iOS DeCal

lecture 7

firebase + midsemester review

cs198-001 : spring 2018

today’s lecture

- announcements
- firebase
- mid-semester review
  - we’ll ask some questions, talk to the people next to you

announcements

- custom app proposal extension
  - due TOMORROW at 11:59pm
- snapchat clone part 2 will be release before lab
  - due after spring break (4/2)
- this week’s lab - help with setting up snapchat clone part 2
- final presentation day not finalized. Either Thursday or Friday of dead week

Overview : Today's Lecture

Sync vs. Async

Recap Closures

Intro to Firebase and BaaS

- Managing Users
- Saving and Retrieving Data
- File Storage

Sync vs. Async Tasks

The problem with network requests

Fact #1: Network requests are *slow.*

The problem with network requests

Fact #1: Network requests are *slow.*

Fact #2: Users hate waiting.

The problem with network requests

Fact #1: Network requests are *slow.*

Fact #2: Users hate waiting.

Fact #3: Users are mean.

The problem with network requests

Fact #1: Network requests are *slow.*
Fact #2: Users hate waiting.
Fact #3: Users are mean.

- We have almost no control over the time it takes to make a request to a server and wait for its response (esp. with poor internet conn.).
- Our goal is to minimize the latency that the user actually sees at any point.
  - Users should never have to sit on a frozen screen.

Synchronous Tasks

Blocks a process until the task is complete

Pros:

- Guarantee that we get results before going on to the next task.
- Somewhat easier implementation (don't have to worry about thread management).

Cons:

- User has to wait for task to finish before being able to do anything else.
  - USERS HATE WAITING!!!

Synchronous Tasks

Blocks a process until the task is complete

Pros:

- Guarantee that we get results before going on to the next task.
- Somewhat easier implementation (don't have to worry about thread management).

Cons:

- User has to wait for task to finish before being able to do anything else.
  - USERS HATE WAITING!!! (& are mean)

Synchronous Tasks: Example

```
func retrieveData {
    let query = PFQuery(classname: "cats")
    let cats = query.findObjects() //
        synchronous call
    for cat in cats {
        // do something
    }
}
```

Asynchronous Tasks

Run out of order, in parallel with the main thread so that code can continue to execute while waiting.

- iOS apps perform network requests in the background
  - Example: loading a TableView and refreshing it once data is returned.
- Introduces a new challenge:
  - What if the next line of code after the network request is evaluated before the request finishes?

Closures Revisited

Closures: self-contained blocks of functionality that can be passed around in your code.

This means we can pass functions around as parameters to other functions!

Why might this be useful for solving our async task problem?

Using Closures as Completion Handlers

Suppose we made an asynchronous network request and wanted to trigger an action only **after** we knew the request had completed.

```
func retrieveData {
    let params = [1, 2, 3]
    makeRequest(params: params, completion:
        { (data) in
        if let data = data {
            // do something with data
        }
    })
    // continue rest of code here
}
```

Implementing functions with completion handlers

We usually look at functions with completion handlers as "black boxes" - we assume they do the heavy lifting, and we just tell them what to do at the end.

- What are they doing behind the scenes?

```
func makeRequest(params: [Int], completion:
    @escaping (Data?) -> Void) {
    // make some API call
    // get data back from call
    if success {
        completion(data)
    } else {
        completion(nil)
    }
}
```

Firebase

How is data usually stored?

Option 1 – Remotely: Make requests to server and let it do the dirty work of saving/retrieving from a database.

Option 2 – Locally: Use something like SQLite or CoreData independently from a server but with more tedious work in terms of actually managing the database.

Using BaaS tools

Backend as a Service tools provide backend cloud storage support to mobile developers through simple API calls.

- Abstracts away the complexities of database implementation
- No need to write any server-side code
- Many offer a lot of additional tools that simple MySQL/ SQLite databases don't support

Using BaaS tools

Firebase

amazon web services™

Backend as a Service tools provide backend cloud storage support to mobile developers through simple API calls.

- Abstracts away the complexities of database implementation
- No need to write any server-side code
- Many offer a lot of additional tools that simple MySQL/SQLite databases don't support

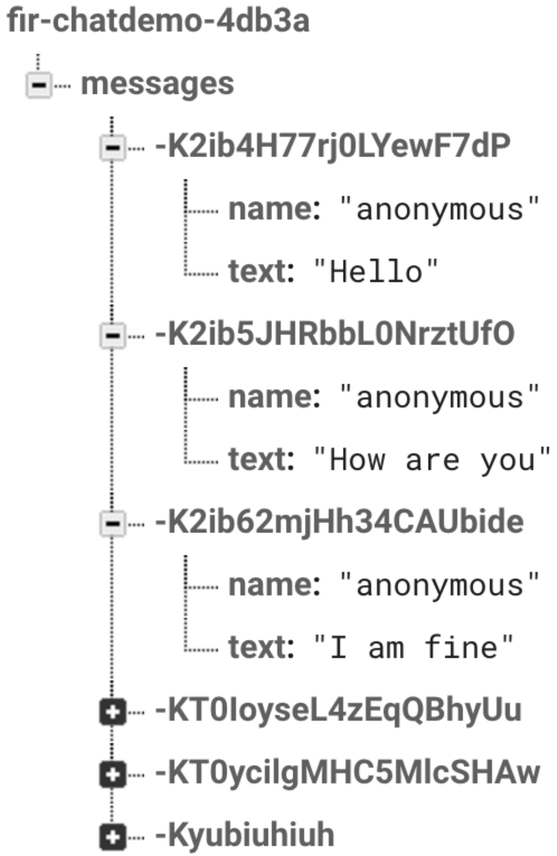
Why Firebase?

1. It's real-time! Allows us to update the view as soon as something in the database changes
2. Has strong support for iOS and Android as well as Web, Unity, C++
3. Thorough documentation - see https://firebase.google.com/docs/ios/setup
4. Can be easily incorporated into project via Cocoapods
5. Supports not only simple data storage but also authentication, file storage, cloud messaging, analytics, and more.
6. Biggest competitor, Parse, shut down in 2015.

How does Firebase work?

Firebase is built on a NoSQL database

- Literally no SQL involved - data stored as a JSON tree



- Data represented as a set of nodes, each with corresponding child nodes
- Retrieve data within app as a dictionary with key-value pairs.

Configuring Firebase

Initializing Firebase

```
import Firebase

class AppDelegate: UIApplicationDelegate... {

    func application(_ application: UIApplication,
didFinishLaunchingWithOptions...) -> Bool {

        FirebaseApp.configure()

         return true
    }
}
```

Managing Users

User-Driven Data

For any application, we need to be able to:

- Create accounts for users
- Store a user's authentication state
- Store a user's basic information (name, profile pic, etc)
- Associate data objects (messages, photos, etc.) to the user who created them.

Firebase allows us to handle this by assigning unique user ID's

Firebase Users

For any user, Firebase (can) store:

- A unique user ID
- Email address
- Display name
- Photo URL

Firebase maintains a `FIRUser` instance which keeps track of the current user.

- Persists the user's state so that closing the app or losing connection doesn't sign the user out.

Creating a new user

```
import FirebaseAuth

Auth.auth().createUser(withEmail: emailText, password:
  passwordText) { (user, error) in

  if let error = error {
    // handle error
  } else {
    // do something
  }

}
```

Signing in

```
Auth.auth().signIn(withEmail: emailText, password:
passwordText) { (user, error) in

  if let error = error {
    // handle error
  } else {
    // do something
  }

}
```

Setting a user's display name

```
let changeRequest = user!.createProfileChangeRequest()
changeRequest.displayName = newName

changeRequest.commitChanges() { (error) in
  if let error = error {
    // handle error
  }

}
```

Getting the current user

If we want to access the properties of the currently signed in user, we can do something like:

```
let user = Auth.auth().currentUser
let name = user?.displayName
let email = user?.email
let uid = user?.uid
let photoURL = user?.photoURL
```

We can also use the currentUser variable to check if a user is already signed in (instead of logging in every time). However, it is safer to use a listener:

```
Auth.auth().addStateDidChangeListener() { (auth, user) in
  // do something with user
}
```

Saving/Retrieving Data

Structuring Data

Recall that data is stored on Firebase as a JSON tree.

