Praktikum Entwicklung von Mediensystemen mit iOS

WS 2011

Prof. Dr. Michael Rohs
michael.rohs@ifi.lmu.de
MHCI Lab, LMU München

Today

- Storyboards
- Automatic Reference Counting
- Animations

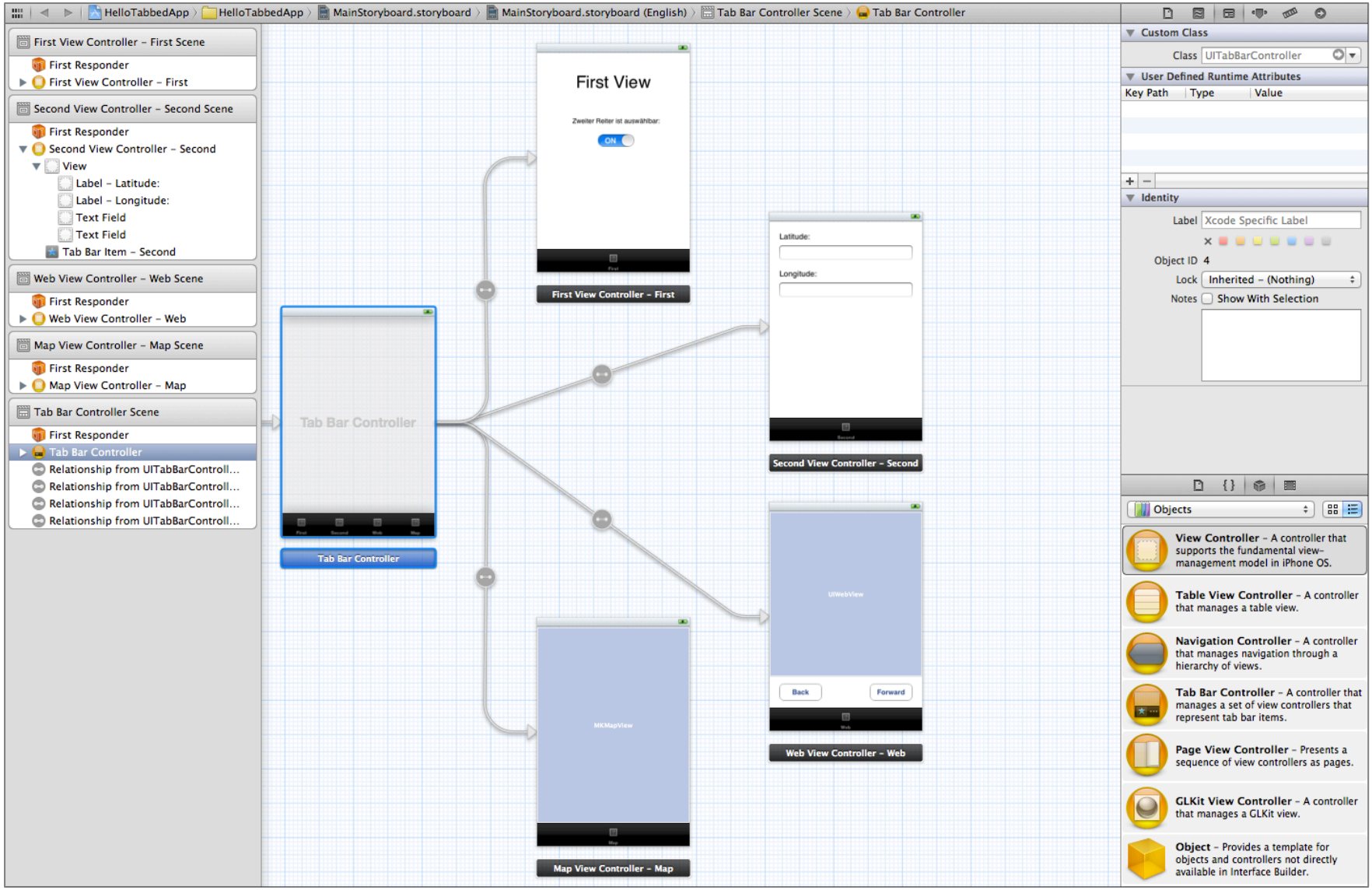
- Exercise 3

Timeline

#	Date	Topic
1	19.10.2011	Introduction and overview of iOS
2	26.10.2011	App architecture, touch input, saving data
3	2.11.2011	Location, networking, sensors
4	9.11.2011	iOS 5, storyboards, automatic reference counting
5	16.11.2011	Interviews, storyboarding; brainstorming
6	30.11.2011	Paper prototyping test, start of software prototype
7	14.12.2011	Heuristic evaluation of software prototype
8	11.1.2012	Think-aloud user study
9	25.1.2012	Completion of software prototype
10	1.2.2012	Final presentation

STORYBOARDS

Storyboards = Scenes + Segues

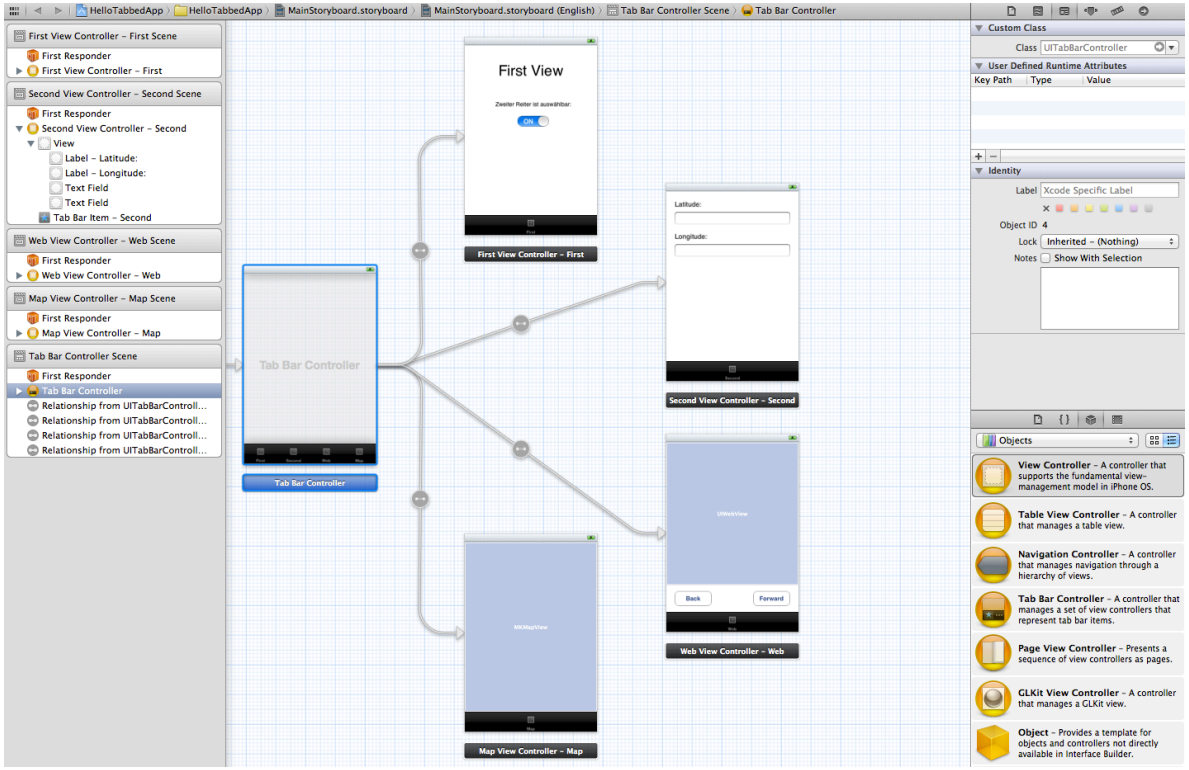


Storyboards = Scenes + Segues

- Scene: A single screen of content
- Segue: Transition between scenes
- One storyboard per project, contains all scenes

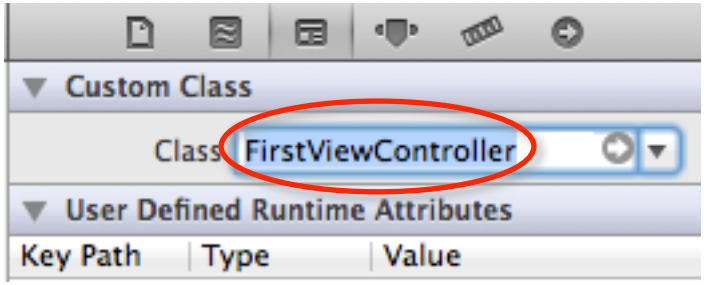
- Zoom in-out: double-click background
- Zoom in to edit scene

