iOS Dev Accelerator
Week 2 Day 2

- Animating Constraints
- UITabBarController
- Parse, all of it
- gitignore

Animating Constraints

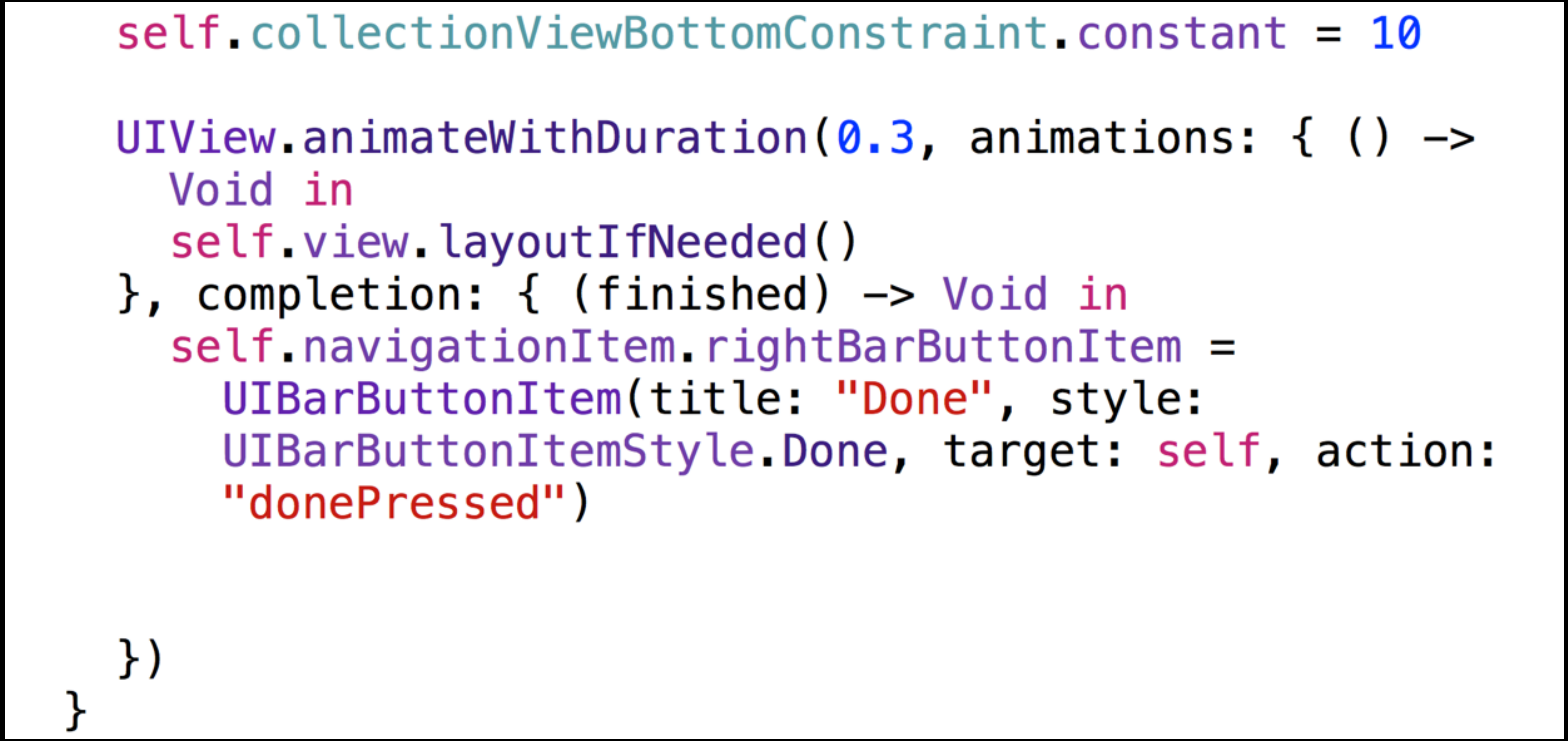
Constraints and animation

- Constraints can be animated, meaning you can smoothly move or change the size of views that are managed by autolayout
- It can be tough at first to figure out. “Well how do I get to the constraints of each view?”
- In code its a bit of a chore, but luckily, we can create IBOutlets to individual constraints from the storyboard. Hooray!

Demo

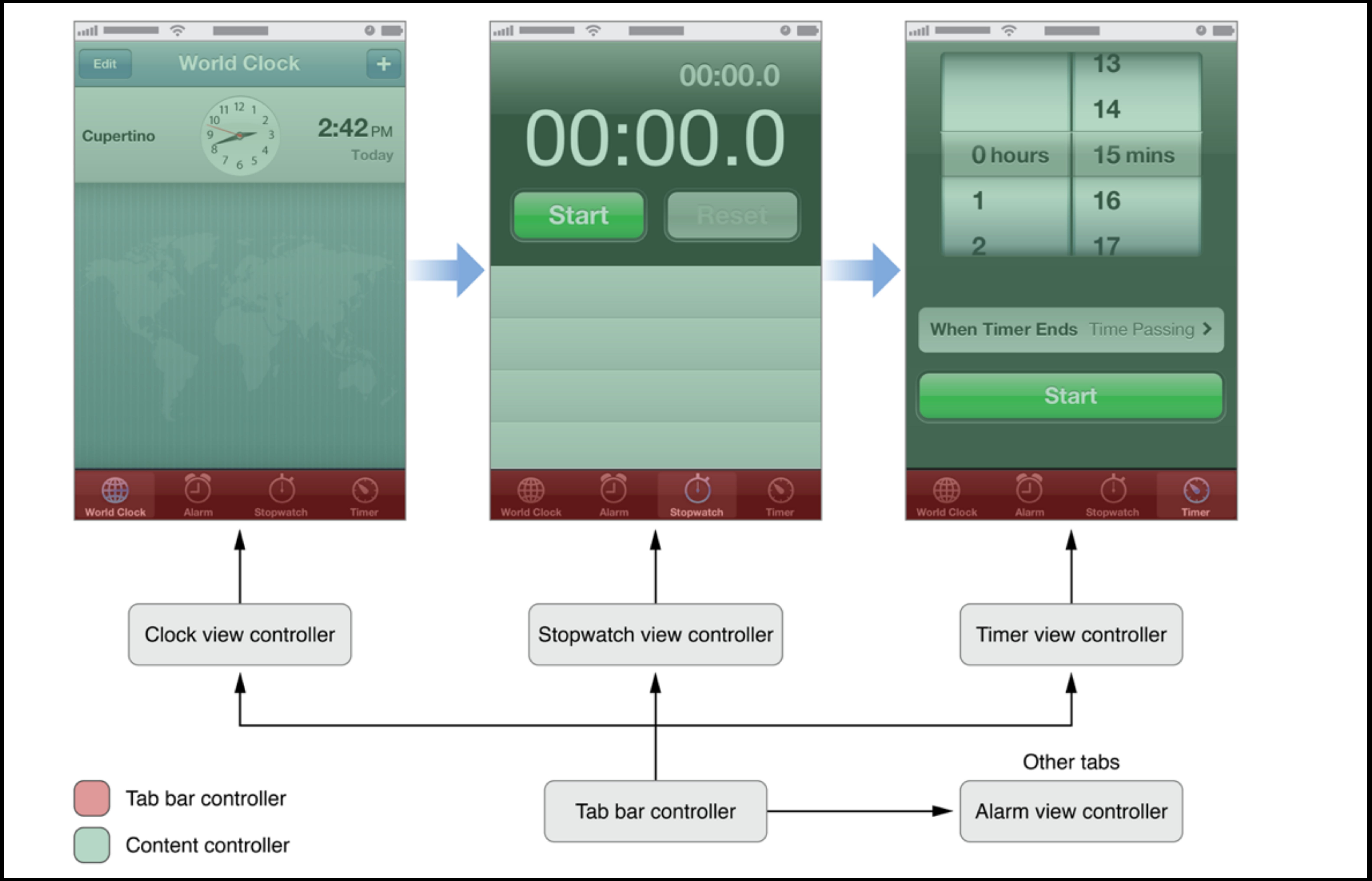
Animating Autolayout changes

1. Change the constant of the constraint
2. In your animation block, call layoutIfNeeded() on the view containing the constraints.



```
    self.collectionViewBottomConstraint.constant = 10

    UIView.animateWithDuration(0.3, animations: { () ->
      Void in
      self.view.layoutIfNeeded()
    }, completion: { (finished) -> Void in
      self.navigationItem.rightBarButtonItem =
        UIBarButtonItem(title: "Done", style:
        UIBarButtonItemStyle.Done, target: self, action:
        "donePressed")


    })
  }
```

