iOS Foundations II
Session 1

- Design Patterns
- MVC
- Objects vs Primitives

iOS Design Patterns

- “A software design pattern is a general reusable solution to commonly occurring problems within a given context in software design”
- No matter what kind of app it is, there are a few fundamental design pattern techniques that all apps use.
- The most important design patterns to know for iOS development:
  - Model View Controller - MVC - Governs the overall structure of your app.
  - Delegation - Facilitates the transfer of data and responsibilities from one object to another.
  - Target-Action - Translates user interactions into code your app can execute.
  - Closures/Blocks - Use these for callbacks and asynchronous code.

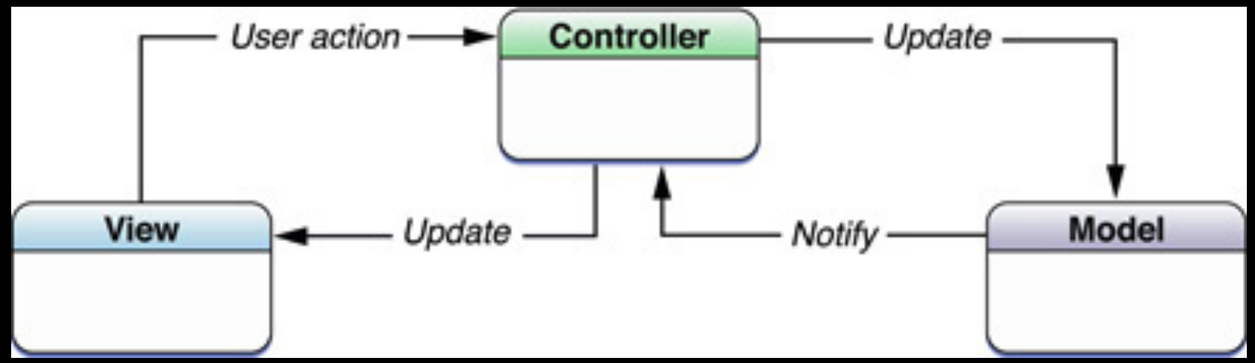
Design Patterns Are Cool

- A benefit of the universal usage of design patterns is commonality between all apps besides just the language they are programmed in.
- If someone is showing you the source code of their app, you can ask things like “What kind of model classes are you using?” or “How many View Controllers do you have?” and sound smart.

MVC (Model–View–Controller)

- A design pattern commonly used in the development of Cocoa Apps and also championed by Apple.
- Assigns objects in an application one of three MVC roles: Model, View, Controller.
- The separate layers are separated by abstract boundaries.

MVC or MCV LOL?



Some people joke its more like MCV, because the controller is the middle man so the C should go in the middle

Classic programming joke

Model Layer

- Model objects encapsulate data and logic that are going to be used by your application.
- The Twitter App has a Tweet model class, a User model class, a Favorite model class, etc.

View Layer

- A View object is an object the user can see and possibly interact with.
- Enables displaying and editing of the app's data.
- Communication between the View and Model layers is made possible by…..

Controller Layer

- Act as the intermediary between the model layer and view layer.
- The most common form of a controller in iOS is a view controller.
- Another common controller is a network controller.
- At first your view controllers will have a lot of code. Eventually you should strive to make them lighter so its easier to understand what they are doing at a glance.

The things inside of an App

- Primitives
- Objects
- Structs
- Enumerations

You will commonly see these referred to as
‘Types’

Primitive types

- Primitive Types, or Data Types, are basic types that represent things like numbers, characters, or strings.
  - “Hello World” is a String, 25 is an Int
- They are ‘built-in’ to Swift. You don’t have to define how they behave yourself.

Number Types – Int

- Int8, Int6, Int32, Int64 bit
- unsigned can’t be negative
- You usually don’t choose, just use Int!

Number Types – Float & Double

- Use Float for 32-bit, Double for 64-bit
- Double is the default
- Must alway have number on both sides of the decimal point

Number Conversions

- Fundamentally different from ObjC's conversion system
- Must cast with Type() for any form of mixing different number types
- Safer and faster

String

- var declares mutable String, let declares immutable
- No longer a reference type, copied when passed (more on this later)
- Use String Literals to easily create Strings
- String interpolation to insert Strings into other Strings

```
var name = “johnny”
name += “clem”
var githubName = “@“ + name
var gmailName = name + “@gmail.com”
```

String

String literals can include the following special characters:

- The escaped special characters `\0` (null character), `\\` (backslash), `\t` (horizontal tab), `\n` (line feed), `\r` (carriage return), `\"` (double quote) and `\'` (single quote)
- Single-byte Unicode scalars, written as `\xnn`, where `nn` is two hexadecimal digits
- Two-byte Unicode scalars, written as `\unnnn`, where `nnnn` is four hexadecimal digits
- Four-byte Unicode scalars, written as `\Unnnnnnnn`, where `nnnnnnnn` is eight hexadecimal digits

Unicode scalar character formats:

```
1 let wiseWords = "\"Imagination is more important
      than knowledge\" - Einstein"
2 // "Imagination is more important than knowledge" -
      Einstein
3 let dollarSign = "\x24"        // $,  Unicode scalar
      U+0024
4 let blackHeart = "\u2665"      // ♥,  Unicode scalar
      U+2665
5 let sparklingHeart = "\U0001F496"  // 💖, Unicode
      scalar U+1F496
```

