Finally, remove the cell.
-        [tableView deleteRowsAtIndexPaths:@[indexPath]
withRowAnimation:UITableViewRowAnimationLeft];
+        if (note != nil)
+            [[DYNoteStorage sharedStorage] deleteNote:note];

  }
}

+// Called when the fetched results controller is about to start reporting
changes.
+- (void)controllerWillChangeContent:(NSFetchedResultsController
*)controller {


+    // Tell the table view to prepare to group together a bunch of
animations.
+    [self.tableView beginUpdates];
+}
+
+// Called when the fetched results controller has finished reporting
changes.
+- (void)controllerDidChangeContent:(NSFetchedResultsController *)controller
{

-    // First, create an index path that describes the position of the cell
+    // Tell the table view that we're done doing updates, so it can perform
the animations
+    // that have been queued up.
+    [self.tableView endUpdates];
+}
+
+// Called when the fetched results controller has a change to report.
```

```
+- (void)controller:(NSFetchedResultsController *)controller
didChangeObject:(id)anObject atIndexPath:(NSIndexPath *)indexPath
forChangeType:(NSFetchedResultsChangeType)type newIndexPath:(NSIndexPath
*)newIndexPath {
+
+    // Different changes need different animations:
+    switch (type) {
+        case NSFetchedResultsChangeInsert:
+            // A new object was inserted, so tell the table view to animate
a new cell in.
+            [self.tableView insertRowsAtIndexPaths:@[newIndexPath]
withRowAnimation:UITableViewRowAnimationTop];
+            break;
+
+        case NSFetchedResultsChangeDelete:
+            // An object was deleted, so tell the table view to delete the
appropriate row.
+            [self.tableView deleteRowsAtIndexPaths:@[indexPath]
withRowAnimation:UITableViewRowAnimationLeft];
+            break;
+
+        case NSFetchedResultsChangeUpdate:
+            // An object was changed, so update its contents by calling
configureCell:atIndexPath.
+            [self configureCell:[self.tableView
cellForRowAtIndexPath:indexPath] atIndexPath:indexPath];
+            break;
+
+        case NSFetchedResultsChangeMove:
+            // An object was move, so delete the row that it used to be in,
and insert one where it's now located.
+            [self.tableView deleteRowsAtIndexPaths:@[indexPath]
withRowAnimation:UITableViewRowAnimationFade];
+            [self.tableView insertRowsAtIndexPaths:@[newIndexPath]
withRowAnimation:UITableViewRowAnimationFade];
+            break;
+    }

-    NSIndexPath* indexPath = [NSIndexPath indexPathForRow:0 inSection:0];
+}
+



+// Called when the user taps the Add button.
+- (IBAction)addNote:(id)sender {

-    // Now add the row
-    [self.tableView insertRowsAtIndexPaths:@[indexPath]
withRowAnimation:UITableViewRowAnimationTop];
+    // Tell the DYNoteStorage to create a new note.
```

```
+    DYNote* newNote = [[DYNoteStorage sharedStorage] createNote];
+    newNote.text = @"New Note";

}

@end
```

1. Update `viewWillDisappear` in DYNoteViewController.m to save when exiting.

DYNoteViewController.m

```
- (void)viewWillDisappear:(BOOL)animated {
  // Store the text that's in the note text view into the note itself.
  self.note.text = self.noteTextView.text;
+
+    // Save the note.
+    NSError* error = nil;
+    [self.note.managedObjectContext save:&error];
+    if (error != nil) {
+        NSLog(@"Failed to save the note! %@", error);
+    }
}
```

08-Searching

We'll now make the table view searchable. (All search implementation is listed under '18-LaunchImages' in the repo)

1. Open the storyboard.
2. Drag in a Search Bar and Search Display Controller. When you start dragging, you'll see a search bar; drag it so that it's above the table view.

Next, we'll make the view controller able to work with the search controller.

1. Open DYNoteListViewController.m
2. Make DYNoteListViewController conform to `UISearchBarDelegate` and `UISearchDisplayDelegate`.

DYNoteListViewController.m

```
+@interface DYNoteListViewController () <NSFetchedResultsControllerDelegate,
UISearchBarDelegate, UISearchDisplayDelegate> {
}
```

1. Add a new strong `NSFetchedResultsController` property called `searchFetchedResultsController`.

DYNoteListViewController.m

```
+@property (strong) NSFetchedResultsController*
searchFetchedResultsController;
```

Next, we'll update the table view and fetched results controller delegate methods to work with the search results table.

1. Update the `numberOfSectionsInTableView:`, `tableView:numberOfRowsInSection:`, `configureCell:atIndexPath:`, `prepareForSegue:sender:`, `controllerWillChangeContent:`, `controllerDidChangeContent:`, and `controller:didChangeObject:atIndexPath:forChangeType:newIndexPath:` methods.

DYNoteListViewController.m

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {

  // Ask the fetched results controller to tell us how many sections there
are.
-    return [[self.fetchedResultsController sections] count];
+    if (tableView == self.searchDisplayController.searchResultsTableView) {
+        return [[self.searchFetchedResultsController sections] count];
+    } else {
+        return [[self.fetchedResultsController sections] count];
+    }
}


// Returns the number of rows (cells) in the given section.
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:
(NSInteger)section {

  // Ask the fetched results controller to tell us about how many rows are
in the section.
-    id <NSFetchedResultsSectionInfo> sectionInfo =
[self.fetchedResultsController sections][section];
-    return [sectionInfo numberOfObjects];
+    if (tableView == self.searchDisplayController.searchResultsTableView) {
+        id <NSFetchedResultsSectionInfo> sectionInfo =
[self.searchFetchedResultsController sections][section];
+        return [sectionInfo numberOfObjects];
+    } else {
+        id <NSFetchedResultsSectionInfo> sectionInfo =
[self.fetchedResultsController sections][section];
+        return [sectionInfo numberOfObjects];
+    }
```

```
}

// Returns a table view cell for use, which shows the data we want to
display.
- (UITableViewCell*) tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath {

  // Get a cell to use.
-    UITableViewCell* cell = [tableView
dequeueReusableCellWithIdentifier:@"NoteCell"];
+    UITableViewCell* cell = [self.tableView
dequeueReusableCellWithIdentifier:@"NoteCell"];

- [self configureCell:cell atIndexPath:indexPath];
+ [self configureCell:cell atIndexPath:indexPath inTableView:tableView];

  // Return the cell to the table view, which will then show it.
  return cell;

}



// Called by either tableView:cellForRowAtIndexPath: or by
// controller:didChangeObject:atIndexPath:forChangeType:newIndexPath:.
- - (void)configureCell:(UITableViewCell *)cell atIndexPath:(NSIndexPath
*)indexPath {
+ - (void)configureCell:(UITableViewCell *)cell atIndexPath:(NSIndexPath
*)indexPath inTableView:(UITableView*) tableView {


  // Work out which note should be shown in this cell.
-    DYNote* note = [self.fetchedResultsController
objectAtIndexPath:indexPath];
+
+    DYNote* note = nil;
+    if (tableView == self.searchDisplayController.searchResultsTableView) {
+        note = [self.searchFetchedResultsController
objectAtIndexPath:indexPath];
+    } else {
+        note = [self.fetchedResultsController objectAtIndexPath:indexPath];
+    }

  // Give the note's text to the cell.
  cell.textLabel.text = note.text;
}

// Called when the view controller is about to move to another view
controller.
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
```

```
  // If this is the "showNote" segue, then we're about to segue to the Note
View Controller because the user tapped on a table view cell.
  if ([segue.identifier isEqualToString:@"showNote"]) {

      // Get the note view controller we're about to move to.
      DYNoteViewController* noteViewController =
segue.destinationViewController;

      // Get the cell that was tapped on.
      UITableViewCell* cell = sender;

      // Work out which row this cell was.
 +    NSIndexPath* indexPath = nil;
 +
 +    DYNote* note = nil;
 +
 +    if (self.searchDisplayController.active) {
 +        indexPath = [self.searchDisplayController.searchResultsTableView
indexPathForCell:cell];
 +        note = [self.searchFetchedResultsController
objectAtIndexPath:indexPath];
 +    } else {
 +        indexPath = [self.tableView indexPathForCell:cell];
 +        note = [self.fetchedResultsController
objectAtIndexPath:indexPath];
 +    }

      noteViewController.note = note;
  }
}

// Called when the user has tapped the Delete button.
- (void)tableView:(UITableView *)tableView commitEditingStyle:
(UITableViewCellEditingStyle)editingStyle forRowAtIndexPath:(NSIndexPath
*)indexPath {

  // We only want to take action if the edit that was made is a deletion.
  if (editingStyle == UITableViewCellEditingStyleDelete) {

      // Find the note that we're talking about
      DYNote* note = [self.fetchedResultsController
objectAtIndexPath:indexPath];

      if (note != nil)
          [[DYNoteStorage sharedStorage] deleteNote:note];

  }
}

// Called when the fetched results controller is about to start reporting
changes.
```

```
- (void)controllerWillChangeContent:(NSFetchedResultsController *)controller
{

  // Tell the table view to prepare to group together a bunch of animations.
-    [self.tableView beginUpdates];
+
+    if (controller == self.searchFetchedResultsController) {
+        [self.searchDisplayController.searchResultsTableView beginUpdates];
+    } else {
+        [self.tableView beginUpdates];
+    }
+
}

// Called when the fetched results controller has finished reporting
changes.
- (void)controllerDidChangeContent:(NSFetchedResultsController *)controller
{

  // Tell the table view that we're done doing updates, so it can perform
the animations
  // that have been queued up.
-    [self.tableView endUpdates];
+
+    if (controller == self.searchFetchedResultsController) {
+        [self.searchDisplayController.searchResultsTableView endUpdates];
+    } else {
+        [self.tableView endUpdates];
+    }
+
+
}

// Called when the fetched results controller has a change to report.
- (void)controller:(NSFetchedResultsController *)controller didChangeObject:
(id)anObject atIndexPath:(NSIndexPath *)indexPath forChangeType:
(NSFetchedResultsChangeType)type newIndexPath:(NSIndexPath *)newIndexPath {

+    UITableView* tableView = nil;
+    if (controller == self.searchFetchedResultsController) {
+        tableView = self.searchDisplayController.searchResultsTableView;
+    } else {
+        tableView = self.tableView;
+    }
+
  // Different changes need different animations:
  switch (type) {
      case NSFetchedResultsChangeInsert:
          // A new object was inserted, so tell the table view to animate a
new cell in.
-            [self.tableView insertRowsAtIndexPaths:@[newIndexPath]
```

```
withRowAnimation:UITableViewRowAnimationTop];
+            [tableView insertRowsAtIndexPaths:@[newIndexPath]
withRowAnimation:UITableViewRowAnimationTop];
          break;

      case NSFetchedResultsChangeDelete:
          // An object was deleted, so tell the table view to delete the
appropriate row.
-            [self.tableView deleteRowsAtIndexPaths:@[indexPath]
withRowAnimation:UITableViewRowAnimationLeft];
+            [tableView deleteRowsAtIndexPaths:@[indexPath]
withRowAnimation:UITableViewRowAnimationLeft];
          break;

      case NSFetchedResultsChangeUpdate:
          // An object was changed, so update its contents by calling
configureCell:atIndexPath.
-            [self configureCell:[self.tableView
cellForRowAtIndexPath:indexPath] atIndexPath:indexPath];
+            [self configureCell:[tableView cellForRowAtIndexPath:indexPath]
atIndexPath:indexPath inTableView:tableView];
          break;

      case NSFetchedResultsChangeMove:
          // An object was move, so delete the row that it used to be in,
and insert one where it's now located.
-            [self.tableView deleteRowsAtIndexPaths:@[indexPath]
withRowAnimation:UITableViewRowAnimationFade];
-            [self.tableView insertRowsAtIndexPaths:@[newIndexPath]
withRowAnimation:UITableViewRowAnimationFade];
+            [tableView deleteRowsAtIndexPaths:@[indexPath]
withRowAnimation:UITableViewRowAnimationFade];
+            [tableView insertRowsAtIndexPaths:@[newIndexPath]
withRowAnimation:UITableViewRowAnimationFade];
          break;
  }

}
```

