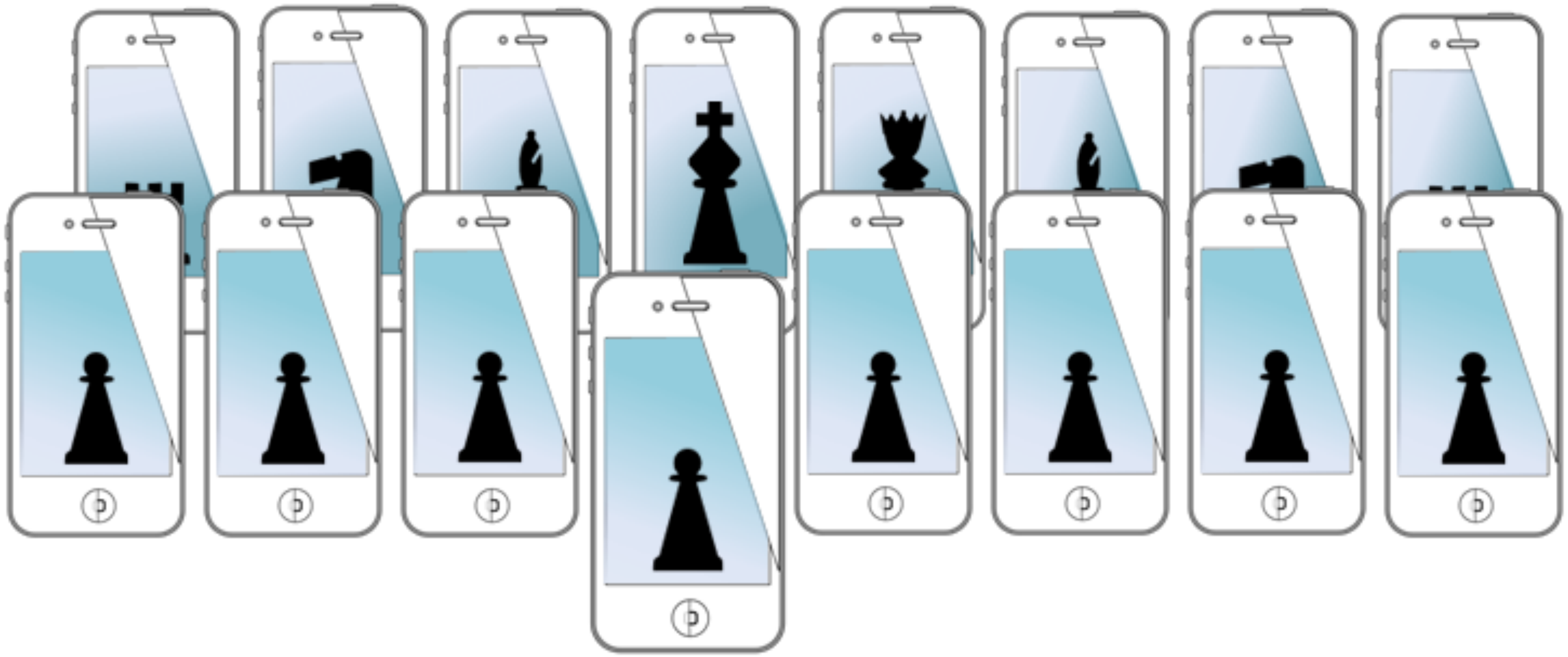
MOBILE SENSING & LEARNING



CSE5323 & 7323

Mobile Sensing & Learning

week one, lecture two: objective-C and !swift?

