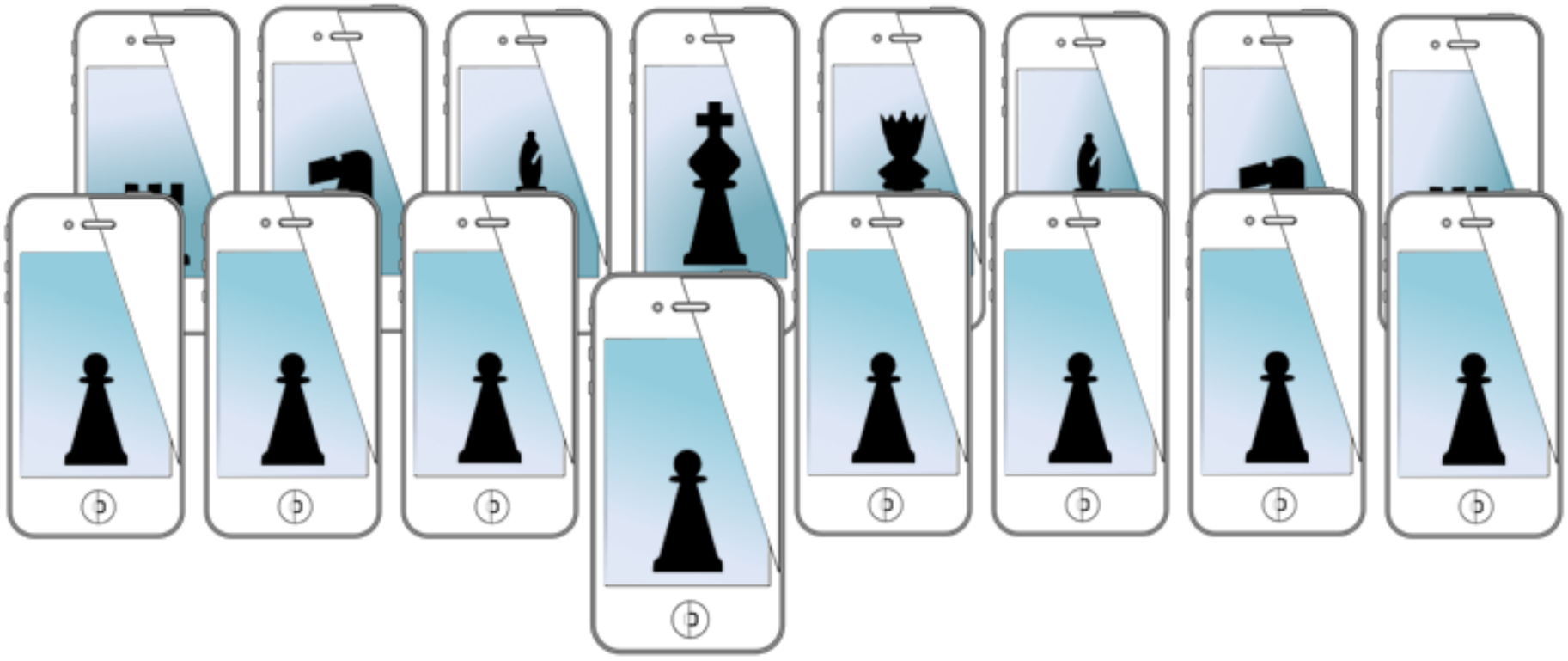
MOBILE SENSING & LEARNING



CS5323 & 7323

Mobile Sensing and Learning

objective-C, swift, and MVC

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

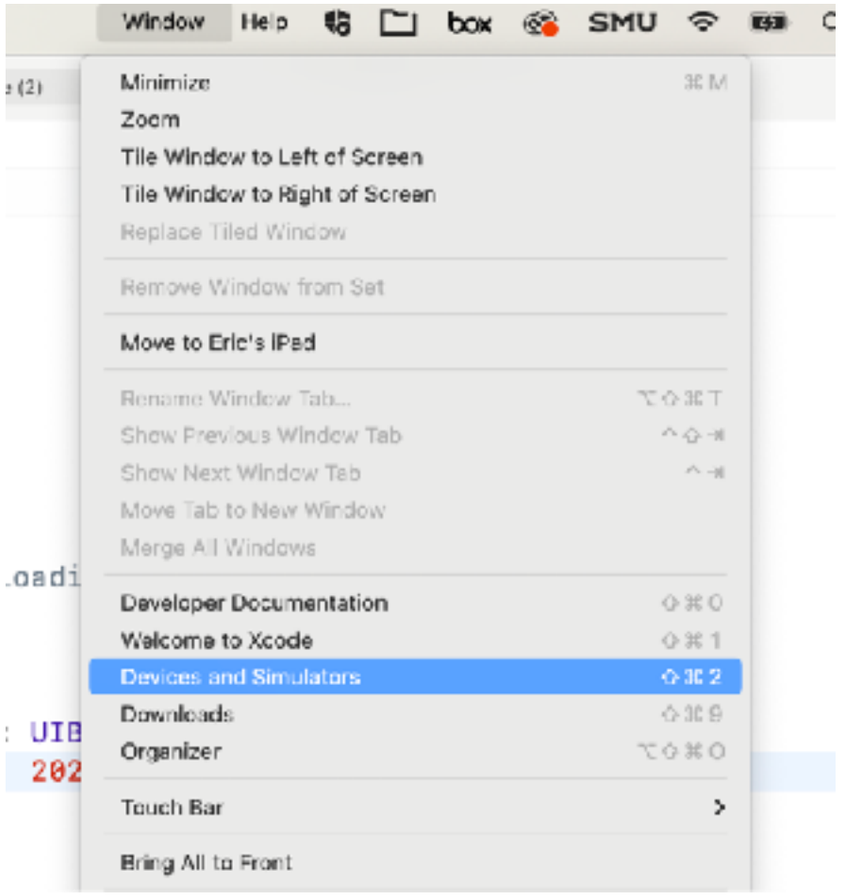
course logistics

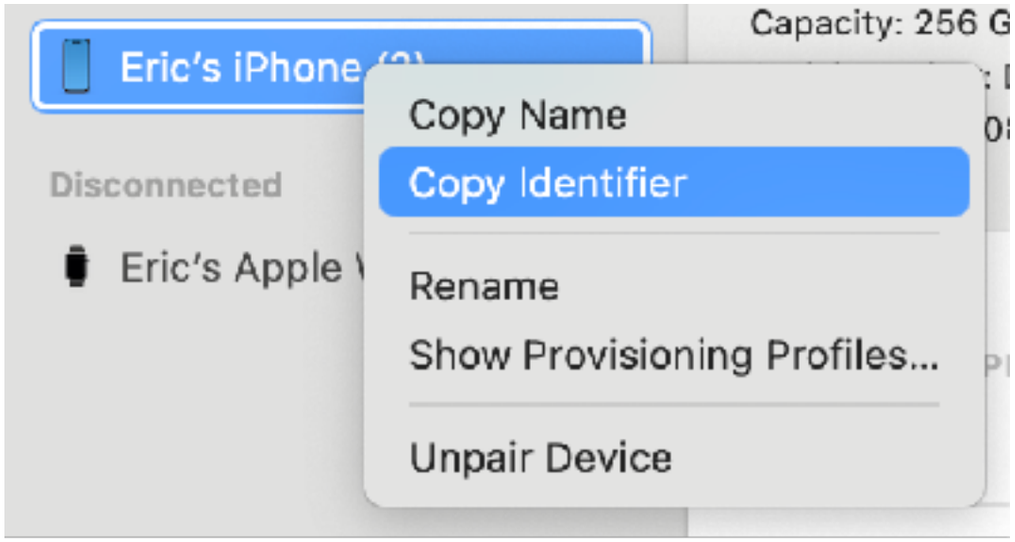
- reminder: no lab this semester
- teams: **get in a team now**! Teams can change throughout semester,
- **equipment checkout**: Phones, macs (must be on a team)
- enrollment in 5000 versus 7000 (ugrad/grad)
- Reminder: Zoom versus in-person and other classes
- Panopto videos access