1. Implement the `searchDisplayControllerWillBeginSearch:`, `searchDisplayControllerWillEndSearch:`, `searchBar:textDidChange:` and `updateSearchQuery:` methods.

DYNoteListViewController.m

```
+// Called when search begins (when the user taps in the search box)
+- (void)searchDisplayControllerWillBeginSearch:(UISearchDisplayController
*)controller {
+    self.searchFetchedResultsController = [[DYNoteStorage sharedStorage]
```

```
createFetchedResultsController];
+    self.searchFetchedResultsController.delegate = self;
+
+    [self updateSearchQuery:self.searchDisplayController.searchBar.text];
+}
+
+// Called when search ends (when the user taps the 'Cancel' button)
+- (void)searchDisplayControllerWillEndSearch:(UISearchDisplayController
*)controller {
+    self.searchFetchedResultsController = nil;
+}
+
+// Called when the search text changes.
+- (void)searchBar:(UISearchBar *)searchBar textDidChange:(NSString
*)searchText {
+    [self updateSearchQuery:searchText];
+}
+
+// Called by searchBar:textDidChange: and
searchDisplayControllerWillBeginSearch: to update
+// the search request.
+- (void) updateSearchQuery:(NSString*)searchQuery {
+
+    // Get the existing fetch request.
+    NSFetchRequest* fetchRequest =
self.searchFetchedResultsController.fetchRequest;
+
+    if ([searchQuery length] > 0) {
+
+        // If the search text is not empty, create a predicate (aka a
search query) that
+        // does a case-insensitive search for the searchQuery text in the
'text' attribute.
+        NSPredicate* predicate = [NSPredicate predicateWithFormat:@"text
contains[c] %@", searchQuery];
+
+        // Give the new predicate to the fetch request.
+        fetchRequest.predicate = predicate;
+    }
+
+    // Now that the fetch request is updated, make the fetched results
controller
+    // use the new fetch request. (This will cause the search results to
update.)
+    NSError* error = nil;
+
+    [self.searchFetchedResultsController performFetch:&error];
+
+    if (error != nil) {
+        NSLog(@"Error fetching search results: %@", error);
+    }
```

```
+
+}
+
@end
```

09-UserDefaults

1. Open the storyboard. Select the Note View Controller. Change its Storyboard ID to 'DYNoteViewController'.
2. Update DYNoteStorage.h and .m to add the `noteWithURL:` method.

DYNoteStorage.m (Note: this is already done by 07 in the repo)

```
+// Returns a note, given its URL in the database.
+- (DYNote *)noteWithURL:(NSURL *)url {
+    // First, we use the URL to ask the persistent store coordinator for an
object ID.
+    NSManagedObjectID* objectID = [self.persistentStoreCoordinator
managedObjectIDForURIRepresentation:url];
+
+    // Next, we get the object with that ID.
+    DYNote* note = (DYNote*)[self.managedObjectContext
objectWithID:objectID];
+
+    return note;
+
+}
+
+
```

1. Update DYNoteListViewController.m's `viewDidLoad` method.

DYNoteListViewController.m

```
-(void)viewDidLoad{
...
+    // Get the URL for the note that we were last editing from user
defaults, if one is set.
+    NSURL* currentNoteURL = [[NSUserDefaults standardUserDefaults]
URLForKey:@"current_note"];
+
+    if (currentNoteURL) {
+
+        // If one is set, we need to use the URL to get the object out of
the database.
+        DYNote* note = [[DYNoteStorage sharedStorage]
noteWithURL:currentNoteURL];
+
+        // If the note exists, then create the note view controller, and
give it the note object
+        if (note != nil) {
+            DYNoteViewController* noteViewController = [self.storyboard
instantiateViewControllerWithIdentifier:@"NoteViewController"];
+            noteViewController.note = note;
+
+            // Next, push the view controller without an animation. This
will make the app start with
+            // the note view controller visible.
+            [self.navigationController
pushViewController:noteViewController animated:NO];
+        }
+
+    }
+
}
```

1. Update DYNoteViewController.m's `viewDidLoad` and `viewWillDisappear` method.

DYNoteViewController.m

```
- (void)viewDidLoad
{
  [super viewDidLoad];
// Do any additional setup after loading the view, typically from a nib.
+
+    // When the screen loads, get the URL of the object that we're showing.
+    // Store it into the user defaults, so that if the app exits while
we're editing this
+    // note, the app will return to it.
+    NSURL* noteURL = [self.note.objectID URIRepresentation];
+    [[NSUserDefaults standardUserDefaults] setURL:noteURL
forKey:@"current_note"];
}

- (void)viewWillDisappear:(BOOL)animated {
  // Store the text that's in the note text view into the note itself.
  self.note.text = self.noteTextView.text;
+    [[NSUserDefaults standardUserDefaults] setURL:nil
forKey:@"current_note"];
}
```

10-Location

1. Add the CoreLocation framework to the project.

Next, we'll set up the UI.

1. Open the storyboard.
2. Add a toolbar to the Note View Controller.
3. Select the bar button item that comes with the toolbar. Rename it to 'Location'.
4. Add a new view controller.
5. Set its title to 'Location'.
6. Connect the Location button the new view controller. Give it a Push segue. Name the segue 'showLocation'.
7. Add a label and an activity indicator.
    - Make the label fill the width of the screen, and center the text.
    - Put the activity indicator beneath the label.
    - Turn on "hides when stopped".
8. Create a new view controller. Call it DYLocationViewController.
9. Set the class of the newly added screen to DYLocationViewController.

Next, we'll link up the UI to code.

1. Add outlets to DYLocationViewController's class extension:
    - Connect the label to an outlet called locationLabel.
    - Connect the activity indicator to an outlet called locationActivity.

Next, make the code know about DYNotes.

1. Open DYLocationViewController.h.
2. Import DYNote.h, and add a property: a DYNote called 'note'.

DYLocationViewController.h

```
+#import <UIKit/UIKit.h>
+#import "DYNote.h"
+
+@interface DYLocationViewController : UIViewController
+
+@property (strong) DYNote* note;
+
+@end
```

We now need to make notes store location info. We'll store locations as archived CLLocation objects.

1. Open Diary.xcdatamodeld.
2. Add a new attribute to the Note entity: a Transformable called 'location'.

Transformable attributes convert between objects and raw data. Any object that conforms to `NSCoding`, which is most data objects, can be stored.

1. Open DYNote.h. `@import` CoreLocation, and add a CLLocation property called 'location'.
2. Open DYNote.m, and mark the property as `@dynamic`.

Before you next re-launch the app, you'll need to erase the app and reinstall it, to prevent errors.

Next, we'll make the DYLocationViewController be given the note when the Location button is tapped.

1. Import DYLocationViewController.h in DYNoteViewController.
2. Add the `prepareForSegue:` method to DYNoteViewController.

DYNoteViewController.m

```
+#import "DYLocationViewController.h"

+// Called when a segue is about to happen.
+- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
+
+    if ([segue.identifier isEqualToString:@"showLocation"]) {
+
+        // If this is the showLocation segue, we're moving to a
DYLocationViewController.
+        DYLocationViewController* locationViewController =
segue.destinationViewController;
+
+        // Give it the note.
+        locationViewController.note = self.note;
+    }
+}
+
```

We'll now make DYLocationViewController get the location, when it appears.

1. Open DYLocationViewController.m.
2. Make DYLocationViewController conform to CLLocationManagerDelegate.
3. Add a new property to DYLocationViewController's class extension: a CLLocationManager called locationManager.
4. Update `viewDidLoad` and implement `locationManager: didUpdateLocations:` and `locationManager: didFailWithError:` in DYLocationManager.m.

