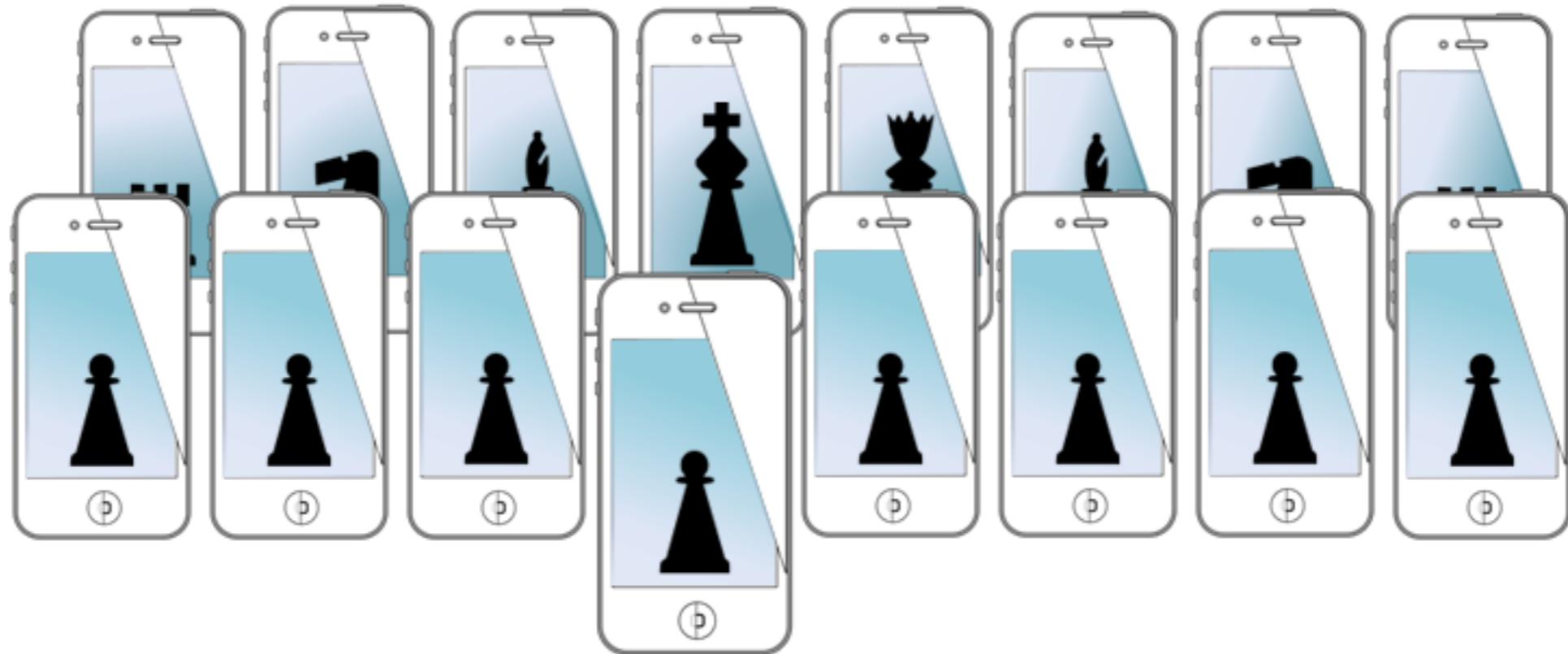


MOBILE SENSING & LEARNING



CSE5323 & 7323
Mobile Sensing and Learning

week one, lecture one: course introduction

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

agenda

- introductions
- class logistics
- what is this mobile sensing course?
 - and what this course is not...
- course goals
- how to do well
- syllabus
 - hardware, lab access, grading, MOD
- Xcode and git

introductions

- education

- undergrad and masters from Oklahoma State
- PhD from the university of Washington, Seattle

- research

- signal, image, and video processing (mobile)
 - how can combining DSP, machine learning, and sensing make seamless computing?
- natural gestures
 - novel interaction techniques and user interface technology
- mobile health
 - moving outside the clinic: how mobile sensing can help patients and doctors
- sustainability
 - how technology can increase awareness

<http://eclarson.com>



introductions

- about you:
 - name (what you go by)
 - grad/undergrad, department, and major
 - something true or false
 - that's all we have time for...

course logistics

- lab: Tues 5-6:30PM, but up for discussion
- we will use canvas for managing the course
- and GitHub for managing code:
 - <https://github.com/SMU-MSLC-2016>
 - Same as previous offerings!!

what is this course (not)

- mobile sensing
 - activity recognition **some, yes!**
 - audio analysis **yes!**
 - vision analysis **yes!**
- machine learning **some, for inference**
- microcontroller communication **yes!**
- general iOS development **some basic skills**
- animation and graphics **no, except to display data**
- user interface design **some, all apps rely on user**

for what we do not cover...

- take the free Stanford iOS course!
- prerequisite: model based coding (or ability to pick it up quick)
- CSE's will find some of this review, EE's will find some of this review – just not at the same time
- creative computation? creativity and design are well rewarded in this class

course goals

- exposure to iOS development, MVCs
- understand how to **use embedded sensors**, on/off phone
- **exposure to machine learning** for mobile sensors
- real time analysis of data streams
 - applications in mobile health
- **present and pitch** applications

how to do well

- complete the lab assignments on time
 - there is no such thing as a late assignment
- start the **lab assignments early, with your team**
- iterate and test your apps
- use good coding practices, lazy instantiation, recycle classes, get on Apple's developer website for more info
- have fun—seriously
- collaborate, collaborate, collaborate
- and come to class

syllabus

- attendance
 - required for lecture, optional for lab
 - video of classes are not available
- hardware is available for checkout
 - need a team formed (do this before the end of the week)
 - mac minis (password protected, do not change)
 - iPhones (password protected, do not change)
 - you can use your own stuff, but will need iPhone 5S or better

syllabus: lab assign.

- 50% of grade (5 labs @ 9% each, 1 lab @ 5%)
- turn in source code via canvas, show me the app or take video
- usually every two weeks, one assignment per team (teams of 2-3)
- deliverables are worth 90% of the lab grade
- each lab deliverable will be graded on
 - how efficient your implementation is
 - how well you use proper coding styles and interface guidelines
 - how well each elements meets specified criteria
- Make it memory and computationally efficient, use hardware acceleration
- Comment code so that it is readable and immediately understandable
- remaining 10% of the points are reserved for exceptional work and work that is above and beyond in one or more elements of the application, incorporating elements not discussed in class and having superior performance
 - Difference between 5000 and 7000 level is in the exceptional work!

syllabus

- in class assignments (flipped assignments)
 - 25% of grade (5 @ 5% each)
 - watch videos before class
 - come ready to work on assignment as a team
 - turn in assignment at end of class
 - absence == no credit

final project

- worth 25% of grade
 - 15% from iOS application
 - 5% from final presentation
 - 5% from video (like kick-starter, with more detail)

The Mother of all demos (MOD)

Students will have the option to “opt out” of the final project website in exchange for a more risky “mother of all demos” demonstration. Groups that opt into this must meet **all** of the proposed specifications of their mobile applications (additionally, these demos should be more difficult). These specifications must work **flawlessly** during the final project presentation. If any specification is not met, the MOD **does not apply and a video summary will be due**. Note that it does not matter how close the specification was to working: if a portion of the application was specified to run in 5 seconds, but takes 5.1 seconds, the MOD is not met (it is **all or nothing**). You will be told immediately after demonstration whether the design specifications were met.

before next class

- look at the class website
- get a team together (groups of 2 or 3, no exceptions)
 - contribute **equally**, **everyone** codes, **everyone** designs
 - **pick good members** with different skills than you
 - take turns **watching each other code** (I know...)
- assignment 1 is already up!
 - let's check it out...

