Day 3

1. My First App revisited (Discussion of homework)
2. Different view controller types
3. Scheme, Targets, Workspaces
4. Sharing Information
5. Autolayout and Size Classes
6. Blocks
7. Core Data
8. Web services (look at AFNetworking)
9. My Second App

Last weeks homework

View Controllers

- UINavigationViewController
- UITabBarController
- UITableViewController
- UICollectionViewController
- UISplitViewController
- UIPageViewController
- UISearchViewController
- UIPopOverController (exception)

Content Controllers vs. Container Controllers

A content view controller presents content on the screen using a view or a group of views organized into a view hierarchy

- To show data to the user
- To collect data from the user
- To perform a specific task
- To navigate between a set of available commands or options, such as on the launch screen for a game

A container view controller contains content owned by other view controllers. These other view controllers are explicitly assigned to the container view controller as its children.

- A container provides its own API to manage its children.
- A container decides whether the children have a relationship between them and what that relationship is.
- A container manages a view hierarchy just as other view controllers do. A container can also add the views of any of its children into its view hierarchy. The container decides when such a view is added and how it should be sized to fit the container's view hierarchy
- A container might impose specific design considerations on its children. For example.

Scheme, Targets, Workspaces

A target specifies a product to build and contains the instructions for building the product from a set of files in a project or workspace

An Xcode project is a repository for all the files, resources, and information required to build one or more software products

A workspace is an Xcode document that groups projects and other documents so you can work on them together

An Xcode scheme defines a collection of targets to build, a configuration to use when building, and a collection of tests to execute.

Sharing Information

1. prepareForSegue
2. shared context
3. database
4. delegation

Autolayout

- Many Devices
- Each has a landscape and portrait option
- Relative layout between any views
- Fine grain control
- No dependence on actual positioning

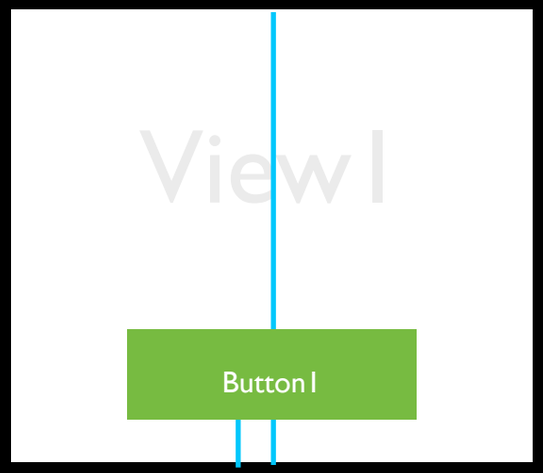
What is the Point?

(or not the point)

- Layout by intent
- Everything in your view is relative
- Layout is insensitive to device and orientation
- Easy to use ? Getting there
- Both visual and programatic use
- Internationalisation - Right to left (automatic support)
- Futureproof

The rule of relativity:

item1.attribute = multiplier * item2.attribute + constant



button1.centreX = 1 * View1.centreX + 0

button1.bottom = 1 * View1.bottom - 20

In code:

```
constraint = [NSLayoutConstraint constraintWithItem:button1
                attribute:NSLayoutAttributeCenterX
                relatedBy:NSLayoutRelationEqual
                toItem:view1
                attribute:NSLayoutAttributeCenterX
                multiplier:1.0f
                constant:0.0f];


   constraint = [NSLayoutConstraint constraintWithItem:button1
                attribute:NSLayoutAttributeBottom
                relatedBy:NSLayoutRelationEqual
                toItem:view1
                attribute:NSLayoutAttributeBottom
                multiplier:1.0f
                constant:-20.0f];

[view1 addConstraint:constraint];
```

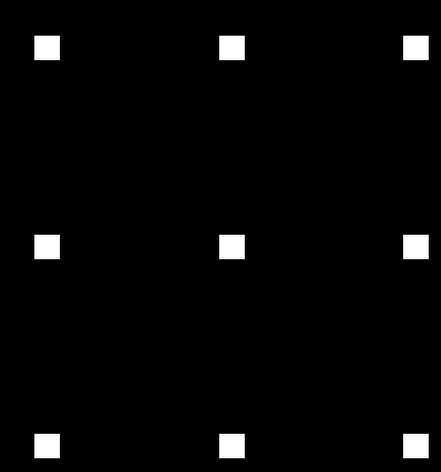
In Code using the Visual Format

- |-[view1]-[view2]-| (defaults to horizontal)
- V:|-[view1]-[view3]-1
- H:|[view4(==200)-100-[view5(==view1)]-|
- V:|-[view7(>=300@500)]-|