DYLocationViewController.m (Complete file)

```
+#import "DYLocationViewController.h"
+
+@interface DYLocationViewController () <CLLocationManagerDelegate>
+@property (weak, nonatomic) IBOutlet UILabel *locationLabel;
+@property (weak, nonatomic) IBOutlet UIActivityIndicatorView
*locationActivity;
+
+@property (strong) CLLocationManager* locationManager;
+
+@end
+
+@implementation DYLocationViewController
+
+- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle
*)nibBundleOrNil
+{
+    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
+    if (self) {
+        // Custom initialization
```

```
+    }
+    return self;
+}
+
+- (void)viewDidLoad
+{
+    [super viewDidLoad];
+
+    if (self.note.location != nil) {
+        // If the note already has a location, show the location in the
text field.
+
+        self.locationLabel.text = [self.note.location description];
+    } else {
+
+        // Otherwise, tell the user that we're looking for a location.
+        self.locationLabel.text = @"Looking for location...";
+        [self.locationActivity startAnimating];
+    }
+
+}
+
+ // Called when the view finishes appearing.
+ - (void)viewDidAppear:(BOOL)animated {
+
+     // Create the location manager, and tell it to start looking for the
location.
+     // We do this in viewDidAppear because if no location hardware is
available
+     // (e.g. we're on the simulator, or permission is denied) then the
location manager
+     // will fail immediately (and we don't want that happening during the
slide-in animation.)
+     self.locationManager = [[CLLocationManager alloc] init];
+     self.locationManager.delegate = self;
+     [self.locationManager startUpdatingLocation];
+ }
+
+// Called when the location manager works out where we are.
+- (void)locationManager:(CLLocationManager *)manager didUpdateLocations:
(NSArray *)locations {
+
+    // Get the first location that was found.
+    CLLocation* location = [locations firstObject];
+
+    // If the note doesn't already have a location..
+    if (self.note.location == nil) {
+
+        // Store it!
+        self.note.location = location;
+
```

```
+        // Show the location to the user
+        self.locationLabel.text = [location description];
+
+        // We're now done looking, so stop the activity indicator and stop
the location manager.
+        [self.locationActivity stopAnimating];
+        [self.locationManager stopUpdatingLocation];
+    }
+
+}
+
+// Called when the location manager fails to work out the user's location.
+// This can be for many reasons - the GPS can't find any satellites, the
user declined to give the app permission, etc.
+- (void)locationManager:(CLLocationManager *)manager didFailWithError:
(NSError *)error {
+
+    // Just go back to the previous view controller.
+    [self.navigationController popViewControllerAnimated:YES];
+}
+
+- (void)didReceiveMemoryWarning
+{
+    [super didReceiveMemoryWarning];
+    // Dispose of any resources that can be recreated.
+}
+


+
+@end
```

Finally, we'll make the user able to delete the location.

1. Open the storyboard, and go to the Location View Controller.
2. Drag in a bar button item into the right side of the navigation bar.
3. Set its identifier to Trash.
4. Connect it to a new method called removeNote:.
5. Implement the method.

DYLocationViewController.m

```
+// Called when the Trash button is tapped.
+- (IBAction)removeNote:(id)sender {
+
+    // Remove the location from the note
+    self.note.location = nil;
+
+    // Return to the previous view controller.
+    [self.navigationController popViewControllerAnimated:YES];
+}
```

11-Maps

1. Make the project use the MapKit framework.
2. Open the Storyboard.
    - Delete the label.
    - Add a Map View to the Location View Controller. Make it fill the screen. Turn on Shows User Location.
    - Change the activity indicator to a Large White one, and keep it above the Map View.
    - Make the map use the view controller as its delegate.

We'll now connect it to the code.

1. Open DYLocationViewController.m.
2. `@import` MapKit.
3. Connect the Map View to an outlet called mapView.

DYLocationViewController.m

```
#import "DYLocationViewController.h"
+@import MapKit;

-@interface DYLocationViewController () <CLLocationManagerDelegate>
+@interface DYLocationViewController () <CLLocationManagerDelegate,
MKMapViewDelegate>
@property (weak, nonatomic) IBOutlet UILabel *locationLabel;
@property (weak, nonatomic) IBOutlet UIActivityIndicatorView
*locationActivity;

@property (strong) CLLocationManager* locationManager;

+@property (weak, nonatomic) IBOutlet MKMapView *mapView;
+
@end
```

1. Implement the `updateAnnotation` method, and update `locationManager:didUpdateLocations:` and `viewDidLoad`.

DYLocationViewController.m

```
- (void)viewDidLoad
{
  [super viewDidLoad];

  if (self.note.location != nil) {
      // If the note already has a location, show the location in the text
field.

      self.locationLabel.text = [self.note.location description];
+
+        [self updateAnnotation];
+
  } else {

      // Otherwise, create and set up the location manager.

      self.locationManager = [[CLLocationManager alloc] init];
      self.locationManager.delegate = self;
      [self.locationManager startUpdatingLocation];

      // Tell the user that we're looking for a location.
      self.locationLabel.text = @"Looking for location...";
      [self.locationActivity startAnimating];
  }

}
...
- (void)locationManager:(CLLocationManager *)manager didUpdateLocations:
(NSArray *)locations {

  // Get the first location that was found.
  CLLocation* location = [locations firstObject];

  // If the note doesn't already have a location..
  if (self.note.location == nil) {

      // Store it!
      self.note.location = location;

      // Show the location to the user
      self.locationLabel.text = [location description];

      // We're now done looking, so stop the activity indicator and stop the
location manager.
      [self.locationActivity stopAnimating];
```

```
      [self.locationManager stopUpdatingLocation];
+
+        [self updateAnnotation];
  }
...
+// Called by viewDidLoad: and locationManager:didUpdateLocations: to update
the annotation.
+
+- (void) updateAnnotation {
+
+    // Remove any existing annotations on the map.
+    [self.mapView removeAnnotations:self.mapView.annotations];
+
+    // Create a new annotation
+    MKPointAnnotation* point = [[MKPointAnnotation alloc] init];
+    point.coordinate = self.note.location.coordinate;
+    point.title = @"Location";
+
+    // Add it to the map
+    [self.mapView addAnnotation:point];
+}
```

12-Audio

1. Open the storyboard, and go to the Note View Controller.
2. Add a Flexible Space to the toolbar at the bottom of the screen.
3. Add a new Bar Button Item to the toolbar. Change its label to 'Audio'.
4. Add a new view controller. Connect the Audio button to the view controller with a Push segue. Name the segue 'showAudio'.
5. Set the title of the view controller to Audio.

Next, we'll create a new view controller to handle the audio-related tasks of the new screen.

1. Create a new UIViewController subclass, called `DYAudioViewController`.
2. Set the class of the 'Audio' view controller in the Storyboard.

The buttons in this view controller will be this: a "play/record/stop" button (which changes state based on whether or not we're recording), and a delete button. The delete button will be in the corner, like in the location view; the play/record/stop button will be in the view.

1. Open the Images.xcassets bundle.
2. Click +, and add a new Image Set.
3. Name it RecordButton, and drop the recording icons onto it.
4. Do the same for the Play and Stop buttons. (You can also drag pairs of images directly into the list of image sets to quickly create them.)
5. Open the Storyboard. Go to the Audio screen.
6. Add a button. Change its type to Custom. Resize it so that it's about 3 times as big,

to increase its touch size.
7. Delete the Title text, and change the Image to RecordButton.
8. Drag a Bar Button Item into the right hand side of the navigation bar. Change its identifier to Trash.

Next, we'll connect up the code.

1. Open DYAudioViewController.m in the assistant.
2. Connect the record button to BOTH an outlet, called controlButton, and an action, called controlButtonTapped.
3. Connect the Trash button to an action called removeAudio:.

Next, we'll make the database be able to store the audio note data.

1. Open Diary.xcdatamodeld, and add a new attribute called 'audioNote'. Make it a binary data.
2. Open DYNote.h, and add a property for the new attribute.

DYNote.h

```
@property (nonatomic, strong) CLLocation* location;

+/// The audio note attached, stored as an NSData.
+@property (nonatomic, strong) NSData* audioNote;
+
```

1. Mark the new property as `@dynamic` in DYNote.m.

DYNote.m @dynamic modifiedDate; @dynamic location; +@dynamic audioNote;

Next, we'll add the code.

1. `@import AVFoundation` in DYAudioViewController.m.
2. Open DYAudioViewController.h. `#import "DYNote.h"`.
3. Add a DYNote* property to the class.
4. Add two new properties to DYAudioViewController.m:
    - `@property (strong) AVAudioPlayer* audioPlayer;` and
    - `@property (strong) AVAudioPlayer* audioRecorder;`

Next, we'll make the note view controller pass its note to the audio view controller.

1. Open DYNoteViewController.m
2. `#import DYAudioViewController.h`.
3. Update `prepareForSegue` to make it pass the note to the audio controller.

DYNoteViewController.m

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {

  if ([segue.identifier isEqualToString:@"showLocation"]) {

      // If this is the showLocation segue, we're moving to a
DYLocationViewController.
      DYLocationViewController* locationViewController =
segue.destinationViewController;

      // Give it the note.
      locationViewController.note = self.note;
  }
+
+    if ([segue.identifier isEqualToString:@"showAudio"]) {
+
+        // If this is the showAudio segue, we're moving to a
DYAudioViewController.
+        DYAudioViewController* audioViewController =
segue.destinationViewController;
+
+        // Give it the note.
+        audioViewController.note = self.note;
+    }
}
```

Implement DYAudioNoteViewController.

1. Add a `BOOL` instance variable to DYAudioNoteViewController's class extension, called `recordingAllowed`.
2. Implement the `controlButtonTapped:`, `startRecording`, `stopRecording`, `startPlaying`, `stopPlaying`, `removeAudio:`, `updateControlButton` and `viewWillDisappear` methods. Update the `viewDidLoad` method.

Next, we'll make the play/pause/stop button update when the end of the recording is reached.

1. Make DYAudioNoteViewController conform to the AVAudioPlayerDelegate protocol.
2. Add the `audioPlayerDidFinishPlaying: successfully:` method.
3. Update the `startPlaying` method to set the delegate.

DYAudioNoteViewController.h

```
+#import <UIKit/UIKit.h>
+#import "DYNote.h"
+
+@interface DYAudioViewController : UIViewController
+
+@property (strong) DYNote* note;
+
+@end
```

DYAudioNoteViewController.m

```
+#import "DYAudioViewController.h"
+
+@import AVFoundation;
+
+@interface DYAudioViewController () <AVAudioPlayerDelegate> {
+    BOOL recordingAllowed;
+}
+
+@property (weak, nonatomic) IBOutlet UIButton *controlButton;
+
+@property (strong) AVAudioPlayer* audioPlayer;
+@property (strong) AVAudioRecorder* audioRecorder;
+
+@end
+
+@implementation DYAudioViewController
+- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle
*)nibBundleOrNil
+{
+    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
+    if (self) {
+        // Custom initialization
+    }
+    return self;
+}
+
+- (void)viewDidLoad
+{
+    [super viewDidLoad];
+
+    // When the view loads, we want to indicate to the system that we will
be both playing and recording audio.
+    // This means that we need to have the recording system ready, by
setting the audio category to 'play and record'.
+    // By default, when we set this category, any playback will go through
the receiver (the small speaker intended
+    // to be held to the ear), if no other route is available.
+    // Because this is a note-taking app which will be held in the user's
```