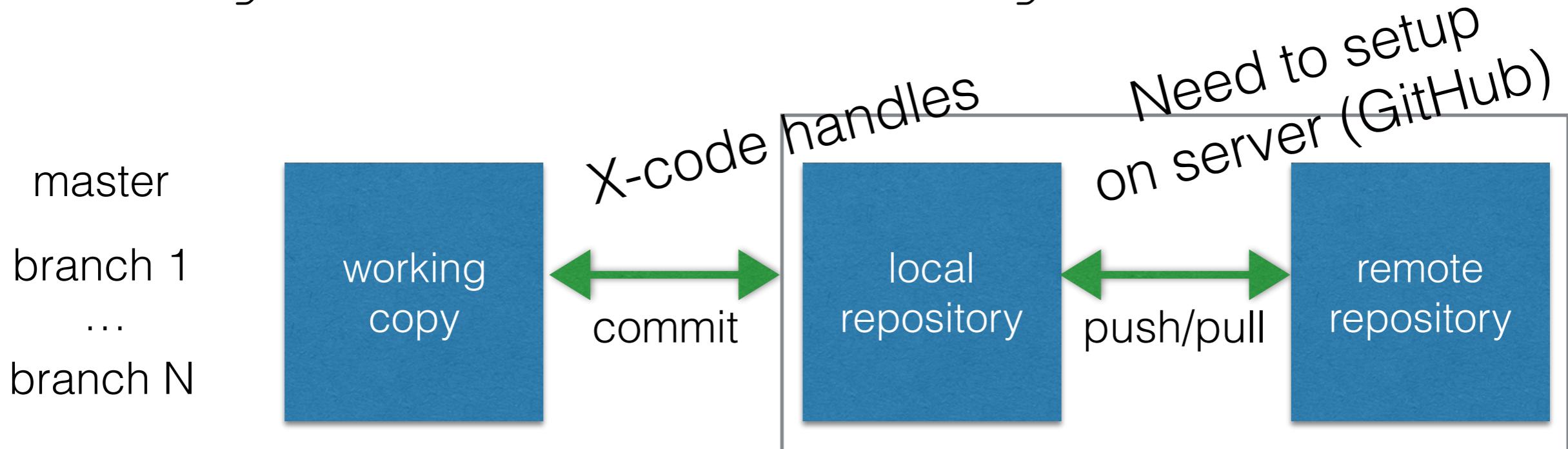
git

- built into unix (and therefore OSX) and Xcode
- use it when developing with teams or just by yourself
- branching, merging, and all the jazz

```
git init
```

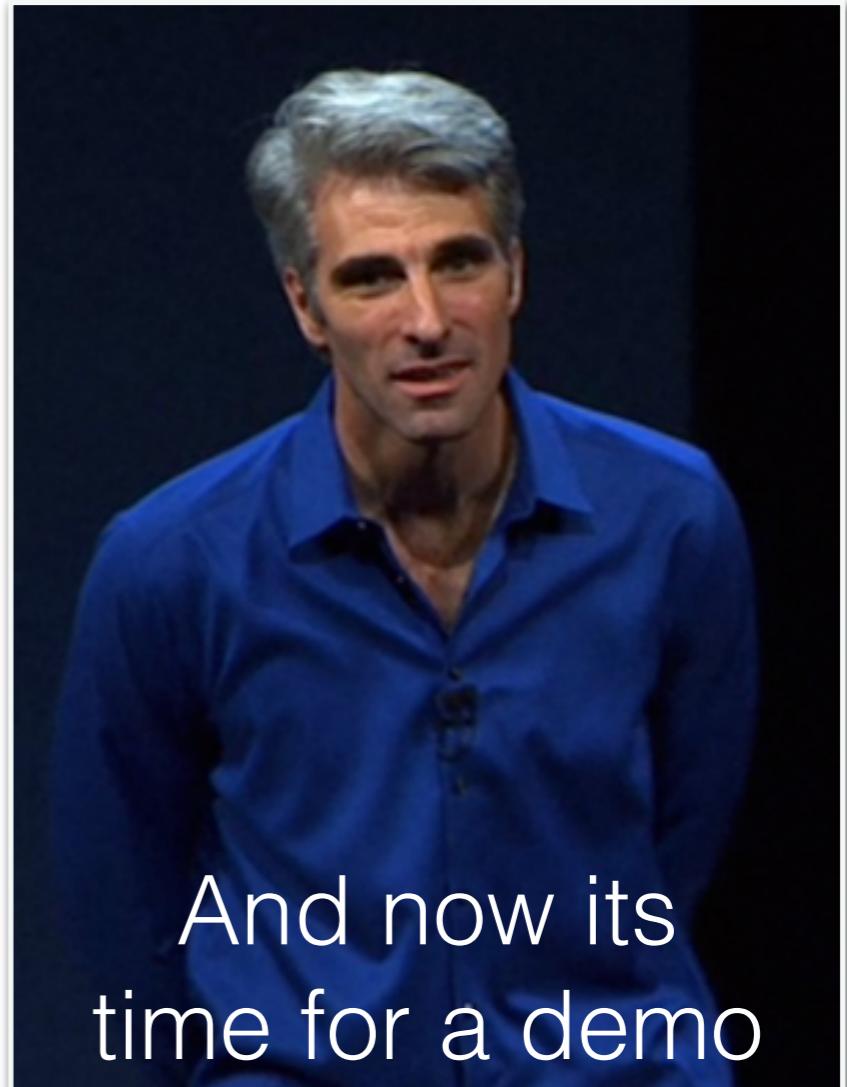
```
git add .
```

```
git commit -m"starting commit"
```



git with Xcode

- provides GUI for most git commands
 - commit, branch, push, pull, etc.
 - plays nicely with submodules
- **rarely** is command line needed
- git is great for code!!
- but not great for storyboards ...