Size Classes (Adaptive Layout)

- Use size classes to enable a storyboard or xib file to work with all available screen sizes
- Core Concept of ‘adaptive layout’ is that size classes are abstracted away from device specific characteristics

Compact, Regular, Any



Four thins you can change between size classes:

1. Constraint existence
2. Constraint constant
3. View existence
4. View font

Core Data

“The Core Data framework provides generalized and automated solutions to common tasks associated with object life-cycle and object graph management, including persistence.The Core Data framework provides generalized and automated solutions to common tasks associated with object life-cycle and object graph management, including persistence.”

A framework for reading and writing your objects too a persistent store

Core Data Features

- Change tracking and undo support.
- Relationship maintenance
- Futures
- Automatic validation of property values
- Schema migration
- Optional integration with the application's controller layer to support user interface synchronization
- Full, automatic, support for key-value coding and key-value observing
- Grouping, filtering, and organizing data in memory and in the user interface

App

NSManaged Object

ELDCorp

ELDVIPClient

ELDVIPClient

ELDExpense

ELDCorp

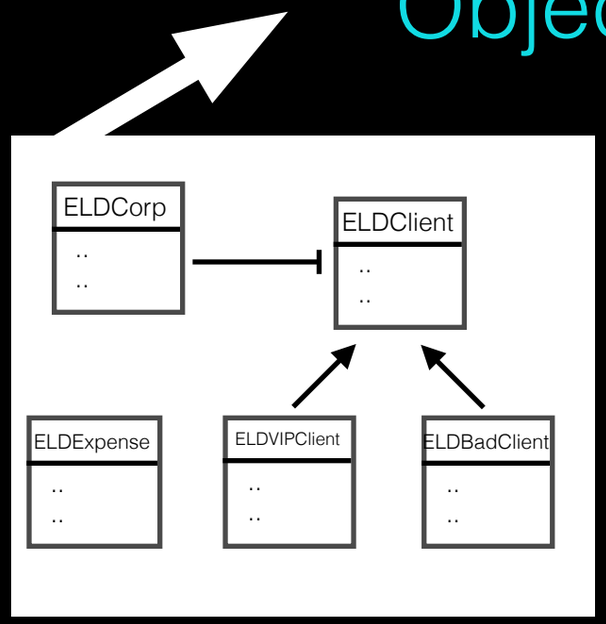
ELDBadClient

ELDBadClient

NSManaged ObjectContext A

NSManaged ObjectContext B

NSManaged ObjectModel



NSPersistant StoreCoordinator

SQLLite

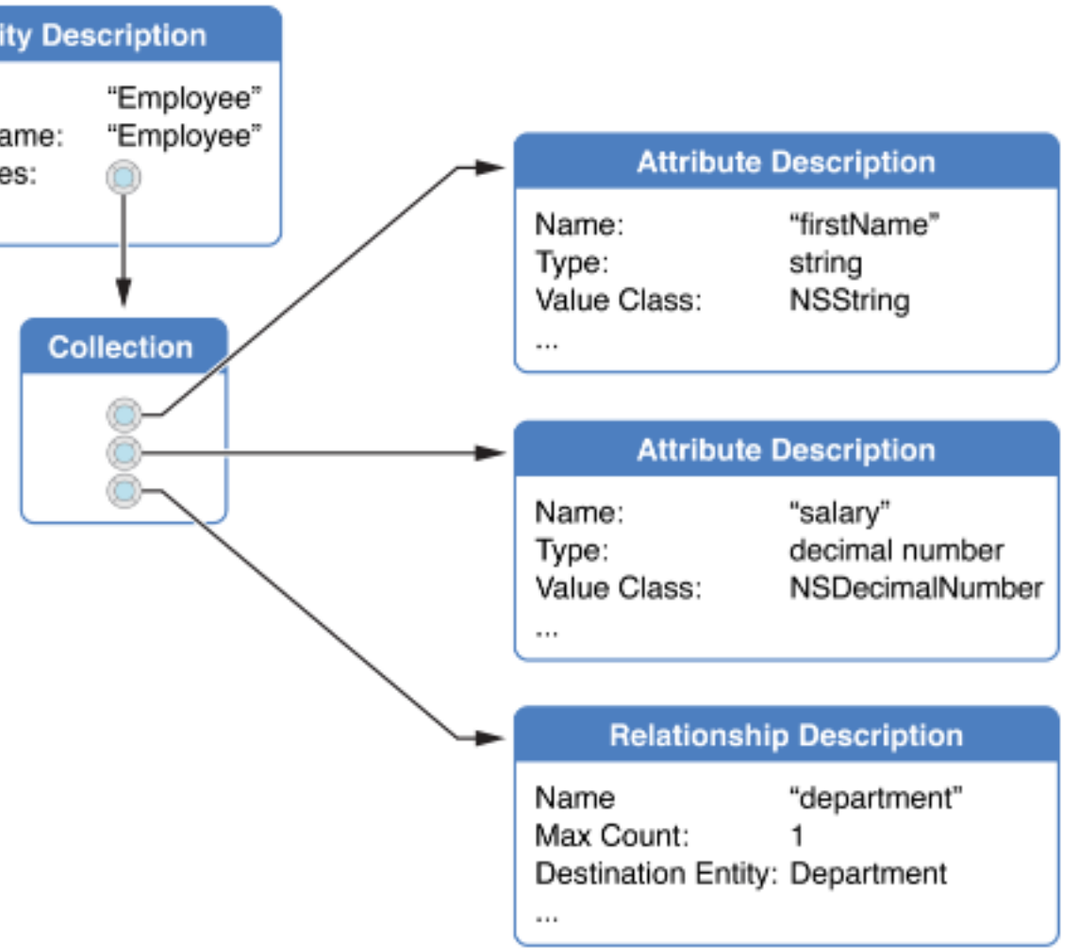
NoSQL

MongoDB

Web Service

NSEntityDescription

Figure 6 Entity description with two attributes and a relationship



Inserting and Deleting

```
NSManagedObject *newEmployee;

newEmployee = [NSEntityDescription insertNewObjectForEntityForName:@“Employee”
                                                inManagedObjectContext:context];
```

```
[context deleteObject:employee];
```

[context save:&error]