- Each time we add data to the tree, it becomes a node in the tree with a key and value.
- We can access a value in the tree by following its key-path in the tree.
- If we attempt to access a node in the database, we get access to all of its children as well.
  - Potential pitfalls of this?

```
"chats": {
  "one": {
    "title": "Historical Tech Pioneers",
    "messages": {
      "m1": { "sender": "ghopper", "message": "Relay malfunction found. Cause:
      "m2": { ... },
      // a very long list of messages
    }
  },
  "two": { ... }
}
```

Writing Data to Firebase

Create a reference to the root node:

```
import FirebaseDatabase
let dbRef = Database.database().reference()
```

Save data to a node:

```
dbRef.child("Users").child(user.uid).setValue(username,
  forKey: "username")
```

We can also specify the entire path directly:

```
dbRef.child("Users/\(user.uid)/username").setValue(username)
```

Save multiple values to a node:

```
let dict: [String: Any] = ["email": email,
                           "preferences": preferences]
dbRef.child("Users/\(user.uid)").setValue(dict)
```

Note: You can also create unique ID's for children using the childByAutoID function.

Reading Data from Firebase

Create a listener (called when a particular node changes):

```
let refHandle = dbRef.child("Users/\(user.uid)").observe(.value,
  with: { (snapshot) in

  if snapshot.exists() {
    if let userDict = snapshot.value as? [String : Any] {
       let newValue = userDict["username"] as! String
     }
  }
})
```

Note that the code inside the closure will execute *every* time the user's node on Firebase (or any of its children) changes.

- We can also query Firebase a single time by calling the observeSingleEvent function instead.

Storing Files

How does Firebase store files?

Firebase's database is only capable of storing numbers, arrays, dictionaries, and strings.

What if we want to store an image? (e.g. Snapchat Clone)

Firebase has a separate module for storage where we can upload all of our files - then we can just store its path in the storage section as a string in the database.

Store an image on Firebase

Just like with the database, we need a reference to the root node of the storage module:

```
import FirebaseStorage
let storageRef = Storage.storage().reference()
```

Then we can upload a file to a specific path as:

```
storageRef.child("images/img.jpg").putData(imageData, metadata: nil) {
error) in

  guard let metadata = metadata else {
    //handle error
    return
  }
  let downloadURL = metadata.downloadURL
}
```

Download an image from Firebase

We can download an image either by using its path:

```
let imageRef = storageRef.child("images/img.jpg")
imageRef.getData(maxSize: 1*1024*1024) { (data, error) in
  if let error = error {
    // handle error
  } else {
    let image = UIImage(data: data!)
  }
}
```

Or by its download URL:

```
let imageRef = Storage.storage().reference(forURL: downloadURL)
imageRef.getData(maxSize: 1*1024*1024) { (data, error) in
  if let error = error {
    // handle error
  } else {
    let image = UIImage(data: data!)
  }
}
```

Check In!

Swift

Syntax

This code doesn’t compile. What’s the problem?

```
func foo(with bar: Double) -> Int {
    return Int(bar + 198.001)
}

let x = foo(bar: 5.0)
```

Syntax

This code doesn't compile. What's the problem?

```
func foo(with bar: Double) -> Int {
    return Int(bar + 198.001)
}

let x = foo(with: 5.0)
```

bar is an internal parameter

! vs ?

What is the difference between types ending with a “!” and a “?”

! vs ?

What is the difference between types ending with a “!” and a “?”

? - can take on a value of `nil`
! - cannot take on value of `nil`

Classes versus structs

Question: What's the main difference between a Class and a Struct?

Classes versus structs

Question: What's the main difference between a Class and a Struct?

Classes - Pass by reference
Structs - Pass by value

Value Types

Question: What are some other value types?

Value Types

Question: What are some other value types?

Structs
Enums
Arrays
String
Dictionary
...

Reference Types

Question: What about reference types?

Reference Types

Question: What about reference types?

Classes
Closures

Syntax

Will this compile?

```
var threads: [String: [Post]] = ["Memes": []]

let p = Post(username: "Oski", read: false)

let memeThread = threads["Memes"]!
memeThread.append(p)

print(threads["Memes"]!.count)
```

Syntax

Will this compile?

```
var threads: [String: [Post]] = ["Memes": []]

let p = Post(username: "Oski", read: false)

let memeThread = threads["Memes"]!
memeThread.append(p)

print(threads["Memes"]!.count)
```

Nope

Syntax

What does this print?

```
var threads: [String: [Post]] = ["Memes": []]

let p = Post(username: "Oski", read: false)

var memeThread = threads["Memes"]!
memeThread.append(p)

print(threads["Memes"]!.count)
```

Syntax

What does this print?

```
var threads: [String: [Post]] = ["Memes": []]

let p = Post(username: "Oski", read: false)

var memeThread = threads["Memes"]!
memeThread.append(p)

print(threads["Memes"]!.count)
```

0

Syntax

What does this print?

```
var threads: [String: [Post]] = ["Memes": []]

let p = Post(username: "Oski", read: false)

threads["Memes"]!.append(p)

print(threads["Memes"]!.count)
```

1

Syntax

And this? (`Post` is a `struct`)

```
var threads: [String: [Post]] = ["Memes": []]

let p = Post(username: "Oski", read: false)

threads["Memes"]!.append(p)

p.username = "Osky"

print(threads["Memes"]!.first!.username)
```

Syntax

And this? (`Post` is a `struct`)

```
var threads: [String: [Post]] = ["Memes": []]

var p = Post(username: "Oski", read: false)

threads["Memes"]!.append(p)

p.username = "Osky"

print(threads["Memes"]!.first!.username)
```

Syntax

And this? (`Post` is a `struct`)

```
var threads: [String: [Post]] = ["Memes": []]
var p = Post(username: "Oski", read: false)
threads["Memes"]!.append(p)
p.username = "Osky"
print(threads["Memes"]!.first!.username)
```

Oski

Syntax

And what if `Post` is a `class`?

```
var threads: [String: [Post]] = ["Memes": []]

var p = Post(username: "Oski", read: false)

threads["Memes"]!.append(p)

p.username = "Osky"

print(threads["Memes"]!.first!.username)
```

Syntax

And what if `Post` is a `class`?

```
var threads: [String: [Post]] = ["Memes": []]
var p = Post(username: "Oski", read: false)
threads["Memes"]!.append(p)
p.username = "Osky"
print(threads["Memes"]!.first!.username)
```

Osky

Syntax

Finally, what is socially unacceptable?

```
var threads: [String: [Post]] = ["Memes": []]

var p = Post(username: "Oski", read: false)

threads["Memes"]!.append(p)

p.username = "Osky"

print(threads["Memes"]!.first!.username)
```

Syntax

Finally, what is socially unacceptable?

```
var threads: [String: [Post]] = ["Memes": []]
var p = Post(username: "Oski", read: false)
threads["Memes"]!.append(p)
p.username = "Osky"
print(threads["Memes"]!.first!.username)
```

force unwrapping optional variables

Variables

```
import UIKit

var view1 = UIView()
view1.alpha = 0.5

let view2 = UIView()
view2.alpha = 0.5 // will this line compile?
```

Question: Will the last line compile? Why or why not?

credit: Antonio Bello

Variables

```
import UIKit

var view1 = UIView()
view1.alpha = 0.5

let view2 = UIView()
view2.alpha = 0.5 // will this line compile?
```

Question: Will the last line compile? Why or why not? Yes it will. Even though view2 is declared with let, it's properties are mutable

credit: Antonio Bello

Storyboard

Prepare for Segue

Question: Can I do this?

```
override func prepare(for segue: UIStoryboardSegue,
sender: Any?) {

   if let dest = segue.destination as?
ChooseThreadViewController {
     dest.chosenImageView.image = selectedImage
   }

}
```

Prepare for Segue

Question: Can I do this?

```
override func prepare(for segue: UIStoryboardSegue,
sender: Any?) {

   if let dest = segue.destination as?
ChooseThreadViewController {
     dest.chosenImageView.image = selectedImage
   }

}
```

No! Why?