- class: UIStoryboard

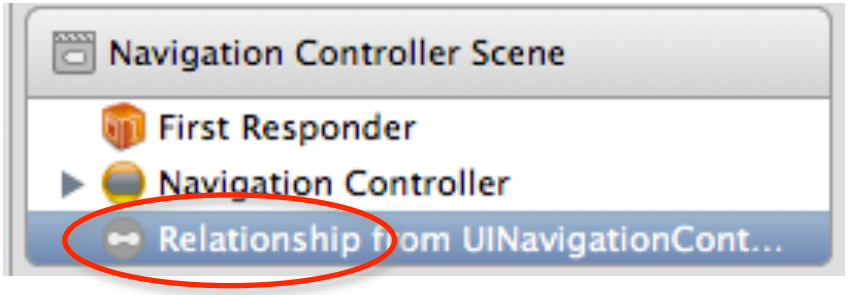


Storyboards = Scenes + Segues

- Scene: A single screen
  - UIViewController subclass
  - Create subclass: File | New File… | UIViewController subclass
  - Set new subclass to scene



- Segue: Transition between scenes
  - UIStoryboardSegue: transitions are objects, too!
    - Performs the visual transition between two view controllers
  - Types: Push, Modal, Custom
  - Relationships link containers (Tab Bar Controller, Navigation Controller) to content views



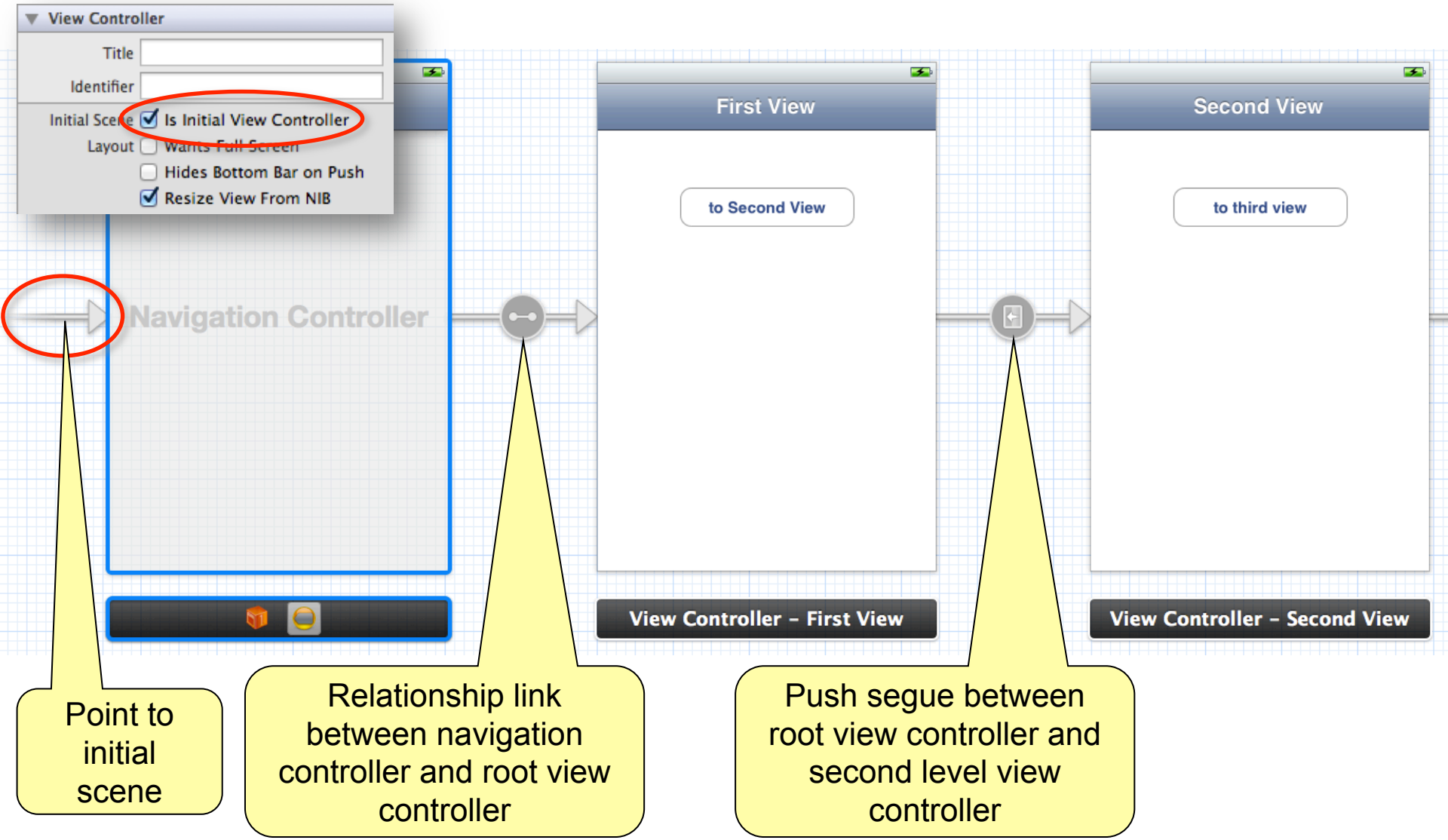
No More MainWindow.xib

- Can still configure projects to use xib-files in iOS 5
  - Adding items from library, IBOutlets, IBActions have not changed
- If using storyboards, then …-Info.plist shows:

Bundle version	String	1.0
Application requires iPhone environmer	Boolean	YES
▶ Required device capabilities	Array	(1 item)
▶ Supported interface orientations	Array	(3 items)
Main storyboard file base name	String	Storyboard

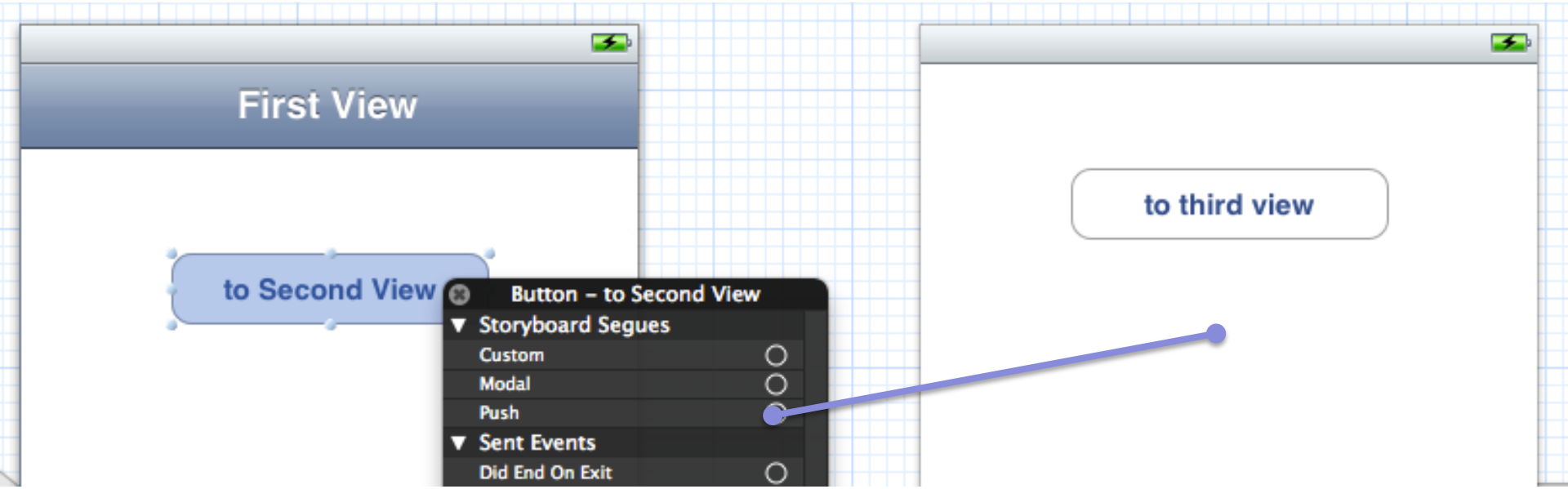
- Automatically instantiates “initial” view controller:

Scenes and Segues



Creating Segues

- Ctrl-click element that invokes second scene (e.g. button)
- Or right-click this element

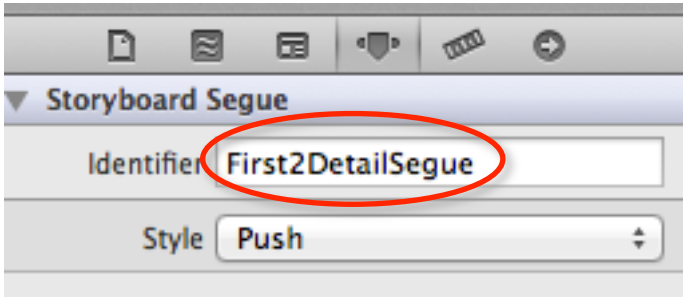


Pass Data Between Scenes (Up)

- Implement method prepareForSegue:sender: in source view controller

```
- (void) prepareForSegue:(UIStoryboardSegue*)segue sender:(id)sender {
    if ([[segue identifier] isEqualToString:@"First2DetailSegue"]) {
        DetailViewController *dvc = (DetailViewController*) [
                                        segue destinationViewController];
        dvc.data = data;
    }
}
```

here the data is passed to the detail controller

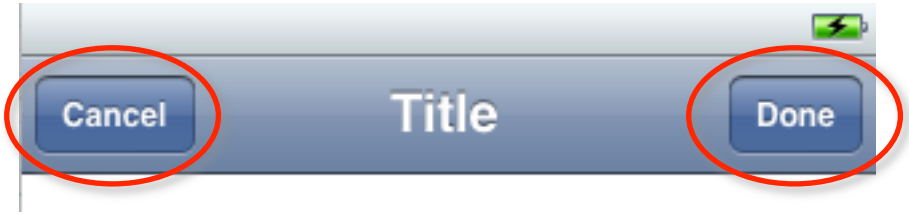


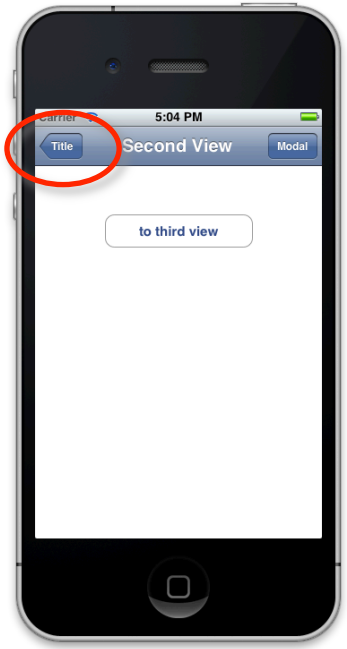
- Set segue identifier in storyboard
- Subclass UIStoryboardSegue for custom transitions
  - Specify as custom in storyboard
  - Override “perform” method

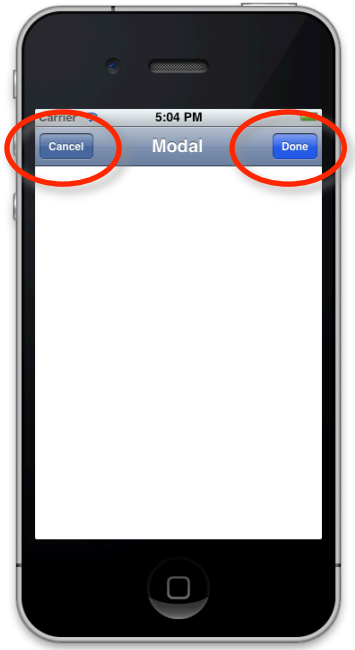
Pass Data Between Scenes (Back)

- For push segues 
  - Update data structure set in prepareForSegue
  - Back button automatically pops view controller

- For modal segues 
  - Use delegate object that processes done/cancel and dismisses modal view controller







Pass Data Back from Modal View

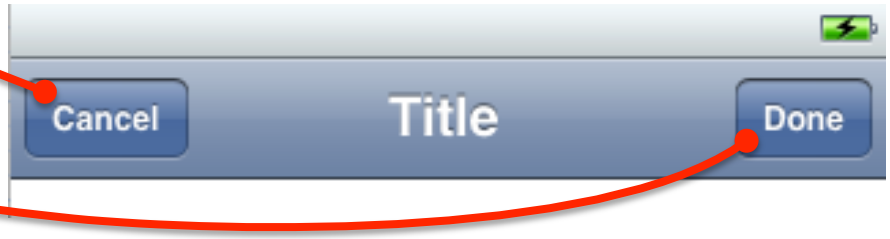
- Modal view defines delegate protocol

```
@class MyModalViewController; // forward declaration

@protocol MyModalViewControllerDelegate
- (void) myModalViewControllerDidCancel:(MyModalViewController*)controller;
- (void) myModalViewControllerDidSave:(MyModalViewController*)controller;
@end

@interface MyModalViewController : UIViewController
@property (nonatomic, strong) id <MyModalViewControllerDelegate> delegate;
- (IBAction)cancel:(id)sender;
- (IBAction)done:(id)sender;
@end
```

tells the compiler that My…Controller is a class (used before declared)

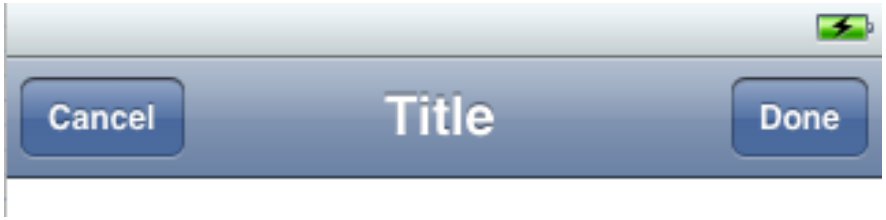


Pass Data Back from Modal View

- MyModalViewController.m, call the delegate

```
- (IBAction)cancel:(id)sender {
    [self.delegate myModalViewControllerDidCancel:self];
}

- (IBAction)done:(id)sender {
    [self.delegate myModalViewControllerDidSave:self];
}
```



Pass Data Back from Modal View

- FirstViewController.h

```
#import "MyModalViewController.h”
@interface FirstViewController : UIViewController
                                 <MyModalViewControllerDelegate>
@end
```

starting view controller implements delegate protocol

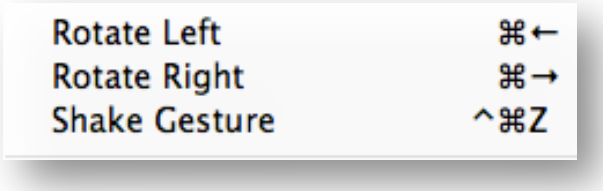
- FirstViewController.m

set segue identifier in storyboard

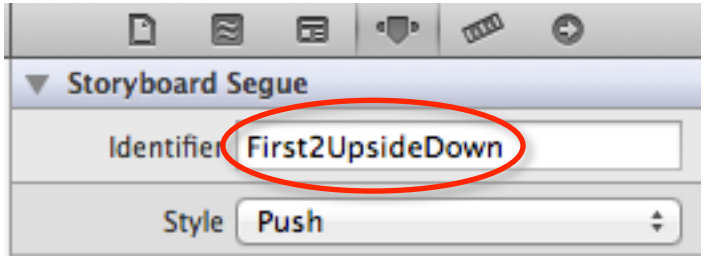
```
- (void) prepareForSegue:(UIStoryboardSegue*)segue sender:(id)sender {
    if ([[segue identifier] isEqualToString:@”First2Modal"]) {
        MyModalViewController *mvc = [segue destinationViewController];
        mvc.delegate = self;
} }
- (void) myModalViewControllerDidCancel:(MyModalViewController*)controller
{   [controller dismissModalViewControllerAnimated:YES];   }
- (void) myModalViewControllerDidSave:(MyModalViewController*)controller
{   [controller dismissModalViewControllerAnimated:YES];   }
```

Programmatically trigger Segues

- performSegueWithIdentifier
- Example: Show a scene when device upside down
  - Rotate simulator: ⌘←, ⌘→