Demo



Tab Bar Controllers

Tab Bar Controllers

- A Tab bar controller is a container view controller that you use to divide your app into two or more **distinct modes of operations**.
- The tab bar has multiple tabs, each representing a child view controller.
- Selecting a tab causes the tab bar controller to display the associated view controller’s view on screen.
- In general, you should use a tab bar if your app displays different types of data to the user, or displays the data in different ways

Getting a tab bar controller into your app

- You can instantiate a UITabBarController in code with a regular initializer.
- Or the easier way, embed them via storyboard, just like you do with a navigation controller

Demo

Tab Bar Controller facts

- Every UIViewController has a property called tabBarController, which points to the tab bar controller it is in, IF it is actually in a tab bar controller (just like the navigationController property)
- UITabBarController has a property called viewControllers which is an array of the view controllers it is managing. They are ordered from left to right.
- By default, the label for each ‘item’ in the tab bar is the title of the view controller it contains.

UITabBarItem

- Every view controller has a UITabBarItem property called tabBarItem
- This item dictates how the view controller is represented in the tab bar
- If you want to provide custom images as the tab icons, you use the UITabBarItem initializer that takes in a title, image and selectedImage.
- Creating custom icons for the tab bar sucks. It has to be only alpha channels. See Apple’s HIG for a guidelines on creating icons
- I will provide you custom icons for this project. You can buy icons specifically designed for the tab bar from many different icon websites as well.

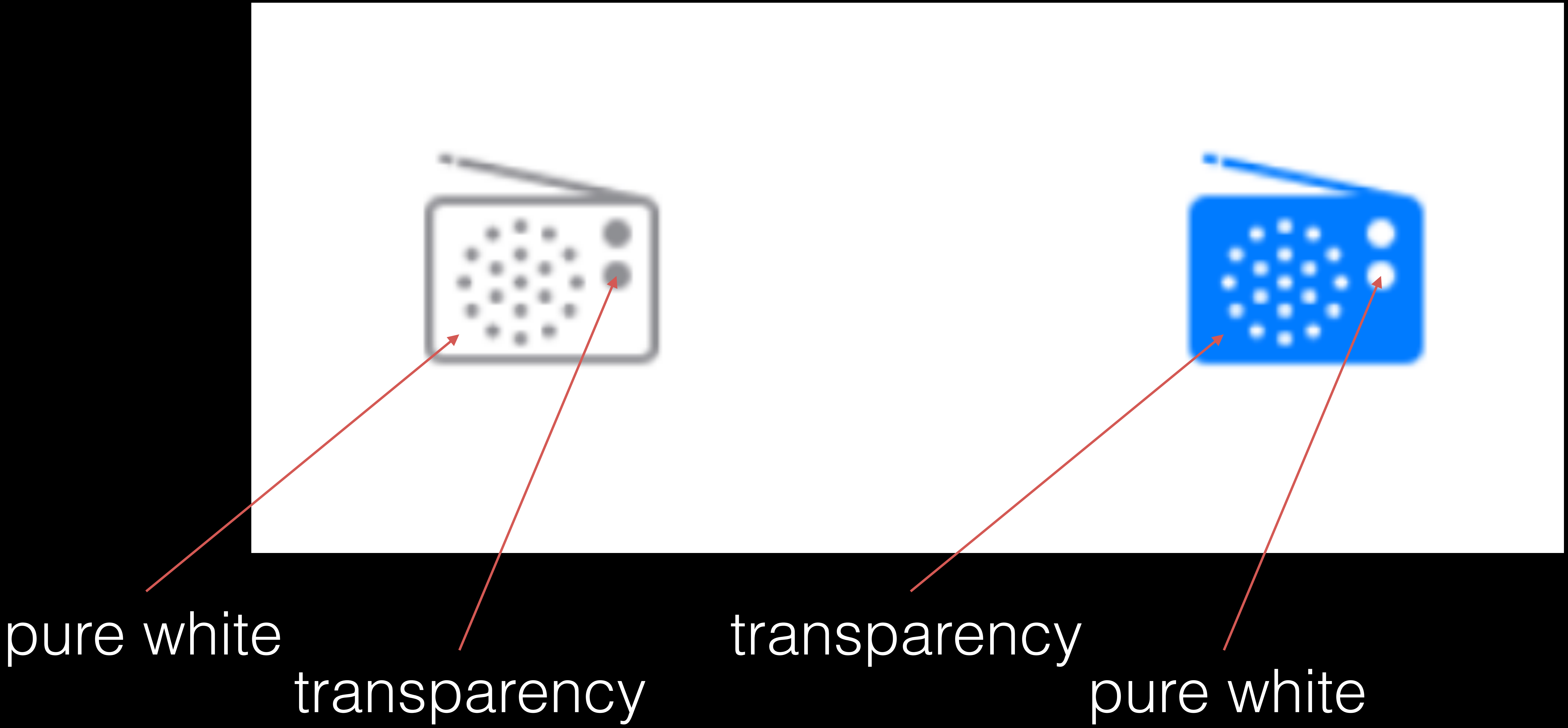
Demo

Custom Bar Button Icons

- If you need custom icons and you can’t find any suitable icons online, here are guidelines to creating your own:
  - provide two versions: one for unselected and one for selected
  - the selected version should just be a ‘filled in’ version of the unselected
  - use a thin stroke, 2-pixel wide for detailed icons and 3-pixel wide for less details.
  - don’t include words in the image, thats what the title property of the tab bar item is for
  - use only pure white with alpha transparency

Custom Bar Button Icons

- “use only pure white with alpha transparency”



Parse

Easy & feature rich backends for your apps

Parse

- Parse is a mobile app platform that allows you to easily setup and work with a backend hosted with Parse
- The main benefit to you, as an iOS Developer, is that you don’t need to know/learn a backend language in order to have a backend.
- Parses’s goal is to completely eliminate the need to write any server code for your mobile apps
- The iOS Parse SDK contains all of the classes and resources our apps need to in order to communicate with Parse

Setting up Parse For your App

- Step 1 is to create a free account at parse.com
- You then have 2 ways to get the Parse SDK in your Swift project:
    - download the starter project they have which comes with the Parse SDK all hooked up and ready to go.
    - Download the standalone framework from https://www.parse.com/downloads/ios/parse-library/latest and then do a bunch of stuff to get it imported into your already existing project.
- We will do the first easy way, but I have included instructions in these slides for the other way as well.

Setting up Parse For your App cont.