Prepare for Segue

Question: Is this better?

```
override func prepare(for segue: UIStoryboardSegue,
sender: Any?) {
   if let dest = segue.destination as?
ChooseThreadViewController {
     dest.chosenImage = selectedImage
   }
}

...
override func viewDidLoad() {
    super.viewDidLoad()
    chosenImageView.image = chosenImage
}
```

Prepare for Segue

Question: Is this better?

```
override func prepare(for segue: UIStoryboardSegue,
sender: Any?)
    if let destination as
ChooseTableViewController {
        destination.chosenImage = chosenImage
    }
}

...
override func viewDidLoad() {
    super.viewDidLoad()
    chosenImageView.image = chosenImage
}
```

YES

View Controller Lifecycle

Question: What's the difference between the methods `viewDidLoad` and `viewWillAppear`?

View Controller Lifecycle

Question: What's the difference between the methods `viewDidLoad` and `viewWillAppear`?

`viewDidLoad` - called once when the view controller is created

`viewWillAppear` - called every time the view appears on the screen

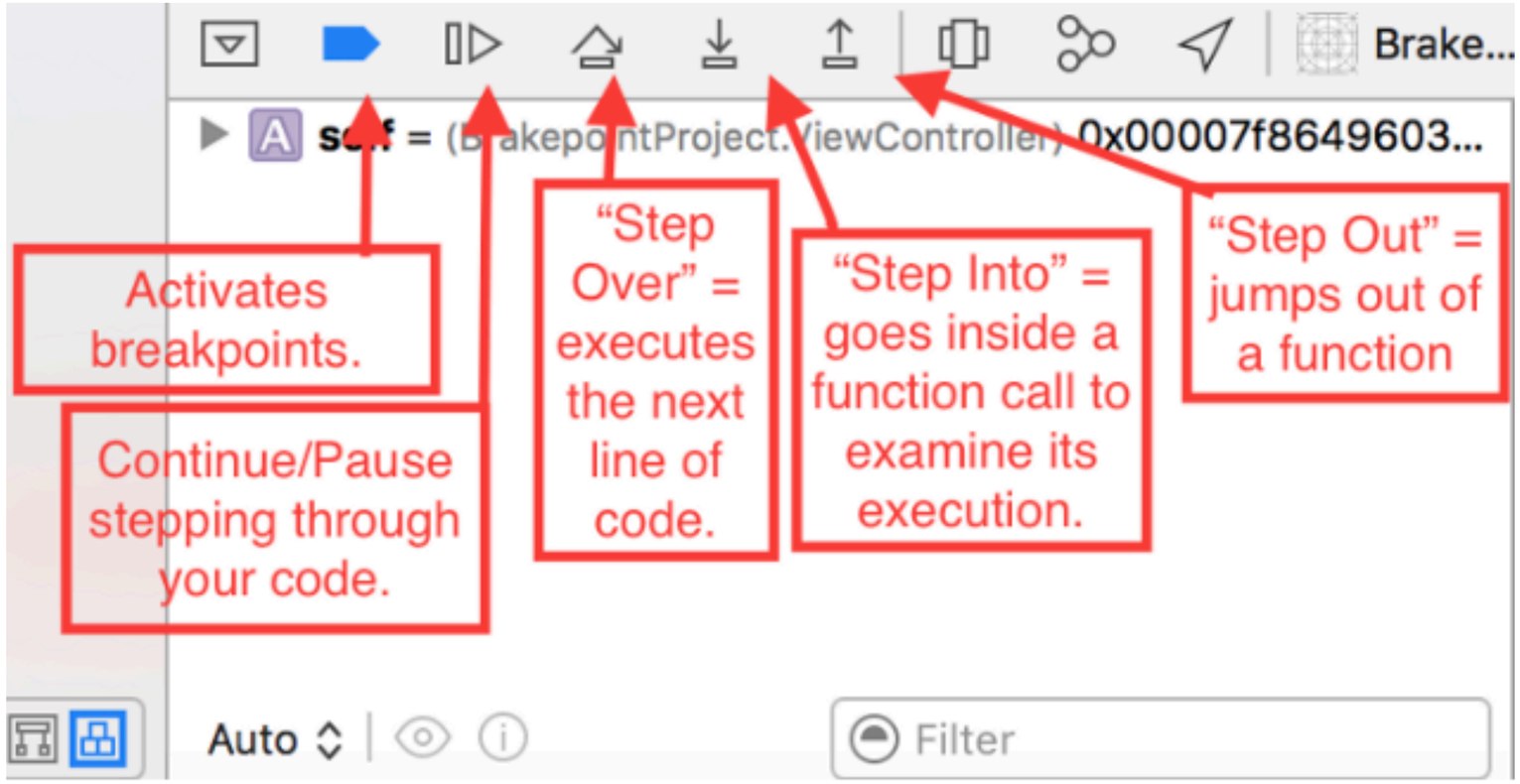
Xcode

breakpoints

Question: What do each of the following buttons do?

breakpoints

Question: What do each of the following buttons do?



Auto Layout

Autolayout?

Question: What are some key benefits of using AutoLayout over frame-based layout?

Autolayout?

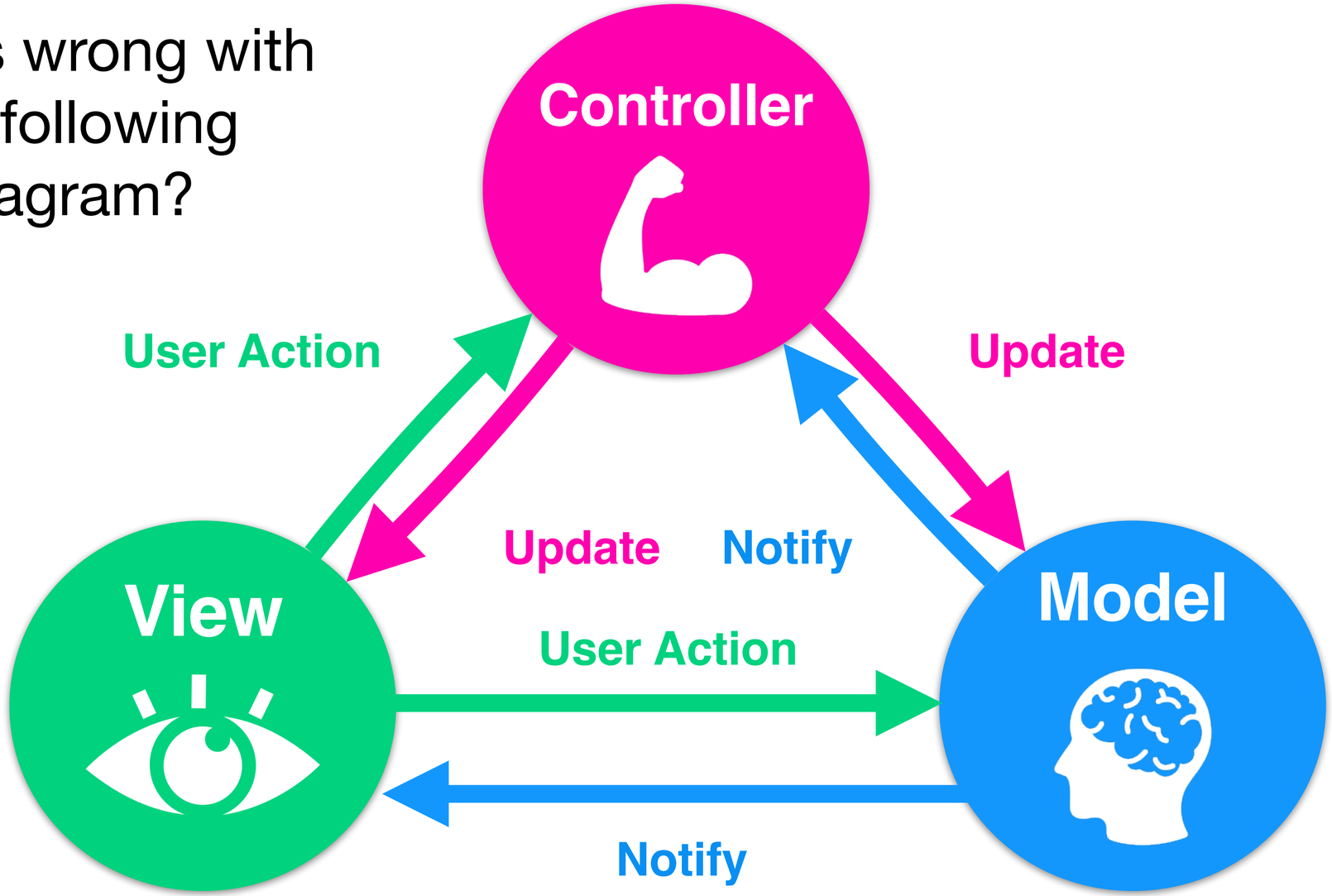
Question: What are some key benefits of using AutoLayout over frame-based layout?