Strings and Characters

- Strings are just unicode character arrays
- Iterating over a String

```
var name = “johnny”
for character in name {
  // do stuff with character
}
```

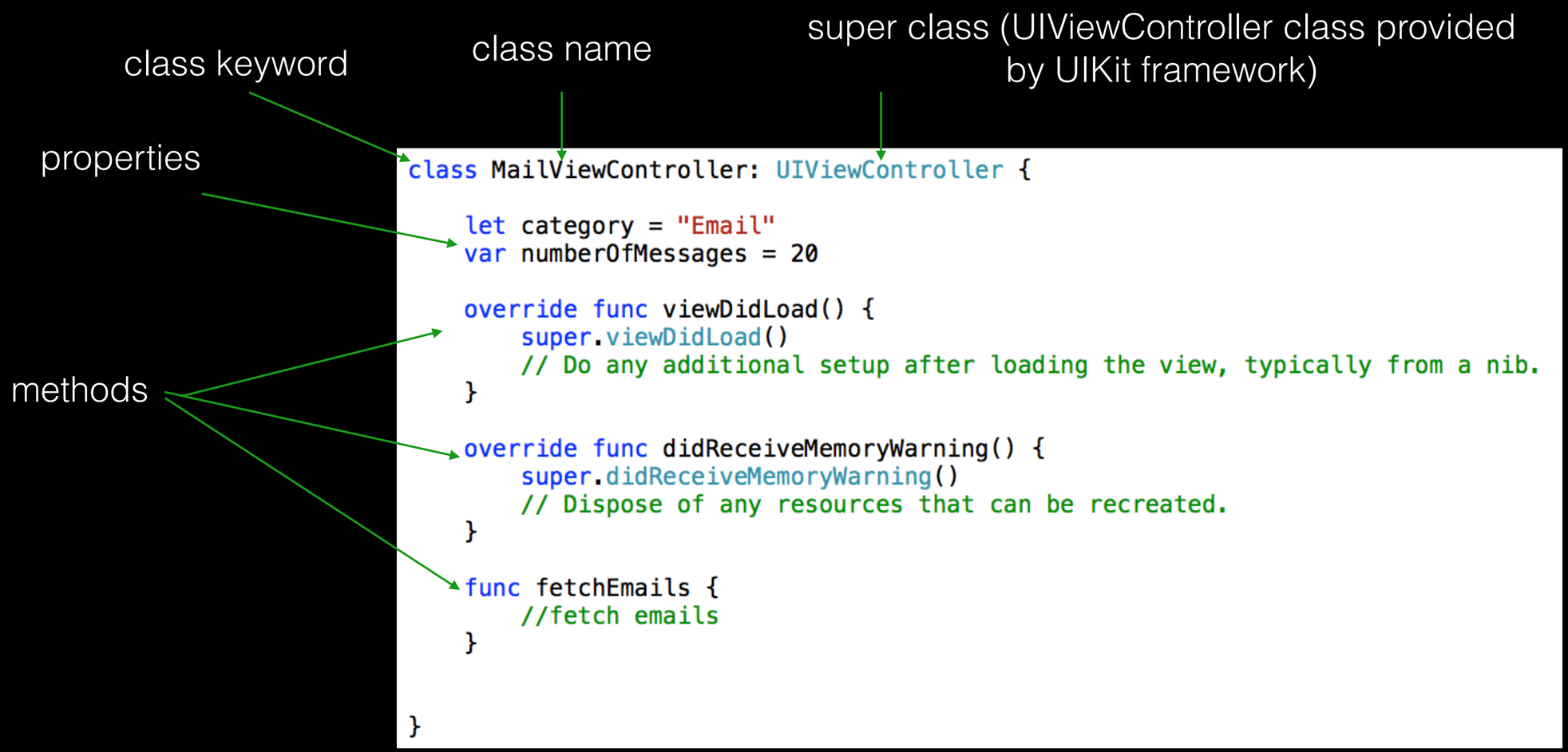
Variables & Constants
or
var and let

- var for variable variables (value can be set to a different value)
- let for constant variables (value cannot be set to different value)
- let isn't just for constant strings
- In some cases let and var are the same as mutable and immutable

Objects

- Objects are instances of Classes.
- Think of Classes as a template for an Object.
- Classes have properties to define what an object contains and methods to define what an object can do.
- Classes can inherit from other Classes. A sub-class class inherits from its super-class.
- You will use a combination of your own custom Classes and Classes provided to you in Frameworks to create your Apps.

What a class looks like



Built In Collection Classes

- Swift provides a number of collection classes to help you keep your data organized.
- Arrays and Dictionaries are the two big ones that everyone relies on.

Collection Types

Arrays

- An ordered collection of entities
- Always clear about what type of values it will store
- Array<ValueType>
- isEmpty property to check if count is 0
- append() or += to add items to end of array

Dictionaries

- An unordered key-value pairing collection (hash table)
- Always clear about type of values AND keys
- Dictionary<KeyType, ValueType>
- .keys and .values properties (for loop)
- can switch on keys now as well