Eric C. Larson, Lyle School of Engineering,
Computer Science and Engineering, Southern Methodist University

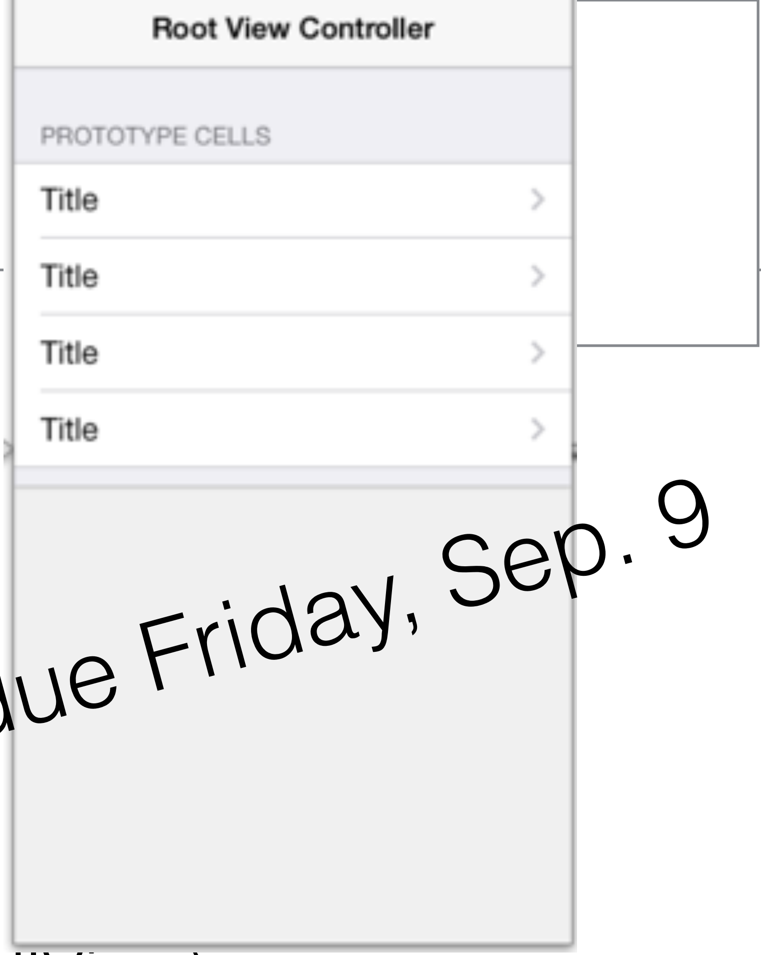
course logistics

- lab time: W 5-7PM
- teams: must be on a team by next lecture
- next class period will be flipped, so view video on canvas!
- get access to:
  - room? mac mini's? iPhones?
- university developer program…
  - or just use my current setup
  - do **NOT** let Xcode manage any certificates, etc.
  - setup an account at developer.apple.com (sample code, video)

assignment one

- You have free reign to create an application that manages some type of mutable information: you might display images from online somewhere, stock exchange information, information from twitter--or movies, or books, or amazon
- The data you load and display can come from anywhere and you can do whatever you want with it.
- must use the interface elements as described (**next slide**). You will need to get creative in order to incorporate ALL the design elements below.
- Create an iOS application in XCode that:
  - uses a **TableViewController** to load different views
  - must implement **three different types of cells and load them dynamically** (i.e., you cannot use a static table).
  - View navigation can be hierarchical in any way you want
  - When loading a new view controller your main view controller should hand off information to the controller that is getting created

assignment one



due Friday, Sep. 9

- Automatic Layout
- Buttons, Sliders, and Labels
- Stepper and Switch
- Picker (you must implement picker delegate)
- Segmented Control
- Timer (which should repeat and somehow update the UIView)
- ScrollView (with scrollable, zoomable content)
- Image View
- Navigation Controller
- Collection View Controller
- Table View Controller with dynamic prototype cells
- (An idea for exceptional Credit) Implement a modal view and handle properly using delegation or subclass view elements to make custom dynamics

agenda

a big syntax demo…

- **objective-c** and swift basics
  - class declaration
  - complex objects
  - common functions
  - encapsulation and primitives
  - memory management

and model view controllers for a breather in between!!!