- ☆ constraints automatically adapt to different screen sizes and orientations, reducing the amount of code you need to write
- ☆ constraints can be set in Storyboard
- ☆ relational rather than calculation-based (could be a con, depending on what you find easier)

MVC

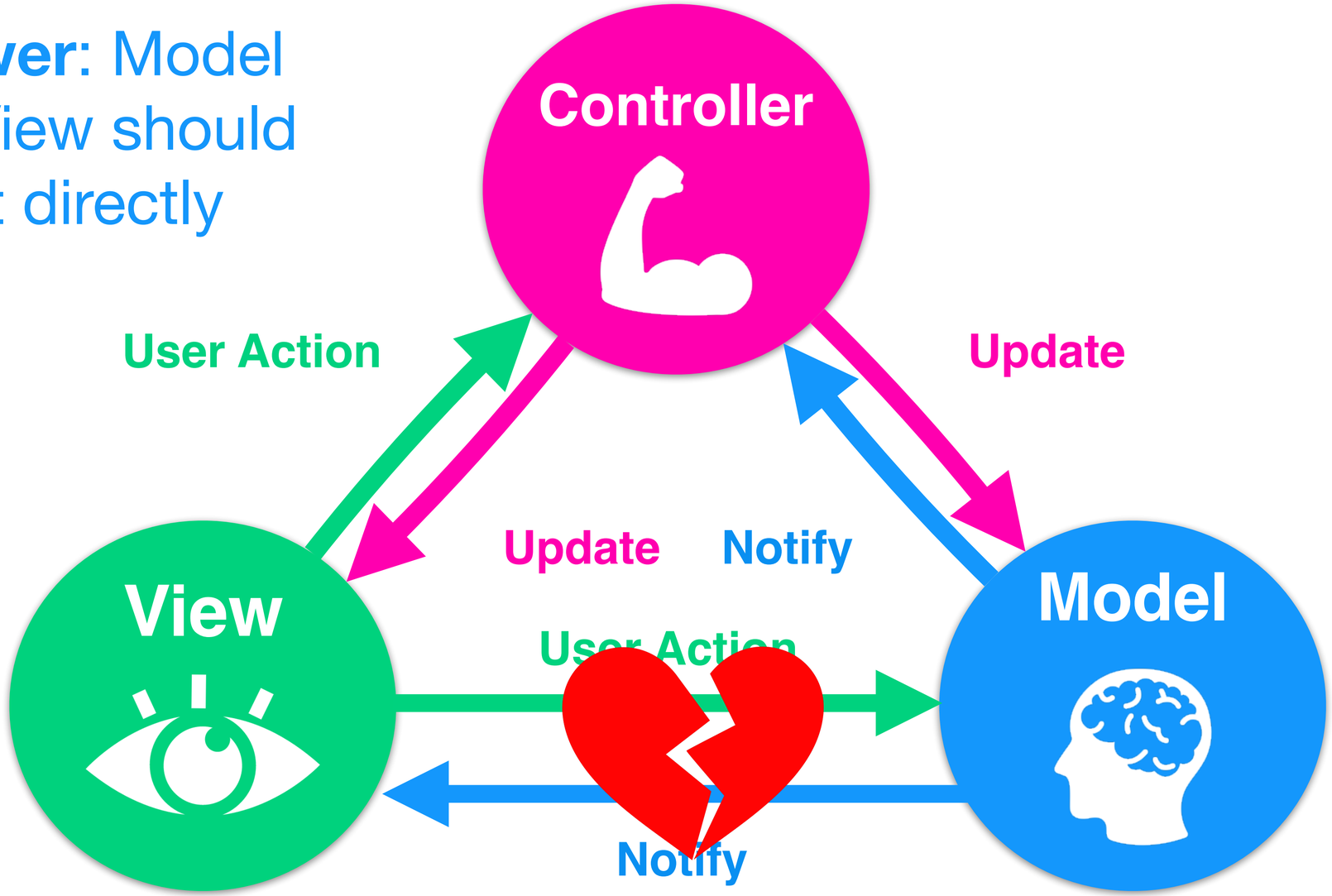
MVC design

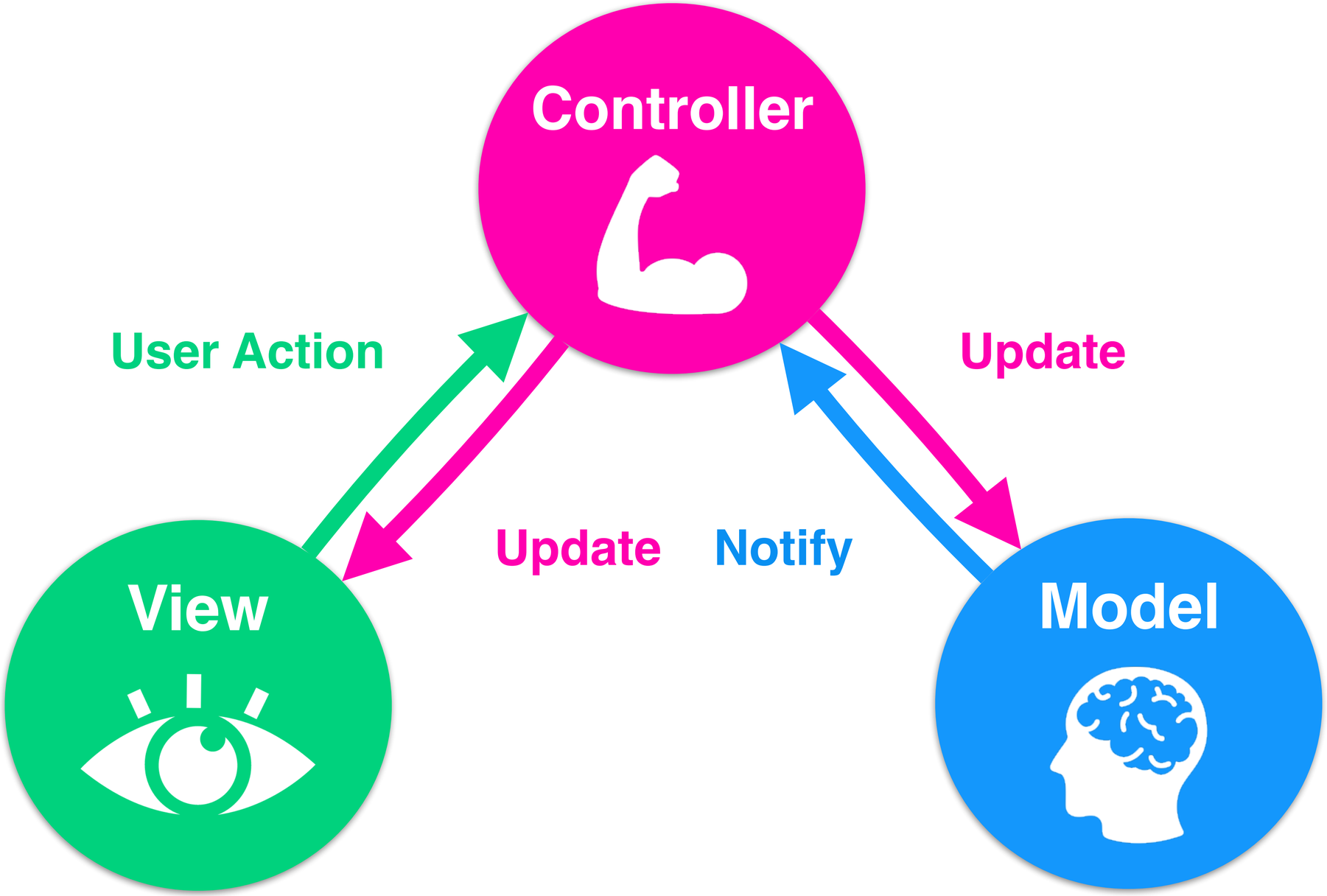
What's wrong with the following diagram?