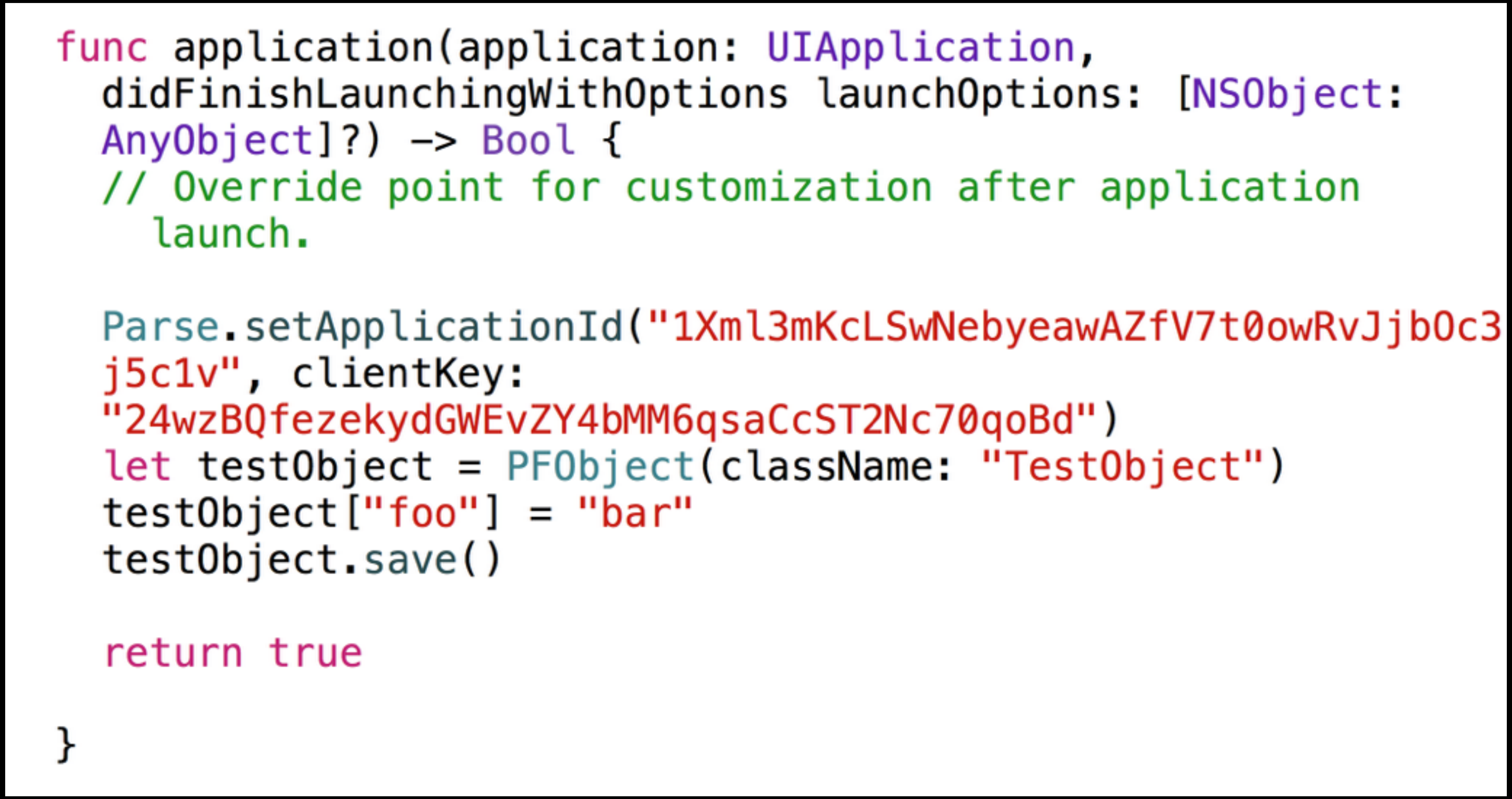
- In your Xcode project, create a blank objective-C file. When it asks you if you want to create an Objective-C bridging header file, say yes.
- Delete the blank .m objective-c file it created, and then in in your bridging header put the following line:
  - #import <Parse/Parse.h>
- Go to the build phases of your app’s target and add the following to Link Binary with Libraries….

Things to add to your build phases – Link Binary with Libraries

- libsqlite3.0.dylib
- Foundation.framework
- SystemConfiguration.framework
- StoreKit.framework
- Security.framework
- QuartzCore.framework
- MobileCoreServices.framework
- libz.dylib
- CoreLocation.framework
- CoreGraphics.framework
- AudioToolbox.framework
- Bolts.framework
- ParseCrashReporting.framework
- ParseFacebookUtils.framework
- ParseUI.framework
- Parse.framework

To test it out

- In your swift project, add the following code to your app delegate's didFinishLaunchingWithOptions method:

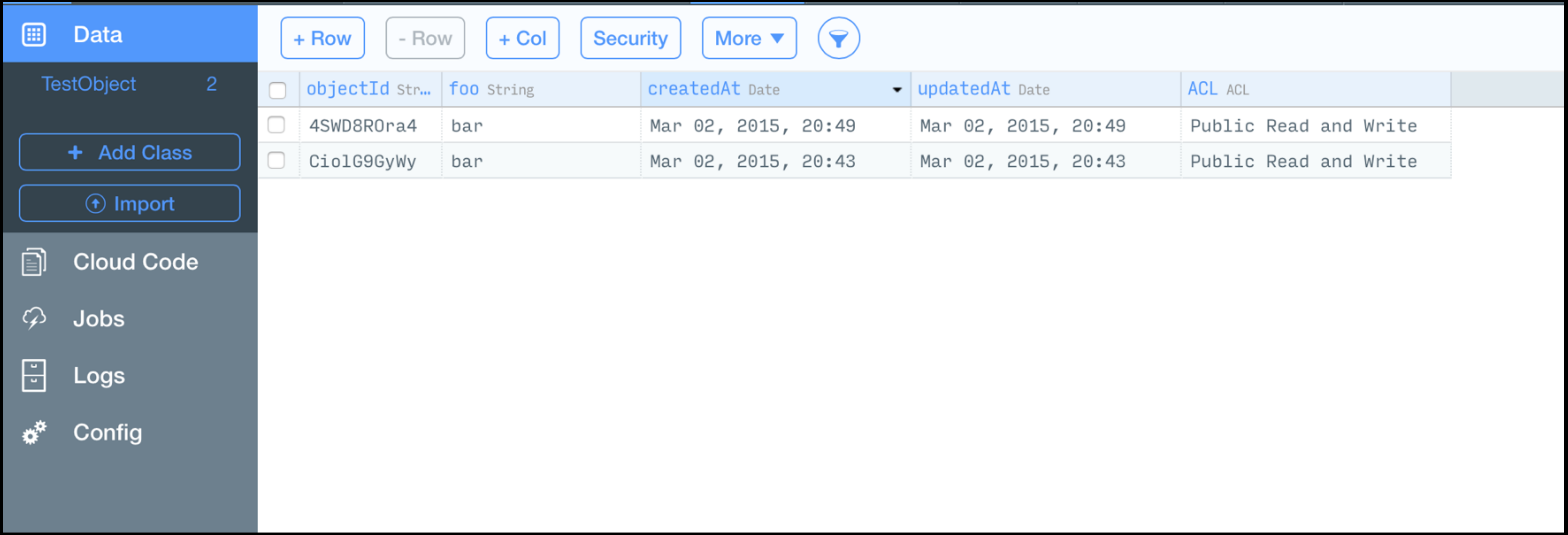


```
func application(application: UIApplication,
  didFinishLaunchingWithOptions launchOptions: [NSObject:
  AnyObject]?) -> Bool {
  // Override point for customization after application
     launch.

  Parse.setApplicationId("1Xml3mKcLSwNebyeawAZfV7t0owRvJjbOc3
  j5c1v", clientKey:
  "24wzBQfezekydGWEvZY4bMM6qsaCcST2Nc70qoBd")
  let testObject = PFObject(className: "TestObject")
  testObject["foo"] = "bar"
  testObject.save()

  return true

}
```

ApplicationID and clientKey should match those created when you created your first app on Parse

To test it out

- Once you have done that, you should be able to log onto your parse dashboard and see instance of TestObject being uploaded, with the value bar set for the field foo:



	objectId Str…	foo String	createdAt Date	updatedAt Date	ACL ACL
	4SWD8ROra4	bar	Mar 02, 2015, 20:49	Mar 02, 2015, 20:49	Public Read and Write
	CiolG9GyWy	bar	Mar 02, 2015, 20:43	Mar 02, 2015, 20:43	Public Read and Write