objective c

- strict superset of c
- a lot like c
- but with “messages”
- so “functions” look funny (i.e., the braces in the logo)

objective c

classes

interface for class

class name

inherits from

property

```
@interface SomeClass  : NSObject

@property (strong, nonatomic) NSString *aString;

@end
```

if in the **.h** file,
it is public

```
@interface SomeClass ()

@property (strong, nonatomic) NSString *aString;

@end

@implementation SomeClass

   … implementation stuff…
@end
```

if in the **.m** file,
it is private

objective c

class properties

```
@interface SomeClass ()
{
    float aFloat;
}
@property (strong, nonatomic) NSString *aString;
@end

@implementation SomeClass
@synthesize aString = _aString;
```

(**rare**) protected class variable:
can't access easily and no custom getter/setter

property declared

backing variable

setter,
auto created

```
-(void)setAString:(NSString *)aString{
    _aString = aString;
}
```

getter,
auto created

```
-(NSString *)aString{
  return _aString;
}
```

lazy instantiation

getter,
custom

```
-(NSString *)aString{
    if(!_aString)
        _aString = @"This string was not set";
    return _aString;
}
@end
```

objective c

ARC

```
@interface SomeClass ()


@property (strong, nonatomic) NSString *aString;
@end

@implementation SomeClass
@synthesize aString = _aString;
```

atomic ~ thread safe
nonatomic ~ faster access

strong ~ keep a reference
weak ~ no reference

automatic reference counting

not garbage collection
when reference count for variable == 0, immediately free memory

strong is usually what you want, else variable is never allocated

weak is used in scenarios where something else holds a reference

```
@end
```

objective c

encapsulation

```
NSNumber *aNum = [[NSNumber alloc]init];
aNum = @3;

NSString *aString = [NSString stringWithFormat:@"The time is always %d past %d",42,9];
aString = @"A string";
```

these are PropertyLists: **serializable**, containers for primitive values

Valid Property Lists: NSData, NSDate, NSNumber (int, float, bool)

```
NSArray *myArray = @[@32,@"a string”,@3.2,@3,@109,@42,@32];
for(id obj in myArray)
    NSLog(@“Obj=%@",obj);
```

can store any object

loop over an NSArray

An **Array** of **PropertyLists** is also a **PropertyList**

Dictionary as a class property

```
@interface SomeClass ()
@property (strong, nonatomic) NSDictionary *aDictionary;
@end
```

An **Dictionary** of **PropertyLists** is also a **PropertyList**

Access self

```
self.aDictionary = @{@"key1":@3,@"key2":@"a string"};
for(id key in self.aDictionary)
    NSLog(@"key=%@, value=%@",key,self.aDictionary[key]);
```

objective c

mutable and immutable

```
NSArray *myArray = @[@32,@"a string”,[[UILabel alloc]init] ];
```

possible to add objects now

all arrays are **nil** terminated
more on that later…

```
NSMutableArray *anArrayYouCanAddTo = [NSMutableArray arrayWithObjects:aNum,@32, nil];

[anArrayYouCanAddTo addObject:someComplexObject];

NSMutableArray *anotherArray = [@[@32,@"string me"] mutableCopy];
```

objective c

functions examples

return type

method name

parameter name

```
-(NSNumber*) addOneToNumber:(NSNumber *)myNumber{

    return @([myNumber floatValue]+1);
}
```

parameter type

```
NSNumber *obj = [self addOneToNumber:@4];
```

receiver class

message

parameter value

throwback to **c**

```
float addOneToNumber(float myNum){
    return myNum++;
}

float val = addOneToNumber(3.0);
```

second parameter name

```
-(NSNumber*)    addToNumber:(NSNumber *)myNumber
            withOtherNumber: (NSNumber *)anotherNumber
```

second parameter

```
NSNumber *obj = [self addToNumber:@4 withOtherNumber:@67];
```