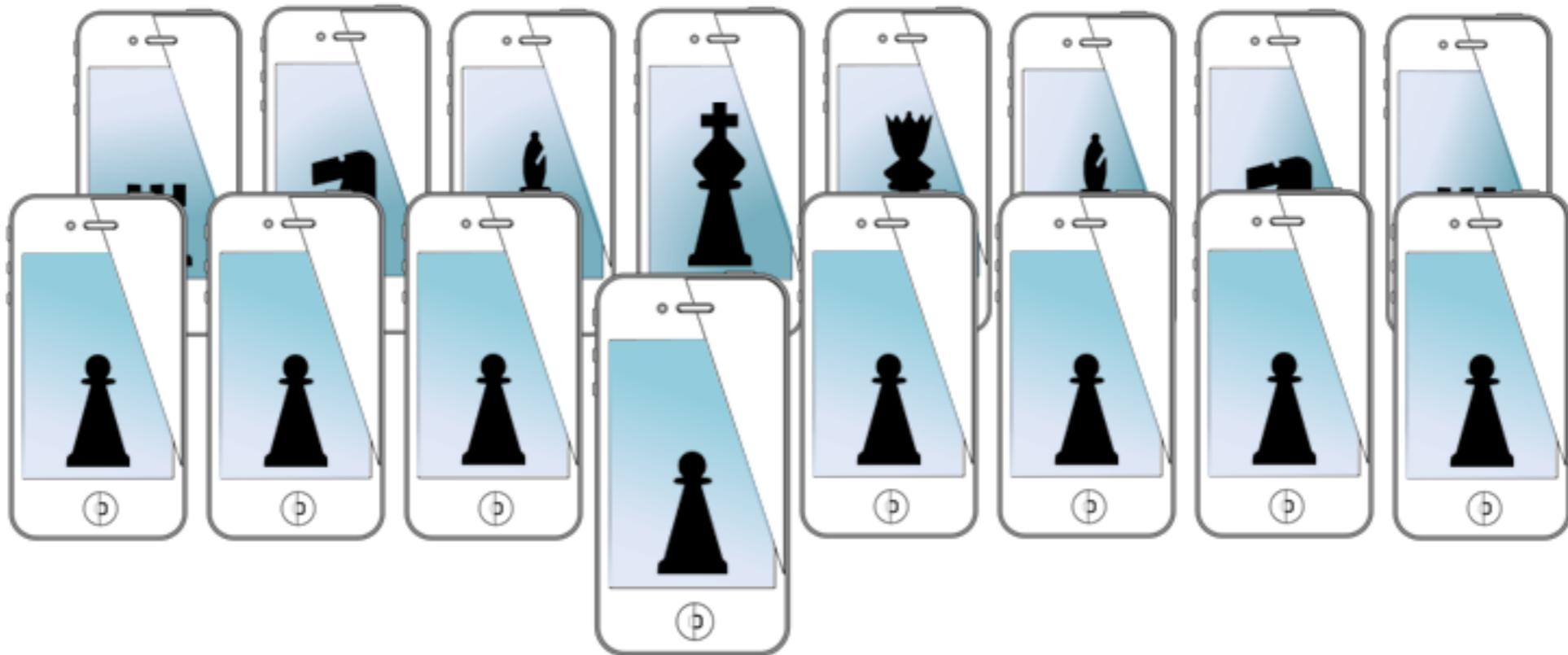


And now its
time for a demo

for next time...