Office Hours Zoom Room:	• **Instructor Office Hours**: Mondays 3:30-5:00PM, Caruth 451, CS Offices ◦ **Instructor Office Hours Zoom Room:** https://smu.zoom.us/j/97583463382 • **Teaching Assistant:** Manan Shukla (please contact via canvas) ◦ **TA Office Hours**: ▪ Tuesdays from 5:00 PM to 7:00 PM ▪ Fridays from 1:00 PM to 3:00 PM ◦ **TA Office Hours Zoom Room**: https://us04web.zoom.us/j/75504346341?pwd=8z8s5PFAArqPU3fbXtPTZpdK9hWdqP.1

Apple Developer Program

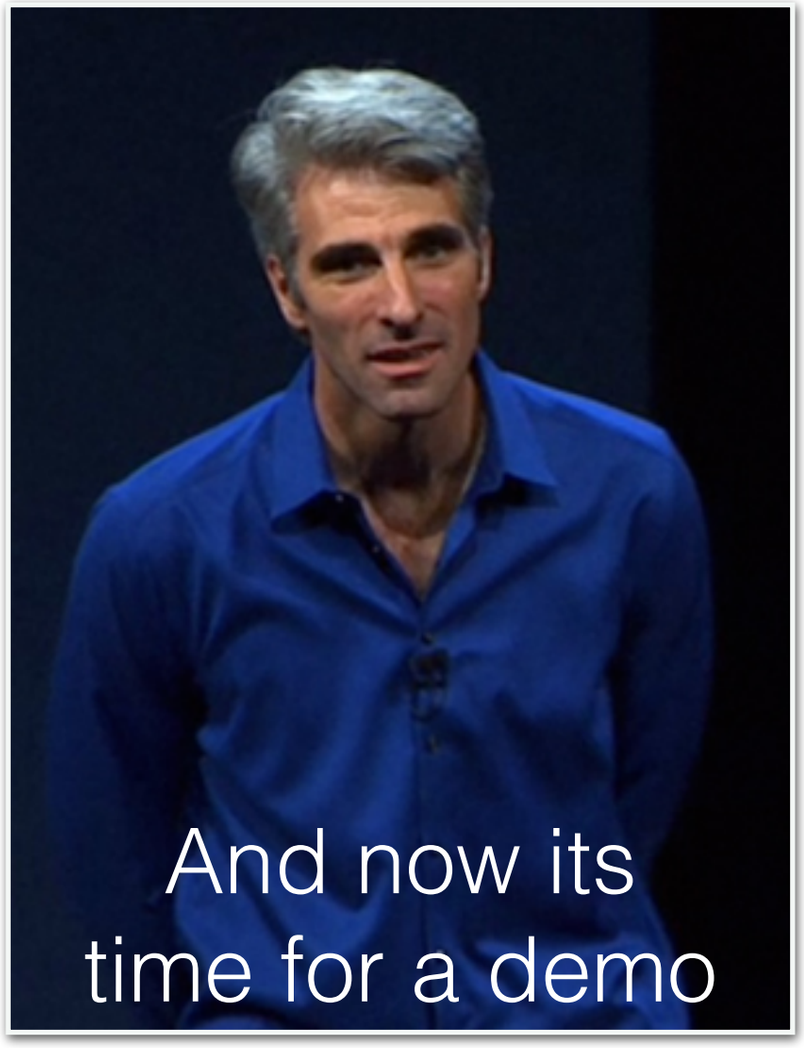
- if you update iOS, **you must update Xcode** (maybe MacOS)
- I will use my personal license, so send me:
    - **add user**: email that you want invite sent to (requires sign up)
    - **add device**: identifier, Xcode "Window>Devices and Simulators"