objective c

common functions

function

NSString to format

object to print

```
NSLog(@"The value is: %@",someComplexObject);
```

%@ is print for serializable objects

```
NSLog(@"The value is: %d",someInt);
NSLog(@"The value is: %.2f”,someFloatOrDouble);
```

```
        someComplexObject = nil;

        if(!someComplexObject)
            printf("Wow, printf works!");
```

set to nothing,
subtract from reference count

this means: **if variable is not nil**

nil only works for objects!
no primitives, structures, or enums

objective c

review

```
@interface SomeViewController ()
{
    float aFloat;
}
@property (strong, nonatomic) NSString *aString;
@property (strong, nonatomic) NSDictionary *aDictionary;
@end

@implementation SomeViewController
@synthesize aString = _aString;

-(NSString *)aString{
    if(!_aString)
        _aString = [NSString stringWithFormat:@"This is a string %d",3];
    return _aString;
}

-(void)setAString:(NSString *)aString{
    _aString = aString;
}

- (void)viewDidLoad
{
    [super viewDidLoad];

    self.aDictionary = @{@"key1":@3,@"key2":@"a string"};
    for(id key in _aDictionary)
        NSLog(@"key=%@, value=%@",key,_aDictionary[key]);


    NSArray *myArray = @[@32,@"a string", self.aString ];
    for(id obj in myArray)
        NSLog(@“Obj=%@",obj);

   self->aFloat = 5.0;

}
```

protected class variable

private properties

backing variable

getter

setter

call from super class

dictionary

dictionary iteration

array

array iteration

protected class variable access

adding to our project

- let's add a slider to our project
- and user lazy instantiation
- and some git branching

MVC's

controller has direct connection to view class

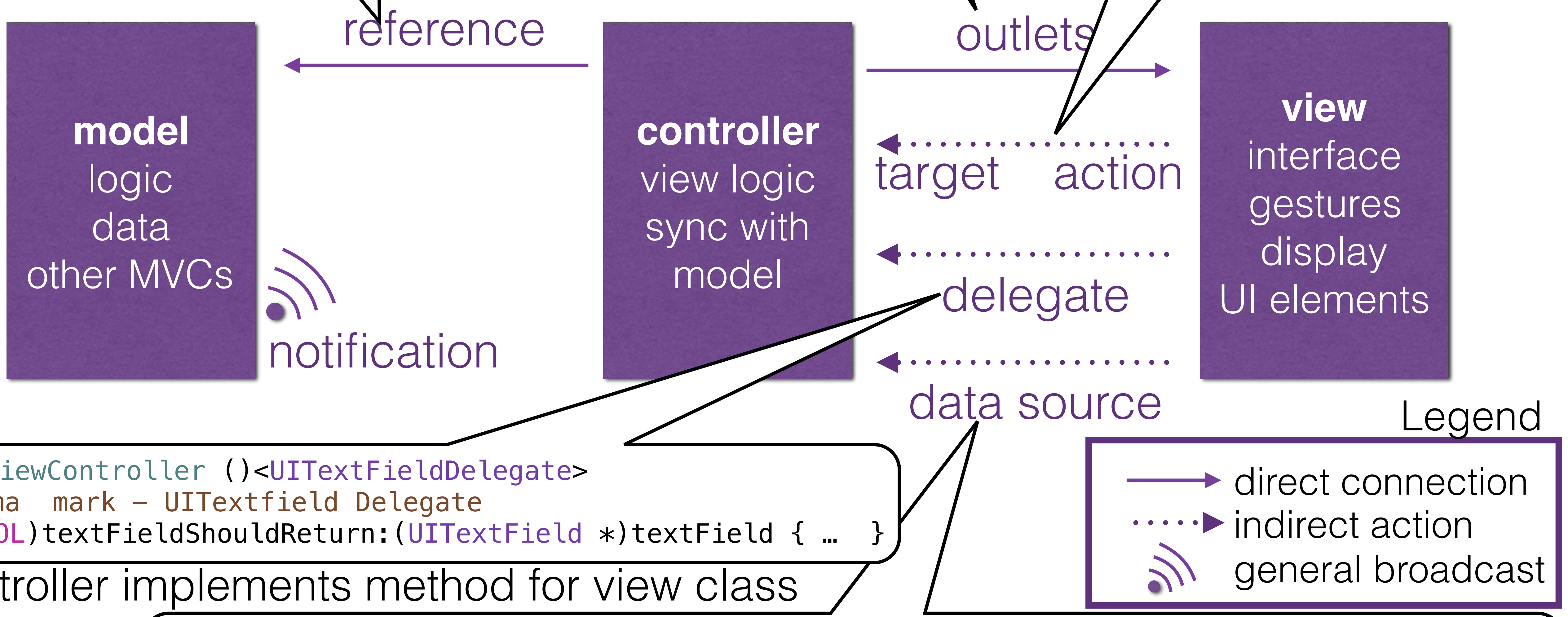
```
@property (weak, nonatomic) IBOutlet UITextField *firstName;
@property (weak, nonatomic) IBOutlet UITextField *lastName;
@property (weak, nonatomic) IBOutlet UITextField *phoneNumber;
```