- have teams figured out
- have hardware checked out to team

MOBILE SENSING & LEARNING

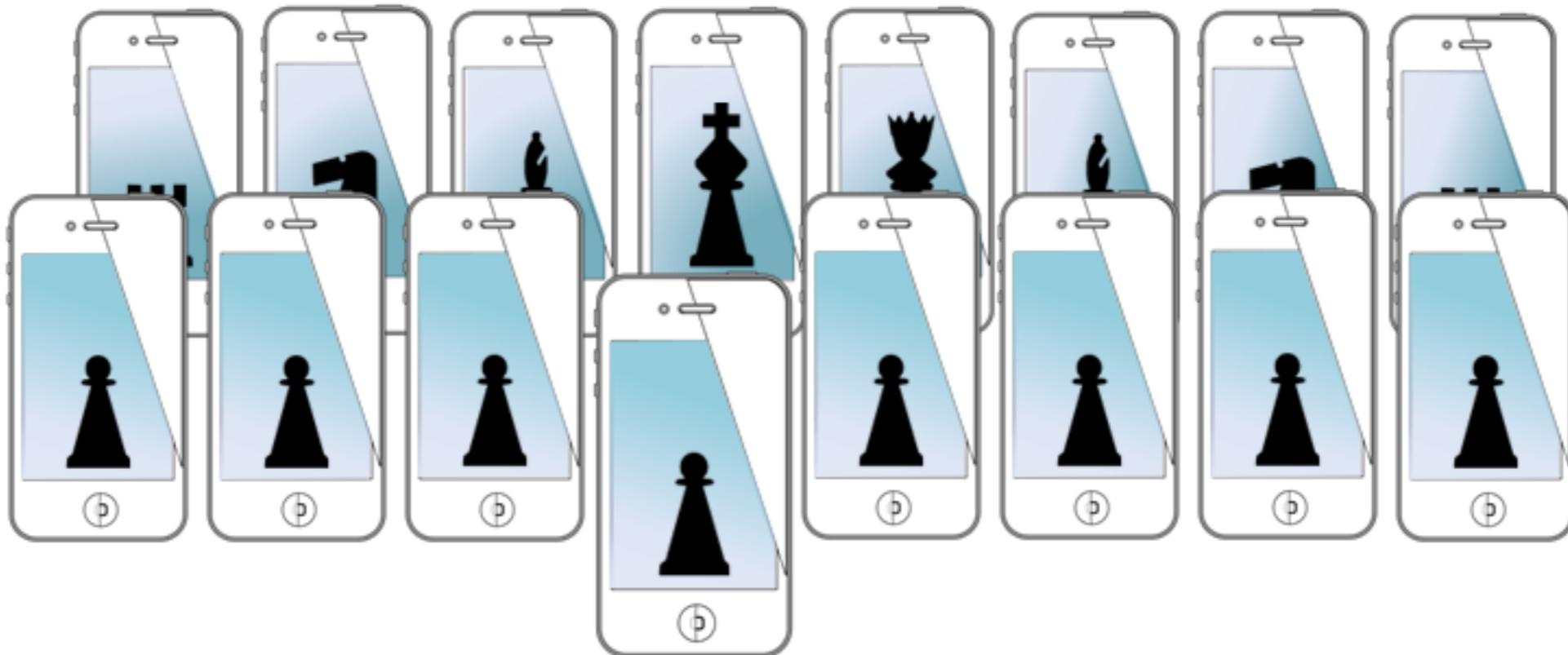


CSE5323 & 7323
Mobile Sensing and Learning

week one, lecture one: course introduction

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

MOBILE SENSING & LEARNING



CSE5323 & 7323
Mobile Sensing & Learning

week one, lecture two: objective-C and MVC

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

course logistics

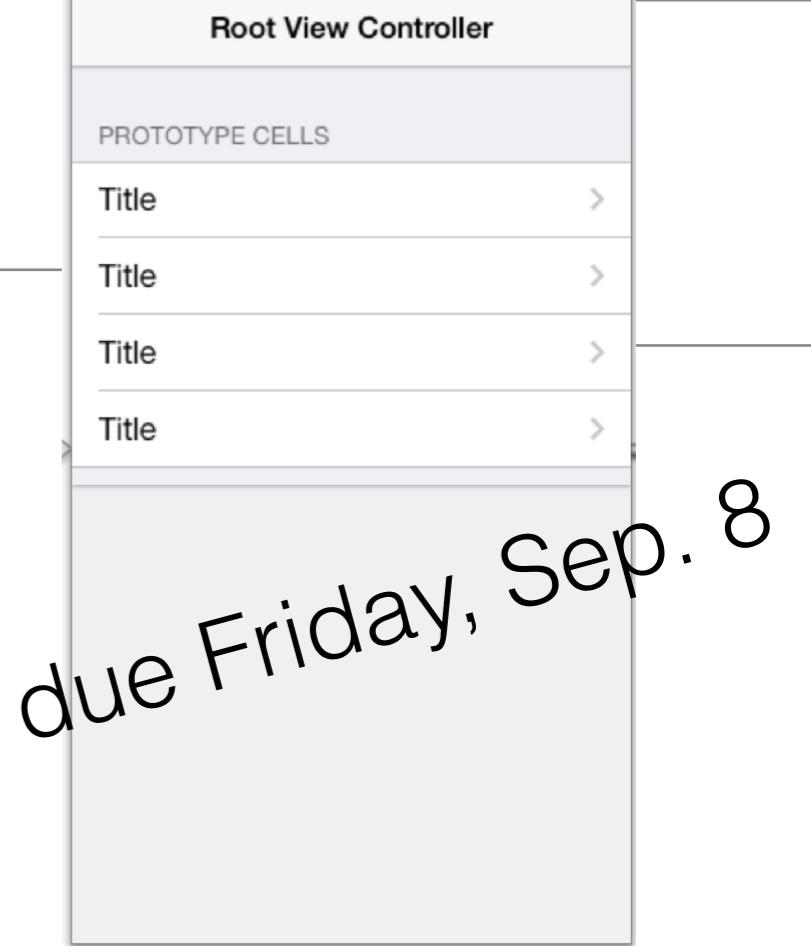
- lab time: Tu5-6:30
- teams: **must be on a team** now!
- next class period will be flipped, **so view video on canvas!**
- get access to (so that you can do the flipped classroom example):
 - mac mini's? iPhones?
 - university developer program is a go!
 - send me an email if you want an invite
 - need your:
 - email that you want invite sent to
 - device Name (i.e., Eric's iPhone)
 - phone SEID: Settings > general > about > SEID

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.
- 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

- 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
- **Refer to the rubric on canvas for full list of required items**



agenda

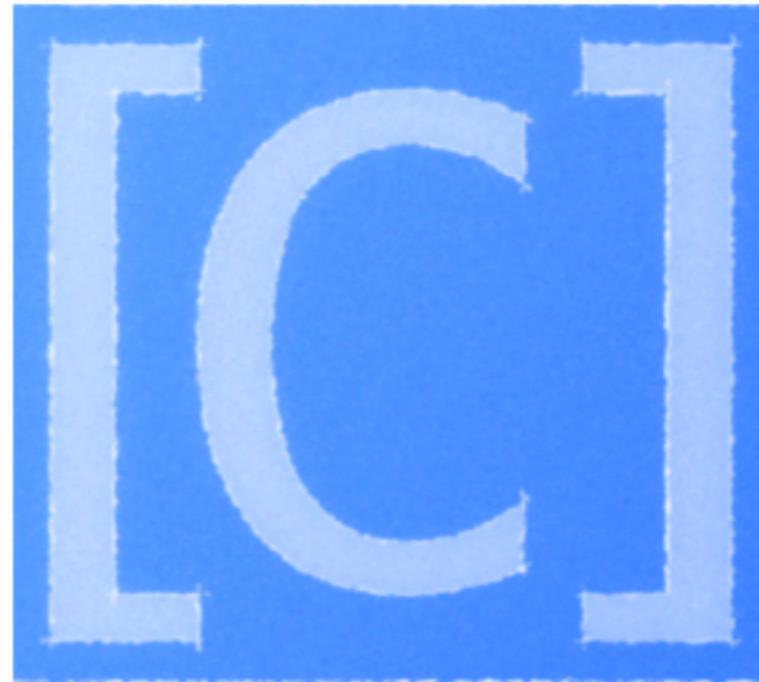
a big syntax demo...

- **objective-c** and (maybe) 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

```
@interface SomeClass : NSObject
```

inherits from

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

```
@end
```

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

property

```
@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; // (rare) protected class variable:  
                  // can't access easily and no custom getter/setter  
}  
@property (strong, nonatomic) NSString *aString; // property declared  
@end  
  
@implementation SomeClass  
@synthesize aString = _aString; // 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 ()
```

atomic ~ thread safe property access
nonatomic ~ faster access

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

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

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;
```

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

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

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

```
NSArray *myArray = @[@32, @"a string", @1.2, @0.1, @10, @42, @32];  
for(id obj in myArray)  
    NSLog(@"%@", 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, 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, structs, or enums