Demo

Parse Apps

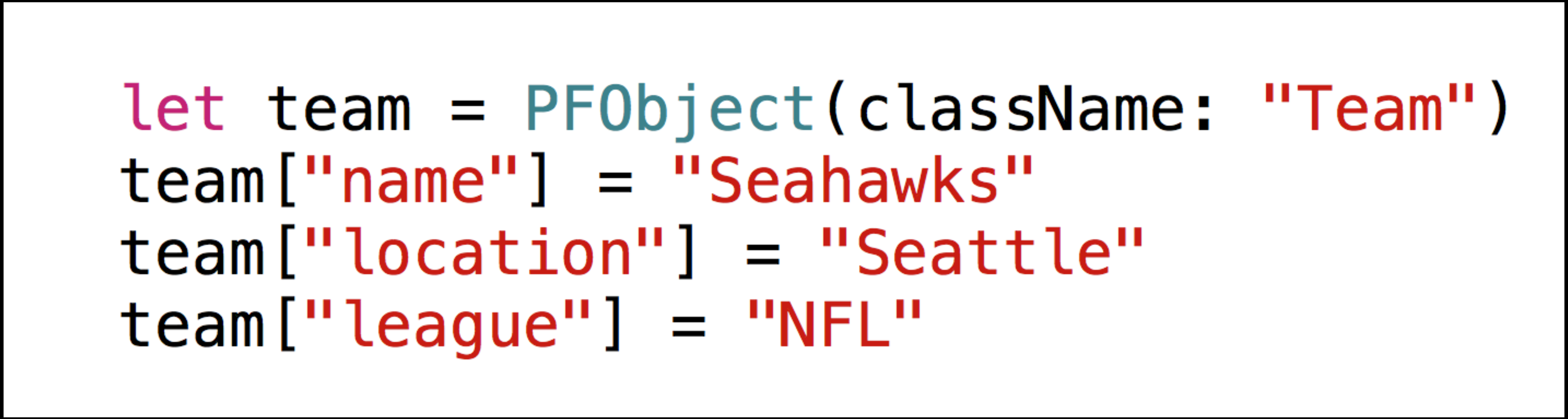
- On Parse, you will need to create a separate Parse app for every separate mobile app you are going to create that needs a Parse backend.
- Each one of your Parse apps has its own Application ID and client key
- Thats how your Xcode project knows which parse app to upload and download data to and from

```swift
func application(application: UIApplication,
  didFinishLaunchingWithOptions launchOptions: [NSObject:
  AnyObject]?) -> Bool {
  // Override point for customization after application
    launch.

  Parse.setApplicationId("1Xml3mKcLSwNebyeawAZfV7t0owRvJjbOc3
  j5c1v", clientKey:
  "24wzBQfezekydGWEvZY4bMM6qsaCcST2Nc70qoBd")
  let testObject = PFObject(className: "TestObject")
  testObject["foo"] = "bar"
  testObject.save()

  return true

}
```

Parse Objects

- Storing and retrieving data on Parse is setup around the class PFObject.
- Each PFObject contains key value pairings (like a dictionary/JSON) of data
- The data in PFObjects is schema-less, which means you don't have to setup schema before you start uploading data. As you upload new key-value pairings, Parse will automatically make a column for it in the table
- Keys must be alphanumeric strings

PFObject

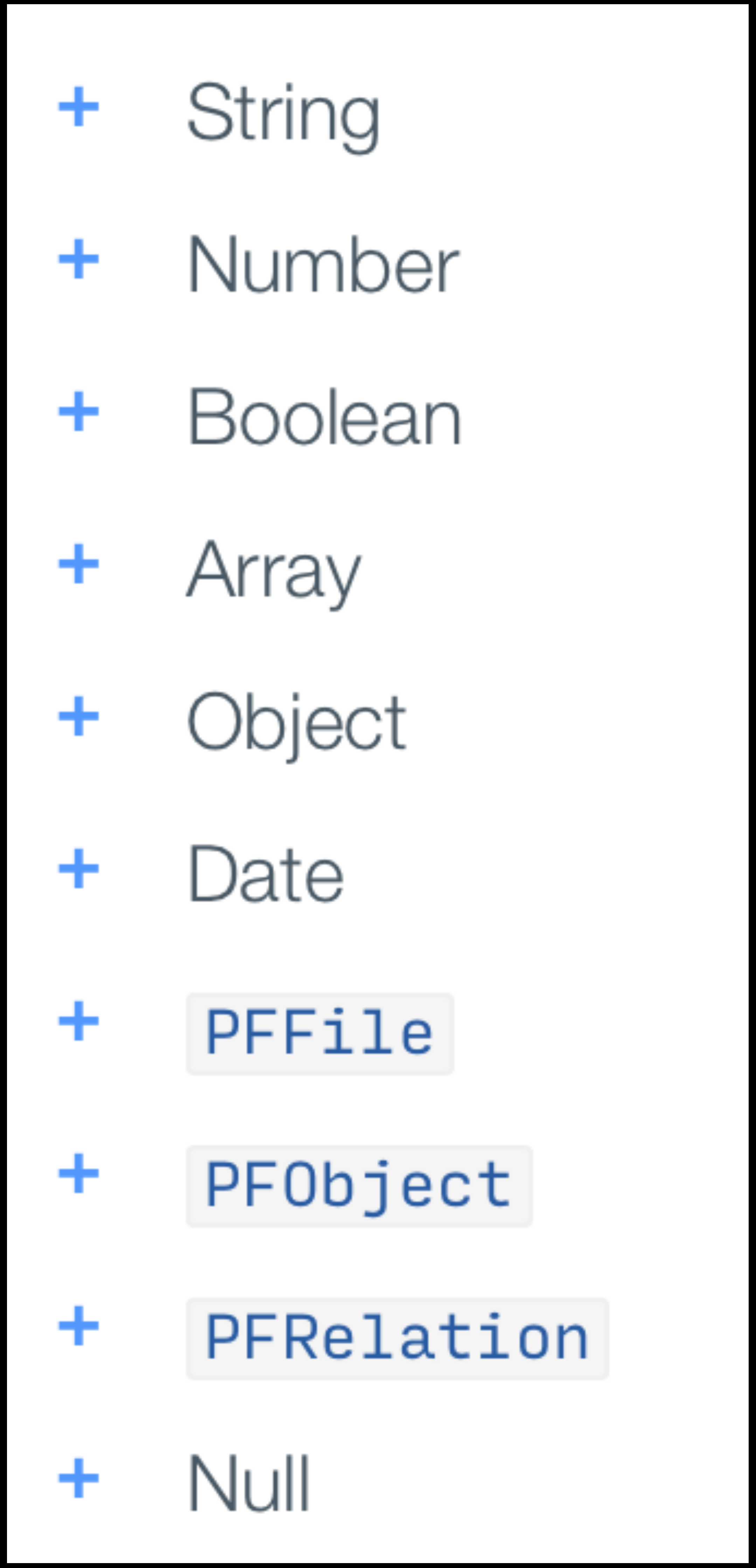


```
let team = PFObject(className: "Team")
team["name"] = "Seahawks"
team["location"] = "Seattle"
team["league"] = "NFL"
```

So the data of this object looks like:
name: “Seahawks”, location: “Seattle”, league:”NFL”

PFObjects and Data

- On Parse's servers in The Cloud™, your data is stored as JSON.
- So this means you can send any data to Parse that can be converted to JSON.
- In addition, it can also handle NSDates!
- Heres a complete list:



- String
- Number
- Boolean
- Array
- Object
- Date
- `PFFile`
- `PFObject`
- `PFRelation`
- Null

Demo

Saving your Objects

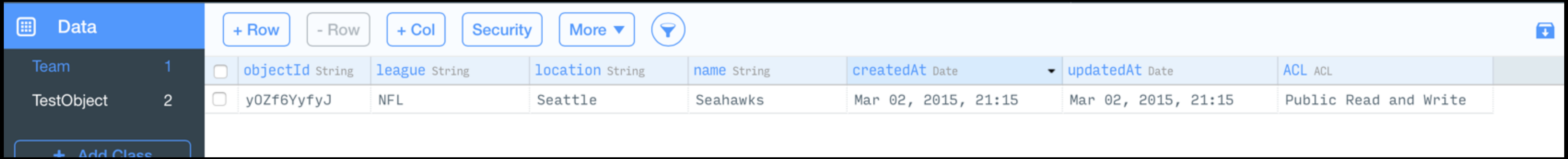
- So once you have created some data you want to upload to your backend, you just need to call save on your objects
- You can call save on your objects by simply calling saveInBackgroundWithBlock()
- The method has one parameter, which is a closure that acts as a callback once the save has been performed by the server or if it timed out.
- Parse uses 10 second time out limits on all async events.