```
hand, we want it to go out the main speaker,
+    // so we set that option.
+    NSError* error = nil;
+    [[AVAudioSession sharedInstance]
setCategory:AVAudioSessionCategoryPlayAndRecord
withOptions:AVAudioSessionCategoryOptionDefaultToSpeaker error:&error];
+
+    if (error != nil) {
+        NSLog(@"Error setting audio session category to record: %@",
error);
+    }
+
+    // Next, we need to ensure that we've got permission to access the
microphone.
+    // The first time this method is called after the app is installed, a
dialog box will appear asking for permission.
+    // When the user either accepts or denies access to the microphone, the
system remembers the decision and calls
+    // the block.
+    // When the method is called subsequent times, the system skips the
dialog and called the block immediately.
+
+    [[AVAudioSession sharedInstance] requestRecordPermission:^(BOOL
granted) {
+        // Do we have permission to record audio?
+
+        if (granted) {
+            // Awesome.
+            recordingAllowed = YES;
+        } else {
+            // Dang.
+            recordingAllowed = NO;
+        }
+    }];
+
+    // Finally, update the control button to indicate what the user can do.
+    [self updateControlButton];
+}
+
+- (void)didReceiveMemoryWarning
+{
+    [super didReceiveMemoryWarning];
+    // Dispose of any resources that can be recreated.
+}
+
+// Called when the user taps the main control button.
+- (IBAction)controlButtonTapped:(id)sender {
+
+    // The action that should be performed depends on what the app is
currently doing.
+
```

```
+    if (self.audioRecorder.isRecording) {
+        // If the app is recording audio, it should stop.
+        [self stopRecording];
+
+    } else if (self.audioPlayer.isPlaying) {
+        // If the app is playing audio, it should also stop.
+        [self stopPlaying];
+
+    } else if (self.note.audioNote == nil) {
+        // If the note currently has no recording, start recording.
+        [self startRecording];
+
+    } else if (self.note.audioNote) {
+        // If the note has a recording, play it.
+        [self startPlaying];
+
+    }
+
+    // Finally, update the control button to make it reflect what the user
can do now.
+    [self updateControlButton];
+}
+
+// Called by controlButtonTapped: to begin audio recording.
+- (void) startRecording {
+
+    // If we're already recording, stop it.
+    [self stopRecording];
+
+    // Remove any existing audio data from the DYNote.
+    self.note.audioNote = nil;
+
+    // Create an NSURL that points at a location where the recording can
temporarily be stored.
+    NSString* temporaryDirectory = NSTemporaryDirectory();
+    NSURL* temporaryFileURL = [[NSURL fileURLWithPath:temporaryDirectory]
URLByAppendingPathComponent:@"Recording.wav"];
+
+    // Create the recorder.
+    NSError* error = nil;
+    self.audioRecorder = [[AVAudioRecorder alloc]
initWithURL:temporaryFileURL settings:nil error:&error];
+
+    if (error != nil) {
+        NSLog(@"Error starting recording! %@", error);
+    }
+
+    // Start recording.
+    [self.audioRecorder record];
+
+}
```

```
+
+// Called by controlButtonTapped: to stop recording an audio note.
+- (void) stopRecording {
+
+    // If no audio recorder exists, bail out now to prevent erasing any
existing audio note.
+    if (self.audioRecorder == nil)
+        return;
+
+    // Stop the recorder.
+    [self.audioRecorder stop];
+
+    // The audio recording is now stored in a disk on file; load that file
into memory.
+    NSData* audioData = [NSData
dataWithContentsOfURL:self.audioRecorder.url];
+
+    // Store the loaded data into the DYNote.
+    self.note.audioNote = audioData;
+
+    // Save the changes.
+    NSError* error = nil;
+    [self.note.managedObjectContext save:&error];
+
+    if (error != nil) {
+        NSLog(@"Failed to save the note: %@", error);
+    }
+
+}
+
+// Called by controlButtonTapped: to begin playback of the audio note.
+- (void) startPlaying {
+
+    // If, for some reason, we're already playing, stop playback.
+    [self stopPlaying];
+
+    // Create a new audio player using the data stored in the note.
+    NSError* error = nil;
+
+    self.audioPlayer = [[AVAudioPlayer alloc]
initWithData:self.note.audioNote error:&error];
+
+    if (error != nil) {
+        NSLog(@"Error loading note! %@", error);
+    }
+
+    // Tell the audio player to contact this object when it changes state.
+    self.audioPlayer.delegate = self;
+
+    // Start playback.
+    [self.audioPlayer play];
```

```
+}
+
+// Called by several other methods to stop audio playback.
+- (void) stopPlaying {
+
+    // No need to check to see if the audioPlayer is nil, since calling
'stop' will do nothing if it's nil
+    [self.audioPlayer stop];
+}
+
+// Called when the user taps the 'delete' button.
+- (IBAction)removeAudio:(id)sender {
+
+    // We may be recording or playing audio. Stop it, in either case.
+    [self.audioRecorder stop];
+    [self.audioPlayer stop];
+
+    // Remove the audio data from the note.
+    self.note.audioNote = nil;
+
+    // Save the change.
+    NSError* error = nil;
+    [self.note.managedObjectContext save:&error];
+
+    if (error != nil) {
+        NSLog(@"Failed to save the note: %@", error);
+    }
+
+    // Update the button to reflect what the user can now do.
+    [self updateControlButton];
+}
+
+// Called by various other methods in this class to update the image
displayed on the main button.
+- (void) updateControlButton {
+
+    // Depending on the state of the audio recorder and audio player, the
name of the image
+    // that we want to show will vary.
+    NSString* imageName = nil;
+
+    if (self.audioRecorder.isRecording || self.audioPlayer.isPlaying) {
+        // If the audio is either recording or playing, the user can stop.
+        imageName = @"StopButton";
+
+    } else if (self.note.audioNote != nil) {
+
+        // If an audio note is present, the user can play it. (They need to
delete it before
+        // recording another one.)
+        imageName = @"PlayButton";
```

```
+
+    } else {
+
+        // If no audio note is present, the user can record one.
+        imageName = @"RecordButton";
+
+        // However, if access to the microphone isn't allowed (see
viewDidLoad), disable the button
+        // and make it semi-transparent (to indicate that it won't work.)
+        if (recordingAllowed == NO) {
+            self.controlButton.enabled = NO;
+            self.controlButton.alpha = 0.5;
+        } else {
+
+            // If recording _is_ allowed, enable the button and make it
fully opaque.
+            self.controlButton.enabled = YES;
+            self.controlButton.alpha = 1.0;
+        }
+
+    }
+
+    // Now that we know the name of the image, we load it up and give it to
the button.
+    UIImage* image = [UIImage imageNamed:imageName];
+
+    [self.controlButton setImage:image forState:UIControlStateNormal];
+}
+
+// Called when the view is about to go off-screen.
+- (void)viewWillDisappear:(BOOL)animated {
+
+    // We might be playing or recording audio. In either case, stop it now
before we move to
+    // another screen.
+    [self stopPlaying];
+    [self stopRecording];
+}
+
+// Called when the audio player finishes playback.
+- (void)audioPlayerDidFinishPlaying:(AVAudioPlayer *)player successfully:
(BOOL)flag {
+
+    // We're no longer playing back, so update the control button.
+    [self updateControlButton];
+}
+
+@end
```

13-EventKit

We'll now make the application notice when you're creating a note during an event that's on your calendar. If you create a note during an event, or within 15 (Provided code does 6 hours) minutes of it starting and ending, the default note text will read "Note created during (event name)".

1. Open DYNoteListViewController and modify the addNote: method to remove the line of code that sets the note's text.

DYNoteListViewController.m

```
  - (IBAction)addNote:(id)sender {

  // Tell the DYNoteStorage to create a new note.
-    DYNote* newNote = [[DYNoteStorage sharedStorage] createNote];
-    newNote.text = @"New Note";
+    [[DYNoteStorage sharedStorage] createNote];

}
```

1. Open DYNoteStorage.m.
2. `@import EventKit`.
3. Add a new `nonatomic` property to DYNoteStorage: an EKEventStore named EventStore.
4. Implement the `eventStore` method (which is a lazy getter for the property.)

DYNoteStorage.m

```
+// Returns the event store.
+- (EKEventStore*) eventStore {
+    if (_eventStore != nil)
+        return _eventStore;
+
+    _eventStore = [[EKEventStore alloc] init];
+
+    return _eventStore;
+}
+
```

1. Update the `createNote` method to request calendar access.

DYNoteStorage.m

```
// Creates a new note, and adds it to the database.
- (DYNote *)createNote {

  // Create the new note, and insert it into the managed object context.
  DYNote* newNote = [NSEntityDescription
insertNewObjectForEntityForName:@"Note"
inManagedObjectContext:self.managedObjectContext];

+    // Note text defaults to "New note"
+    newNote.text = @"New note";
+
+    // Query the calendar, and try to find out if we're creating this note
during an event.
+    // First, ensure that we have access to the calendar.
+    [self.eventStore requestAccessToEntityType:EKEntityTypeEvent
completion:^(BOOL granted, NSError *error) {
+        if (granted) {
+            // If we have permission, find an appropriate event and prepare
the note using that.
+            [self prepareNoteWithCalendarEvent:newNote];
+        } else {
+            // We don't have permission - leave the note as-is.
+        }
+
+    }];
+
  // Try to save the database, which ensures that the note is stored.
  NSError* error = nil;

  [self.managedObjectContext save:&error];

  if (error != nil) {
      NSLog(@"Couldn't save the context: %@", error);
      return nil;
  }

  // Return the new note.
  return newNote;
}
```

1. Implement the `prepareNoteWithCalendarEvent:` method, which queries the calendar and finds events to use.

DYNoteStorage.m

```
+// Called by createNote, to try to find the event that the note is being
created in.
+- (void) prepareNoteWithCalendarEvent:(DYNote*)note {
```