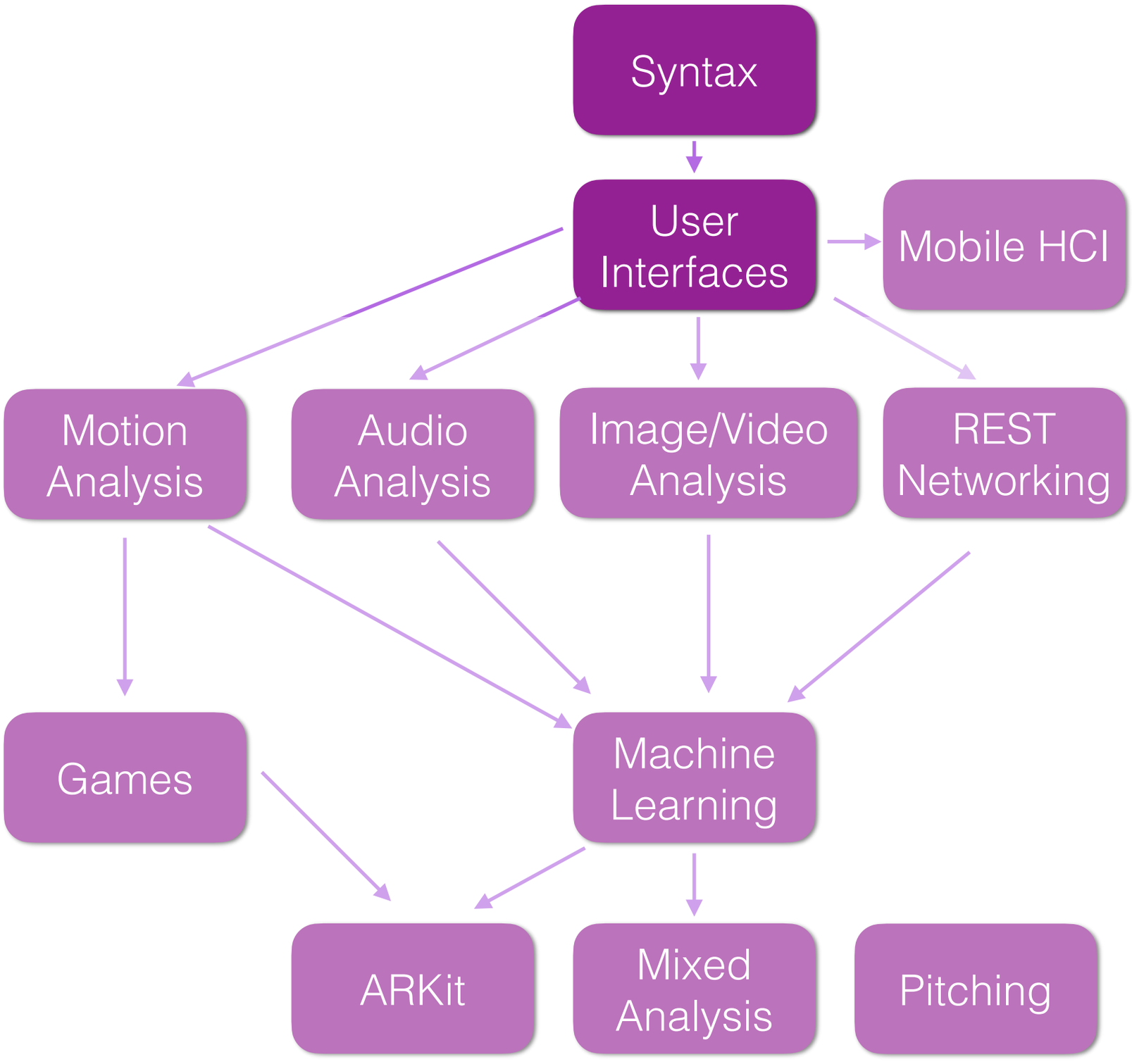


our first app with Xcode

- provides GUI for most git commands
  - commit, branch, push, pull, etc.
- **rarely** is command line needed
- git is great for code but not storyboards
- and some auto layout too!



class progression



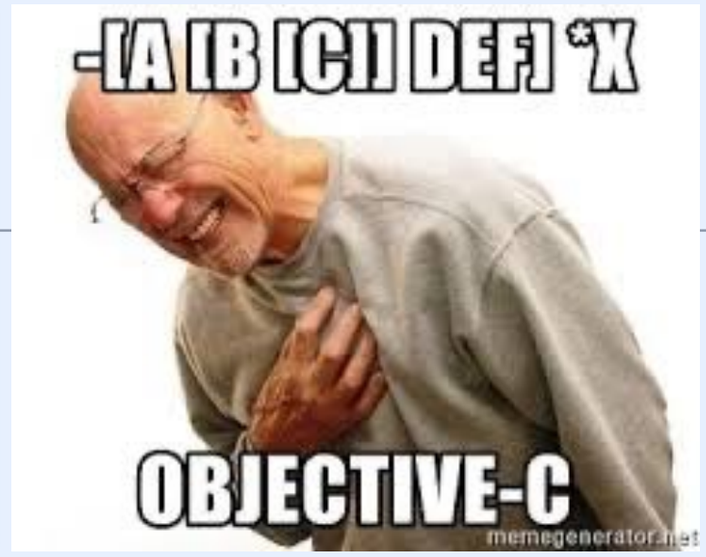
agenda

a big syntax demo…

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

and model view controllers, if time
…also available on flipped module video…

objective c



- strict superset of c
  - but with “messages”
    - so “functions” look very different (i.e., the braces in the logo)