Saving your Objets

```
let team = PFObject(className: "Team")
team["name"] = "Seahawks"
team["location"] = "Seattle"
team["league"] = "NFL"

team.saveInBackgroundWithBlock { (finished, error) -> Void
  in
  if error != nil {
    println("Error during save: %@",error.
      localizedDescription)
  }
}
```

Saving your Objets

- Whats cool about that save is no where did we have to define the scheme for the Team class.
- Parse lazily sets all this stuff up once it sees we are uploading a new class to the backend.



objectId String	league String	location String	name String	createdAt Date	updatedAt Date	ACL ACL
yOZf6YyfyJ	NFL	Seattle	Seahawks	Mar 02, 2015, 21:15	Mar 02, 2015, 21:15	Public Read and Write

- Also take note that there are a few fields that we didn't create explicitly in our code, Parse generates these for us.
- Of particular importance is the objectID field, which is how parse recognizes each individual record in this table. This is guaranteed to be unique.

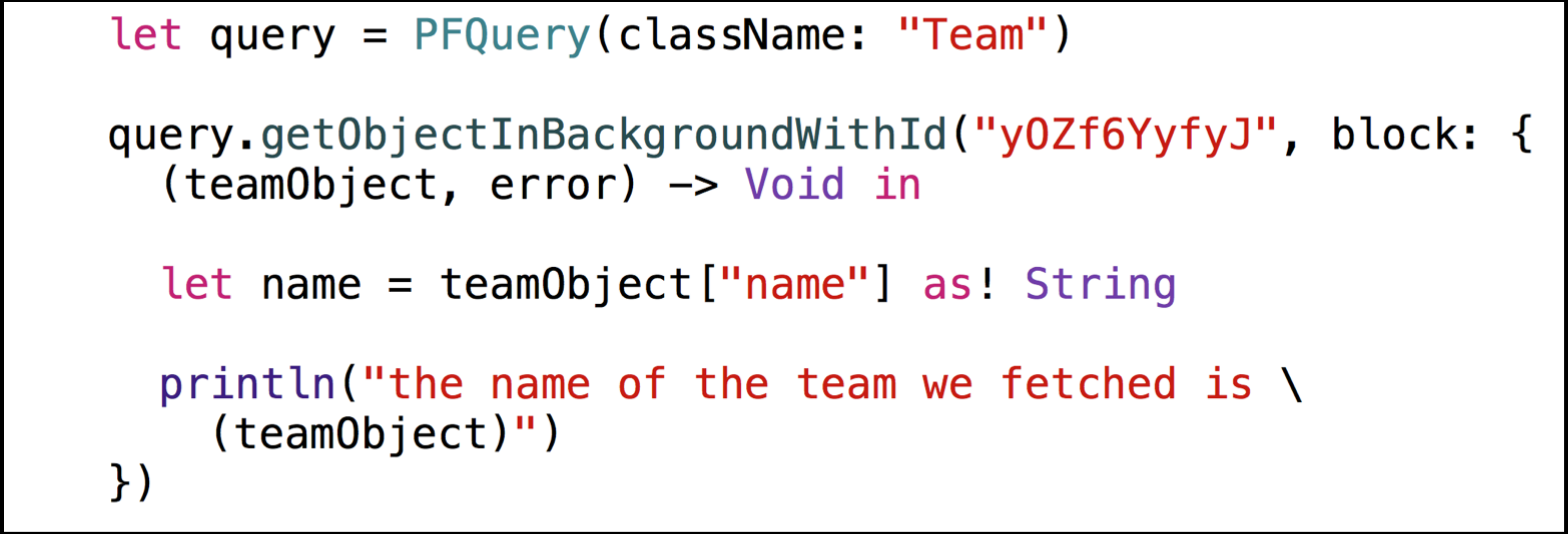
Demo

Fetching your objects

- So we know how to create and save our objects, but how do we fetch them from the cloud?
- We do this using the PFQuery class.
- The PFQuery class offers many different ways to fetch objects from your backend.
- Lets look at a few of the most common and powerful ones

Fetching an object with an ID

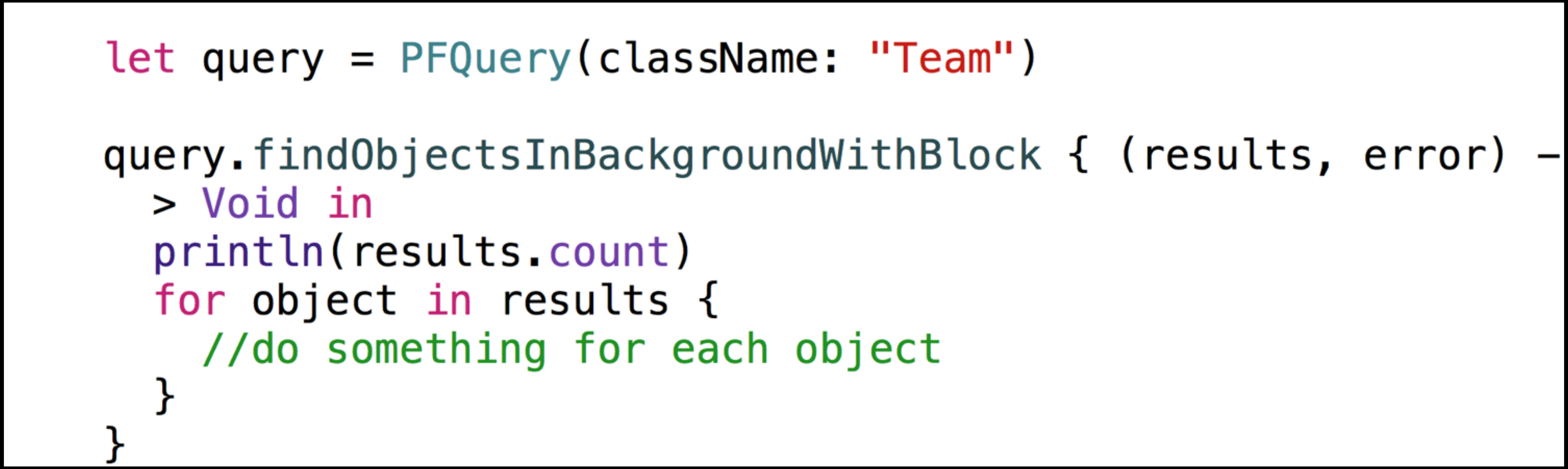
- If you know the objectID of the object you are trying to fetch, use getObjectInBackgroundWithId:block:() to fetch a single object back



```
let query = PFQuery(className: "Team")

query.getObjectInBackgroundWithId("yOZf6YyfyJ", block: {
  (teamObject, error) -> Void in

  let name = teamObject["name"] as! String

  println("the name of the team we fetched is \
    (teamObject)")
})
```