MVC design

Answer: Model and View should not directly





Model View Controller

MVC q2

```
var threads: [String: [UIImage]] = ["memes": [], "dog
spots": [], "random": []]

var read: [String: [Bool]] = ["memes": [], "dog spots": [],
"random": []]

var postTime: [String: [NSDate]] = ["memes": [], "dog
spots": [], "random": []]

var username: [String: [String]] = ["memes": [], "dog
spots": [], "random": []]
```

Question: This is one implementation of the Model for the Snapchat project. What's a better way to implement this? (hint: how can we make this object oriented)

MVC q2

```
class Post {
    var read: Bool = false
    let username: String
    let thread: String
    let datePosted: Date = NSDate()
    let imageName: String
    let postId: String
   ///implementation hidden
}

var threads: [String: [Post]] = ["memes": [], "dog
                                 spots": [], "random": []]
```

Question: This is one implementation of the Model for the Snapchat project. What's a better way to implement this?

Create a Post / Snap Model class! See example Post class here: github.com/iosdecal/hw3-part2/blob/master/SnapchatClone/Post.swift

Collection views and table views

Collection/table views

What are the 2 required methods for Tableviews?

Collection/table views

What are the 2 required methods for Tableviews?

```
func cellForRow(at indexPath:
IndexPath) -> UITableViewCell?

func numberOfRows(inSection
section: Int) -> Int
```

```
class ImagePickerController: UIViewController, UICollectionViewDataSource
{

    @IBOutlet var imageCollectionView: UICollectionView!

    override func viewDidLoad() {
        super.viewDidLoad()
        imageCollectionView.dataSource = self
    }

    func collectionView(_ collectionView: UICollectionView,
         numberOfItemsInSection section: Int) -> Int {
        // implementation hidden
    }

    func collectionView(_ collectionView: UICollectionView,
     cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {
        // implementation hidden
    }

    func collectionView(_ collectionView: UICollectionView,
       didSelectItemAt indexPath: IndexPath) {
        // implementation hidden
    }
}
```

My collectionview isn’t behaving correctly. What problem do you think I have, and how could I solve it?

```
class ImagePickerController: UIViewController,
UICollectionViewDataSource, UICollectionViewDelegate {

    @IBOutlet var imageCollectionView: UICollectionView!

    override func viewDidLoad() {
        super.viewDidLoad()
        imageCollectionView.dataSource = self
        imageCollectionView.delegate = self
    }

    func collectionView(_ collectionView: UICollectionView,
         numberOfItemsInSection section: Int) -> Int {
        // implementation hidden
    }

    func collectionView(_ collectionView: UICollectionView,
     cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {
        // implementation hidden
    }

    func collectionView(_ collectionView: UICollectionView,
       didSelectItemAt indexPath: IndexPath) {
        // implementation hidden
    }
}
```

`didSelectItemAt` will never be called, since the delegate is not set

```
class ImagePickerController: UIViewController {

    @IBOutlet var imageCollectionView: UICollectionView!

    override func viewDidLoad() {
        super.viewDidLoad()
        imageCollectionView.dataSource = self
    }
}

class ImagePickerDataSource: NSObject,
                UICollectionViewDataSource {
    func collectionView(_ collectionView: UICollectionView,
          numberOfItemsInSection section: Int) -> Int {
        // implementation hidden
    }

    func collectionView(_ collectionView: UICollectionView,
            cellForItemAt indexPath: IndexPath) ->
                  UICollectionViewCell {
        // implementation hidden
    }
}
```

What’s wrong with my code? How can I fix it (multiple answers)

```
class ImagePickerController: UIViewController {

    @IBOutlet var imageCollectionView: UICollectionView!

    override func viewDidLoad() {
        super.viewDidLoad()
        imageCollectionView.dataSource = self
    }
}

class ImagePickerDataSource: NSObject,
                UICollectionViewDataSource {
    func collectionView(_ collectionView: UICollectionView,
          numberOfItemsInSection section: Int) -> Int {
        // implementation hidden
    }

    func collectionView(_ collectionView: UICollectionView,
            cellForItemAt indexPath: IndexPath) ->
                  UICollectionViewCell {
        // implementation hidden
    }
}
```

We are setting the dataSource to self, but ImagePickerController does not conform to `UICollectionViewDataSource`

```
class ImagePickerController: UIViewController {

    @IBOutlet var imageCollectionView: UICollectionView!

    override func viewDidLoad() {
        super.viewDidLoad()
        imageCollectionView.dataSource = ImagePickerDataSource()
    }
}

class ImagePickerDataSource: NSObject,
                UICollectionViewDataSource {
    func collectionView(_ collectionView: UICollectionView,
          numberOfItemsInSection section: Int) -> Int {
        // implementation hidden
    }

    func collectionView(_ collectionView: UICollectionView,
            cellForItemAt indexPath: IndexPath) ->
                  UICollectionViewCell {
        // implementation hidden
    }
}
```

Solution 1: create instance of ImagePickerDataSource() and use it as the datasource

```
class ImagePickerController: UIViewController,
                UICollectionViewDataSource {

    @IBOutlet var imageCollectionView: UICollectionView!

    override func viewDidLoad() {
        super.viewDidLoad()
        imageCollectionView.dataSource = self
    }

    func collectionView(_ collectionView: UICollectionView,
          numberOfItemsInSection section: Int) -> Int {
        // implementation hidden
    }

    func collectionView(_ collectionView: UICollectionView,
            cellForItemAt indexPath: IndexPath) ->
                  UICollectionViewCell {
        // implementation hidden
    }
}
```

Solution 2: make your ImagePickerController conform to UICollectionViewDataSource, and move code into it

```
class ImagePickerController: UIViewController {

    @IBOutlet var imageCollectionView: UICollectionView!

    override func viewDidLoad() {
        super.viewDidLoad()
        imageCollectionView.dataSource = self
    }
}

extension ImagePickerController: UICollectionViewDataSource {

    var cellImages = [UIImage(named: "dog"), UIImage(named: "dog2")]

    func collectionView(_ collectionView: UICollectionView,
      numberOfItemsInSection section: Int) -> Int {
        return cellImages.count
    }

    func collectionView(_ collectionView: UICollectionView,
     cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {
        // implementation hidden
    }
}
```

What’s wrong with my code? How can I fix it (multiple answers)