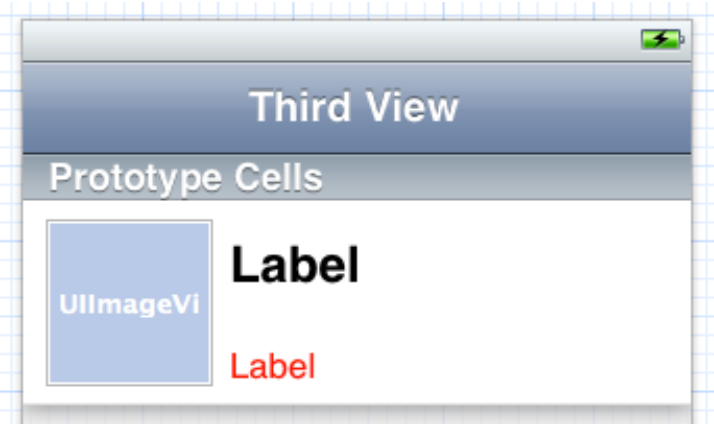
- In UIViewController subclass



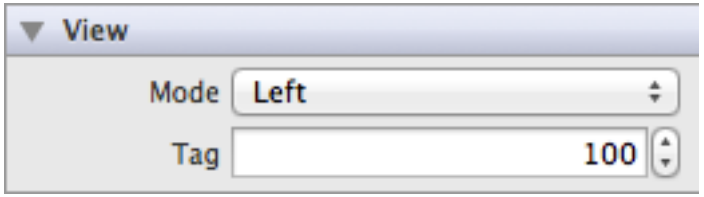

```
- (BOOL)shouldAutorotateToInterfaceOrientation:
            (UIInterfaceOrientation)interfaceOrientation
{
    if (interfaceOrientation == UIInterfaceOrientationPortraitUpsideDown) {
        [self performSegueWithIdentifier:@"First2UpsideDown" sender:self];
    }
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}
```

Prototype Cells in Tables

- Prototype cells define the layout of table cells
- Set cell identifier in storyboard
- Use in cellForRowAtIndexPath

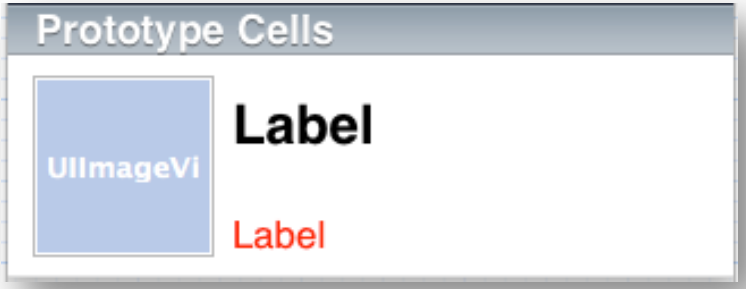



```
- (UITableViewCell *)tableView:(UITableView *)tableView
                 cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    static NSString *CellIdentifier = @"MyCustomCell";
    UITableViewCell *cell = [tableView
                             dequeueReusableCellWithIdentifier:CellIdentifier];
    UILabel *label = (UILabel*) [cell viewWithTag:100];
    label.text = [data objectAtIndex:indexPath.row];
    return cell;
}
```



- Or: sublass UITableViewCell

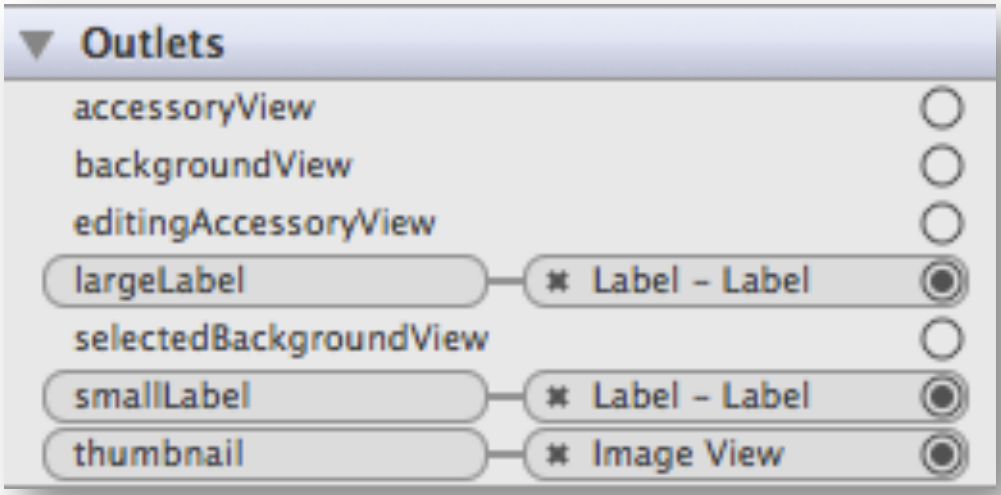
Custom TableViewCell



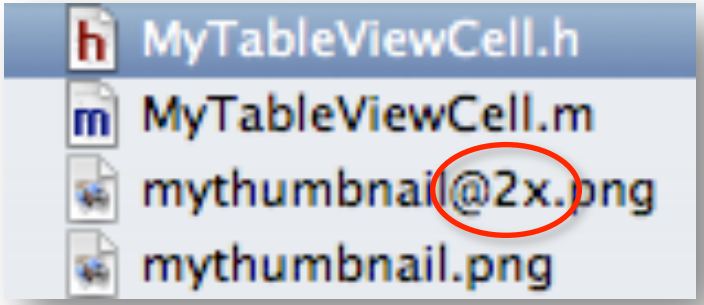
- MyTableViewCell.h

```
@interface MyTableViewCell : UITableViewCell
@property (nonatomic, strong) IBOutlet UILabel* largeLabel;
@property (nonatomic, strong) IBOutlet UILabel* smallLabel;
@property (nonatomic, strong) IBOutlet UIImageView* thumbnail;
@end
```

- Select prototype cell
- Connect objects to outlets



Custom TableViewCell

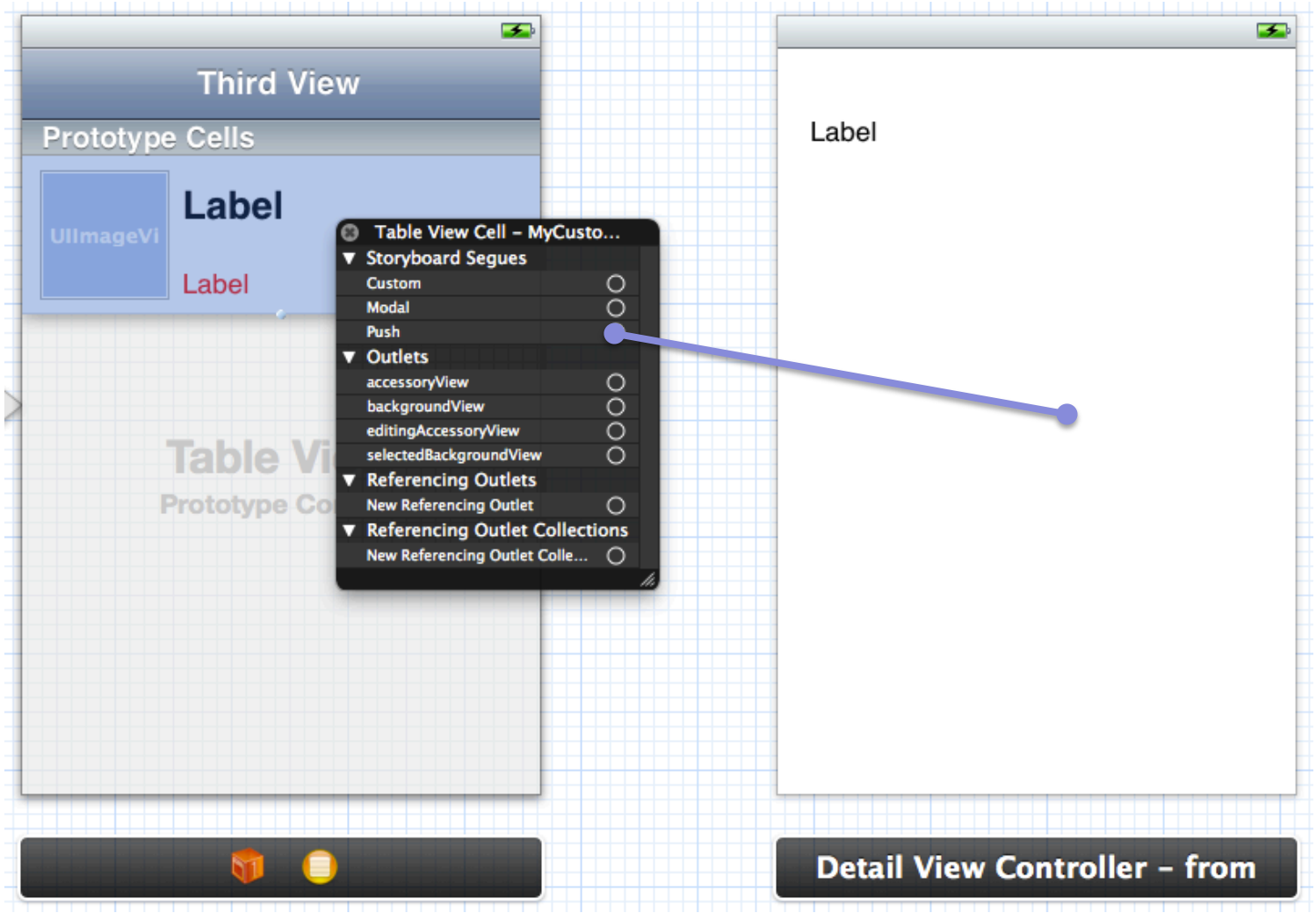



```
@interface MyTableViewCell : UITableViewCell
@property (nonatomic, strong) IBOutlet UILabel* largeLabel;
@property (nonatomic, strong) IBOutlet UILabel* smallLabel;
@property (nonatomic, strong) IBOutlet UIImageView* thumbnail;
@end
```