swift

- syntax is nothing like objective-c
- but uses the same libraries…
- similarities with python syntax
  - weakly typed, no need for semicolons

an example class

```objc
@interface SomeViewController ()

@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);


}
```

```swift
class SomeViewController: UIViewController {

    lazy var aString = {
       return "This is a string \(3)"
    }()

    var aDictionary:[String : Any] = [:]

    override func viewDidLoad() {
        super.viewDidLoad()

        self.aDictionary = ["key1":3, “key2":
                    "String value"] as [String : Any]

        for (_,val) in self.aDictionary {
            print(val)
        }

        let myArray: [Any] = [32,"a string”,
                              self.aString]

        for val in myArray{
            print(val)
        }

    }
}
```

let’s work our way up to understanding both of these examples

variables, pointers, and optionals

```
aString = nil
```

```
aString = nil
```

nil
similar to NULL_POINTER, points to nothing,
can evaluate to “false” in expression

```
double aDouble;
float aFloat;
char aChar;
int aInt;
unsigned int anUnsignedInt;
...
```

Primitives
Direct Access via Stack
CANNOT be nil

```
mutable? name:Type = Value
   var aDouble:Double = 0.0
   var aFloat:Float = 0.0
   let aChar:Character = "c"
   var aInt:Int = 0
   let unsignedInt:UInt = 0
   ...
```

Next Step **Encapsulated**
Pointers to the Heap

```
                                shorthand
NSString *myString;               @“ ”
NSNumber *myNum;                  @( )
NSArray *myArray;                 @[ ]
NSDictionary *myDictionary;       @{ }
NSMutableArray *arrayYouCanMutate;
```

Swift **Optionals**
Pointers to the Heap

```
let myString:String? = "Const"
var myNum:Double? = nil
let myArray:[Any]? = nil
var arrayYouCanMutate:[Any]? = nil
var myDictionary:[String:Any]? = nil
```

classes

class name

inherits from

SomeClass.h

```
@interface SomeClass  : NSObject

@property (strong, nonatomic) NSString *aPublicStr;

@end
```

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

obj-c property:
NOT variables, but they provide *access* to backing variables

```
@interface SomeClass ()

@property (strong, nonatomic) NSString *aPrivateStr;

@end

@implementation SomeClass

   //… implementation stuff…
@end
```

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

SomeClass.m

Declared in the **.swift** file

class name

inherits from

```
class SomeClass : NSObject{

    var aPublicString = "..."
    private var aPrivateString = "..."

    // implementation stuff
}
```

swift defaults to **public properties**
must use **private** keyword

swift property:

- special variables
- can add functionality through observers and overrides

objective c

class property: access a variable in class

```
@interface SomeClass ()
@property (strong, nonatomic) NSString *aString;
@end
@implementation SomeClass
@synthesize aString = _aString;
```

property declared

backing variable: usually implicit to compiler

setter, **auto** created
`self.aString=val;`

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

getter, **auto** created
`val=self.aString;`

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

property access `self.aString`
direct access `_aString`

getter, **custom overwrites** auto creation

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

lazy instantiation

```
@end
```

objective c

class properties

```
@interface SomeClass ()

  @property (strong, nonatomic) NSString *aString;


@end

@implementation SomeClass

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

   -(void)someFunction{

       NSString *eric =_aString;

       NSString *eric2 = self.aString;

   }
@end
```

What does this do?

swift

class properties

```
class SomeClass : NSObject{

    var aPublicString = "..."
    private var aPrivateString = “..."

    var noDefaultVal:Int
    override init() {
        self.noDefaultVal = 0
    }

    lazy var aString = "Default val if not set"
    lazy var aStringAlso = {
        // could do other things here
        return "Value"
    }()


    var watchedVariable:Float = 0.0 {
        willSet(newValue){
            print("setting value to \(newValue)")
        }
        didSet{
            print("\(oldValue) set to \(watchedVariable)")
        }
    }

}
```

property declared in class directly

if no default value, must be setup in `init()`

lazy instantiation, set to values if accessed

property observers: willSet and didSet

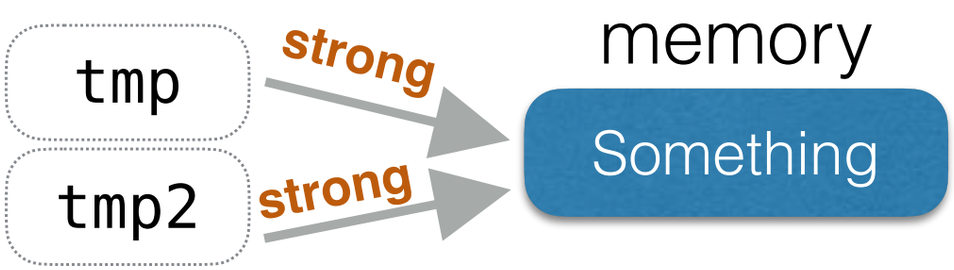
can also override “set” and “get” methods, but this is rare to need