Custom Getters and Setters

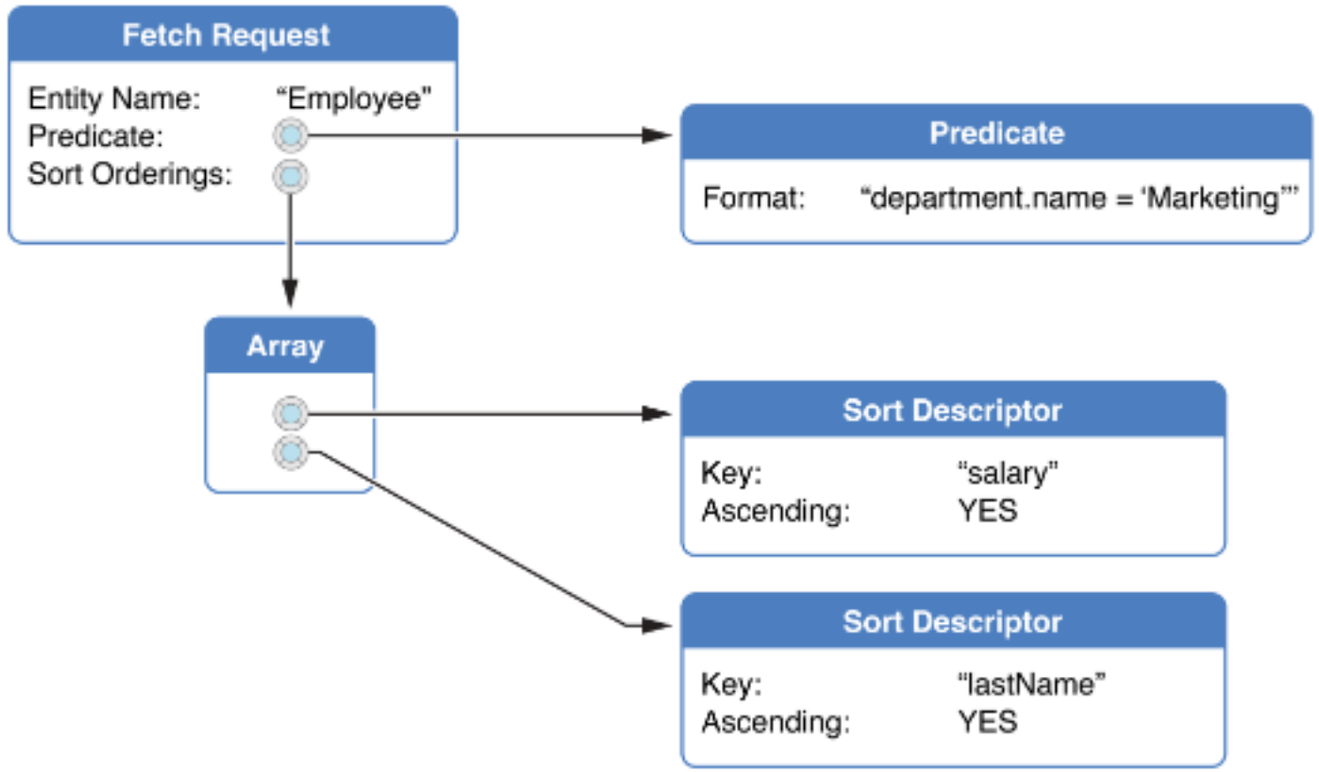
```
- (NSString *)name
  {
    [self willAccessValueForKey:@"name"];

    NSString *myName = [self primitiveName];

    [self didAccessValueForKey:@"name"];

    return myName;
  }



- (void)setName:(NSString *)newName
  {
    [self willChangeValueForKey:@"name"];

    [self setPrimitiveName:newName];

    [self didChangeValueForKey:@"name"];
  }
```

```
- (void)addEmployeesObject:(Employee *)value
  {
    NSSet *changedObjects = [[NSSet alloc]
      initWithObjects:&value count:1];

    [self willChangeValueForKey:@"employees"
          withSetMutation:NSKeyValueUnionSetMutation
          usingObjects:changedObjects];

    [[self primitiveEmployees] addObject:value];

    [self didChangeValueForKey:@"employees"
          withSetMutation:NSKeyValueUnionSetMutation
          usingObjects:changedObjects];
  }


- (void)removeEmployeesObject:(Employee *)value
  {
    NSSet *changedObjects = [[NSSet alloc]
        initWithObjects:&value count:1];

    [self willChangeValueForKey:@"employees"
          withSetMutation:NSKeyValueMinusSetMutation
          usingObjects:changedObjects];

    [[self primitiveEmployees] removeObject:value];

    [self didChangeValueForKey:@"employees"
          withSetMutation:NSKeyValueMinusSetMutation
          usingObjects:changedObjects];
  }
```

```
- (void)addEmployees:(NSSet *)value
  {
    [self willChangeValueForKey:@“employees" withSetMutation:NSKeyValueUnionSetMutation
usingObjects:value];

    [[self primitiveEmployees] unionSet:value];

    [self didChangeValueForKey:@“employees" withSetMutation:NSKeyValueUnionSetMutation
usingObjects:value];
  }



- (void)removeEmployees:(NSSet *)value
  {
    [self willChangeValueForKey:@“employees" withSetMutation:NSKeyValueMinusSetMutation
usingObjects:value];

    [[self primitiveEmployees] minusSet:value];

    [self didChangeValueForKey:@“employees" withSetMutation:NSKeyValueMinusSetMutation
usingObjects:value];
  }
```

Fetch Requests

Figure 3 An example fetch request



3 Components:

1. Entity
2. Predicate (filter)
3. Sort Descriptors (ordering)

NSPredicate

```