```
+
+    if ([EKEventStore authorizationStatusForEntityType:EKEntityTypeEvent]
!= EKAuthorizationStatusAuthorized) {
+        // We don't have permission to access the calendar. Give up.
+        return;
+    }
+
+    // We're looking for events that started 6 hours ago, and end 6 hours
from now.
+    // We'll use NSDateComponents to perform the calculation, because this
kind of calculation is _tricky_.
+
+
+    // First, get the user's current calendar. (In Western countries, this
will probably be Georgian, but could be different.)
+    NSCalendar* calendar = [NSCalendar currentCalendar];
+
+    // Get the current time.
+    NSDate* now = [NSDate date];
+
+    // Construct an NSDateComponent that represents "6 hours before".
+    NSDateComponents* startOffset = [[NSDateComponents alloc] init];
+    [startOffset setHour:-6];
+
+    // Construct an NSDateComponent that represents "6 hours after".
+    NSDateComponents* endOffset = [[NSDateComponents alloc] init];
+    [endOffset setHour:6];
+
+    // Get an NSDate that represents 6 hours before now by adding this
offset to now.
+    NSDate* startDate = [calendar dateByAddingComponents:startOffset
toDate:now options:0];
+
+    // Get an NSDate that represents 6 hours from now by adding this offset
to now.
+    NSDate* endDate = [calendar dateByAddingComponents:endOffset toDate:now
options:0];
+
+    // We now have dates that refer to 6 hours ago and 6 hours from now.
+    // Create a predicate (aka a search query) that finds events within the
range of startDate and endDate, on all calendars.
+    NSPredicate* predicate = [self.eventStore
predicateForEventsWithStartDate:startDate endDate:endDate calendars:nil];
+
+    // Find the events.
+    NSArray* events = [self.eventStore eventsMatchingPredicate:predicate];
+
+    EKEvent* eventToUse = nil;
+
+    for (EKEvent* event in events) {
+        // If we're in the middle of this event - that is, start time is
```

```
before now, and end time is after now - then use this event.
+
+        if ([event.startDate compare:now] == NSOrderedAscending && //
startDate is before now
+            [event.endDate compare:now] == NSOrderedDescending) { //
endDate is after now
+
+            // The event is happening right now! We'll use it.
+            eventToUse = event;
+
+            // Stop looking for events.
+            break;
+
+        }
+    }
+
+    // If we now have an event, grab info from it!
+    if (eventToUse) {
+        note.text = [NSString stringWithFormat:@"Note created during %@",
eventToUse.title];
+    } else {
+        // We have no event, so just leave the note
+    }
+
+    // Save the note's changes. This method might be run on a background
thread, so to keep things safe, we'll save it
+    // on the main operation queue.
+
+    // Get the main queue
+    NSOperationQueue* mainQueue = [NSOperationQueue mainQueue];
+
+    // Schedule the save operation.
+    [mainQueue addOperationWithBlock:^{
+        NSError* error = nil;
+
+        [note.managedObjectContext save:&error];
+
+        if (error != nil) {
+            NSLog(@"Failed to save the note: %@", error);
+        }
+    }];
+
+
+}
+
```

14-LocalNotifications

We'll make it so that you can set a time for reminding you of a note.

1. Create a new subclass of UIViewController, called DYReminderViewController.
2. Open the Storyboard.
3. Go to the Note View Controller.
4. Drag in a Flexible Space into the toolbar at the bottom.
5. Drag in a Bar Button item, labelled "Reminder". Arrange everything so it goes Location - Reminder - Audio.
6. Drag in a new view controller.
7. Set its class to `DYReminderViewController`.
8. Create a segue from the Reminder button to the new view controller. Set the segue's identifier to 'showReminder'.
9. Set the title of the view controller's navigation bar to 'Reminder'.

Now, we'll set up the interface.

1. Drag in a Date Picker. Put it right in the middle of the Reminders screen.
2. Drag in a Switch. Put it above the Date Picker, and to the right.
3. Drag in a Label. Set the text to "Show reminder". Align it with the Switch.

Next, we'll connect the interface to code.

1. Open DYReminderViewController.h in the assistant.
2. `#import` DYNote.h.
3. Add a `DYNote` property called `note`.
4. Switch to DYReminderViewController.m.
5. Connect the switch to an outlet called `reminderSwitch`, **and** to an action called `reminderSwitchChanged`.
6. Connect the date picker to an outlet called `datePicker`.

Next, we'll make the note view controller pass the DYNote to the DYReminderViewController.

1. Open DYNoteViewController.m
2. Import DYReminderViewController.
3. Update `prepareForSegue:` to pass the note when the `showReminder` segue is happening.

DYNoteViewController.m

```
// Called when a segue is about to happen.
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {

  if ([segue.identifier isEqualToString:@"showLocation"]) {

      // If this is the showLocation segue, we're moving to a
DYLocationViewController.
      DYLocationViewController* locationViewController =
segue.destinationViewController;

      // Give it the note.
      locationViewController.note = self.note;
  }

  if ([segue.identifier isEqualToString:@"showAudio"]) {

      // If this is the showAudio segue, we're moving to a
DYAudioViewController.
      DYAudioViewController* audioViewController =
segue.destinationViewController;

      // Give it the note.
      audioViewController.note = self.note;
  }
+
+    if ([segue.identifier isEqualToString:@"showReminder"]) {
+
+        // If this is the showReminder segue, we're moving to a
DYReminderViewController.
+        DYReminderViewController* reminderViewController =
segue.destinationViewController;
+
+        // Give it the note.
+        reminderViewController.note = self.note;
+    }
}
```

Next, we'll add support for setting reminders.

1. Open DYNote.h.
2. Add a nonatomic `NSDate` property called `reminderDate`.

DYNote.h

```
@property (nonatomic, strong) NSData* audioNote;

+/// The date at which a reminder will appear.
+@property (nonatomic) NSDate* reminderDate;
+
```

1. Open DYNote.m.
2. Declare the `reminderDate` property as `@dynamic`.
3. Implement the `localNotification`, `setReminderDate:` and `reminderDate` methods.

DYNote.m

```
+/// Finds the notification associated with this note, if any exists.
+- (UILocalNotification*) localNotification {
+
+    // Notes are associated with notifications using the note's URL.
+    NSString* urlRepresentation = [[self.objectID URIRepresentation]
absoluteString];
+
+    // Go through all currently-scheduled notifications
+    for (UILocalNotification* notification in [UIApplication
sharedApplication].scheduledLocalNotifications) {
+
+        // Get the note URL from this notification
+        NSURL* notificationNote = notification.userInfo[@"note"];
+
+        // If it's the same as this note's URL, then we've found the
notification associated with this note
+        if ([notificationNote isEqual:urlRepresentation])
+            return notification;
+    }
+
+    // We didn't find it; return nil.
+    return nil;
+}
+/// Schedules a reminder to go off at reminderDate; cancels any existing
reminders.
+- (void)setReminderDate:(NSDate *)reminderDate {
+
+    // Get the existing notification, if any, and cancel it
+    UILocalNotification* existingNotification = [self localNotification];
+
+    if (existingNotification) {
+        [[UIApplication sharedApplication]
cancelLocalNotification:existingNotification];
+    }
+
```

```
+    // If the date is set to nil, don't schedule a new notification
+    if (reminderDate == nil)
+        return;
+
+    // Create a new notification
+    UILocalNotification* newNotification = [[UILocalNotification alloc]
init];
+
+    // Set the time when it's going to go off
+    newNotification.fireDate = reminderDate;
+
+    // Setting the timezone means that if the user changes time zones, the
notification's fire date will be updated.
+    // If you don't do this, the fire date is considered to be in GMT and
won't update when the user changes time zones.
+    newNotification.timeZone = [NSTimeZone defaultTimeZone];
+
+    // When the alert fires, show the text of this note.
+    newNotification.alertBody = self.text;
+
+    // Associate the notification with this object's identifier.
+    newNotification.userInfo = @{@"note": [[self.objectID
URIRepresentation] absoluteString]};
+
+    // Schedule the notification.
+    [[UIApplication sharedApplication]
scheduleLocalNotification:newNotification];
+
+}
+
+// Returns the date of the associated reminder, if any exists.
+- (NSDate *)reminderDate {
+
+    // Get the notification, and return its fire date.
+    UILocalNotification* notification = [self localNotification];
+
+    return notification.fireDate;
+}
```

Finally, we'll write the code that gets and sets the reminder date.

The way that this works is as follows:

- when the view appears, if the note has a reminder, turn on the switch, update the date picker to use the reminder date, and make the date picker available. If it has no reminder, turn off the switch and disable the date picker.
- When the switch is turned on or off, enable or disable the date picker.
- When the view is exited, if the switch is on, set the reminder date. Otherwise, clear the reminder date entirely (removing the notification.)

- Add the `updateInterface` and `viewWillDisappear:` methods
- Update the `viewDidLoad` and `reminderSwitchChanged:` methods.

DYReminderViewController.h

```
+#import <UIKit/UIKit.h>
+#import "DYNote.h"
+
+@interface DYReminderViewController : UIViewController
+
+@property (strong) DYNote* note;
+
+@end
```

DYReminderViewController.m

```
+#import "DYReminderViewController.h"
+
+@interface DYReminderViewController ()
+@property (weak, nonatomic) IBOutlet UISwitch *reminderSwitch;
+@property (weak, nonatomic) IBOutlet UIDatePicker *datePicker;
+
+@end
+
+@implementation DYReminderViewController
+
+- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle
*)nibBundleOrNil
+{
+    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
+    if (self) {
+        // Custom initialization
+    }
+    return self;
+}
+
+- (void)viewDidLoad
+{
+    [super viewDidLoad];
+
+    // When the view loads, check to see if we have a reminder scheduled.
+    if (self.note.reminderDate) {
+
+        // If we do, turn on the switch, and make the date picker show the
reminder date.
+        self.reminderSwitch.on = YES;
+        self.datePicker.date = self.note.reminderDate;
+
```