in MyTableViewController:

```
- (UITableViewCell *)tableView:(UITableView *)tableView
                 cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    MyTableViewCell *cell = [tableView
                    dequeueReusableCellWithIdentifier:@"MyCustomCell"];
    cell.largeLabel.text = [data objectAtIndex:indexPath.row];
    cell.smallLabel.text = @"this is a small label";
    cell.thumbnail.image = [UIImage imageNamed:@"mythumbnail"];
    return cell;
}
```

Link Prototype Cells to View Controllers



Passing Data from Table to Detail View

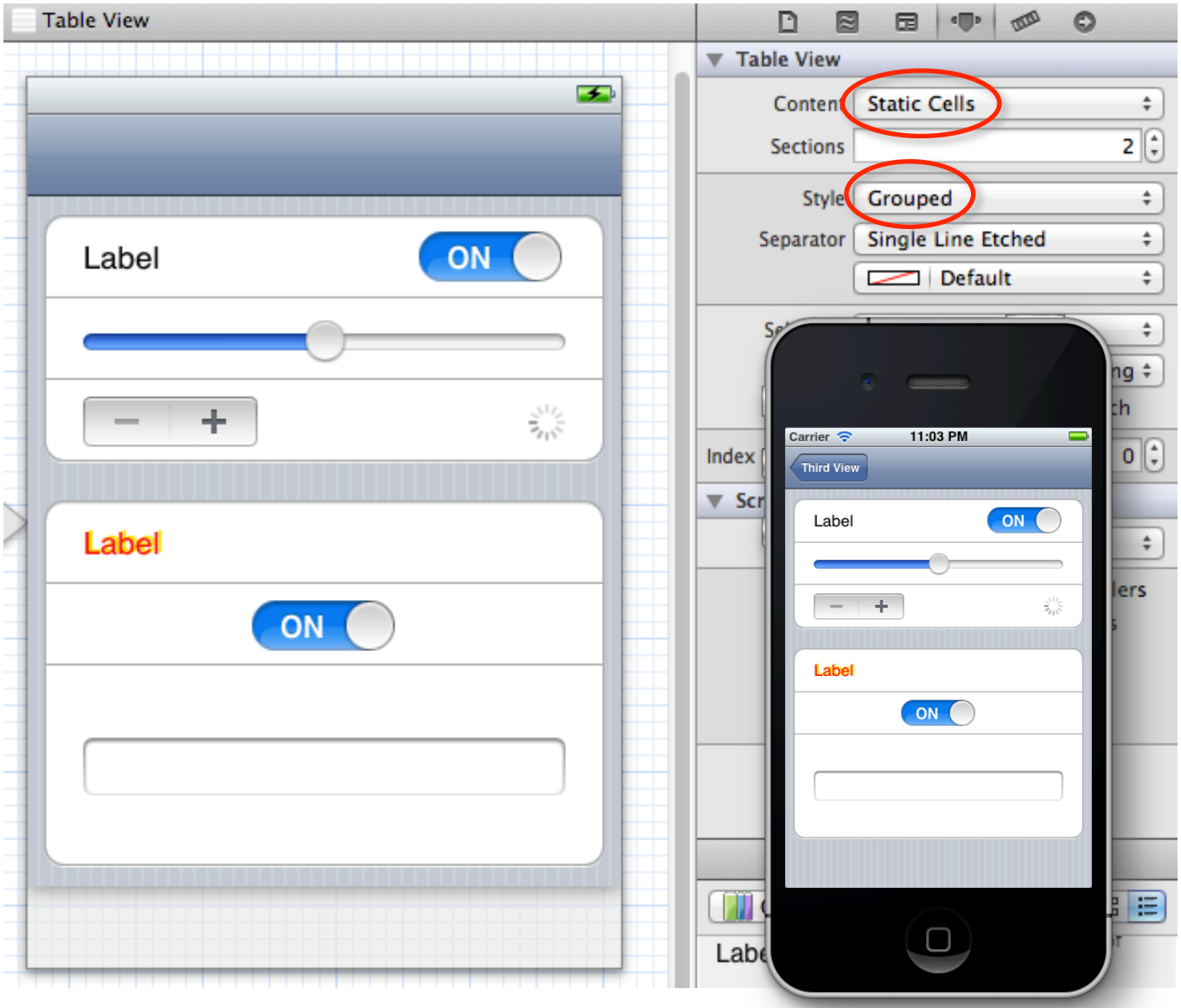
in MyTableViewController:

```
- (void) prepareForSegue:(UIStoryboardSegue*)segue sender:(id)sender
{
    if ([[segue identifier] isEqualToString:@"Third2Detail"]) {
        DetailViewController *dvc = (DetailViewController*)
                                            [segue destinationViewController];
        UITableViewCell *cell = sender;
        UILabel *label = (UILabel*) [cell viewWithTag:100];
        dvc.data = label.text;
    }
}
```

here the data is passed to the detail controller

Static Table Cells

- Fixed cell content
- Appears on device as is
- Nice for grouped tables (edit views)
- Hook up to controllers via outlets



AUTOMATIC REFERENCE COUNTING (ARC)

Automatic Reference Counting (ARC)

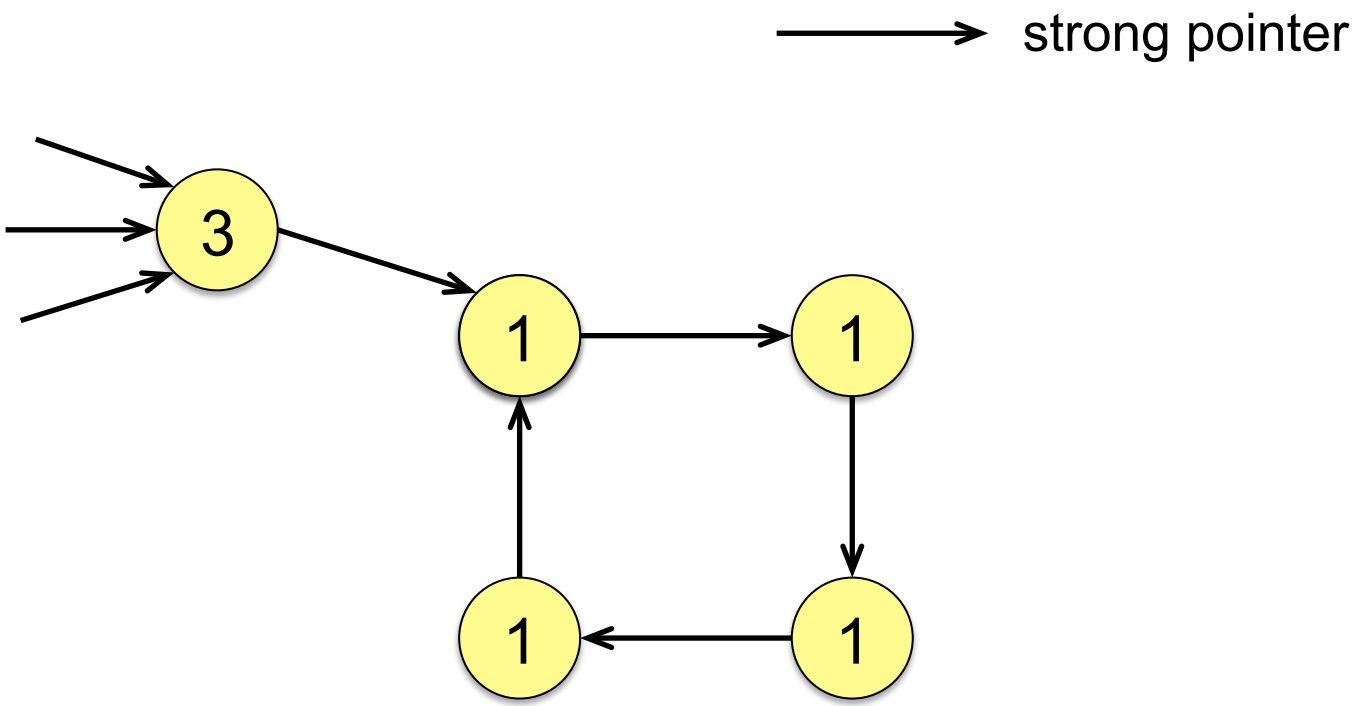
- Automates contain and release calls
  - Writing code without thinking about retain/release! ☺
  - Still uses reference counting internally
  - Retain, release, etc. not allowed; dealloc rarely necessary
- Objective-C language extensions
  - Automatic retain/release on entry/exit of scopes
  - Compiler knows about naming conventions (alloc, new, copy, …)
  - @autoreleasepool { … }
- ARC is a compile-time mechanism
  - Not a new runtime memory model
  - Not a garbage collector
  - Does not cover malloc/free, core foundation