[NSPredicate predicateWithFormat:@"(lastName like[cd] %@) AND (birthday > %@)", lastNameSearchString, birthdaySearchDate];
```

```
[NSPredicate predicateWithFormat:@"lastName like[c] \"S*\""];


[NSPredicate predicateWithFormat:@"anAttribute == %@", [NSNumber numberWithBool:aBool]];


[NSPredicate predicateWithFormat:@"%K like %@", attributeName, attributeValue];


[NSPredicate predicateWithFormat:@"lastName like[c] $LAST_NAME"];
[predicateTemplate predicateWithSubstitutionVariables:[NSDictionary dictionaryWithObject:@"Turner" forKey:@"LAST_NAME"]];
[NSPredicate predicateWithFormat:@"date = $DATE"];
[predicate predicateWithSubstitutionVariables:[NSDictionary dictionaryWithObject:[NSNull null] forKey:@"DATE"]];
```

```
[NSMutableArray arrayWithObjects:@"Nick", @"Ben", @"Adam", @"Melissa", nil];
NSPredicate *bPredicate = [NSPredicate predicateWithFormat:@"SELF beginswith[c] 'a'"];
NSArray *beginWithB = [array filteredArrayUsingPredicate:bPredicate];
// beginWithB contains { @"Adam" }.
NSPredicate *sPredicate = [NSPredicate predicateWithFormat:@"SELF contains[c] 'e'"];
[array filterUsingPredicate:sPredicate];
// array now contains { @"Nick", @"Ben", @"Melissa" }


[NSPredicate predicateWithFormat:@"ANY employees.firstName like ‘Matthew'"];


[NSPredicate predicateWithFormat:@"(firstName == %@) || (firstName = nil)", firstName]


[NSPredicate predicateWithFormat: @"attributeName BETWEEN %@", @[@1, @10]]
```

```
NSManagedObjectContext *moc = [self managedObjectContext];

NSEntityDescription *entityDescription = [NSEntityDescription entityForName:@"Employee"
inManagedObjectContext:moc];

NSFetchRequest *request = [[NSFetchRequest alloc] init];

[request setEntity:entityDescription];


// Set example predicate and sort orderings…

NSNumber *minimumSalary = ...;

NSPredicate *predicate = [NSPredicate predicateWithFormat:@"(lastName LIKE[c] 'Worsley') AND (salary > %@)",
minimumSalary];

[request setPredicate:predicate];


NSSortDescriptor *sortDescriptor = [[NSSortDescriptor alloc] initWithKey:@"firstName" ascending:YES];

[request setSortDescriptors:@[sortDescriptor]];


NSError *error;

NSArray *array = [moc executeFetchRequest:request error:&error];

if (array == nil)
  {
    // Deal with error...
  }
```