Fetching a group of objects

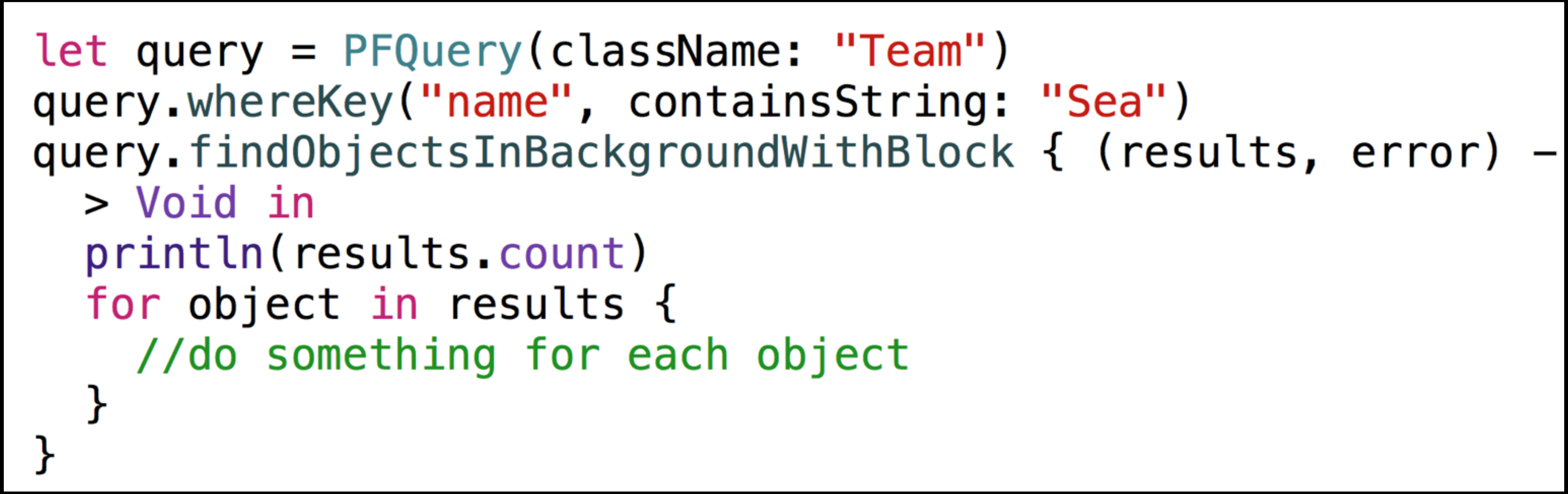
- You can run a query against an entire class of objects, without an id, by using the findObjectsInBackgroundWithBlock() method



```
let query = PFQuery(className: "Team")

query.findObjectsInBackgroundWithBlock { (results, error) -
  > Void in
  println(results.count)
  for object in results {
    //do something for each object
  }
}
```

Fetching a group of objects

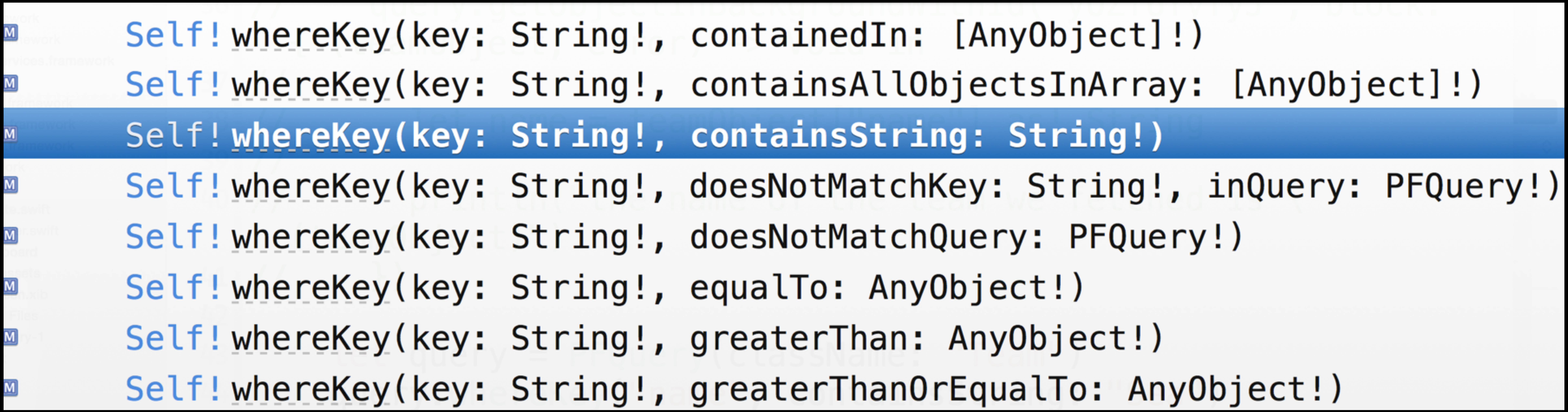
- So the previous query will return ALL the objects (up to a limit of 1000 per query, default is 100) in that class. What if we want to be more specific? We can attach filters to our queries:



```
let query = PFQuery(className: "Team")
query.whereKey("name", containsString: "Sea")
query.findObjectsInBackgroundWithBlock { (results, error) -
  > Void in
  println(results.count)
  for object in results {
    //do something for each object
  }
}
```

Query filters

- Theres a ton!



```
Self! whereKey(key: String!, containedIn: [AnyObject]!)
Self! whereKey(key: String!, containsAllObjectsInArray: [AnyObject]!)
Self! whereKey(key: String!, containsString: String!)
Self! whereKey(key: String!, doesNotMatchKey: String!, inQuery: PFQuery!)
Self! whereKey(key: String!, doesNotMatchQuery: PFQuery!)
Self! whereKey(key: String!, equalTo: AnyObject!)
Self! whereKey(key: String!, greaterThan: AnyObject!)
Self! whereKey(key: String!, greaterThanOrEqualTo: AnyObject!)
```

- While its recommended you use these methods, you can use NSPredicate class if you are coming in with some CoreData knowledge, of if you need to do some more advanced fetching.

Demo

Saving files to your backend

- The PFFile class lets you store files in the cloud that would be too large to put inside regular PFObjects.
- PFFile is needed for things like images, videos, music, any other binary data (up to 10mb at a time).
- There is a pretty simple workflow you follow to upload your files to the cloud. Its a relatively easy 2 step process.

PFFiles workflow step 1

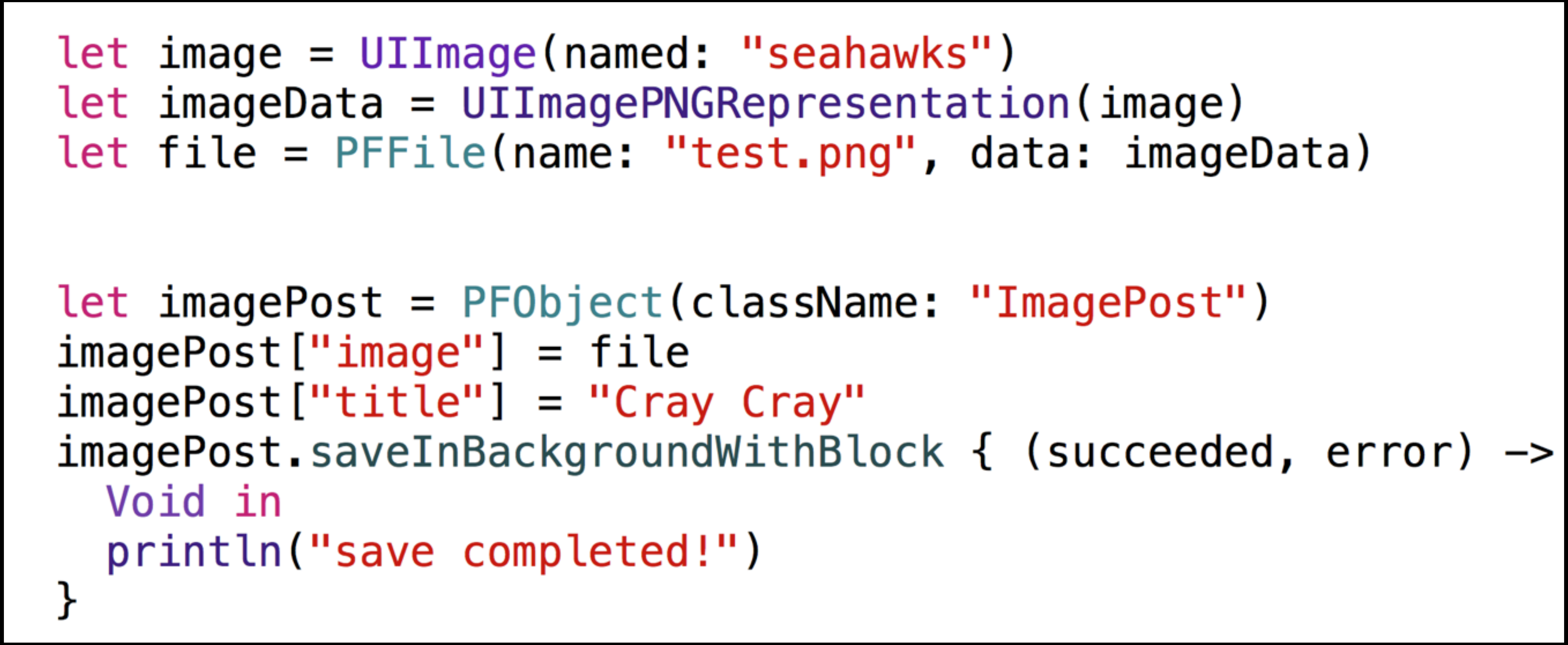
- First thing is to convert the file you are trying to upload (like a UIImage) into NSData.
- And then create a PFFile with the data:

```swift
let image = UIImage(named: "MyImage")
let imageData = UIImagePNGRepresentation(image)
let file = PFFile(name: "test.png", data: imageData)
```