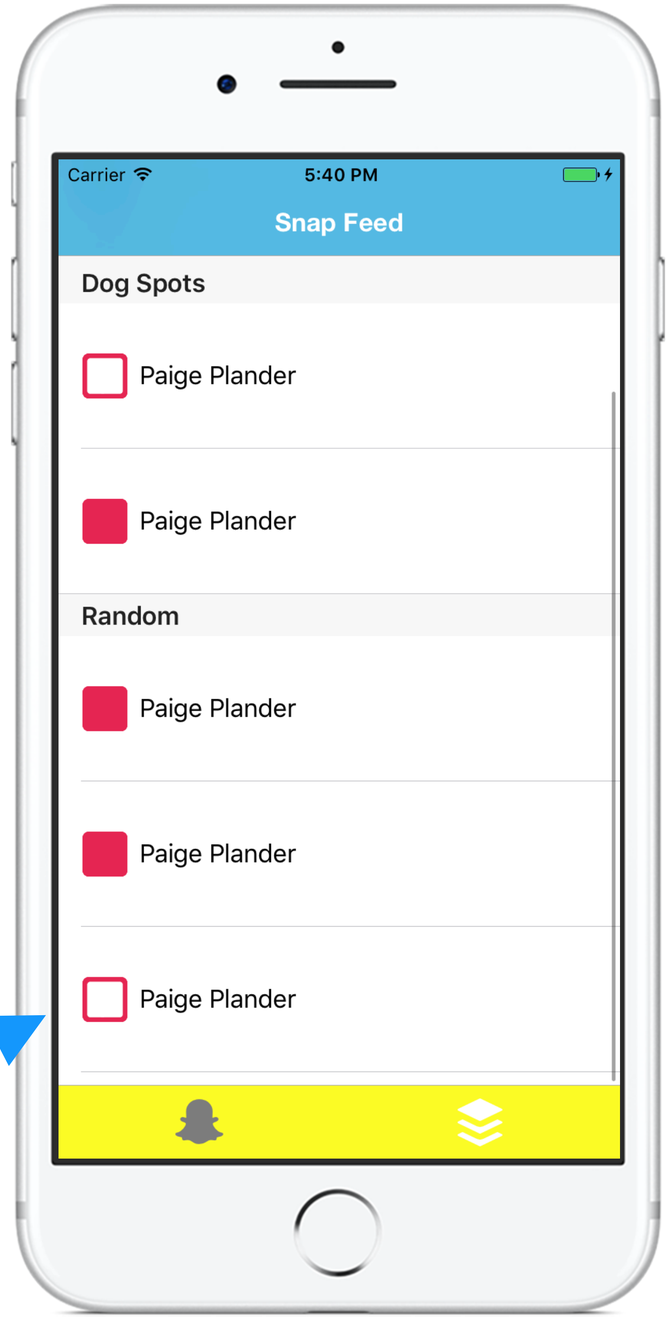
```
class ImagePickerController: UIViewController {

    var cellImages = [UIImage(named: "dog"), UIImage(named: "dog2")]
    @IBOutlet var imageCollectionView: UICollectionView!

    override func viewDidLoad() {
        super.viewDidLoad()
        imageCollectionView.dataSource = self
    }
}

extension ImagePickerController: UICollectionViewDataSource {

    func collectionView(_ collectionView: UICollectionView,
      numberOfItemsInSection section: Int) -> Int {
        return cellImages.count
    }

    func collectionView(_ collectionView: UICollectionView,
     cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {
        // implementation hidden
    }
}
```

extensions can’t store properties - to fix, move it into the main class

Collection/table views

Question: Some of my cells show up as read, even though they shouldn't be - what's the problem in my code?

Wasn't read, but still clickable!



Collection/table views

Question: Some of my cells show up as read, even though they shouldn't be - what's the problem in my code?

```
func tableView(_ tableView: UITableView, cellForRowAt
    indexPath: IndexPath) -> UITableViewCell {
    let cell =
        tableView.dequeueReusableCell(withIdentifier:
        "postCell", for: indexPath) as! PostsTableViewCell
    if let post = getPostFromIndexPath(indexPath:
        indexPath) {
        if post.read {
            cell.readImageView.image = UIImage(named:
                "read")
        }
    }
    return cell
}
```

Collection/table views

Since Table view Cells are **recycled** , you need to check if the cell has is not read, and set image to “unread”

```
func tableView(_ tableView: UITableView, cellForRowAt
    indexPath: IndexPath) -> UITableViewCell {
    let cell =
        tableView.dequeueReusableCell(withIdentifier:
        "postCell", for: indexPath) as! PostsTableViewCell
    if let post = getPostFromIndexPath(indexPath:
        indexPath) {
        if post.read {
            cell.readImageView.image = UIImage(named:
                "read")
        }
    }
    return cell
}
```

Collection/table views

```
func tableView(_ tableView: UITableView, cellForRowAt indexPath:
    IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier:
        "postCell", for: indexPath) as! PostsTableViewCell
    if let post = getPostFromIndexPath(indexPath: indexPath) {
        if post.read {
            cell.readImageView.image = UIImage(named: "read")
        }
        else {
            cell.readImageView.image = UIImage(named: "unread")
        }
        cell.usernameLabel.text = post.username
    }
    return cell
}
```

Solution Code

Networking

Async vs Sync q1

Question: What's the difference between an asynchronous block of code and a synchronous block of code?

Async vs Sync q1

Question: What's the difference between an asynchronous block of code and a synchronous block of code?

Synchronous: waits until the task has completed
Asynchronous: completes a task in background and can notify you when complete

Async vs Sync q2

Question: Give an example of a scenario in which we would need to use a callback?

Async vs Sync q2 (ans)

API call

```
let client = TWTRAPIClient()
client.loadTweetWithID("20") { (tweet, error) -> Void in
    // handle the response or error
}
```

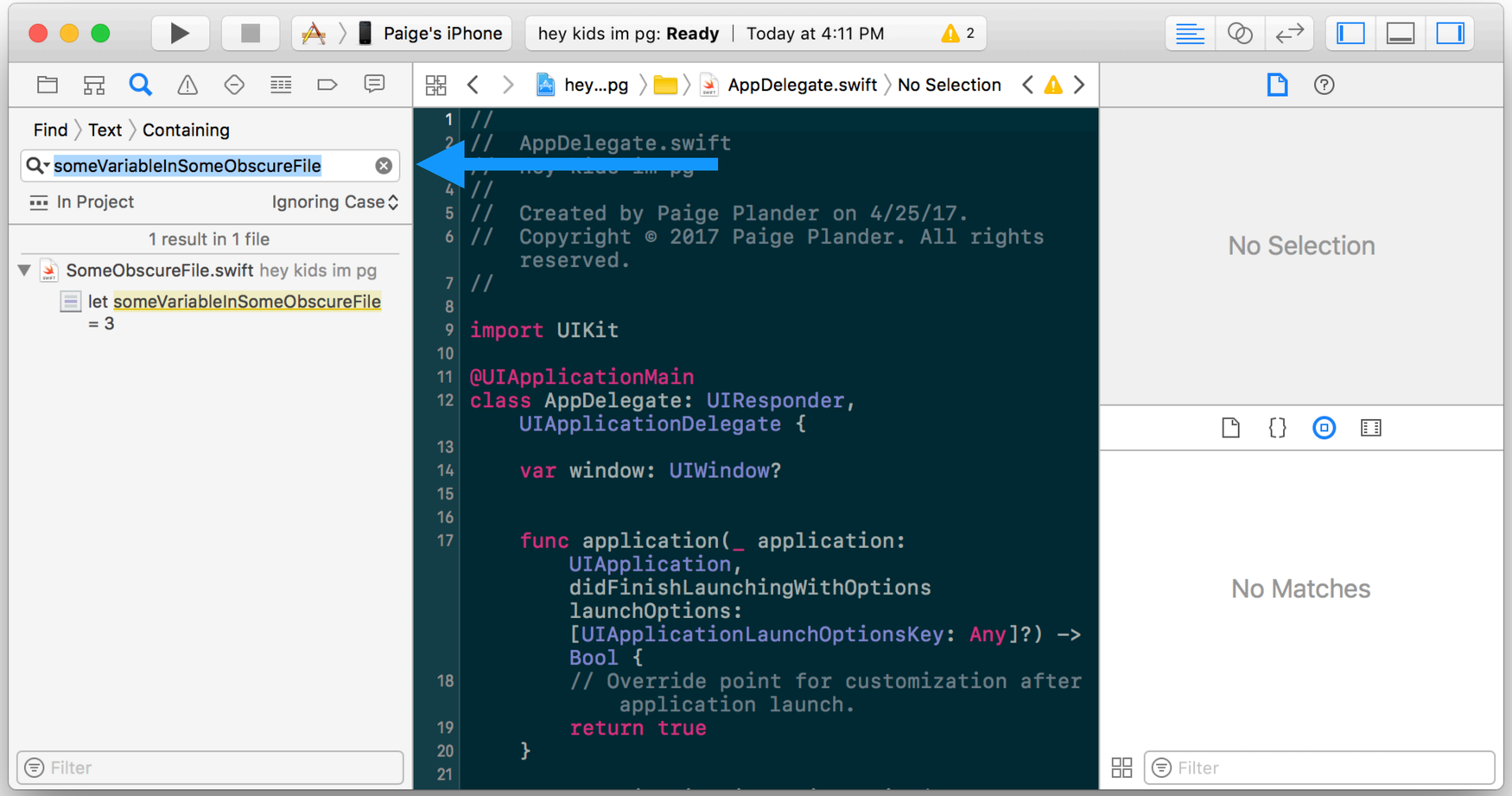
URLSession

```
func dataTask(with url: URL,
              completionHandler: @escaping (Data?,
                  URLResponse?, Error?) -> Void) ->
                  URLSessionDataTask
```