```
+    } else {
+
+        // If we don't, just disable the switch.
+        self.reminderSwitch.on = NO;
+    }
+
+    // Update whether or not the date picker can be used.
+    [self updateInterface];
+
+}
+
+// Called when the view loads, and when the switch changes state.
+- (void) updateInterface {
+
+    if (self.reminderSwitch.on) {
+
+        // If the switch is on, we can work with the date picker, so enable
it.
+        self.datePicker.enabled = YES;
+        self.datePicker.alpha = 1.0;
+
+    } else {
+
+        // Otherwise, we can't work with the date picker, so disable it.
+        self.datePicker.enabled = NO;
+        self.datePicker.alpha = 0.5;
+
+    }
+}
+
+- (void)didReceiveMemoryWarning
+{
+    [super didReceiveMemoryWarning];
+    // Dispose of any resources that can be recreated.
+}
+
+// Called when the user taps the 'Show reminder' switch.
+- (IBAction)reminderSwitchChanged:(id)sender {
+
+    // The 'on' state of the button has already changed, so update the date
picker
+    // by calling updateInterface.
+    [self updateInterface];
+}
+
+// Called when the view is about to go away.
+- (void)viewWillDisappear:(BOOL)animated {
+
+    if (self.reminderSwitch.on == YES) {
+
+        // If the reminder switch is on, the user wants to set the
```

```
reminder.
+        // Set the date that's currently shown in the picker.
+
+        NSDate* dateToSet = self.datePicker.date;
+
+        if ([dateToSet timeIntervalSinceNow] <= 0) {
+            // This date is in the past, so we can't use it as a reminder.
Instead,
+            // use a date that's 10 seconds in the future.
+            dateToSet = [NSDate dateWithTimeIntervalSinceNow:10];
+        }
+
+        self.note.reminderDate = dateToSet;
+    } else {
+
+        // The switch is off, which means the user wants to have no
reminder. Clear it by
+        // setting the reminder date to nil.
+        self.note.reminderDate = nil;
+    }
+}
```

15-NicerTextView

Currently, the text view never dismisses the keyboard. We're going to make it so that tapping the text view dismisses the keyboard, when it's up.

1. Open the storyboard.
2. Drag a Tap Gesture Recognizer onto the text view.
3. Connect the gesture recognizer to a new method, called textViewTapped.
4. Implement the `textViewTapped:` method in DYNoteViewController.m

DYNoteViewController.m

```
+// Called when the user taps on the text view.
+- (IBAction)textViewTapped:(id)sender {
+
+    // If the text view is currently the first responder (i.e. owns the
keyboard),
+    // make it resign the keyboard.
+    if ([self.noteTextView isFirstResponder])
+        [self.noteTextView resignFirstResponder];
+
+    // Otherwise, make it _become_ the first responder.
+    else
+        [self.noteTextView becomeFirstResponder];
+}
```

Now, we'll make it so that the text field adjusts its size to account for the keyboard. Additionally, because the text field is under the toolbar, we also want to take into account the toolbar's size.

1. Open the storyboard.
2. Connect the toolbar to a new outlet called 'toolbar'.
3. Implement the `updateTextInsetWithBottomHeight:`, `keyboardWillShow:` and `keyboardWillHide:` methods.

DYNoteViewController.m

```
+// Called when the keyboard is about to appear.
+- (void) keyboardWillShow:(NSNotification*)notification {
+
+    // Get the frame (position and size) of the keyboard.
+    CGRect keyboardFrame =
[notification.userInfo[UIKeyboardFrameEndUserInfoKey] CGRectValue];
+
+    // We only care about the keyboard's height. Make the text view account
for the
+    // fact that it's taking up space.
+    [self updateTextInsetWithBottomHeight:keyboardFrame.size.height];
+
+}
+
+// Called when the keyboard is about to disappear.
+- (void) keyboardWillHide:(NSNotification*)notification {
+
+    // The keyboard's about to be gone; make the text view adjust.
+    [self updateTextInsetWithBottomHeight:self.toolbar.frame.size.height];
+
+}
+
+// Called by viewDidLoad, keyboardWillShow and keyboardWillHide to adjust
the
+// scrolling region of the text view.
+- (void) updateTextInsetWithBottomHeight:(float)height {
+
+    // The text view never actually changes size. Instead, the scrollable
inset changes,
+    // which makes the area that gets scrolled around in smaller or bigger
to account for
+    // the change in visible area.
+
+    // Get the current insets - we only want to override the bottom value,
so we aren't
+    // replacing the entire thing.
+    UIEdgeInsets insets = self.noteTextView.contentInset;
+
+    // Make the insets use the provided height.
+    insets.bottom = height;
+
+    // Adjust both the content inset of the text view, as well as the inset
of the scroll bars.
+    self.noteTextView.contentInset = insets;
+    self.noteTextView.scrollIndicatorInsets = insets;
+
+}
```

1. Update the `viewDidLoad`, `viewWillAppear` and `viewWillDisappear` methods.

DYNoteViewController.m

```
- (void)viewDidLoad
{
    [super viewDidLoad];
  // Do any additional setup after loading the view, typically from a nib.

    // When the screen loads, get the URL of the object that we're showing.
    // Store it into the user defaults, so that if the app exits while we're
editing this
    // note, the app will return to it.
    NSURL* noteURL = [self.note.objectID URIRepresentation];
    [[NSUserDefaults standardUserDefaults] setURL:noteURL
forKey:@"current_note"];

+    // When the view first appears, the keyboard is not up, so the toolbar
will be the
+    // biggest thing covering the text view.
+    // So, make the text view adjust to the toolbar's height.
+    [self updateTextInsetWithBottomHeight:self.toolbar.frame.size.height];

}

// Called when the view controller is about to appear.
- (void)viewWillAppear:(BOOL)animated {
    // Make the note text view use the text that's in the note.
    self.noteTextView.text = self.note.text;

+    // Register to be notified when the keyboard appears or disappears.
+    [[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(keyboardWillShow:) name:UIKeyboardWillShowNotification
object:nil];
+    [[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(keyboardWillHide:) name:UIKeyboardWillHideNotification
object:nil];
+
+}
+
// Called just before the view controlled is about to go away.
- (void)viewWillDisappear:(BOOL)animated {
    // Store the text that's in the note text view into the note itself.
    self.note.text = self.noteTextView.text;

    // Save the note.
    NSError* error = nil;
    [self.note.managedObjectContext save:&error];
    if (error != nil) {
        NSLog(@"Failed to save the note! %@", error);
    }

    [[NSUserDefaults standardUserDefaults] setURL:nil
```

```
forKey:@"current_note"];

+    // Unregister to be notified about the keyboard.
+    [[NSNotificationCenter defaultCenter] removeObserver:self];
}
```

16-Images

We'll add the ability to take photos and put them in notes.

First, we'll need to store the photo data in notes.

1. Open Diary.xcdatamodeld.
2. Add a new Attribute to the Note entity: a Binary Data called `image`.
3. Open `DYNote.h`. Add a new `nonatomic` property: an `NSData` called `image`.

DYNote.h

```
@property (nonatomic, strong) NSData* audioNote;

+/// The image, stored as an NSData
+@property (nonatomic, strong) NSData* image;
+
```

1. Open `DYNote.m`. Declare the `image` property as `@dynamic`.

DYNote.m @dynamic location; @dynamic audioNote; +@dynamic image;

Next, we'll set up the view controller that lets the user take, view and remove photos.

1. Create a new `UIViewController` subclass called `DYPhotoViewController`.
2. Open the storyboard. Go to the Note View Controller.
3. Add a Bar Button Item to the top-right of the navigation bar. Set its identifier to Camera.
4. Drag in a View Controller. Set its class to `DYPhotoViewController`.
5. Control-drag from the new Camera button to the new view controller, and create a new Push segue. Name the segue 'showPhoto'.
6. Set the title of the new view controller to 'Photo'.

Next, we'll set up the interface of this new view controller.

1. Drag in a Bar Button Item, and put it at the top-right of the navigation bar. Set its Identifier to Trash.
2. Drag in a Toolbar. Change the Title of the existing button to Take Photo. Drag a Flexible Space in and put it at the left hand side (so that the Take Photo button is placed at the right.)
3. Drag in a Label. Change its title to 'No Photo'. Change its font to System Bold 24pt,

and change its color to Light Grey (it's one of the system colors.)
4. Drag in another Label. Change its title to "Tap Take Photo to take a new photo." Change its font to System 17pt, and change its color to Light Grey. Change the number of lines to 0, and resize it so that the text is on two lines of roughly equal length. Place it under the larger label.
5. Select both labels, and position them so that they're centered in the screen.
6. Drag in an Image View. Make it fill the screen. Change its mode to Aspect Fit. Make sure that the image is above the labels (check the Outline pane to be sure.)

Next, we'll make the photo view controller able to receive a note; we'll also make the photo view controller pass the note to the photo view controller.

1. Open DYPhotoViewController.h.
2. Import DYNote.h.
3. Add a new `strong` property: a `DYNote` called `note`.
4. Open DYNoteViewController. Import DYPhotoViewController.h.
5. Modify

DYNoteViewController.m

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {

  if ([segue.identifier isEqualToString:@"showLocation"]) {

      // If this is the showLocation segue, we're moving to a
DYLocationViewController.
      DYLocationViewController* locationViewController =
segue.destinationViewController;

      // Give it the note.
      locationViewController.note = self.note;
  }

  if ([segue.identifier isEqualToString:@"showAudio"]) {

      // If this is the showAudio segue, we're moving to a
DYAudioViewController.
      DYAudioViewController* audioViewController =
segue.destinationViewController;

      // Give it the note.
      audioViewController.note = self.note;
  }

  if ([segue.identifier isEqualToString:@"showReminder"]) {

      // If this is the showReminder segue, we're moving to a
DYReminderViewController.
      DYReminderViewController* reminderViewController =
segue.destinationViewController;

      // Give it the note.
      reminderViewController.note = self.note;
  }
+
+    if ([segue.identifier isEqualToString:@"showPhoto"]) {
+
+        // If this is the showPhoto segue, we're moving to a
DYPhotoViewController.
+        DYPhotoViewController* photoViewController =
segue.destinationViewController;
+
+        // Give it the note.
+        photoViewController.note = self.note;
+    }
}
```

Next, we'll connect up the interface to the code.

1. Open the storyboard, and go to the Photo View Controller. Open DYPhotoViewController.m in the Assistant.
2. Connect the image view to a new outlet called `imageView`.
3. Connect the Take Photo button to a new action called `takePhoto`.
4. Connect the Trash button to a new action called `deletePhoto`.

Next, we'll implement the code.

1. Make DYPhotoViewController conform to `UINavigationControllerDelegate` and `UIImagePickerControllerDelegate`.
2. Update the `viewDidLoad` method, and implement the `takePhoto:`, `deletePhoto:`, `imagePickerController:didFinishPickingMediaWithInfo:` and `imagePickerControllerDidCancel:` methods.