automatic reference counting

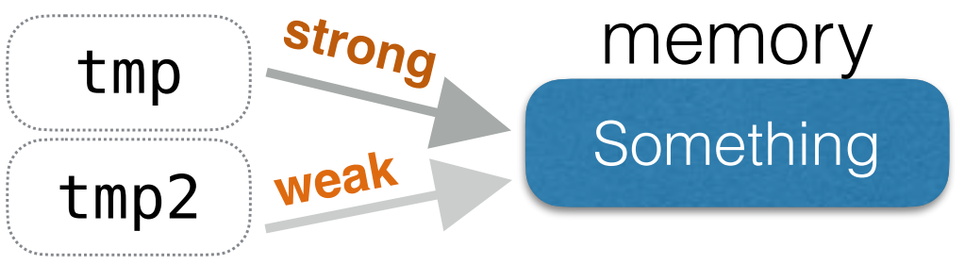
- not garbage collection
- when reference count for variable == 0, trigger event to free memory
  - **strong** pointer adds to reference count
  - **weak** pointer does not add to reference count
  - **unowned** special case of weak, always assumes there is a strong reference with longer lifetime

```
var tmp:String? = "Something"
var tmp2 = tmp
tmp = nil
tmp2 = nil
```

```
NSString* tmp = @"Something";
NSString* tmp2 = tmp;
tmp = nil;
tmp2 = nil;
```