Strong and Weak Pointers

- Strong pointers
  - Strong pointers keep objects alive
  - Strong pointers are like “retain” properties (+1 ref. count)
  - Default for all variables (instance variables, local variables, etc.)
  - Keyword: __strong
  - Example: __strong NSString *name;
- Weak pointers
  - Weak pointers do not keep objects alive
  - Weak pointers are like “assign” properties (+0 ref. count)
  - Weak pointers get nil when object is deallocated
  - Keyword: __weak
  - Example: __weak NSString *name;

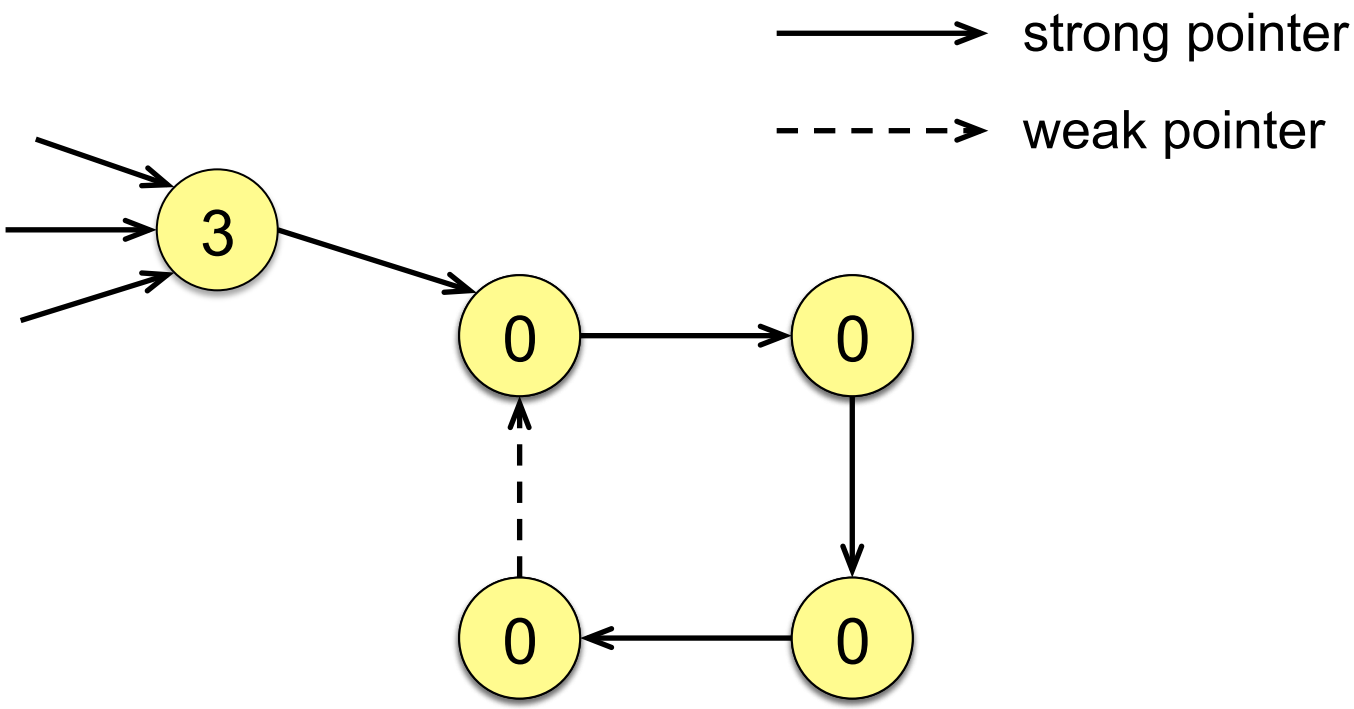
Cycles in Object Graphs

- Problem: Strong pointers keep objects in cycle alive
  - Memory leak!



Cycles in Object Graphs

- Problem: Strong pointers keep objects in cycle alive
- Solution: Weak pointers do not keep objects alive
  - A weak pointer gets nil when object it points to get deallocated



ARC Variable Modifiers

- __strong
  Strong pointer, the default
  Initialized to nil: NSString *name; → NSString *name = nil;
- __weak
  Weak pointer, initialized to nil, set to nil when object deallocated
- __unsafe_unretained
  Traditional variable, unretained, not set to nil
  Sometimes needed for non-Objective-C code
- __autoreleasing
  Out-parameters, not for general use

Writing a __strong Variable

- Code you write

```
name = newName
```

- What the compiler adds

```
[newName retain];
NSString *oldName = name;
name = newName;
[oldName release];
```

Scoping of __strong Variables

- Code you write

```
if (a < 10) {
    NSString *name =
                [[NSString alloc] init…];
    // using name…

}
```

- What the compiler adds

```
if (a < 10) {
    NSString *name =
                [[NSString alloc] init…];
    // using name…
    [name release];
}
```

ARC Deallocation Examples

- Automatic deallocation when variable goes out of scope
  - method exit
  - if-cause ends
  - etc.
- Object graph deallocation
  - Referenced object deallocated when root goes out of scope

Dealloc Object

```
- (void) myCallerMethod {
    [self myMethod];
    NSLog(@"after myMethod");
}

- (void) myMethod {
    MyObject *o = [[MyObject alloc] init];
    NSLog(@"%@", [o description]);
    NSLog(@"myMethod exit");
}
```

```
@interface MyObject : NSObject
@end

@implementation MyObject
- (void)dealloc {
    NSLog(@"MyObject::dealloc");
}
@end
```

Output:

<MyObject: 0x685b3f0>
myMethod exit
MyObject::dealloc
after myMethod

Dealloc Object after If-Clause

```
- (void) myCallerMethod {
    [self myMethod:TRUE];
    NSLog(@"after myMethod");
}

- (void) myMethod:(BOOL)condition {
    if (condition) {
        MyObject *o = [[MyObject alloc] init];
        NSLog(@"%@", [o description]);
    }
    NSLog(@"myMethod exit");
}
```

```
@interface MyObject : NSObject
@end

@implementation MyObject
- (void)dealloc {
    NSLog(@"MyObject::dealloc");
}
@end
```

Output:

<MyObject: 0x8844fd0>
MyObject::dealloc
myMethod exit
after myMethod

Dealloc Object in Array

```
- (void) myCallerMethod {
    [self myMethod];
    NSLog(@"after myMethod");
}

- (void) myMethod {
    MyObject *o = [[MyObject alloc] init];
    NSArray *a = [[NSArray alloc]
                  initWithObjects:o, nil];
    NSLog(@"%@", [a description]);
    NSLog(@"%@", [o description]);
    NSLog(@"myMethod exit");
}
```

```
@interface MyObject : NSObject
@end

@implementation MyObject
- (void)dealloc {
    NSLog(@"MyObject::dealloc");
}
@end
```

Output:

("<MyObject: 0x6831990>")
<MyObject: 0x685b3f0>
myMethod exit
MyObject::dealloc
after myMethod