DYPhotoViewController.h (Completed files)

```
+#import <UIKit/UIKit.h>
+#import "DYNote.h"
+
+@interface DYPhotoViewController : UIViewController
+
+@property (strong) DYNote* note;
+
+@end
```

DYPhotoViewController.m

```
+#import "DYPhotoViewController.h"
+
+@interface DYPhotoViewController () <UINavigationControllerDelegate,
UIImagePickerControllerDelegate>
+
+@property (weak, nonatomic) IBOutlet UIImageView *imageView;
+@end
+
+@implementation DYPhotoViewController
+
+- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle
*)nibBundleOrNil
+{
+    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
+    if (self) {
+        // Custom initialization
+    }
+    return self;
+}
+
+- (void)viewDidLoad
```

```
+{
+    [super viewDidLoad];
+
+    // When the view loads, the note may already have an image.
+    // Attempt to load it, and if that succeeds, show the image.
+    UIImage* image =  [UIImage imageWithData:self.note.image];
+
+    if (image) {
+        self.imageView.image = image;
+    }
+}
+
+- (void)didReceiveMemoryWarning
+{
+    [super didReceiveMemoryWarning];
+    // Dispose of any resources that can be recreated.
+}
+
+// Called when the Take Photo button is tapped.
+- (IBAction)takePhoto:(id)sender {
+
+    // Create a new image picker, and configure it.
+    UIImagePickerController* picker = [[UIImagePickerController alloc]
init];
+
+    // If a camera is available, make the picker use it.
+    if ([UIImagePickerController
isSourceTypeAvailable:UIImagePickerControllerSourceTypeCamera]) {
+
+        picker.sourceType = UIImagePickerControllerSourceTypeCamera;
+
+    } else {
+        // If no camera is available (we could be on an early model
+        // iPad, an iPod touch, or the iOS simulator), use the photo
+        // library instead.
+        picker.sourceType = UIImagePickerControllerSourceTypePhotoLibrary;
+    }
+
+    // We want to be told about when the picker finishes picking.
+    picker.delegate = self;
+
+    // Present the picker to the user!
+    [self presentViewController:picker animated:YES completion:nil];
+
+}
+
+// Called when the user taps the Delete Photo button.
+- (IBAction)deletePhoto:(id)sender {
+
+    // First, get rid of the image data, and empty the image view.
+    self.note.image = nil;
```

```
+    self.imageView.image = nil;
+
+    // Next, save the changes.
+    NSError* error = nil;
+
+    [self.note.managedObjectContext save:&error];
+
+    if (error != nil) {
+        NSLog(@"Failed to save the note: %@", error);
+    }
+}
+
+// Called when the user finishes taking a picture.
+- (void)imagePickerController:(UIImagePickerController *)picker
didFinishPickingMediaWithInfo:(NSDictionary *)info {
+
+    // Get the image from the info dictionary.
+    UIImage* image = info[UIImagePickerControllerOriginalImage];
+
+    // Show the image to the user.
+    self.imageView.image = image;
+
+    // We also want to store the image in the database. We need to
+    // convert the image into a format that can be stored - either
+    // PNG or JPEG. In this example, we'll go with JPEG.
+    // JPEG lets you choose how much to compress by, from 0 to 1;
+    // numbers closer to 0 are smaller files, but numbers closer to
+    // 1 are better quality. 0.8 is a decent compromise.
+    NSData* imageData = UIImageJPEGRepresentation(image, 0.8);
+
+    // Store the data in the note object, and save the database.
+    self.note.image = imageData;
+
+    NSError* error = nil;
+
+    [self.note.managedObjectContext save:&error];
+
+    if (error != nil) {
+        NSLog(@"Failed to save the note: %@", error);
+    }
+
+    // Finally, we need to make the image picker go away.
+    [self dismissViewControllerAnimated:YES completion:nil];
+}
+
+// Called when the user taps the Cancel button in the image picker.
+- (void)imagePickerControllerDidCancel:(UIImagePickerController *)picker {
+
+    // The user cancelled, so we should just get rid of
+    // the image picker.
+    [self dismissViewControllerAnimated:YES completion:nil];
```

```
+
+}
+
+@end
```

17-iPad

We'll now port this entire app to the iPad.

1. Open the Project (select it at the top of the Project Navigator.)
2. Change Devices from iPhone to Universal. Xcode will ask if you want to copy the Storyboard file. Click Don't Copy.

   (Despite it promising to copy the file, clicking 'Copy' will actually just create a group called iPad.)

3. Still on the same page, switch from iPhone to iPad. Change the Main Interface from 'Main' to 'Main-iPad'.

Next, we'll create the storyboard for the iPad.

1. Create a new Storyboard file. (You'll find storyboards in the User Interface category.) Set the Device Family to iPad. Name it 'Main-iPad'.
2. Go to the Info tab, and ensure that the "Main storyboard file base name (iPad)" is set. If "Main nib file base name (iPad)" is set instead, change it to "storyboard". Weird Xcode bug, I guess?
3. Open the new storyboard.

The iPad app will work in a different way to the iPhone one. Whereas the iPhone version created a brand-new DYNoteViewController when the list is tapped, the iPad version will only ever have one. When a note is tapped, the text in the Note View Controller is replaced.

1. Drag in a Split View Controller.
2. Select the table view. Change its class to DYNoteListViewController.
3. Select the prototype cell, and change its identifier to "NoteCell". Change its style to Basic.

Weirdly, you can't drag a bar button item directly into the navigation bar. Instead, you need do it in a roundabout way.

1. Drag a Bar Button Item into the Note List View controller, *in the outline view*. Put it just under the First Responder item.
2. Make it use the Add identifier, and control-drag from it to the View Controller to connect it to the addItem: method.
3. Select the Navigation Item. Go to the Connections inspector. Connect the Right Bar Button Item outlet to the Bar Button Item.
4. Select the Navigation Item, and set the Title to 'Notes'. (Double-clicking the title won't work.)

When you launch the app, you can create new notes, and delete them. Next up: connecting tapping the notes to showing them.

In order to have a navigation bar that doesn't have an ugly gap above it on the iPad, you need to use a navigation controller. Therefore, the right-hand-side needs to have a navigation controller, inside of which is the note view controller. It's kind of an ugly hack.

1. Delete the larger view controller.
2. Drag in a Navigation Controller.
3. Control-drag from the Split View Controller to the Navigation Controller, and make it the Detail View Controller. (Alternatively to 4 and 5, drag in a ViewController, embed in NavController and then hook it up)
4. The navigation controller comes with a table view controller. Delete it.
5. Drag in a View Controller. Make it the root view of the Navigation Controller.
6. Select the new view controller, and set its class to DYNoteViewController.

Next, we need to set up the new Note View Controller to actually show content.

1. Drag a Text View into the note view controller. Make it fill the screen.
2. Drag a toolbar into the note view controller. Place it at the bottom of the screen.
3. Connect the text view to the `noteTextView` outlet, and the toolbar to the `toolbar` outlet.
4. Drag in three additional bar button items. Rename them so that they look like this:

`Location |--| Audio |--| Reminder <-------> (camera icon)`

(`|--|` = fixed space; `<--->` = flexible space.)

You can now type into the text field. Additionally, it will handle rotation correctly, too.

The toolbar needs to be kept at the bottom of the screen. Select it, click the Pin menu, and pin the left, bottom and right edges.

Next up, we need to make the app switch between notes. To do that, the DYNoteListViewController needs to be able to talk to the DYNoteViewController, and give it the updated note view when the selection changes. Additionally, we need to make DYNoteViewController respond when the note changes.

1. Open DYNoteViewController.h. Make the `note` property be `nonatomic`.

DYNoteViewController.h

```
@interface DYNoteViewController : UIViewController

-@property (strong) DYNote* note;
+@property (nonatomic, strong) DYNote* note;

@end
```

1. Open DYNoteViewController.m. Implement the `setNote:` method.

DYNoteViewController.m

```
+// Called when the view controller is given a new DYNote.
+- (void)setNote:(DYNote *)note {
+
+    // First, we need to store the current text into the note.
+
+    // Before we try to save, ensure that the note is still valid.
+    // (The underlying data may have been deleted, so check first.)
+    if (self.note.isFault == NO)
+        self.note.text = self.noteTextView.text;
+
+    // Update to use the new DYNote.
+    _note = note;
+    self.noteTextView.text = self.note.text;
+
+    // Dismiss the keyboard, if it's up.
+    [self.noteTextView resignFirstResponder];
+
+    // Update the current note URL to reflect the fact that
+    // we're looking at a new note.
+    NSURL* noteURL = [self.note.objectID URIRepresentation];
+    [[NSUserDefaults standardUserDefaults] setURL:noteURL
forKey:@"current_note"];
+}
```

Next, we'll make the note change when a new note is tapped.

1. Control-drag from the Split View Controller to the Note List View Controller, and make the split view controller use the note list as its delegate.
2. Open DYNoteListViewController.m.
3. Make the class conform to `UISplitViewControllerDelegate`.

DYNoteListViewController.m

```
-@interface DYNoteListViewController () <NSFetchedResultsControllerDelegate,
UISearchBarDelegate, UISearchDisplayDelegate> {
+@interface DYNoteListViewController () <NSFetchedResultsControllerDelegate,
UISearchBarDelegate, UISearchDisplayDelegate, UISplitViewControllerDelegate>
{
}
```

1. Update `viewDidLoad` to make the split view controller use this view controller as its delegate.

DYNoteListViewController.m

```
- (void)viewDidLoad {

  // Get the table view's "Edit" button, which will put the table into Edit
mode when tapped
  self.navigationItem.leftBarButtonItem = self.editButtonItem;

  // Get the note storage system to give us a fetched results controller,
which we can use
  // to get the notes themselves.
  self.fetchedResultsController = [[DYNoteStorage sharedStorage]
createFetchedResultsController];

  // Tell the fetched results controller to let us know when data changes.
  self.fetchedResultsController.delegate = self;

  // Finally, tell the controller to start getting objects, and watching for
changes.
  NSError* error = nil;
  [self.fetchedResultsController performFetch:&error];

  if (error != nil) {
      NSLog(@"Problem fetching results! %@", error);
  }

  // Get the URL for the note that we were last editing from user defaults,
if one is set.
  NSURL* currentNoteURL = [[NSUserDefaults standardUserDefaults]
URLForKey:@"current_note"];

  if (currentNoteURL) {

      // If one is set, we need to use the URL to get the object out of the
database.
      DYNote* note = [[DYNoteStorage sharedStorage]
noteWithURL:currentNoteURL];

      // If the note exists, then create the note view controller, and give
it the note object
      if (note != nil) {
          DYNoteViewController* noteViewController = [self.storyboard
instantiateViewControllerWithIdentifier:@"NoteViewController"];
          noteViewController.note = note;

          // Next, push the view controller without an animation. This will
make the app start with
          // the note view controller visible.
          [self.navigationController pushViewController:noteViewController
animated:NO];
      }
```

```
  }

+    // Make the split view controller use this object as its delegate, so
that swiping
+    // to present and dismiss the last of notes works. (This only has an
effect on the iPad.)
+    self.splitViewController.delegate = self;
+
}
```

1. Implement the `splitViewController: willHideViewController: withBarButtonItem: forPopoverController:` method (which is empty - it just needs to exist.)