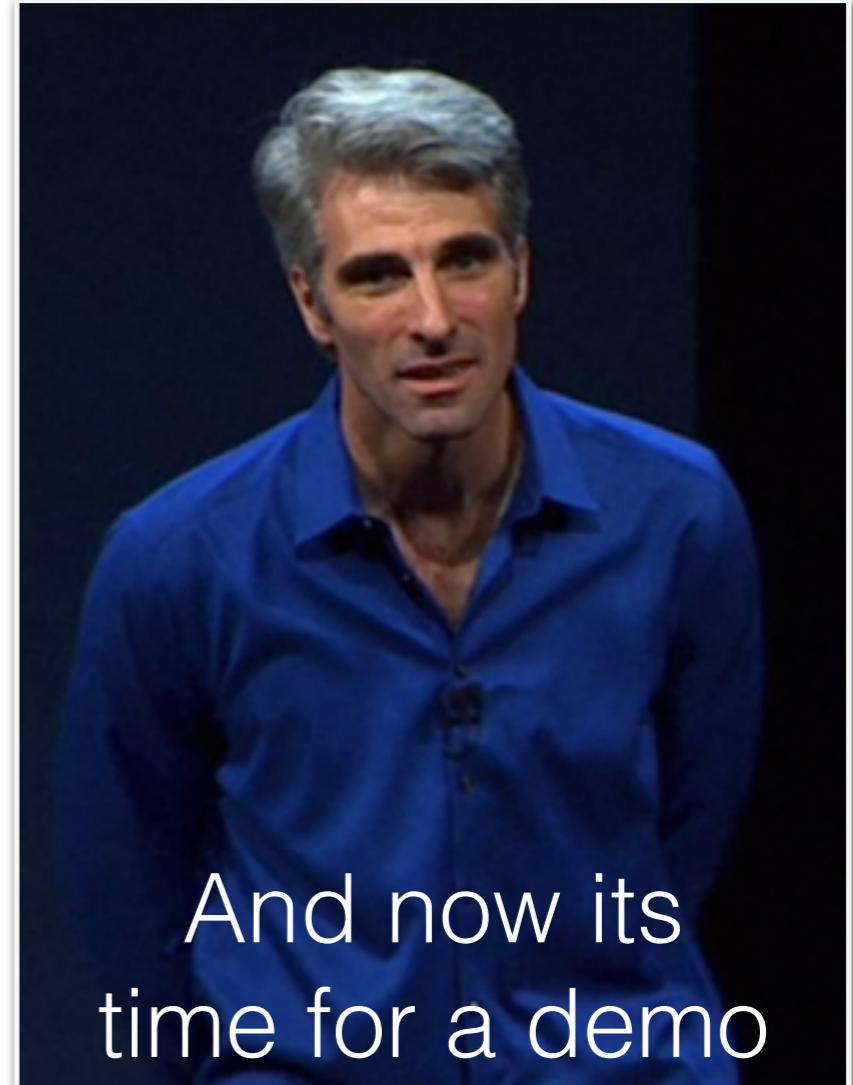
objective c

review

```
@interface SomeViewController ()  
{  
    float aFloat; // protected class variable  
}  
@property (strong, nonatomic) NSString *aString;  
@property (strong, nonatomic) NSDictionary *aDictionary;  
@end  
  
@implementation SomeViewController  
@synthesize aString = _aString; // private properties  
  
-(NSString *)aString{ // backing variable  
    if(!_aString)  
        _aString = [NSString stringWithFormat:@"This is a string %d",3];  
    return _aString; // getter  
}  
  
-(void)setAString:(NSString *)aString{ // setter  
    _aString = aString;  
}  
  
- (void)viewDidLoad // call from super class  
{  
    [super viewDidLoad];  
  
    self.aDictionary = @{@"key1":@3,@"key2":@"a string"}; // dictionary  
    for(id key in _aDictionary) // dictionary iteration  
        NSLog(@"key=%@, value=%@",key,_aDictionary[key]);  
  
    NSArray *myArray = @[@32, @"a string", self.aString ]; // array  
    for(id obj in myArray) // array iteration  
        NSLog(@"Obj=%@",obj);  
  
    self->aFloat = 5.0; // protected class variable access  
}
```

adding to our project

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



And now its
time for a demo

MVC's

controller has direct connection to view class

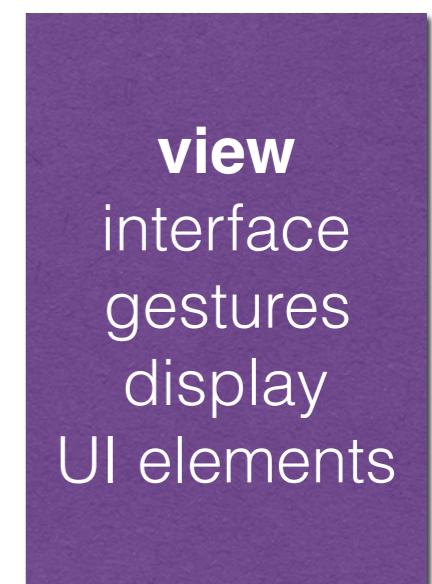
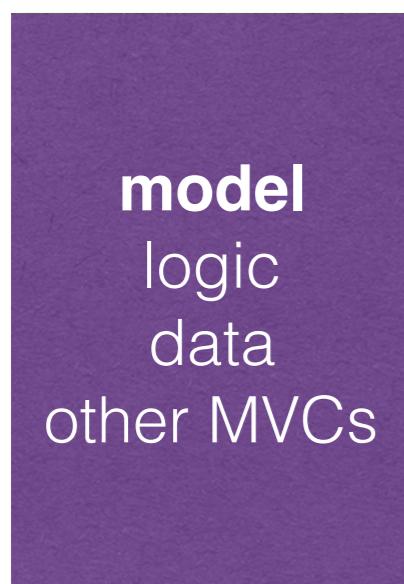
```
@property (weak, nonatomic) IBOutlet UITextField *firstName;
@property (weak, nonatomic) IBOutlet UITextField *lastName;
@property (weak, nonatomic) IBOutlet UITextField *phoneNumber;
```

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;



reference

outlets

target

action

delegate

data source

Legend

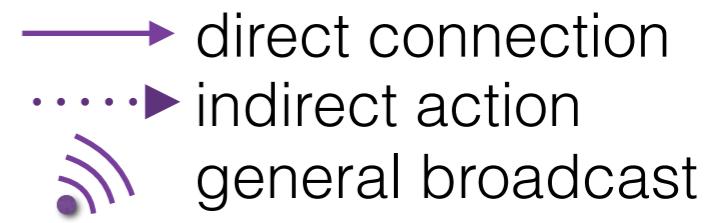
MainViewController ()<UITextFieldDelegate>

#pragma mark - UITextField Delegate

- (BOOL)textFieldShouldReturn:(UITextField *)textField { ... }

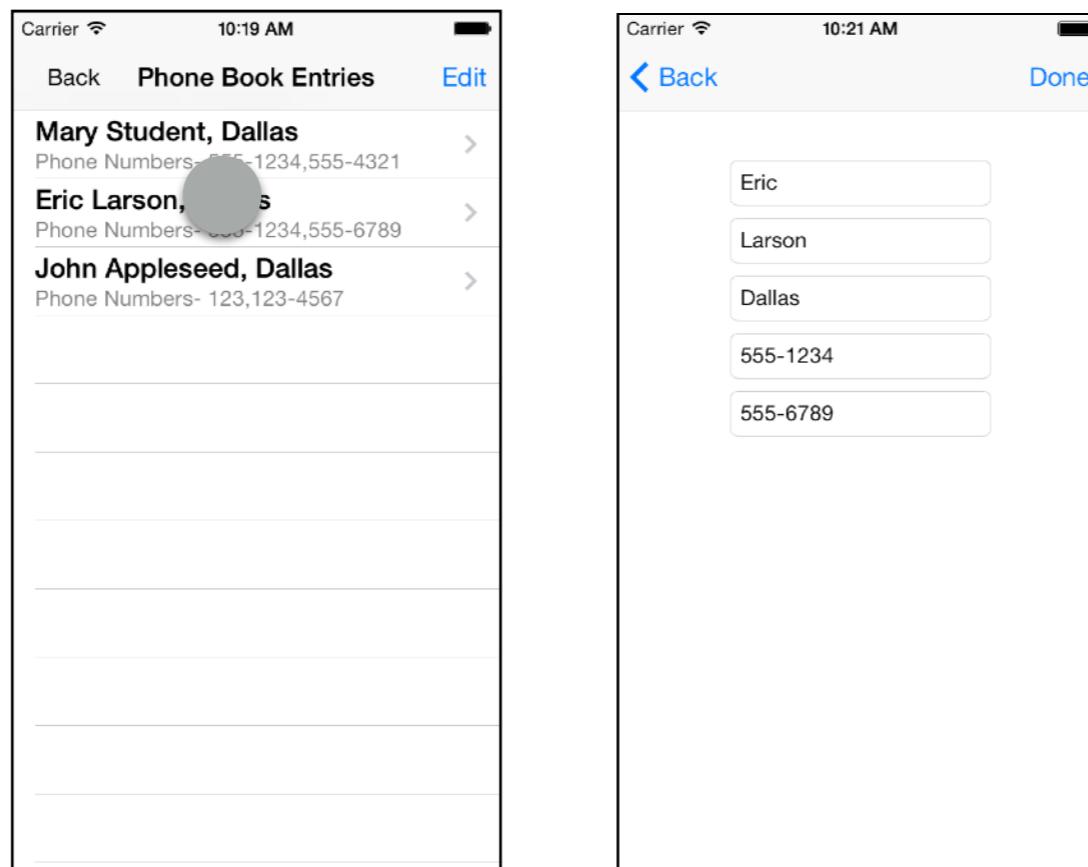
controller implements method for view class