…after If-Clause in Array

```
- (void) myCallerMethod {
    [self myMethod:TRUE];
    NSLog(@"after myMethod");
}

- (void) myMethod:(BOOL)condition {
    if (condition) {
        MyObject *o = [[MyObject alloc] init];
        NSArray *a = [[NSArray alloc]
                        initWithObjects:o, nil];
        NSLog(@"%@", [a description]);
    }
    NSLog(@"myMethod exit");
}
```

```
@interface MyObject : NSObject
@end

@implementation MyObject
- (void)dealloc {
    NSLog(@"MyObject::dealloc");
}
@end
```

Output:

```
( "<MyObject: 0x68748d0>" )
MyObject::dealloc
myMethod exit
after myMethod
viewDidLoad
```

Normal and Retained Returns

- ARC knows method naming conventions
  - first part of name (capitalization subdivides name parts)
- Transfer of ownership if first name part
  - alloc, init, copy, mutableCopy, new
  - returned objects are not autoreleased
  - “retained returns”
- Otherwise no transfer of ownership
  - returned objects are autoreleased
  - “normal returns”
  - @autoreleasepool { … } determines when autoreleased objects are deallocated

Normal (Autoreleased) Returns

- Code you write

```
- (NSString*) name {
    return myName;
}
```

- What the compiler adds

```
- (NSString*) name {
    return [[myName retain] autorelease];
}
```

Retained (Non-Autoreleased) Returns

- Code you write

```
- (NSString*) newName {
    return myName;
}
```

- What the compiler adds

```
- (NSString*) newName {
    return [myName retain];
}
```

Method name starts with “new” → transfer of ownership

Output of these programs?

```
@autoreleasepool {
    [self myMethod];
    NSLog(@"after myMethod");
}

- (MyObject*) myMethod {
    MyObject *s = [[MyObject alloc] init];
    NSLog(@"myMethod exit");
    return s;
}
```

autorelease pool emptied here

```
myMethod exit
after myMethod
MyObject::dealloc
```

```
@autoreleasepool {
    [self newMethod];
    NSLog(@"after newMethod");
}

- (MyObject*) newMethod {
    MyObject *s = [[MyObject alloc] init];
    NSLog(@"newMethod exit");
    return s;
}
```

```
newMethod exit
MyObject::dealloc
after newMethod
```

ARC Autoreleased Array

```
@interface MyObject : NSObject
@end
```

```
- (void) myCallerMethod {
    @autoreleasepool {
        [self myMethod];
        NSLog(@"after myMethod");
    }
}
- (void) myMethod {
    MyObject *o = [[MyObject alloc] init];
    NSArray *a = [NSArray arrayWithObject:o];
    NSLog(@"%@", [a description]);
    NSLog(@"%@", [o description]);
    NSLog(@"myMethod exit");
}
```

autorelease pool emptied here

```
@implementation MyObject
- (void)dealloc {
    NSLog(@"MyObject::dealloc");
}
@end
```

Output:

```
( "<MyObject: 0x6d3f5d0>" )
<MyObject: 0x6d3f5d0>
myMethod exit
after myMethod
MyObject::dealloc
```

Example: Weak Pointers

- Output of this program?

```
__weak MyObject *o = [[MyObject alloc] init];
NSLog(@"MyObject is: %@", [o description]);
```

- Output:

```
MyObject::dealloc
MyObject is: (null)
```

- Why?
  - Weak pointer does not keep object alive

```
@interface MyObject : NSObject
@end

@implementation MyObject
- (void)dealloc {
    NSLog(@"MyObject::dealloc");
}
@end
```

ANIMATIONS

UIView Class

- Rectangular area on the screen
  - Renders content in that area
  - Handles interactions in that area
- Base class of anything visible on the screen
  - UILabel, UIButton, UIImageView, UITableView, etc.
- A view contains zero or more subviews
- Can be subclassed for custom views
  - Drawing: drawRect method, UIGraphicsGetCurrentContext
  - Trigger redrawing: setNeedsDisplay method
  - Touch events (implements UIResponder)
- Call UIView methods from the main thread only

UIView Animatable Properties

- Some properties can be gradually changed over time, i.e. animated
  - @property frame
  - @property center
  - @property transform
  - @property alpha
  - @property backgroundColor
  - (@property bounds)
  - (@property contentStretch)

UIView Animation with Blocks

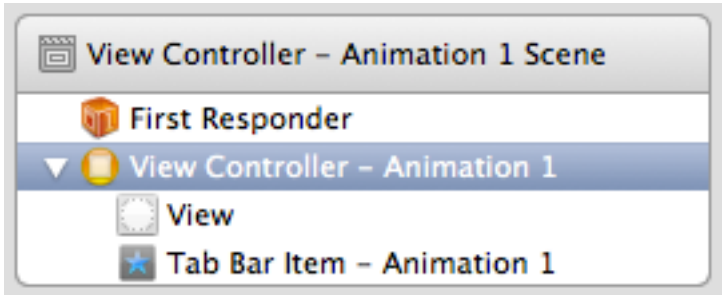
- Animate properties of one or more views over time
  - Set properties to initial values (before animation starts)
  - Set desired final values of properties
  - Optionally: Set acceleration/deceleration
  - Optional: Set operation to perform when animation is done
- Scheme:

```
view.property1 = <initial value>;
view.property2 = <initial value>;
[UIView animateWithDuration:<duration in sec>
        animations:^{
                view.property1 = <final value>;
                view.property2 = <final value>;
        }
];
```

Add Image View as Subview

- Add image view as subview when view did load
  - viewDidLoad is a method of your UIViewController subclass
  - any UIView object has zero or more subviews

```
- (void)viewDidLoad {
    [super viewDidLoad];
    UIImage *image = [UIImage imageNamed:@"testimage"];
    UIImageView *iv = [[UIImageView alloc]
                       initWithImage:image];
    [self.view addSubview:iv];
}
```



From Transparent to Opaque

- Animate view when view controller becomes active
  - Change alpha from 0 (transparent) to 1 (opaque) in 3 sec

```
- (void)viewWillAppear:(BOOL)animated {
    [super viewWillAppear:animated];
    UIImageView *iv = [[self.view subviews] objectAtIndex:0];
    iv.alpha = 0.0;
    [UIView animateWithDuration:3.0
            animations:^{
                    iv.alpha = 1.0;
            }
    ];
}
```

View Animations with Options

- Animate alpha from 0 to 1 in 1 sec; accelerate/decelerate, reverse the animation; repeat forever

```
UIImageView *iv = [[self.view subviews] objectAtIndex:0];
iv.alpha = 0.0;
[UIView animateWithDuration:1.0
        delay:0.0
        options:UIViewAnimationOptionRepeat |
                UIViewAnimationOptionCurveEaseInOut |
                UIViewAnimationOptionAutoreverse
        animations:^{
                iv.alpha = 1.0;
        }
        completion:nil];
```

Simultaneous Animation of Two Properties

- Animate multiple properties simultaneously
  - Change alpha and move center from left/top to display center in 3s

```
- (void)viewWillAppear:(BOOL)animated {
    [super viewWillAppear:animated];
    UIImageView *iv = [[self.view subviews] objectAtIndex:0];
    iv.alpha = 0.0;
    iv.center = CGPointMake(0, 0);
    CGRect frame = self.view.frame;
    [UIView animateWithDuration:3
            animations:^{
                    iv.alpha = 1.0;
                    iv.center = CGPointMake(frame.size.width / 2, frame.size.height / 2);
            }
    ];
}
```

UIView frame

- CGRect **frame**: the view's location and size in the coordinate system of its superview
  - struct CGRect { CGPoint origin; CGSize size; };
  - struct CGPoint { CGFloat x; CGFloat y; };
  - struct CGSize { CGFloat width; CGFloat height; };
  - typedef float CGFloat;
- **frame** is invalid if **transform** is not the identity transform
- Coordinates are specified in "points"
  - iPhone and iPod touch: 320 x 480 points
  - iPad: 768 x 1024 points
  - "Old" iPhones: 1 point = 1 pixel
  - "New" iPhones 1 point = 2 pixels

UIView center

- CGPoint **center**: the view's center in the coordinate system of its superview
  - struct CGPoint { CGFloat x; CGFloat y; };
- Changing center updates frame.origin
- Coordinates are specified in "points"