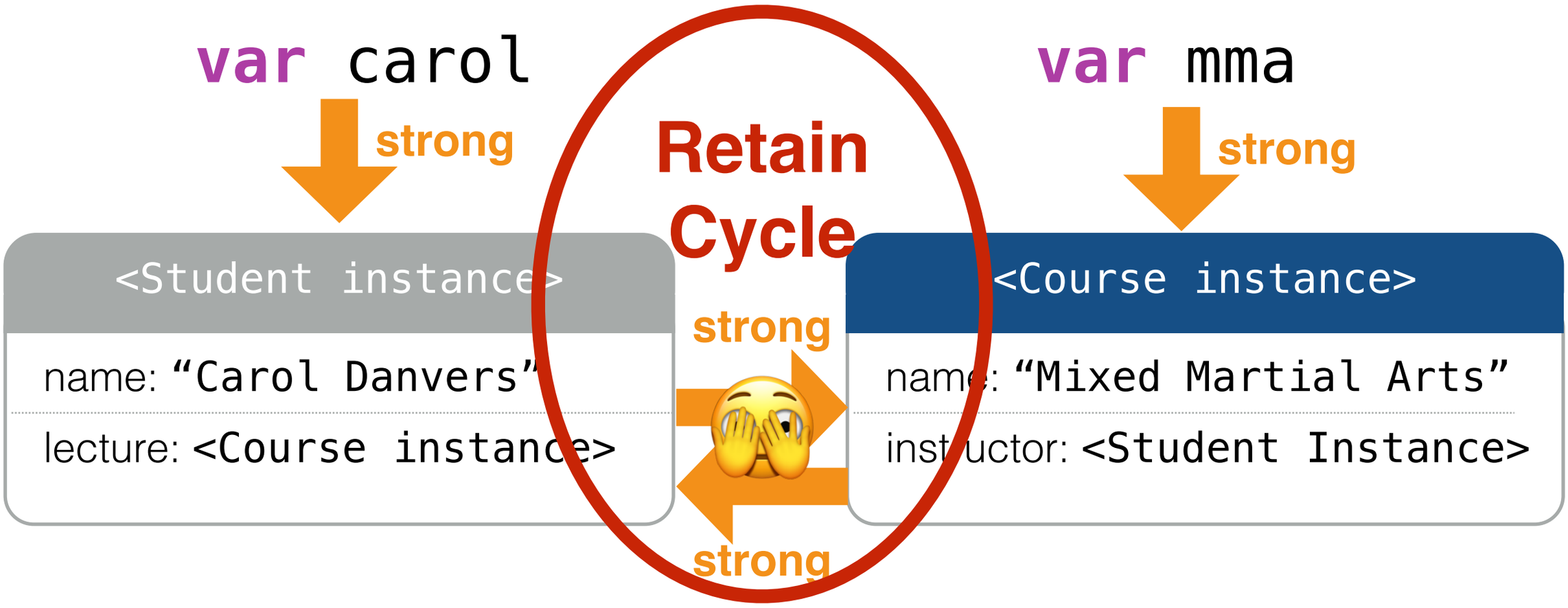


- deallocated after **both references** are `nil`



- deallocated after **strong reference** is `nil`

automatic reference counting

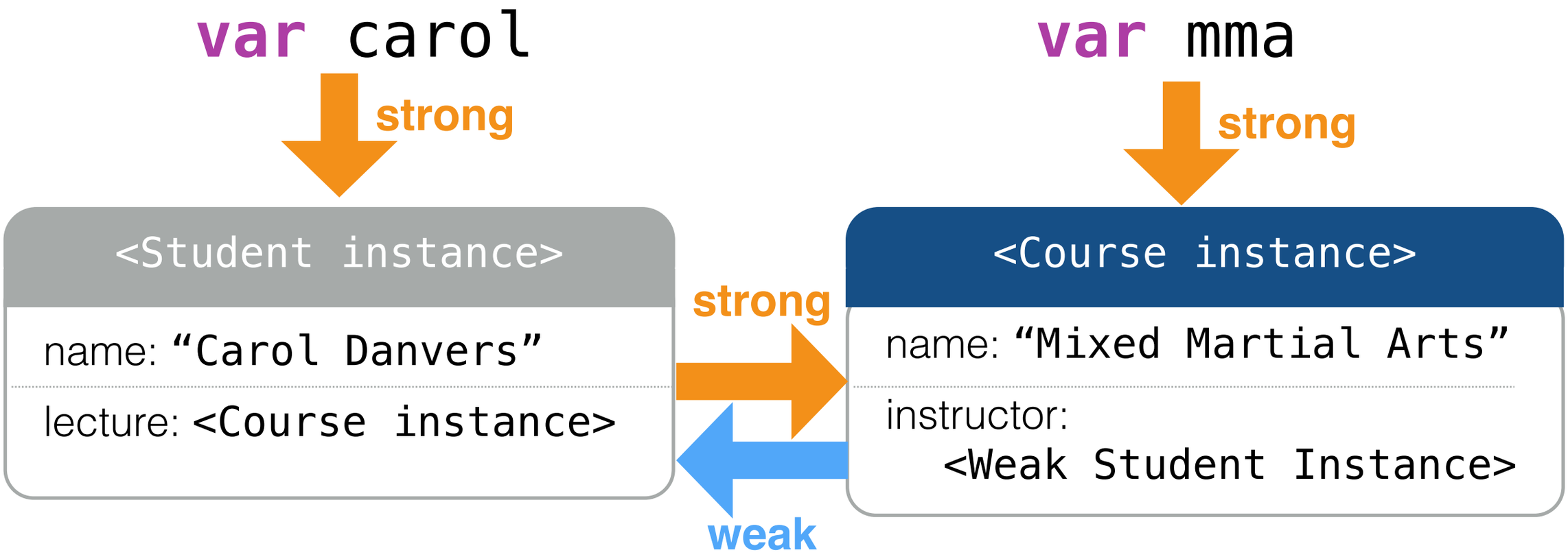



```
carol.lecture = mma
mma.instructor = carol

mma = nil
carol = nil
```

- memory never deallocated because reference cycle
- results in a memory leak if done repeatedly
- solution: weak pointers

automatic reference counting



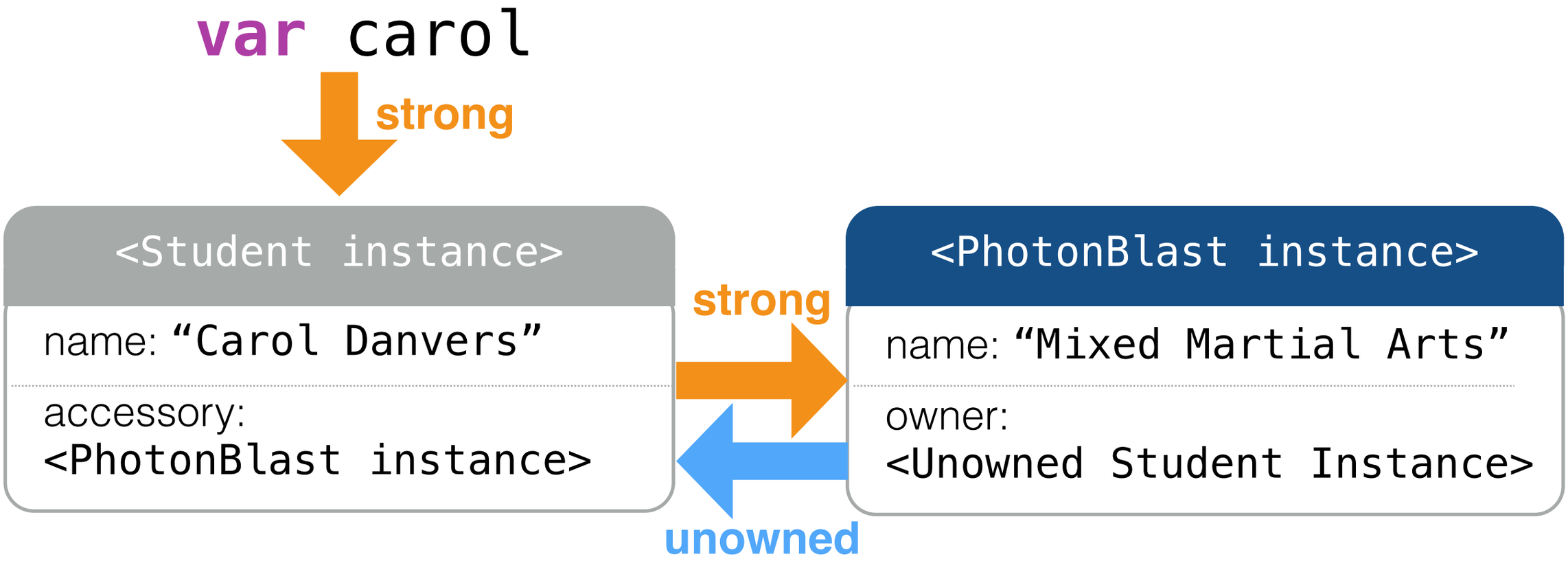

```
carol.lecture = mma
mma.instructor = carol

carol = nil
mma = nil
```

- references to parent instance cascade into properties
- all memory released immediately for use in app
- common example in Views:

```
@IBOutlet weak var someLabel: UILabel!
```

unowned usage



- used primarily when there is no need for referencing a class instance without the parent instance
- typically one-to-one class instances

using strong, weak, unowned

atomic ~ thread safe property access
nonatomic ~ faster access

```
@property (strong, nonatomic) Student *aStudent;
```

strong ~ keep a reference
weak ~ no reference

```
weak var aStudent: Student?
unowned var aStudent: Student?
```

strong by default in swift
weak used when needed

```
self.aStudent = [[Student alloc] init];
```

most common initialization
syntax for obj-c and swift

```
self.aStudent = Student()
```

properties are accessed
through `self` (like c++)

mutable and immutable

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

arrays are **nil** terminated

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

[anArrayYouCanAddTo addObject:someComplexObject];
```

possible to add objects now

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