- (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



controller life cycle

Source Controller

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

Destination Controller

view is unfrozen, property memory allocated

view outlets are ready for interaction

viewDidLoad

viewWillAppear

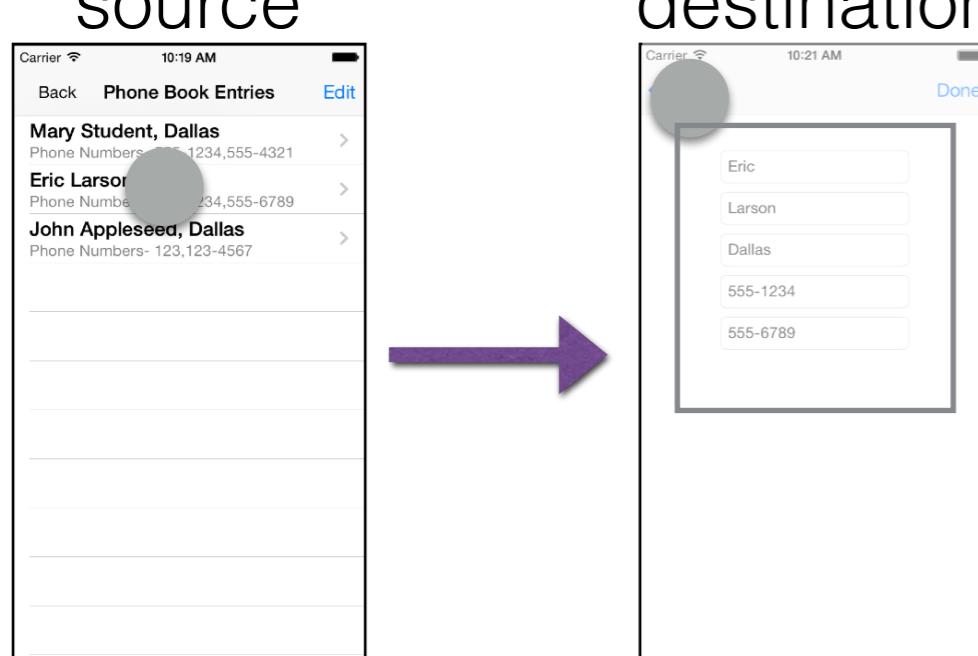
viewDidAppear

viewWillDisappear

viewDidDisappear

memory deallocated when app is ready

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
```

```
+ (MyCustomClass*)sharedInstance
{
    static MyCustomClass * _sharedInstance = nil;
    static dispatch_once_t oncePredicate;
    dispatch_once(&oncePredicate, ^{
        _sharedInstance = [[MyCustomClass alloc] init];
    });
    return _sharedInstance;
}
```

called like a backing variable
in .m file

don't worry about syntax
until next week...

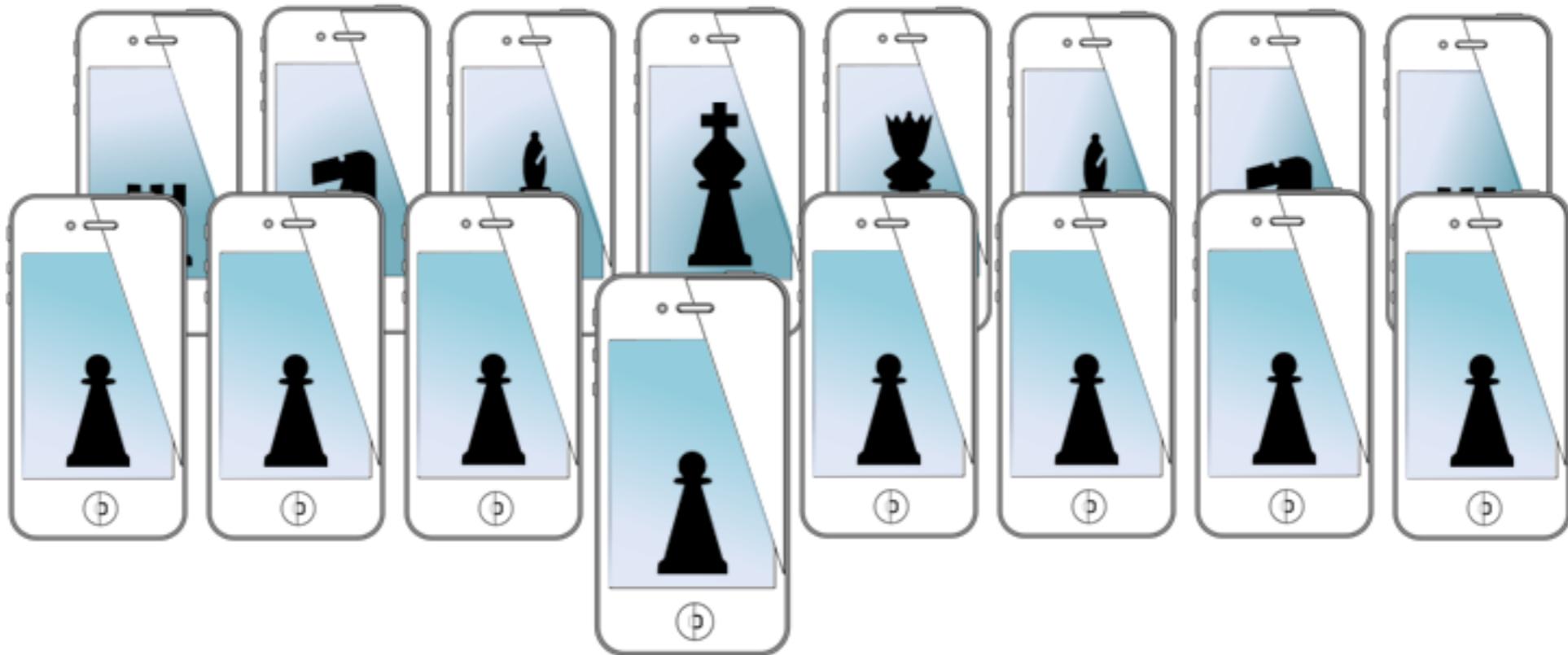
Need more help on MVC's ? Check out Ray Wenderlich:

<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 assignment

MOBILE SENSING & LEARNING



CSE5323 & 7323
Mobile Sensing & Learning

week one, lecture two: objective-C and MVC

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

if time slides!

swift

- syntax is nothing like objective c
- a lot like python syntax (but not)
- weakly typed, no need for semicolons
- can be hard to read or interpret
- powerful use of *tuples,optionals,switch*
- you need to look online for more material than this lecture
 - https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/TheBasics.html



Swift

variables

```
let maximumNumberOfLoginAttempts = 10  
var currentLoginAttempt = 0
```

not mutable

mutable

```
let pi = 3.14159  
// pi is inferred to be of type Double  
let three = 3  
let pointOneFourOneFiveNine = 0.14159  
let pi = Double(three) + pointOneFourOneFiveNine  
// pi equals 3.14159, and is inferred to be of type Double  
let meaningOfLife = 42  
// meaningOfLife is inferred to be of type Int
```

and then there
is this...

