iOS Dev Accelerator Week1 Day3

- AutoLayout
- SizeClasses
- Network Controller
- Singleton Pattern
- Web tools wednesday

`Autolayout`

- A constraint-based layout system for making user interfaces:
  - `dynamic` (constraint rules are calculated and applied at runtime)
  - `adaptive` (general rules for all sizes plus optional per-size rules)
  - `responsive` (changes in orientation or constraints)

Autolayout

- AutoLayout needs to know 2 things about every view in your interface:
    1. How big the view is going to be
    2. Where the object is going to be located
- You accomplish this using constraints.

Constraints

- Constraints are the fundamental building block of autolayout.
- Constraints contain rules for the layout of your interface's elements.
- You could give a 50 point height constraint to an imageView, which constrains that view to always have a 50 point height. Or you give it a constraint to always be 20 points from the bottom of its superview.
- Constraints can work together, but sometimes they conflict with other constraints.
- At runtime Autolayout considers all constraints, and then calculates the positions and sizes that bests satisfies all the constraints.

Attributes

- When you attach constraints to a view, you attach them using attributes.
- The attributes are: **left/leading, right/trailing, top, bottom, width, height, centerX, centerY.**
- So if you attach a constraint of 50 points from a button’s left attribute to its container’s left attribute, thats saying “I want this button to be 50 points over from its super view’s left side”

Constraints + Attributes = Math time

- “You can think of a constraint as a mathematical representation of a human-expressable statement”
- So if you say “the left edge should be 20 points from the left edge of its containing view”
- This translates to button.left = container.left x 1.0 + 20
- which is in the form of y = mx+b
- first attribute = second attribute * multiplier + constant
- In the case of an absolute value, like pinning height, width, centerX, or centerY, the second attribute is nil.
- You can change the constants in code as an easy way to programmatically adjust your interface.

Storyboard and Autolayout

- Storyboard makes setting up autolayout pretty intuitive and painless, and even though you can setup autolayout completely in code, Apple strongly recommends doing it in storyboard.
- Xcode will let you build your app even if you have constraints that are conflicting and incorrect, but Apple says you should never ship an app like that.
- When you drag a object onto your interface, it starts out with no constraints.

Intrinsic Content Size

- Intrinsic content size is the minimum size a view needs to display its content.
- Its available for certain UIView subclasses:
    - UIButton & UILabel: these views are as large as they need to display their full text
    - UIImageView: image views have a size big enough to display their entire image. This can change with its content mode.
- You will know a view has an intrinsic content size if autolayout doesn't require its size to be described with constraints.

Demo

Size Classes

Size Classes

- “Size classes are traits assigned to a user interface element, like a screen or a view”
- There are only two types of size classes, Regular and Compact.
- Size classes, together with displayScale and userInterfaceIdiom (iPhone or iPad) make up a trait collection.
- Everything on screen has a trait collection, including the screen itself, and view controllers as well.
- The storyboard uses a view controller’s trait collection to figure out which layout should be currently displayed to the user.