UIAlert

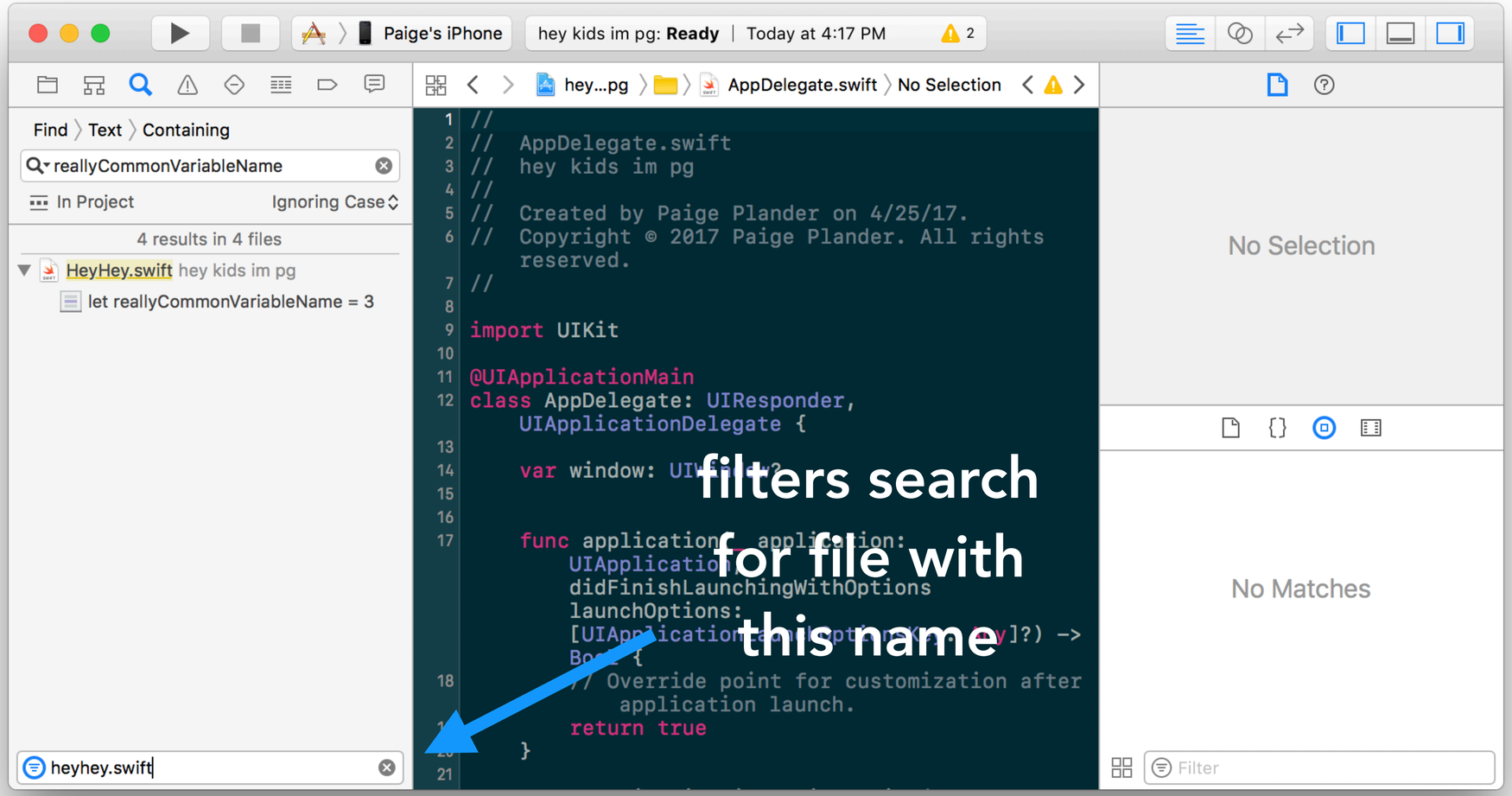
```
let alert = UIAlertController(title: "My Alert", message: “heyo!",
             preferredStyle: .alert)

let okAction = UIAlertAction(title: "OK", style: .default, handler:
{(action: UIAlertAction) -> Void in
    // user pressed okay… let’s do something
})
alertController.addAction(OKAction)
```

extra slides



Searching for a specific variable / method / comment

Search with filename filtering (saves a lot of time!)



Search with filename filtering (saves a lot of time!)

Paige's iPhone | hey kids im pg: **Ready** | Today at 4:19 PM

Find 〉 Text 〉 Containing

reallyCommonVariableName

In Project — Ignoring Case

4 results in 4 files

HeyHey.swift hey kids im pg

let reallyCommonVariableName = 3

heyhey.swift

h...g 〉 AppDelegate.swift 〉 AppDelegate

```
1  //
2  //  AppDelegate.swift
3  //  hey kids im pg
4  //
5  //  Created by Paige Plander on 4/25/17.
6  //  Copyright © 2017 Paige Plander. All rights reserved.
7  //
8
9  import UIKit
10
11 @UIApplicationMain
12 class AppDelegate: UIResponder, UIApplicationDelegate {
13
14     var window: UIWindow?
15
16
17     func application(_ application: UIApplication,
           didFinishLaunchingWithOptions launchOptions:
           [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
18         // Override point for customization after
               application launch.
19         return true
20     }
21
22     func applicationWillResignActive(_ application:
           UIApplication) {
23         // Sent when the application is about to move from
               active to inactive state. This can occur for
               certain types of temporary interruptions (such
               as an incoming phone call or SMS message) or
               when the user quits the application and it
               begins the transition to the background state.
24         // Use this method to pause ongoing tasks, disable
               timers, and invalidate graphics rendering
```