Fetch Results Controller

You use a fetched results controller to efficiently manage the results returned from a Core Data fetch request to provide data for a UITableView object.

- Offers change tracking
- Can cache results

```
 fetchRequest = [NSFetchRequest fetchRequestWithEntityName:@"Employee"];
  sortDescriptor = [[NSSortDescriptor alloc] initWithKey:@"lastName" ascending:YES];
  [fetchRequest setSortDescriptors:@[sortDescriptor]];

  controller = [[NSFetchedResultsController alloc] initWithFetchRequest:fetchRequest
                                                    managedObjectContext:_managedObjectContext
                                                      sectionNameKeyPath:@"department.name"
                                                               cacheName:nil];

[controller performFetch:&error]
```

Blocks

Implementation

```
^ {
   NSLog(@“This is a block”);
   }
```

Invocation

```
simpleBlock();
```

Definition

```
void (^simpleBlock)(void);
double (^additionBlock)(double, double)
```

Blocks can capture variables from the enclosing scope

```
simpleBlock = ^
   {
   NSLog(@“This is a block”);
   };
```

```
NSString* name = @“Bob”
simpleBlock = ^
   {
   NSLog(@“Hello %@“, name);
   };
```

AFNetworking

“AFNetworking is a delightful networking library for iOS and Mac OS X. It's built on top of the Foundation URL Loading System, extending the powerful high-level networking abstractions built into Cocoa. It has a modular architecture with well-designed, feature-rich APIs that are a joy to use.”

https://github.com/AFNetworking/AFNetworking.git

https://github.com/AFNetworking/AFNetworking

Homework

1) Create a sorting button for the employees tab that sorts the employees by position.
2) Create a positions tab
3) Make the Departments table cells selectable such that they navigate you to a new page with information about that department. (also Listing its employees)
4) Add the ability to add employees from the employees tab

References

- Xcode Concepts, https://developer.apple.com/library/ios/featuredarticles/XcodeConcepts/Concept-Targets.html
- View Controller Programming Guide, https://developer.apple.com/library/ios/featuredarticles/ViewControllerPGforiPhoneOS/AboutViewControllers/AboutViewControllers.html
- Launch Arguments and Environment Variables, http://nshipster.com/launch-arguments-and-environment-variables/
- Programming with Objective-C, https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/ErrorHandling/ErrorHandling.html
- Autolayout Guide, https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/AutolayoutPG/VisualFormatLanguage/VisualFormatLanguage.html
- Size Classes Design Help, https://developer.apple.com/library/ios/recipes/xcode_help-IB_adaptive_sizes/chapters/AboutAdaptiveSizeDesign.html
- Adaptive Layout Tutorial: Getting Started, http://www.raywenderlich.com/83276/beginning-adaptive-layout-tutorial
- Core Data Programming Guide, https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/CoreData/Articles/cdBasics.html
- Predicate Programming Guide, https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/Predicates/AdditionalChapters/Introduction.html#//apple_ref/doc/uid/TP40001798-SW1
- View Controller Catalog for iOS, https://developer.apple.com/library/ios/documentation/WindowsViews/Conceptual/ViewControllerCatalog/Chapters/NavigationControllers.html#//apple_ref/doc/uid/TP40011313-CH2