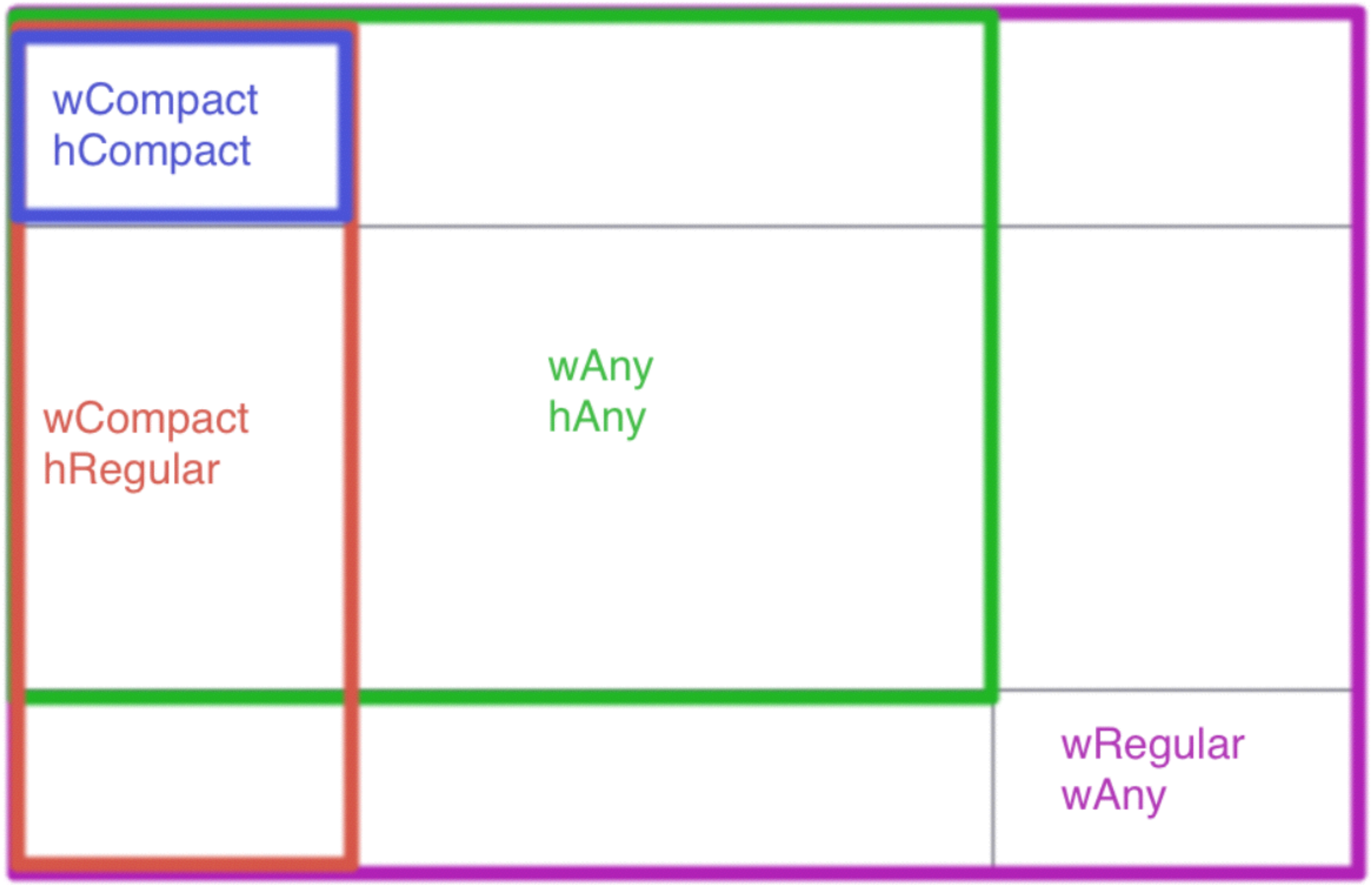
Size Classes and Storyboard

- Size classes allow you to have different constraints and layouts for each configuration on the storyboard.
- By default, every size class configuration will pull from the base configuration, which is wAny hAny.
- If you change your storyboard’s configuration, certain changes you make will only apply when your app is running in that specific size class.

Size Classes

- Specifically, there are 4 things you can change in each configuration on your storyboard:
    1. constraint constants.
    2. font and font sizes
    3. turning constraints off
    4. turning view on and off

Size Classes



- iPad Landscape and Portrait
- Base configuration
- iPhone Portrait
- iPhone Landscape

Demo

Network Controller

Network code

- Right now we have our network code inside of our view controllers.
- This works, but what if a bunch of different view controllers are making the same network calls? Thats not efficient.
- Anytime you have duplicate code anywhere, that is a ‘code smell’ or a hint that you need to do some refactoring.

Network code best practices

- There are two spots that experienced iOS developers like to put their network code:
  - In the models classes themselves.
  - In a consolidated network class.
- I greatly prefer the consolidated network class, because it makes our model classes a lot less complicated, and I like only having to go to one place for network code issues/debugging.
- Theres a few name patterns for a class like this:
  - <Name of the web api>Service (ex: TwitterService)
  - <Name of the web api>API (ex: TwitterAPI)
  - NetworkController
  - NetworkManager

Demo

Singleton Pattern

Singleton Pattern

- “A singleton class returns the same instance no matter how many times an application requests it”
- Any regular class will allow callers to create as many instance of that class as they want.
- The singleton object provides a global point of access to the resources of its class.
- If your class provides some general service or resource, consider the singleton pattern.
- Apple uses singleton pattern for a number of different classes, such as NSFileManager and UIApplication.

Demo

Web Tools Wednesday: Environment Setup

Welcome to Web Tools Wednesday

- The purpose of these days is to introduce you to Javascript, Node, and server side programming.
- This will make working with web API’s make a lot more sense.
- And javascript/node will look great on your resumes.
- Everybody wins, except the people who aren't here for web tools Wednesday. They lose.

Environment setup

- We are spoiled as iOS developers, as the only tool we really need is Xcode.
- Not so much for nodeJS developers.
- We will go step by step and download all the tools you will need for node development.

Homebrew

- Homebrew is a package manager used for installing software on the Mac OS X operating system.
- Go to brew.sh for installation instructions.
- But really just go into terminal and type brew install wget and hit enter.

NVM (Node Version Manager)

- NVM is used for installing and managing different version of node and installing local version into repos.
- Use home-brew to install nvm
- Type brew install nvm into your terminal and hit enter

node.js install

- we can now use nvm to install node.js onto your computer.
- in terminal, type nvm install 0.10.32
- This installs the latest version of node
- You can then type node —version into your terminal to see which version of node you have installed

MongoDB

- We will be using mongoDB as the persistence layer of our web app.
- We can use home-brew to install monogodb
- Type brew install mongodb