UIView Transforms

- view.transform property
  - Describes view's affine transformation

- Identity: CGAffineTransformIdentity
- Translation: CGAffineTransformMakeTranslation(tx, ty)
- Rotation: CGAffineTransformMakeRotation(angle)
- Scaling: CGAffineTransformMakeScale(sx, sy)

- Concatenation with current view.transform
  - CGAffineTransformRotate(CGAffineTransform t, angle)
  - CGAffineTransformScale(CGAffineTransform t, sx, sy)
  - CGAffineTransformTranslate(CGAffineTransform t, tx, ty)

UIView Transforms in Animations

- Animate multiple properties simultaneously
  - Change alpha, scale from 10% to 100%, and rotate by 90° in 3 sec

```
- (void)viewWillAppear:(BOOL)animated {
    [super viewWillAppear:animated];
    UIImageView *iv = [[self.view subviews] objectAtIndex:0];
    iv.alpha = 0.0;
    iv.transform = CGAffineTransformMakeScale(0.1, 0.1);
    [UIView animateWithDuration:3
        animations:^{
            iv.alpha = 1.0;
            iv.transform = CGAffineTransformMakeRotation(M_PI_2);
        }
    ];
}
```

Two Consecutive Animations

- Rotate 90° (3 sec), then rotate back and scale to 50%

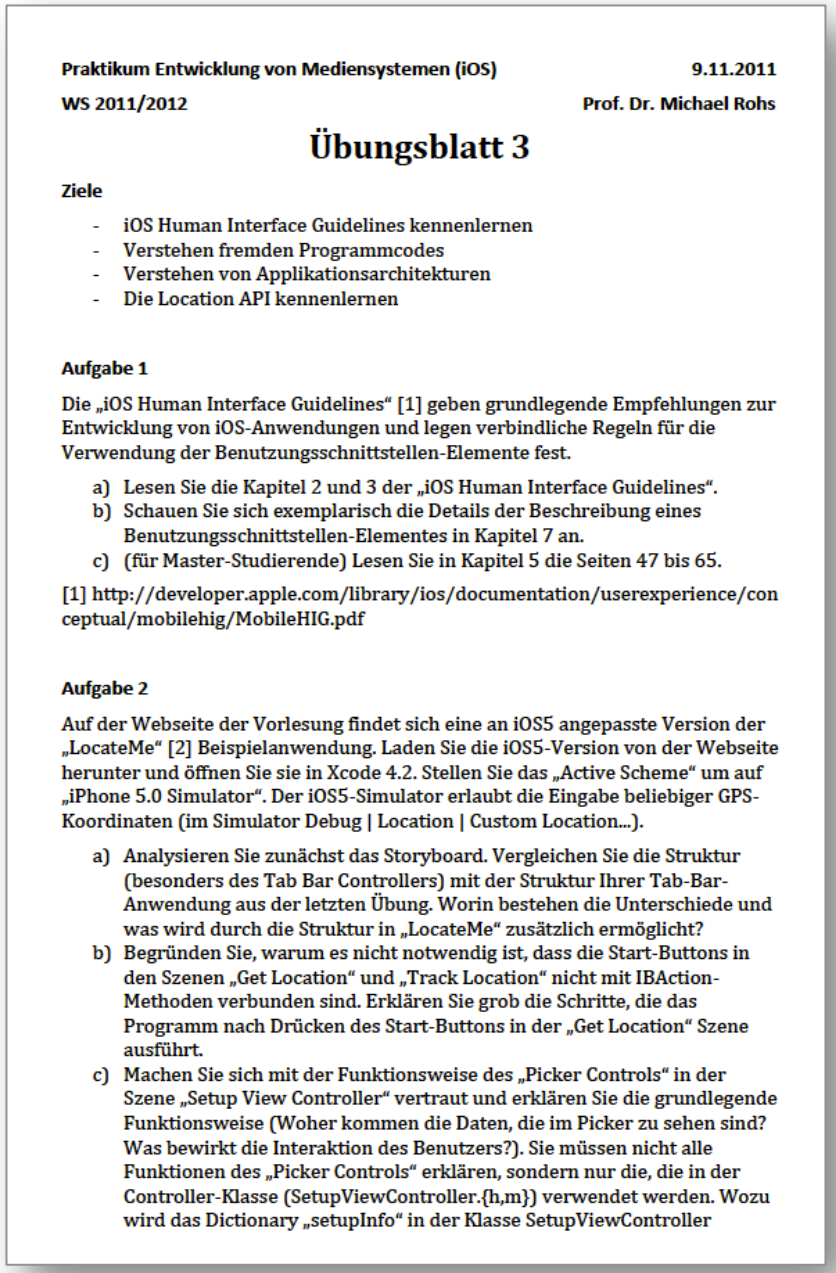
```
- (void)viewWillAppear:(BOOL)animated {
    [super viewWillAppear:animated];
    UIImageView *iv = [[self.view subviews] objectAtIndex:0];
    iv.transform = CGAffineTransformIdentity;
    [UIView animateWithDuration:3 delay:0
        options:UIViewAnimationOptionCurveEaseInOut
        animations:^{
            iv.transform = CGAffineTransformMakeRotation(M_PI_2); }
        completion:^(BOOL finished) {
            [UIView animateWithDuration:3
                animations:^{
                    iv.transform = CGAffineTransformMakeScale(0.5, 0.5); }
            ];
    }];
}
```

Other New Featuers in iOS5

- Storyboards ✔
- Automati Reference Counting ✔
- iCloud
- Twitter integration
- UIKit additions / changes
- Newsstand
- Notification center
- Location simulation
- and more…

Exercise 3

- Ziele
  - iOS Human Interface Guidelines kennenlernen
  - Verstehen fremden Programmcodes
  - Verstehen von Applikationsarchitekturen
  - Die Location API kennenlernen
- LocateMe für iOS5
  - Storyboards
  - ARC



Praktikum Entwicklung von Mediensystemen (iOS) 9.11.2011

WS 2011/2012 Prof. Dr. Michael Rohs

Übungsblatt 3

Ziele

- iOS Human Interface Guidelines kennenlernen
- Verstehen fremden Programmcodes
- Verstehen von Applikationsarchitekturen
- Die Location API kennenlernen

Aufgabe 1

Die „iOS Human Interface Guidelines" [1] geben grundlegende Empfehlungen zur Entwicklung von iOS-Anwendungen und legen verbindliche Regeln für die Verwendung der Benutzungsschnittstellen-Elemente fest.

a) Lesen Sie die Kapitel 2 und 3 der „iOS Human Interface Guidelines".
b) Schauen Sie sich exemplarisch die Details der Beschreibung eines Benutzungsschnittstellen-Elementes in Kapitel 7 an.
c) (für Master-Studierende) Lesen Sie in Kapitel 5 die Seiten 47 bis 65.

[1] http://developer.apple.com/library/ios/documentation/userexperience/conceptual/mobilehig/MobileHIG.pdf

Aufgabe 2

Auf der Webseite der Vorlesung findet sich eine an iOS5 angepasste Version der „LocateMe" [2] Beispielanwendung. Laden Sie die iOS5-Version von der Webseite herunter und öffnen Sie sie in Xcode 4.2. Stellen Sie das „Active Scheme" um auf „iPhone 5.0 Simulator". Der iOS5-Simulator erlaubt die Eingabe beliebiger GPS-Koordinaten (im Simulator Debug | Location | Custom Location...).

a) Analysieren Sie zunächst das Storyboard. Vergleichen Sie die Struktur (besonders des Tab Bar Controllers) mit der Struktur Ihrer Tab-Bar-Anwendung aus der letzten Übung. Worin bestehen die Unterschiede und was wird durch die Struktur in „LocateMe" zusätzlich ermöglicht?
b) Begründen Sie, warum es nicht notwendig ist, dass die Start-Buttons in den Szenen „Get Location" und „Track Location" nicht mit IBAction-Methoden verbunden sind. Erklären Sie grob die Schritte, die das Programm nach Drücken des Start-Buttons in der „Get Location" Szene ausführt.
c) Machen Sie sich mit der Funktionsweise des „Picker Controls" in der Szene „Setup View Controller" vertraut und erklären Sie die grundlegende Funktionsweise (Woher kommen die Daten, die im Picker zu sehen sind? Was bewirkt die Interaktion des Benutzers?). Sie müssen nicht alle Funktionen des „Picker Controls" erklären, sondern nur die, die in der Controller-Klasse (SetupViewController.{h,m}) verwendet werden. Wozu wird das Dictionary „setupInfo" in der Klasse SetupViewController