```
let orangesAreOrange = true  
let turnipsAreDelicious = false
```

```
let π = 3.14159  
let 你好 = "你好世界"  
let 🐶🐮 = "dogcow"
```

```
var friendlyWelcome = "Hello World!"  
var friendlyWelcome: String = "Hello World!"  
  
println(friendlyWelcome)  
  
println("The current value of friendlyWelcome is \(friendlyWelcome)")
```

no need to set

see this: https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/TheBasics.html

Swift tuples

```
let http404Error = (404, "Not Found")
// http404Error is of type (Int, String), and equals (404, "Not Found")
```

```
let (statusCode, statusMessage) = http404Error
println("The status code is \(statusCode)")
// prints "The status code is 404"
println("The status message is \(statusMessage)")
// prints "The status message is Not Found"
```

```
println("The status code is \(http404Error.0)")
// prints "The status code is 404"
println("The status message is \(http404Error.1)")
// prints "The status message is Not Found"
```

```
let http200Status = (statusCode: 200, description: "OK")
```

```
println("The status code is \(http200Status.statusCode)")
// prints "The status code is 200"
println("The status message is \(http200Status.description)")
// prints "The status message is OK"
```

see this: https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/TheBasics.html

Swift

optionals

```
let possibleNumber = "123"
let convertedNumber = possibleNumber.toInt()
// convertedNumber is inferred to be of type "Int?", or "optional Int"
```

```
var serverResponseCode: Int? = 404
// serverResponseCode contains an actual Int value of 404
serverResponseCode = nil
// serverResponseCode now contains no value
```

can now set to nil :)

```
var surveyAnswer: String?
// surveyAnswer is automatically set to nil
```

```
if convertedNumber != nil {
    println("convertedNumber has an integer value of \(convertedNumber!)\")
}
// prints "convertedNumber has an integer value of 123."
```

```
if let actualNumber = possibleNumber.toInt() {
    println("\\" \(possibleNumber) \" has an integer value of \(actualNumber)")
} else {
    println("\\" \(possibleNumber) \" could not be converted to an integer")
}
// prints "'123' has an integer value of 123"
```

see this: https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/TheBasics.html

Swift

accessing optionals

optional

! **unwrap** output to be **string**. Else: **error**

```
let possibleString: String? = "An optional string"  
let forcedString: String = possibleString! // requires an exclamation mark
```

implicit unwrap

```
let assumedString: String! = "An implicitly unwrapped optional string."  
let implicitString: String = assumedString // no need for an exclamation mark
```

output always unwrapped to be **string**. Else: **error**

```
if assumedString != nil {  
    println(assumedString)  
}  
// prints "An implicitly unwrapped optional string."
```

```
if let definiteString = assumedString {  
    println(definiteString)  
}  
// prints "An implicitly unwrapped optional string."
```

Optional unwrapping is not my favorite part of swift

see this: https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/TheBasics.html

Swift arrays

```
var shoppingList = ["Eggs", "Milk"]  
  
println("The shopping list contains \(shoppingList.count) items.")  
// prints "The shopping list contains 2 items."
```

```
if shoppingList.isEmpty {  
    println("The shopping list is empty.")  
} else {  
    println("The shopping list is not empty.")  
}  
// prints "The shopping list is not empty."
```

```
shoppingList += ["Baking Powder"]  
shoppingList += ["Chocolate Spread", "Cheese", "Butter"]
```

```
var firstItem = shoppingList[0]  
// firstItem is equal to "Eggs"
```

```
shoppingList[0] = "Six eggs"
```

like a dequeue

```
let butter = shoppingList.removeLast()  
let sixEggs = shoppingList.removeAtIndex(0)
```

like a pop

see this: https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/TheBasics.html

Swift

dictionaries

```
var airports = ["YYZ": "Toronto Pearson", "DUB": "Dublin"]  
airports["LHR"] = "London"  
// the airports dictionary now contains 3 items
```

```
if let oldValue = airports.updateValue("Dublin Airport", forKey: "DUB") {  
    println("The old value for DUB was \(oldValue).")  
}  
// prints "The old value for DUB was Dublin."
```

```
airports["APL"] = "Apple International"  
// "Apple International" is not the real airport for APL, so delete it  
airports["APL"] = nil  
// APL has now been removed from the dictionary
```

```
let airportCodes = [String](airports.keys)  
// airportCodes is ["YYZ", "LHR"]  
  
let airportNames = [String](airports.values)  
// airportNames is ["Toronto Pearson", "London Heathrow"]
```

see this: https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/TheBasics.html

Swift loops

```
for index in 1...3 {  
    println("\(index) times 5 is \(index * 5)")  
}  
// 1 times 5 is 5  
// 2 times 5 is 10  
// 3 times 5 is 15
```

```
let names = ["Anna", "Alex", "Brian"]  
for name in names {  
    println("Hello, \(name)!")  
}  
// Hello, Anna!  
// Hello, Alex!  
// Hello, Brian!
```

```
for (index, value) in enumerate(names) {  
    println("Item \(index + 1): \(value)")  
}  
// Item 1: Anna  
// Item 2: Alex  
// Item 3: Brian
```

```
let number0fLegs = ["spider": 8, "ant": 6, "cat": 4]  
for (animalName, legCount) in number0fLegs {  
    println("\(animalName)s have \(legCount) legs")  
}  
// ants have 6 legs  
// cats have 4 legs  
// spiders have 8 legs
```

see this: https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/TheBasics.html

Swift

switch

```
let someCharacter: Character = "e"
switch someCharacter {
    case "a", "e", "i", "o", "u":
        println("\(someCharacter) is a vowel")
    case "b", "c", "d", "f", "g", "h", "j", "k", "l", "m",
        "n", "p", "q", "r", "s", "t", "v", "w", "x", "y", "z":
        println("\(someCharacter) is a consonant")
    default:
        println("\(someCharacter) is not a vowel or a consonant")
}
// prints "e is a vowel"
```

no pass through

```
let somePoint = (1, 1)
switch somePoint {
    case (0, 0):
        println("(0, 0) is at the origin")
    case (_, 0):
        println("\(somePoint.0), 0) is on the x-axis")
    case (0, _):
        println("(0, \(somePoint.1)) is on the y-axis")
    case (-2...2, -2...2):
        println("\(somePoint.0), \(somePoint.1)) is inside the box")
    default:
        println("\(somePoint.0), \(somePoint.1)) is outside of the box")
}
// prints "(1, 1) is inside the box"
```

"any" value

see this: https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/TheBasics.html

Swift

switch continued...