Identity and Type

Name: AppDelegate.swift

Type: Default - Swift Source

Location: Relative to Group

AppDelegate.swift

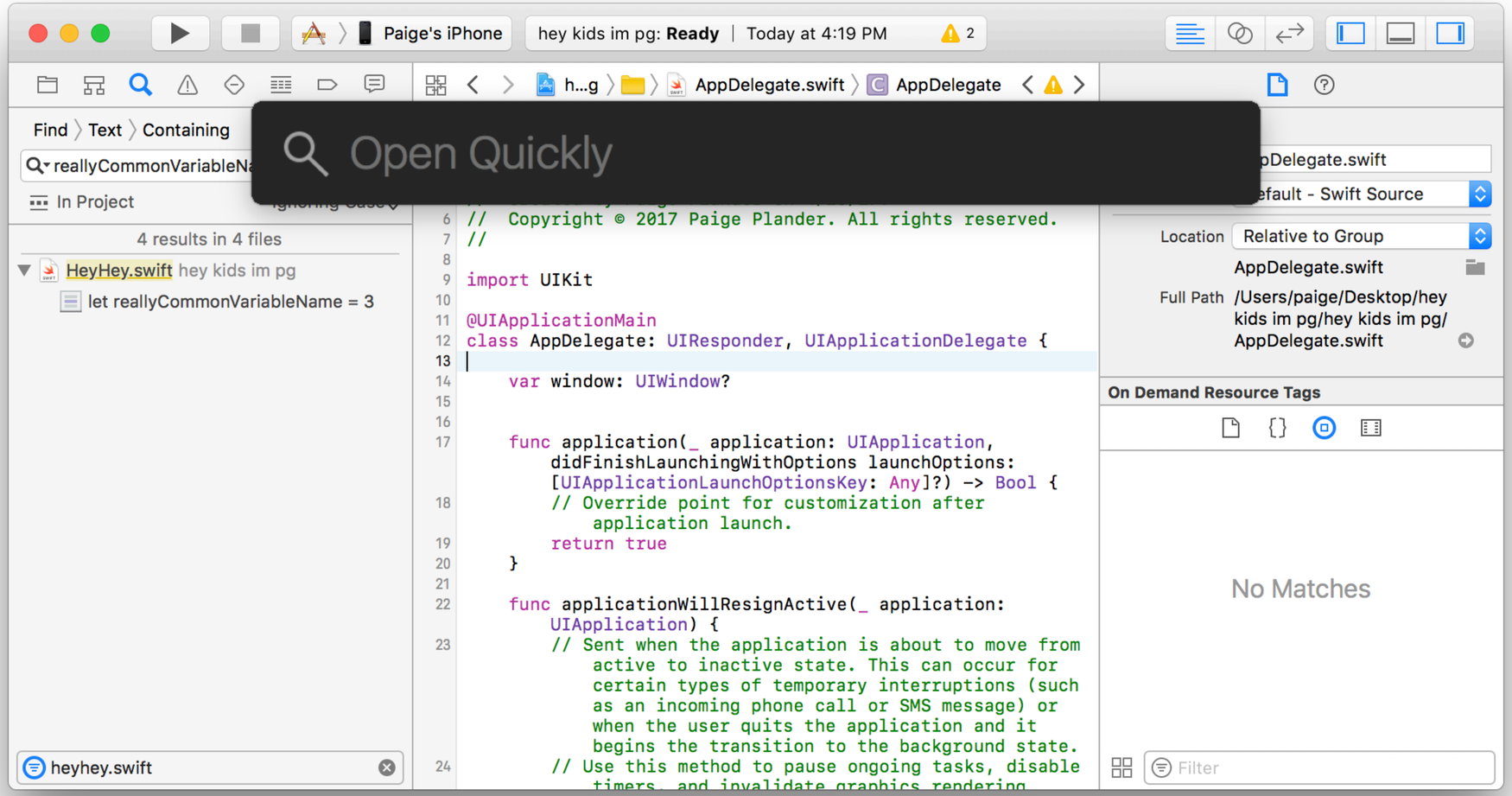
Full Path: /Users/paige/Desktop/hey kids im pg/hey kids im pg/AppDelegate.swift

On Demand Resource Tags

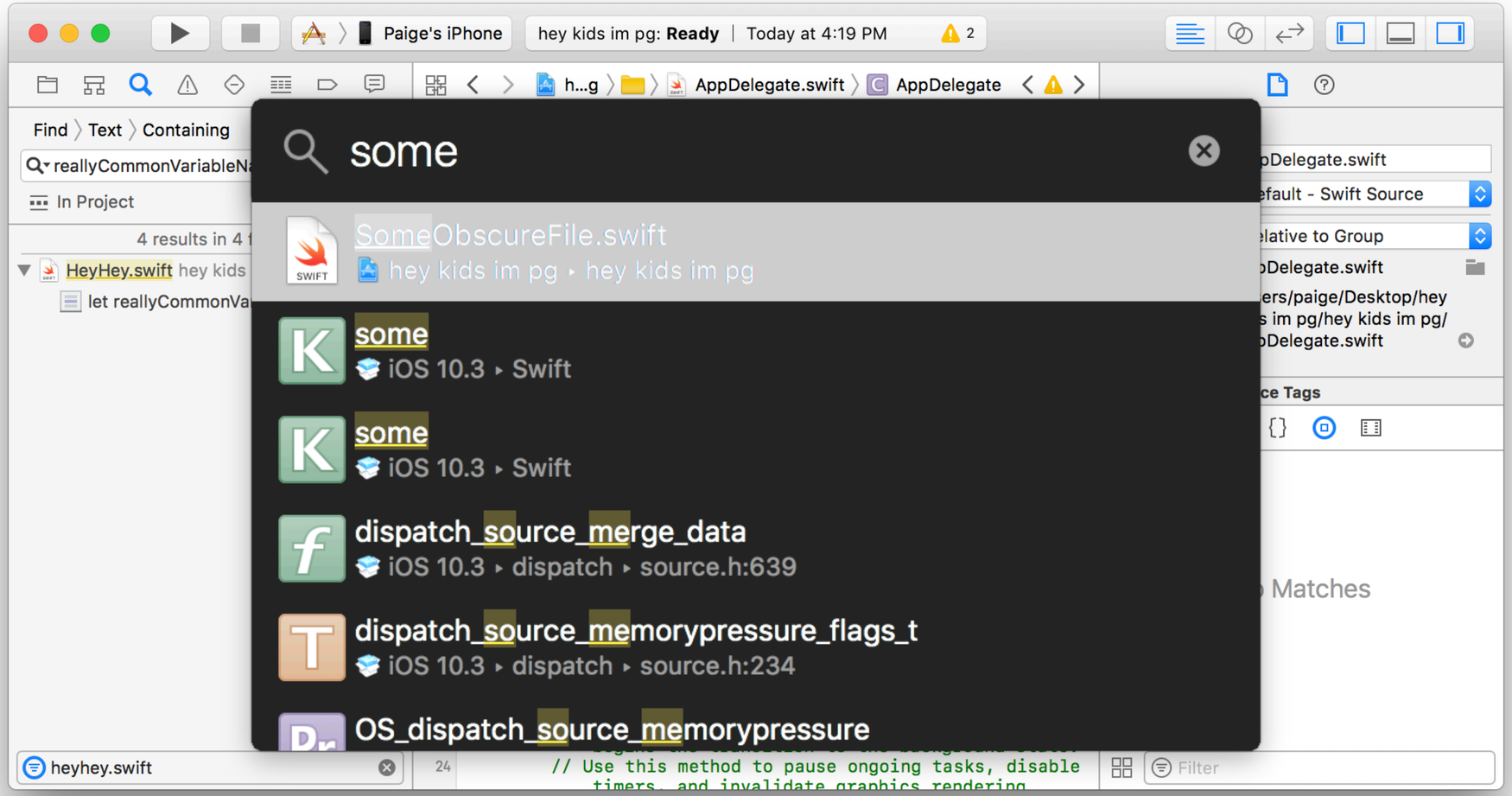
No Matches

Filter

Quick Search - Command + Shift + O



Quick Search - Command + Shift + O



Quick Search - Command + Shift + O

Strong vs Weak?

Strong - Two objects both increase each other's reference counts and are in memory forever.
Weak - Only one object increases reference count, so when one gets deallocated so does the other.

Retain Cycles

```
@class Child;
@interface Parent : NSObject {
    Child *child; //instance variables implicitly __strong
}
@end
@interface Child : NSObject {
    Parent *parent; //also implicitly __strong
}
@end
```