The name parameter of the PFFile does not have to be unique server side, but the file type is required so the parse server knows how to handle the file

PFFiles workflow step 2

- You then need to associate the PFFile with a PFObject, and then save it to the cloud



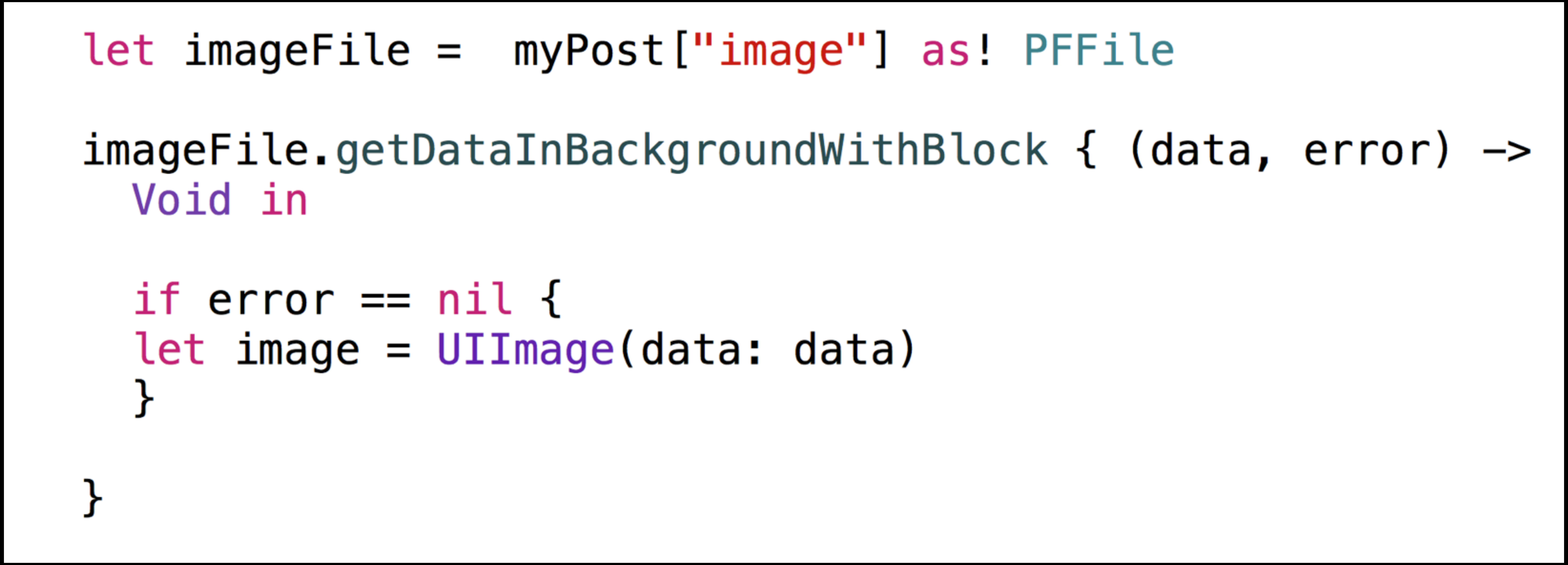
```
let image = UIImage(named: "seahawks")
let imageData = UIImagePNGRepresentation(image)
let file = PFFile(name: "test.png", data: imageData)


let imagePost = PFObject(className: "ImagePost")
imagePost["image"] = file
imagePost["title"] = "Cray Cray"
imagePost.saveInBackgroundWithBlock { (succeeded, error) ->
  Void in
  println("save completed!")
}
```

Demo

Get that file back from the cloud

- To get back files from the cloud, like getting back an image to show, you first need to run a regular PFQuery to return the PFObject(s) that contain the data you need.
- You then need to access that particular field on the PFObject that contains the data, creating a PFFile to hold the reference.
- And finally call getDataInBackground() to download the actual data. The data doesn't come down when you do your PFQuery. That would make PFQuerys very slow.

Get that file back from the cloud



```
let imageFile =  myPost["image"] as! PFFile

imageFile.getDataInBackgroundWithBlock { (data, error) ->
  Void in

  if error == nil {
  let image = UIImage(data: data)
  }

}
```

Demo

Errors

- Anytime you are working with a server, you need to keep error handling in mind.
- Tons of things can go wrong!
- There are two primary ways it can go wrong:
  - The request you made was bad
  - There was a problem connecting with the server

Parse Errors

- Parse will provide us with information about the error
- Since interacting with our Parse server is done asynchronously, we deal with Parse errors in callbacks.
- Parse callbacks often take this form:

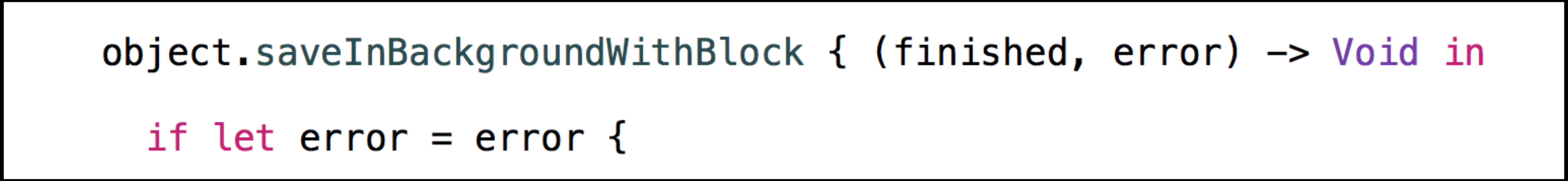
```
(result: PFObject?, error: NSError?) -> Void
```

the object(s) we want

an error optional, if this is nil then everything went great!

Handling the errors

- The first thing we can do in our callbacks is check for the presence of an error:

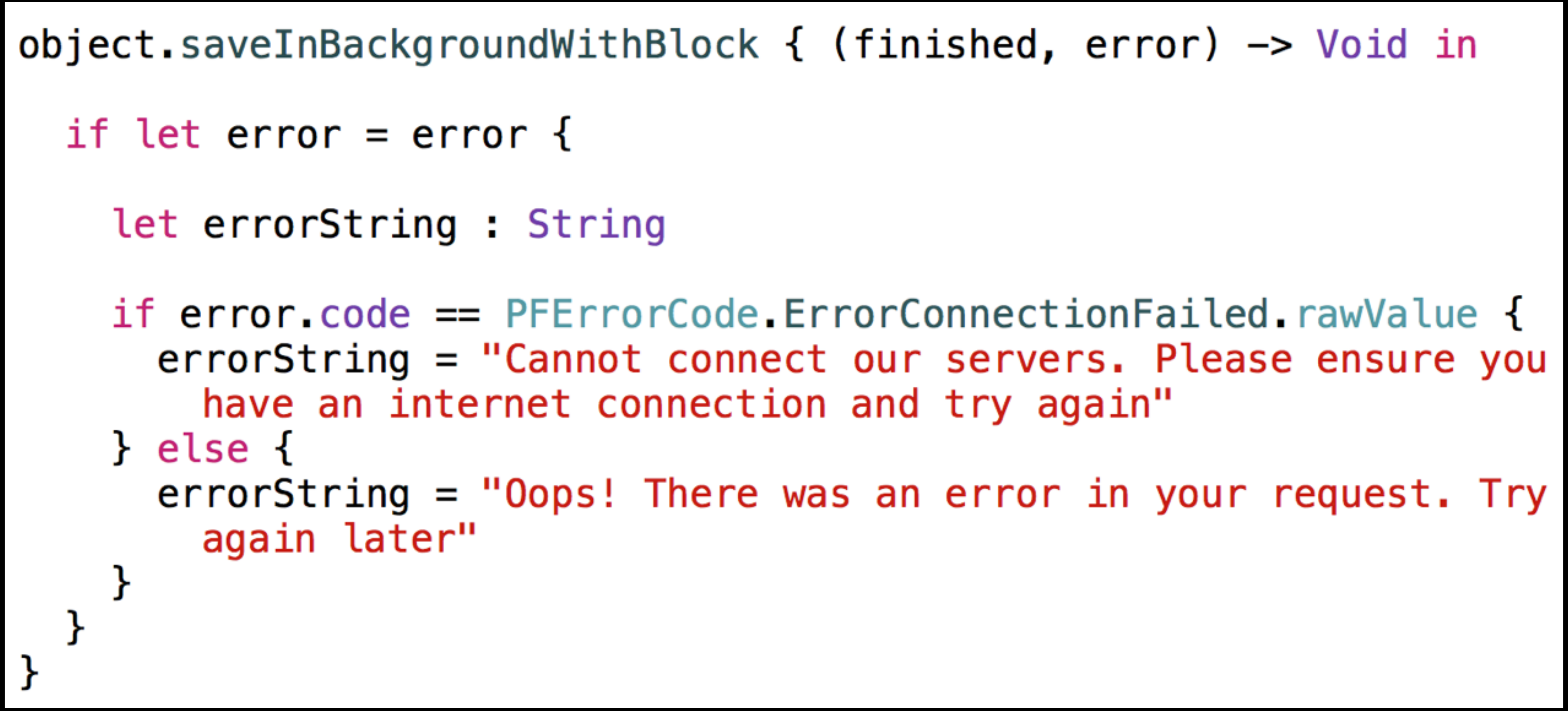


```
object.saveInBackgroundWithBlock { (finished, error) -> Void in
  if let error = error {
```

- If there is an error, we now need to handle the different error codes.

Handling the errors

- Based on what operation you are trying to perform, your error handling can differ. For a simple save, we can just if the error was a bad connection:



```
object.saveInBackgroundWithBlock { (finished, error) -> Void in

  if let error = error {

    let errorString : String

    if error.code == PFErrorCode.ErrorConnectionFailed.rawValue {
      errorString = "Cannot connect our servers. Please ensure you
        have an internet connection and try again"
    } else {
      errorString = "Oops! There was an error in your request. Try
        again later"
    }
  }
}
```

Eventually you will want to create a class that has a single responsibility of handling errors and giving you back something to show the user.

Demo

Animating Constraints

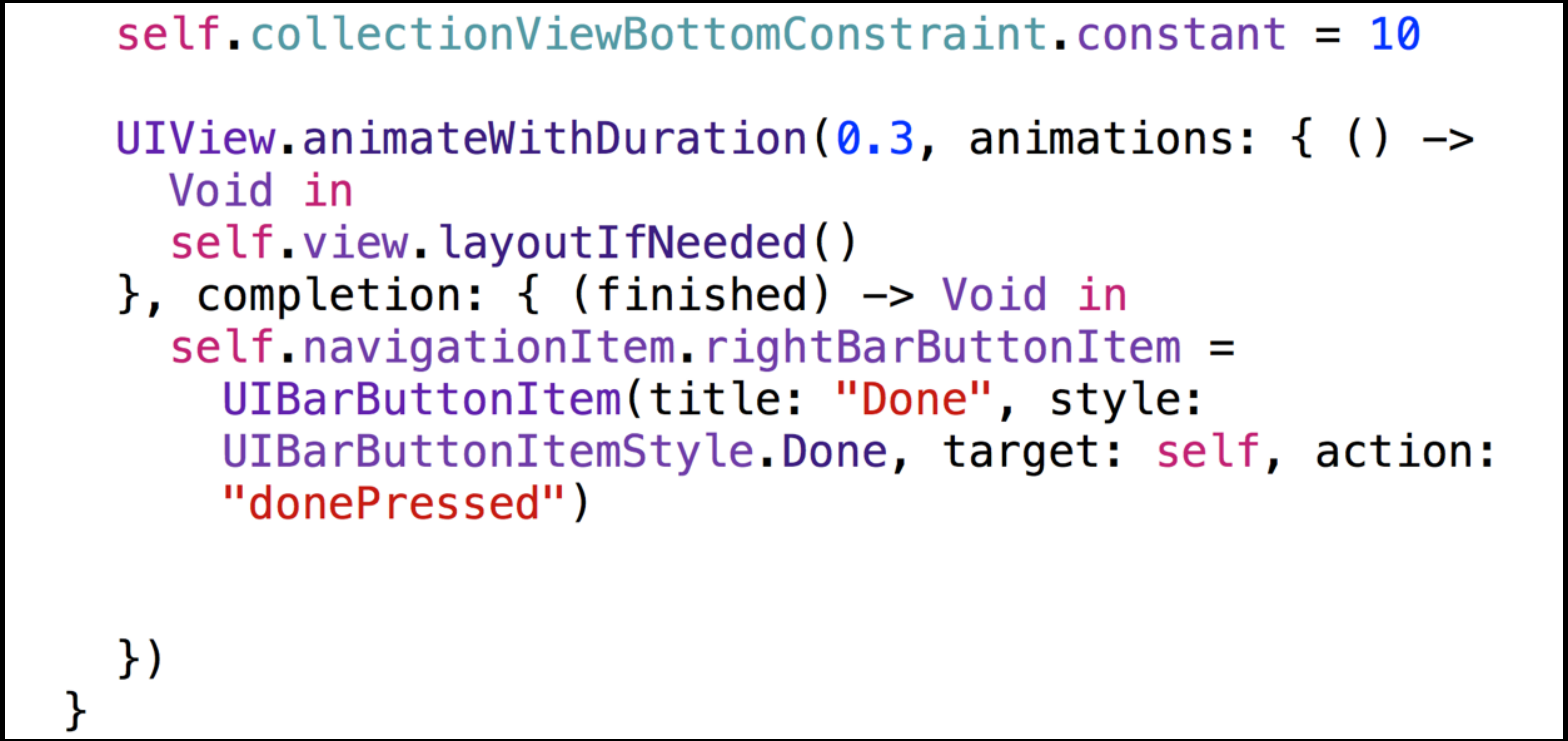
Constraints and animation

- Constraints can be animated, meaning you can smoothly move or change the size of views that are managed by autolayout
- It can be tough at first to figure out. “Well how do I get to the constraints of each view?”
- In code its a bit of a chore, but luckily, we can create IBOutlets to individual constraints from the storyboard. Hooray!

Demo

Animating Autolayout changes

1. Change the constant of the constraint
2. In your animation block, call layoutIfNeeded() on the view containing the constraints.



```
    self.collectionViewBottomConstraint.constant = 10

    UIView.animateWithDuration(0.3, animations: { () ->
      Void in
      self.view.layoutIfNeeded()
    }, completion: { (finished) -> Void in
      self.navigationItem.rightBarButtonItem =
        UIBarButtonItem(title: "Done", style:
        UIBarButtonItemStyle.Done, target: self, action:
        "donePressed")


    })
  }
```

Demo

.gitignore

- Often times you will have a set of files which git shouldn’t track.
- These are generally automatically generated logs, file system files, large SDKs or resources, or even secret API keys.
- In these cases, you can create a file called .gitignore that contains a list of the files or file patterns you want ignored by git.
- In this .gitignore file, you can list specific files, directories, or specific file types you want ignored.
- Use a # for comment lines
- github has a master list of best practice .gitignores for many programming languages.

.gitignore workflow

- Change directory in terminal to the root directory of your project (where the xcodeproj file lives)
- run the command touch .gitignore
- run open .gitingore
- paste in the list of files/directories to ignore and save the file
- add the .gitignore file (add it individually instead of with git add -A)
- Once the .gitignore is added, git will ignore any files listed in the .gitignore.
- It is important that adding the .gitignore be the first file you add to your git repository.