```
let anotherPoint = (2, 0)
switch anotherPoint {
case (let x, 0):
    println("on the x-axis with an x value of \(x)")
case (0, let y):
    println("on the y-axis with a y value of \(y)")
case let (x, y):
    println("somewhere else at (\(x), \(y))")
}
// prints "on the x-axis with an x value of 2"
```

“any” value and set

```
let yetAnotherPoint = (1, -1)
switch yetAnotherPoint {
case let (x, y) where x == y:
    println("( \(x) , \(y) ) is on the line x == y")
case let (x, y) where x == -y:
    println("( \(x) , \(y) ) is on the line x == -y")
case let (x, y):
    println("( \(x) , \(y) ) is just some arbitrary point")
}
// prints "(1, -1) is on the line x == -y"
```

very powerful, concise,
readable

see this: https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/TheBasics.html

swift

functions

```
func sayHello(personName: String) -> String {  
    let greeting = "Hello, " + personName + "!"  
    return greeting  
}
```

internal name

input type

return type

```
func join(string s1: String, toString s2: String, withJoiner joiner: String)  
-> String {  
    return s1 + joiner + s2  
}
```

return type

external name

passed value

```
join(string: "hello", toString: "world", withJoiner: ", ")  
// returns "hello, world"
```

see this: https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/TheBasics.html

Swift

functions

There are too many ways of defining functions to cover it all. For instance you can also setup default values...

array of ints

```
func minMax(array: [ Int ]) -> (min: Int , max: Int ) {  
    var currentMin = array[0]  
    var currentMax = array[0]  
    for value in array[1..        if value < currentMin {  
            currentMin = value  
        } else if value > currentMax {  
            currentMax = value  
        }  
    }  
    return (currentMin, currentMax)  
}
```

return tuple

tuple keys are external names!!

```
let bounds = minMax([8, -6, 2, 109, 3, 71])  
println("min is \(bounds.min) and max is \(bounds.max)")  
// prints "min is -6 and max is 109"
```

see this: https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/TheBasics.html

Swift

classes and properties

```
class DataImporter {  
    var fileName = "data.txt"  
    // the DataImporter class would provide data importing functionality here  
}
```

class variable

```
class DataManager {  
    lazy var importer = DataImporter()  
    var data = [String]()  
    // the DataManager class would provide data management functionality here  
}
```

lazy instantiation

```
let manager = DataManager()  
manager.data.append("Some data")  
manager.data.append("Some more data")  
// the DataImporter instance for the importer property has not yet been created
```

class initialized, but importer is not set

```
println(manager.importer.fileName)  
// the DataImporter instance for the importer property has now been created  
// prints "data.txt"
```

when accessed first, sets value

see this: https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/TheBasics.html

```
class StepCounter {  
    var totalSteps: Int = 0 {  
        willSet(newTotalSteps) {  
            println("About to set totalSteps to \(newTotalSteps)")  
        }  
        didSet {  
            if totalSteps > oldValue {  
                println("Added \(totalSteps - oldValue) steps")  
            }  
        }  
    }  
}
```

no custom setter or getters

but we can still do custom actions around the property access

```
let stepCounter = StepCounter()  
stepCounter.totalSteps = 200  
// About to set totalSteps to 200  
// Added 200 steps  
stepCounter.totalSteps = 360  
// About to set totalSteps to 360  
// Added 160 steps  
stepCounter.totalSteps = 896  
// About to set totalSteps to 896  
// Added 536 steps
```

see this: https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/TheBasics.html

```
class Counter {  
    var count = 0  
    func increment() {  
        count++  
    }  
    func incrementBy(amount: Int) {  
        count += amount  
    }  
    func reset() {  
        count = 0  
    }  
}
```

```
class Counter {  
    var count: Int = 0  
    func incrementBy(amount: Int, numberOfTimes: Int) {  
        count += amount * numberOfTimes  
    }  
}
```

see this: https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/TheBasics.html

see this: https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/TheBasics.html

Lots more on the inter-webs!

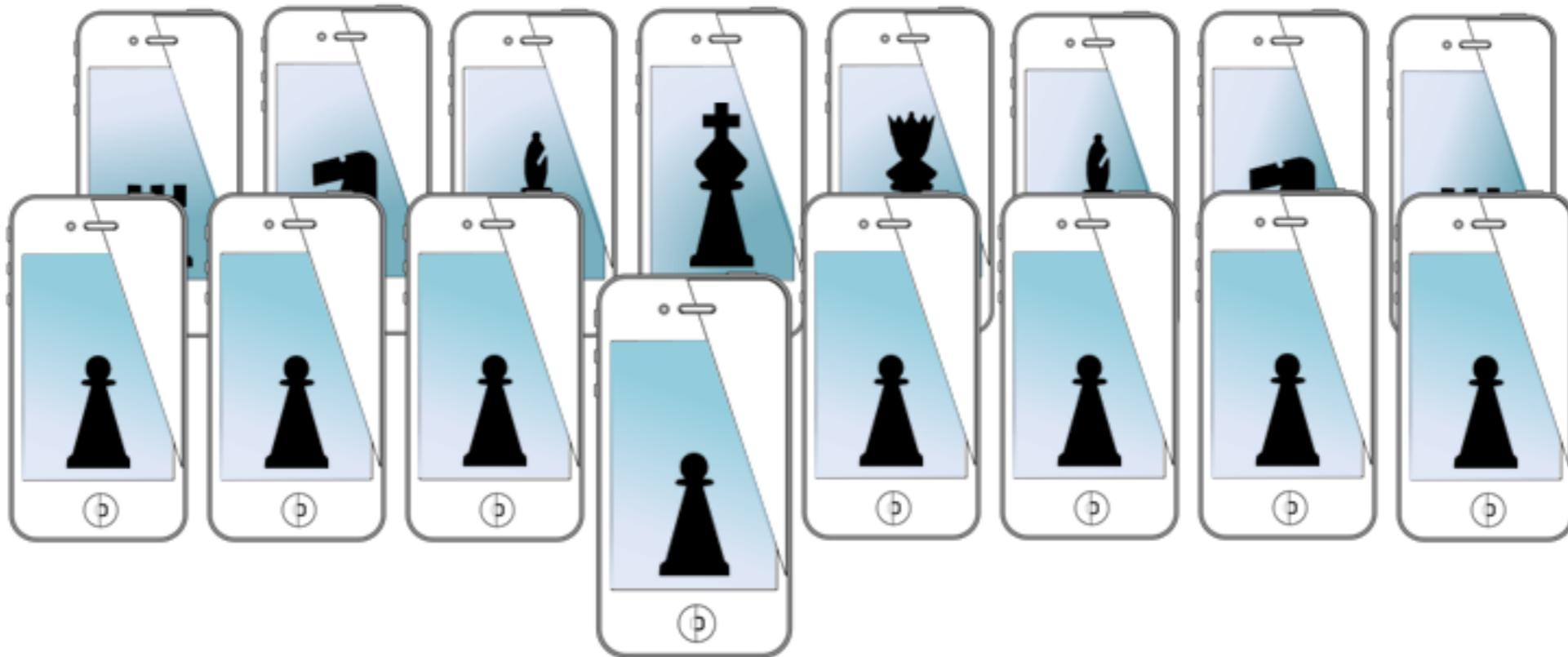
Need more help on MVC's ? Check out Ray Wenderlich:

<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