```
let myConstArray = [34, 22, 1]
var myArray = [22, 34, 12]
```

more explicit in swift regarding mutability

iteration on objects

```
NSArray *myArray = @[@32,@"a string”, [@3.1,@3,@5,@9,@42,@32]];
for(id obj in myArray)
    NSLog(@“Obj=%@",obj);
```

can store any object

loop over an NSArray

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

Dictionary as a class property

Access self

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

```
let myArray: [Any] = [32,"a string”, self.aString]
for val in myArray{
      print(val)
}

self.aDictionary = ["key1":3, “key2":"String value"] as [String : Any]

for (_,val) in self.aDictionary {
      print(val)
}
```

declaration requires specifying **any** if the data is not consistent

Dictionary loops through as tuple **(key, varName)**

functions examples

return type | method name | parameter type | parameter name

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

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

second

receiver class | parameter name/value

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

throwback to **c**

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

float val = addOneToNumber(3.0);
```

(+ —) instance versus class method

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

```
func addOneToNumber(myNumber:Float) -> (Float){
    return myNumber+1
}

func addOneToNumber(myNum:Float, withOtherNumber myNum2:Float) -> (Float){
    return myNum+myNum2+1
}

var obj = self.addOneToNumber(myNumber: 3.0)
var obj = self.addOneToNumber(myNum: 3.0, withOtherNumber: 67)
```

(varName:Type) -> (Return Type)

similar named second parameter syntax in swift

common logging functions

function

NSString to format

object to print

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

%@ is print for serializable objects

```
someComplexObject = nil;

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

set to nothing,
subtract from reference count

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

```
var complexObj:Float? = nil

if let obj = complexObj{
    print("The value is: \(obj)")
}
```

`if let` syntax, **safely unwraps** optional

print variable within string using
`\( varName )`

review

```objc
@interface SomeViewController ()

@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);


}
```

private properties

backing variable

getter

setter

call from super class

dictionary iteration

array iteration

[C]

```swift
class SomeViewController: UIViewController {

    private lazy var aString = {
       return "This is a string \(3)"
    }()

    private var aDictionary:[String : Any] = [:]



    override func viewDidLoad() {
        super.viewDidLoad()

        self.aDictionary = ["key1":3, “key2":
                    "String value"] as [String : Any]

        for (_,val) in self.aDictionary {
            print(val)
        }



        let myArray: [Any] = [32,"a string”,
                              self.aString]
        for val in myArray{
            print(val)
        }

    }
}
```

private properties

call from super class

dictionary iteration

array iteration

adding to our project

- let's add to our project
  - an objective-c class and swift class
  - and practice using lazy instantiation

for next time…

- **next time**: more dual language programming
- **one week**: flipped assignment
- **then**: mobile HCI

MVC's

controller has direct connection to view class

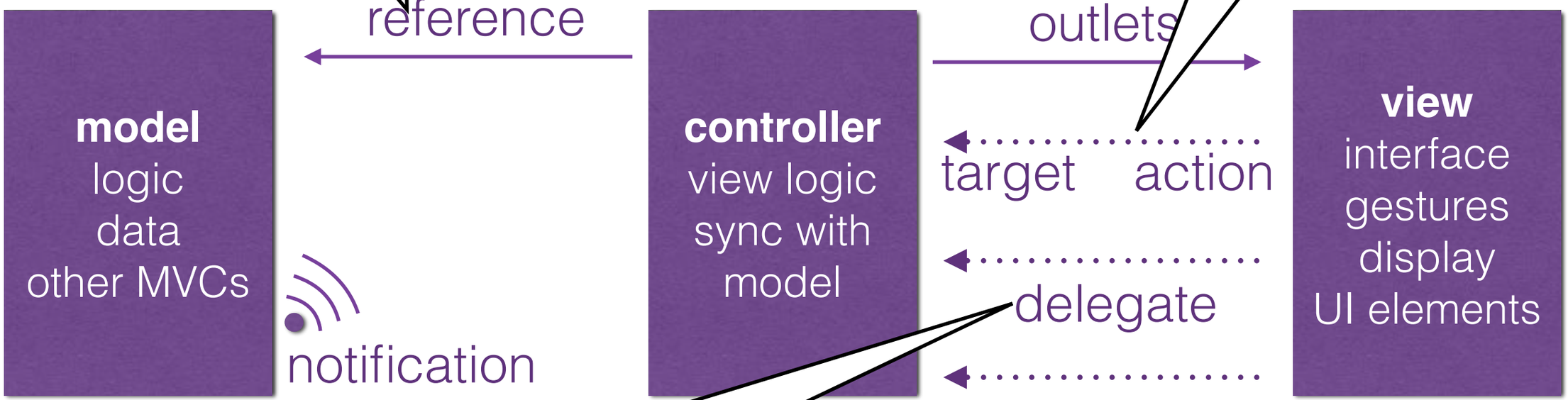
```
@property (weak, nonatomic) IBOutlet UITextField *firstName;
@property (weak, nonatomic) IBOutlet UITextField *lastName;
@property (weak, nonatomic) IBOutlet UITextField *phoneNumber;
```

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

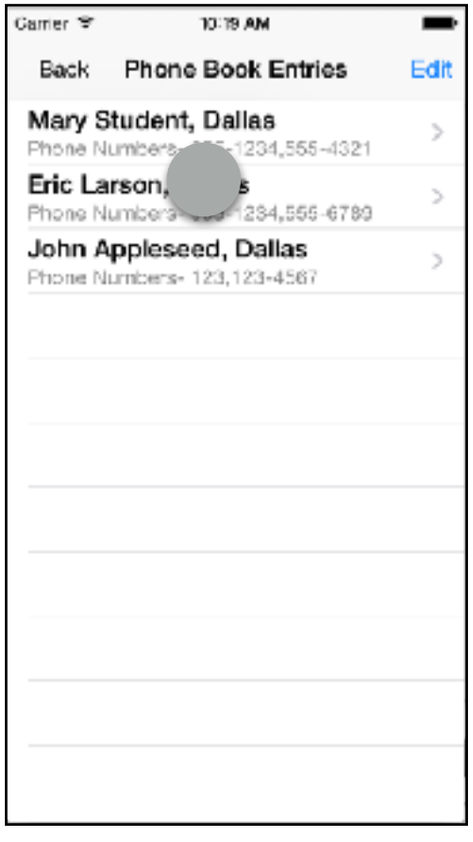
Legend

direct connection
indirect action
general broadcast