DYNoteListViewController.m

```
+// When this method is implemented, and this class is being used as the
delegate for the
+// split view controller, you can swipe to bring up and dismiss the master
view controller
+// when in portrait mode.
+- (void)splitViewController:(UISplitViewController *)svc
willHideViewController:(UIViewController *)aViewController
withBarButtonItem:(UIBarButtonItem *)barButtonItem forPopoverController:
(UIPopoverController *)pc {
+
+}
```

1. Implement the `noteViewController`

DYNoteListViewController.m

```
+// Called by tableView:didSelectRowAtIndexPath: to get the
DYNoteViewController.
+- (DYNoteViewController*) noteViewController {
+
+    // This code should only be called when we're on the iPad.
+    if (UI_USER_INTERFACE_IDIOM() != UIUserInterfaceIdiomPad)
+        return nil;
+
+    // Get the right-hand view controller, which is a navigation
controller.
+    UINavigationController* detailNavigationController =
self.splitViewController.viewControllers[1];
+
+    // Get the view controller inside it.
+    DYNoteViewController* noteViewController =
(id)detailNavigationController.topViewController;
+
+    // Return it.
+    return noteViewController;
+
+}
```

1. Implement the `tableView:didSelectRowAtIndexPath:` method.

DYNoteListViewController.m

```
+// Called when a table view cell is tapped.
+- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:
(NSIndexPath *)indexPath {
+
+    // Only run this code if we're on the iPad.
+    if (UI_USER_INTERFACE_IDIOM() != UIUserInterfaceIdiomPad)
+        return;
+
+    // Work out which note was selected (taking into account whether or not
we're searching.)
+    DYNote* selectedNote = nil;
+
+    if (tableView == self.searchDisplayController.searchResultsTableView) {
+        selectedNote = [self.searchFetchedResultsController
objectAtIndexPath:indexPath];
+    } else {
+        selectedNote = [self.fetchedResultsController
objectAtIndexPath:indexPath];
+    }
+
+    // Get the note view controller, and give it the new note.
+    [self noteViewController].note = selectedNote;
+
+}
+
```

Tapping on notes in the master view controller now switches between notes. Additionally, you can swipe to bring up the list of notes when in portrait mode.

Next up, we'll start adding the additional features.

1. Drag in a View Controller.
2. Change its Size to Freeform; then, select the view inside it, and change its size to be 320 wide and 480 high.
3. Change its class to DYPhotoViewController.
4. Reconstruct the UI to match the iPhone version. You'll need to create the Navigation bar yourself - you don't get one for free, because this view isn't being presented in a Navigation Controller. Don't forget to connect the outlets and actions.
5. Control-drag from the Camera button in the Note View Controller to the new Photo View Controller. Choose the Popover segue style. Name the segue 'showPhoto' (same as the iPhone version's.)
6. Repeat the same process for the DYLocationViewController, the DYAudioNoteViewController and the DYReminderViewController. The names for the segues for each are `showLocation`, `showAudio` and `showReminder`.

18-iPadPolish

There are a few remaining tweaks that we should add to the iPad version, before it's ready

to ship:

- If there's no note selected, disable the right hand view controller.
- If the current note is deleted, the right hand view controller should act as if no note is selected.
- Tapping the Trash button in the Location view controller does nothing, because the original code used the navigation controller (and the iPad version doesn't have that.)
- Open the storyboard. Go to the Note view controller.
- Drag in a Label. Make it use the System Bold 24pt font. Change the text to "No Note Selected".
- Open the Align menu, and add the Horizontal Center in Container and Vertical Center in Container constraints. Once that's done, select the lable and press Option-Command-= to re-position it in the center.
- Open DYNoteViewController.m in the assistant. Connect the label to a new outlet called `noNoteLabel`.
- Implement the `updateInterface` method. Update `setNote:` and `viewDidLoad` to call the new method.

DYNoteViewController.m

```
+@property (weak, nonatomic) IBOutlet UILabel *noNoteLabel;
...
- (void)viewDidLoad
{
  [super viewDidLoad];
// Do any additional setup after loading the view, typically from a nib.

  // When the screen loads, get the URL of the object that we're showing.
  // Store it into the user defaults, so that if the app exits while we're
editing this
  // note, the app will return to it.
  NSURL* noteURL = [self.note.objectID URIRepresentation];
  [[NSUserDefaults standardUserDefaults] setURL:noteURL
forKey:@"current_note"];

  // When the view first appears, the keyboard is not up, so the toolbar
will be the
  // biggest thing covering the text view.
  // So, make the text view adjust to the toolbar's height.
  [self updateTextInsetWithBottomHeight:self.toolbar.frame.size.height];
+    // When the view loads, we may not have a note to show. Update the
interface in case we don't.
+    [self updateInterface];
+

}
...
- (void)setNote:(DYNote *)note {
```

```
  // First, we need to store the current text into the note.

  // Before we try to save, ensure that the note is still valid.
  // (The underlying data may have been deleted, so check first.)
  if (self.note.isFault == NO)
      self.note.text = self.noteTextView.text;

  // Update to use the new DYNote.
  _note = note;
  self.noteTextView.text = self.note.text;

  // Dismiss the keyboard, if it's up.
  [self.noteTextView resignFirstResponder];

  // Update the current note URL to reflect the fact that
  // we're looking at a new note.
  NSURL* noteURL = [self.note.objectID URIRepresentation];
  [[NSUserDefaults standardUserDefaults] setURL:noteURL
forKey:@"current_note"];
+
+    // Update the interface so that the view is either enabled or
+    // disabled (depending on whether we have a note or not
+    [self updateInterface];
+}
...
+// Called by viewDidLoad and setNote: to update the interface depending on
whether we have a note or not.
+- (void) updateInterface {
+
+    // If we have no note, make the 'no note' label visible, disable the
+    // view so that no buttons can be tapped, and make everything grey.
+    if (self.note == nil) {
+        self.noNoteLabel.hidden = NO;
+        self.view.userInteractionEnabled = NO;
+        self.view.tintColor = [UIColor grayColor];
+    } else {
+
+        // Otherwise, if we have a note, hide the 'no note' label, enable
user interaction, and return the colours to normal.
+        self.noNoteLabel.hidden = YES;
+        self.view.userInteractionEnabled = YES;
+        self.view.tintColor = nil;
+    }
}
```

Next, we'll make deleting the current note deselect the note.

1. Open DYNoteListViewController.m.
2. Update `controller: didChangeObject: atIndexPath: forChangeType:`

`newIndexPath:` to set the note view controller's note to nil if it's currently displaying the note that was deleted.

DYNoteListViewController.m

```
- (void)controller:(NSFetchedResultsController *)controller didChangeObject:
(id)anObject atIndexPath:(NSIndexPath *)indexPath forChangeType:
(NSFetchedResultsChangeType)type newIndexPath:(NSIndexPath *)newIndexPath {

  UITableView* tableView = nil;
  if (self.searchDisplayController.active) {
      tableView = self.searchDisplayController.searchResultsTableView;
  } else {
      tableView = self.tableView;
  }

  // Different changes need different animations:
  switch (type) {
      case NSFetchedResultsChangeInsert:
          // A new object was inserted, so tell the table view to animate a
new cell in.
          [tableView insertRowsAtIndexPaths:@[newIndexPath]
withRowAnimation:UITableViewRowAnimationTop];
          break;

      case NSFetchedResultsChangeDelete:
          // An object was deleted, so tell the table view to delete the
appropriate row.
          [tableView deleteRowsAtIndexPaths:@[indexPath]
withRowAnimation:UITableViewRowAnimationLeft];
+
+            // iPad only: if the note view controller is currently showing
the object that just got deleted,
+            // set its 'note' property to nil. This will disable the
interface, preventing user error.
+            if ([self noteViewController].note == anObject) {
+                [self noteViewController].note = nil;
+            }
+
          break;

      case NSFetchedResultsChangeUpdate:
          // An object was changed, so update its contents by calling
configureCell:atIndexPath.
          [self configureCell:[tableView cellForRowAtIndexPath:indexPath]
atIndexPath:indexPath];
          break;

      case NSFetchedResultsChangeMove:
          // An object was move, so delete the row that it used to be in,
```

```
and insert one where it's now located.
          [tableView deleteRowsAtIndexPaths:@[indexPath]
withRowAnimation:UITableViewRowAnimationFade];
          [tableView insertRowsAtIndexPaths:@[newIndexPath]
withRowAnimation:UITableViewRowAnimationFade];
          break;
  }

}
```

Finally, we'll make it so that tapping the Trash button in the Location popover removes the location.

1. Open DYLocationViewController.m.
2. Update removeNote: to remove all annotations, stop the location manager, and set it to nil.

DYLocationViewController.m

```
- (IBAction)removeNote:(id)sender {

  // Remove the location from the note
  self.note.location = nil;

+    // Hide the pin by deleting all annotations.
+    [self.mapView removeAnnotations:self.mapView.annotations];
+
+    // Disable the location manager.
+    [self.locationManager stopUpdatingLocation];
+    self.locationManager = nil;
+
  // Return to the previous view controller.
  [self.navigationController popViewControllerAnimated:YES];
}
```

19-LaunchImages

The last step is to provide launch images.

1. Open the Images.xcassets file.
2. Go to the LaunchImage image set. Go to the Attributes inspector, and turn on iPad Portrait and Landscape.
3. Drag the appropriate files into the slots.

Ship it.