controller has direct connection to model class

```
ModelClass *myModel = [get global handle to model]
PhoneNumberStruct * phNumber = [myModel getNumber];
self.phoneNumberLabel.text = phNumber.number;
```

view sends a targeted message

```
- (IBAction)buttonPressed:(id)sender;
- (IBAction)showPhBookPressed:(id)sender;
```



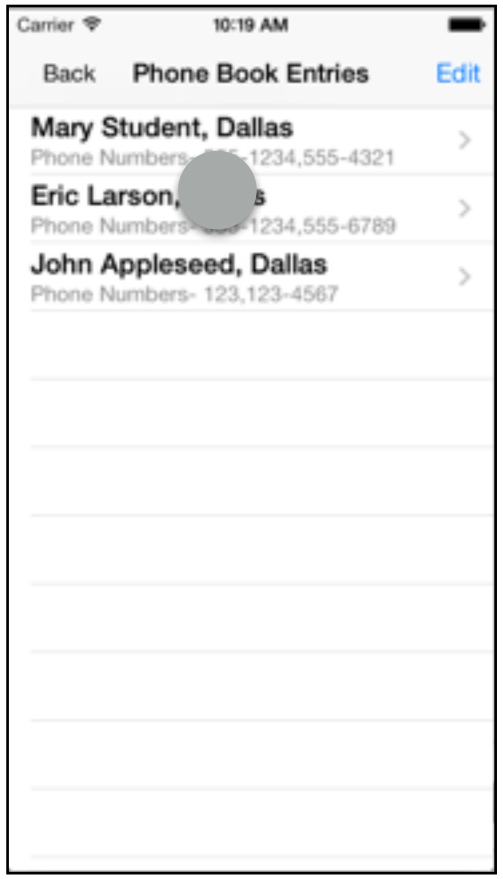
```
MainViewController ()<UITextFieldDelegate>
#pragma  mark - UITextfield Delegate
- (BOOL)textFieldShouldReturn:(UITextField *)textField { …  }
```