```
- (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`

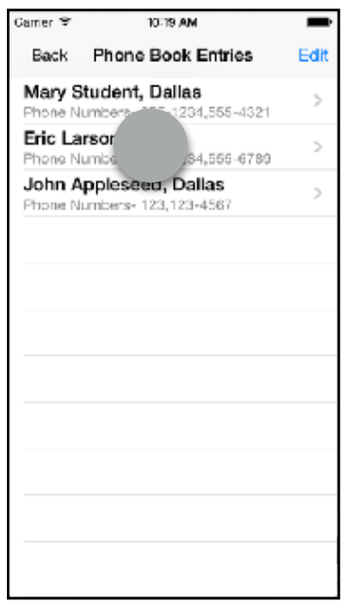
`viewDidAppear`

`viewWillDisappear`

`viewDidDisappear`

memory deallocated when app is ready

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…

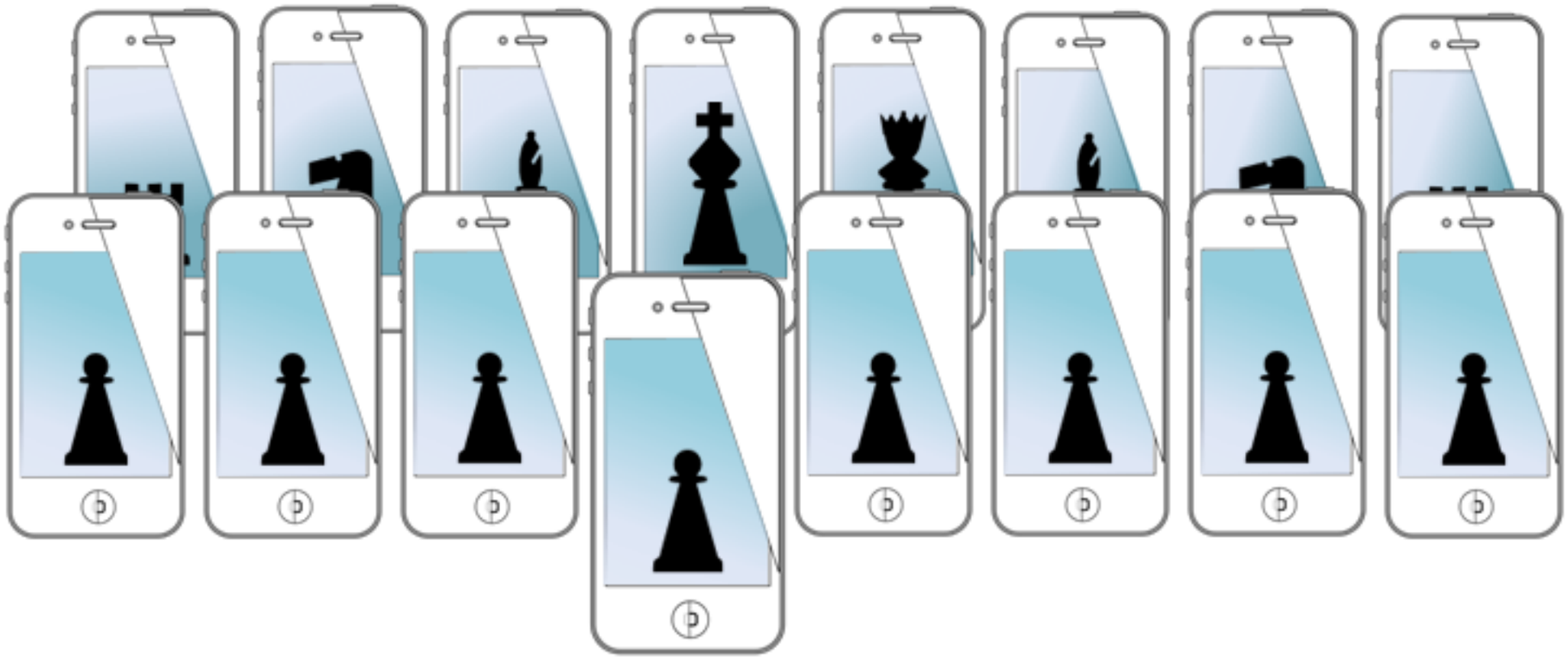
Need more help on MVC's ? Check out Ray Wenderlich:

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

for next time…

- Swift
- Mobile HCI

MOBILE SENSING & LEARNING



CS5323 & 7323

Mobile Sensing and Learning

objective-C and MVC

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