controller implements method for view class

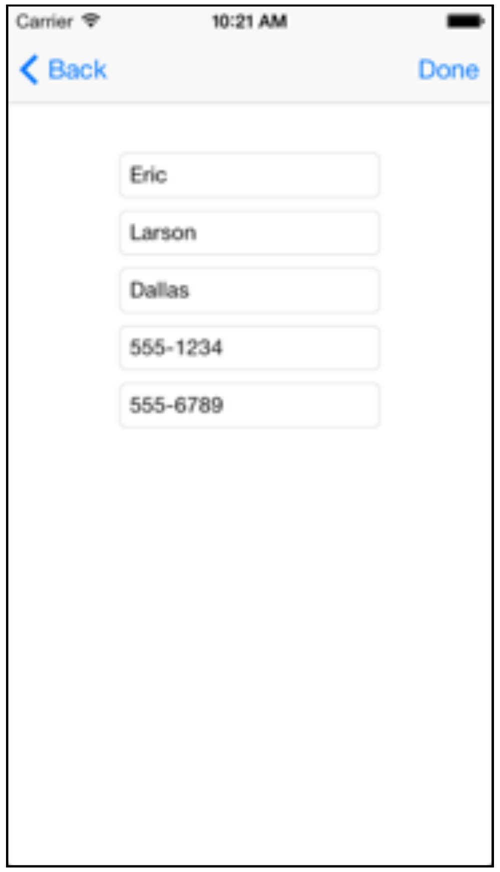
```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
```

MVC life cycle

- problem: we need to handoff control of the screen to a new view
- the app itself is handling most of this transition
    - app will “unfreeze” the new view and its class properties
    - **you** need to send information from **source** ViewController to **destination** ViewController



Mary Student, Dallas
Phone Numbers- 555-1234,555-4321
Eric Larson, Dallas
Phone Numbers- 555-1234,555-6789
John Appleseed, Dallas
Phone Numbers- 123,123-4567



Eric
Larson
Dallas
555-1234
555-6789

controller life cycle

Source Controller

Destination Controller

view is unfrozen, property memory allocated

`prepareForSegue`
prepare to leave the screen
set properties of destination, if needed

view outlets are ready for interaction

`viewDidLoad`

`viewWillAppear`

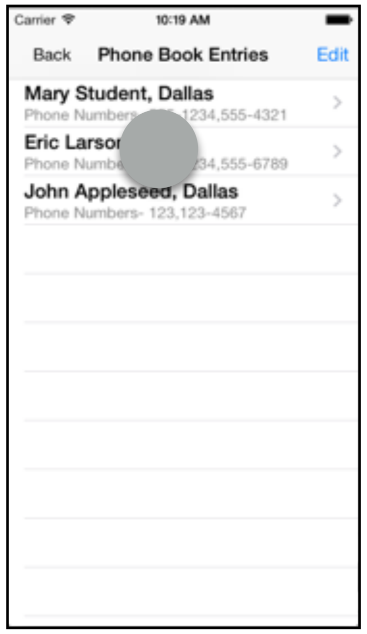
`viewDidAppear`

`viewWillDisappear`

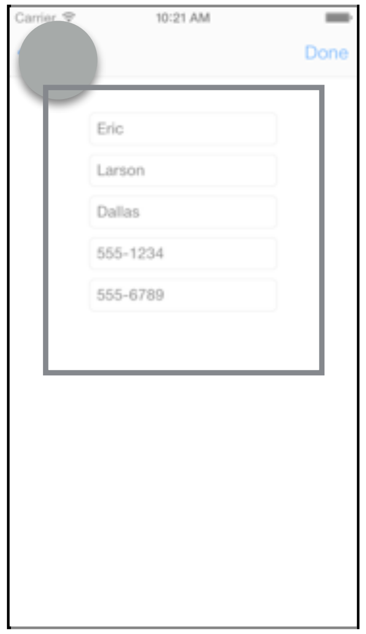
`viewDidDisappear`

memory deallocated when app is ready

source



destination



user

MVC's

- sometimes the best way to create a model is through a Singleton

in .h file (so its public)

```
@interface MyCustomClass : NSObject

+ (MyCustomClass*)sharedInstance;

@end
```

+ means its a
class method
don't need instance to call it

custom getter

called like a backing variable

in .m file

```
+ (MyCustomClass*)sharedInstance
{
    static MyCustomClass * _sharedInstance = nil;

    static dispatch_once_t oncePredicate;

    dispatch_once(&oncePredicate, ^{
            _sharedInstance = [[MyCustomClass alloc] init];
     });
     return _sharedInstance;
}
```

don't worry about syntax
until next week…

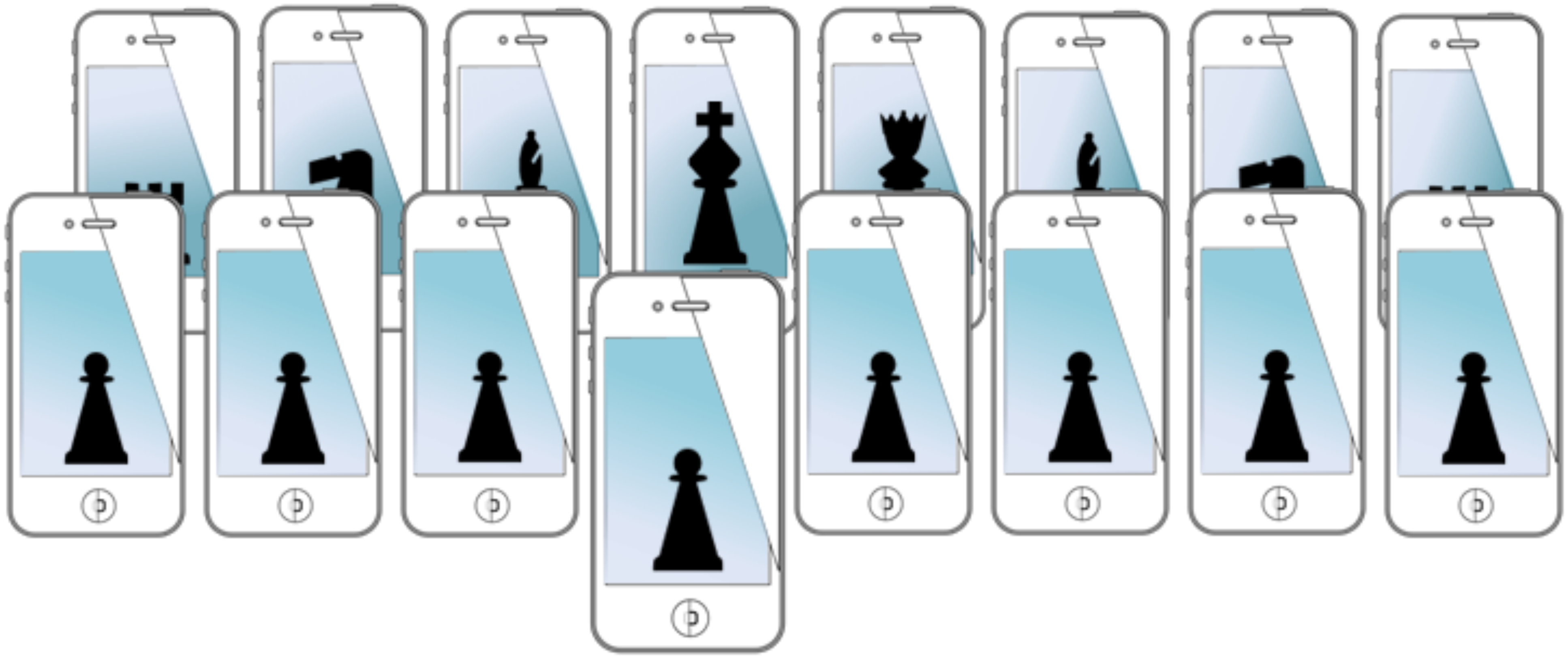
Need more help on MVC's ? Check out Ray Wenderlich:

http://www.raywenderlich.com/46988/ios-design-patterns

for next time…

- View Controllers in iOS
  - Watch videos **before class**
- Come ready to work in teams on an in-class project

MOBILE SENSING & LEARNING



CSE5323 & 7323

Mobile Sensing & Learning

week one, lecture two: objective-C and swift

Eric C. Larson, Lyle School of Engineering,
Computer Science and Engineering, Southern Methodist University