iOS Dev Accelerator Week 7 Day 2

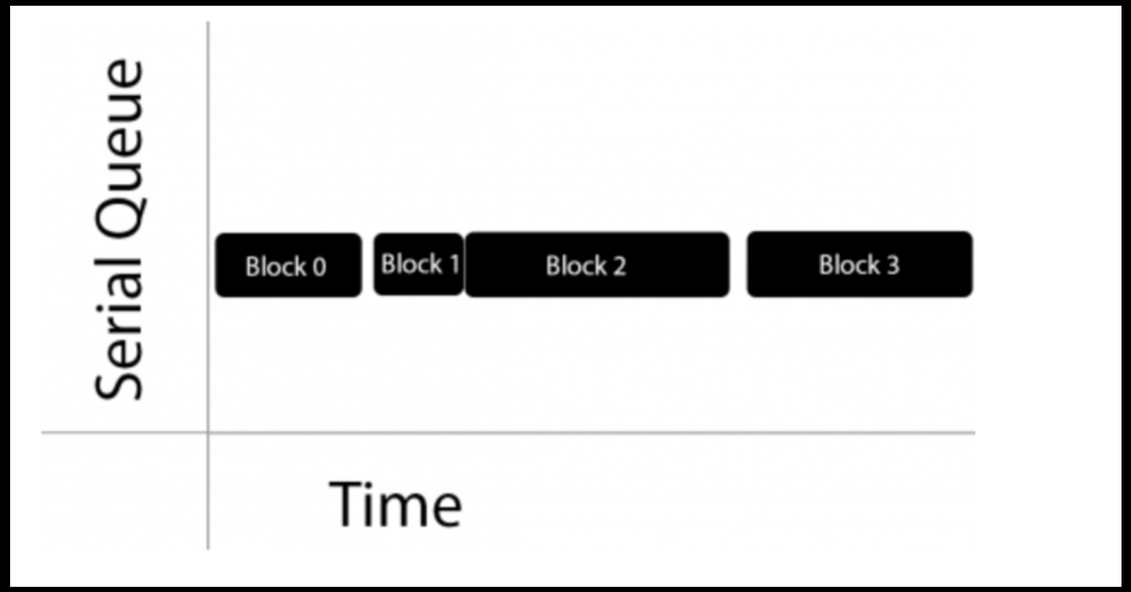
- Grand Central Dispatch
- Cocoa Pods
- More Objective-C things

Grand Central Dispatch

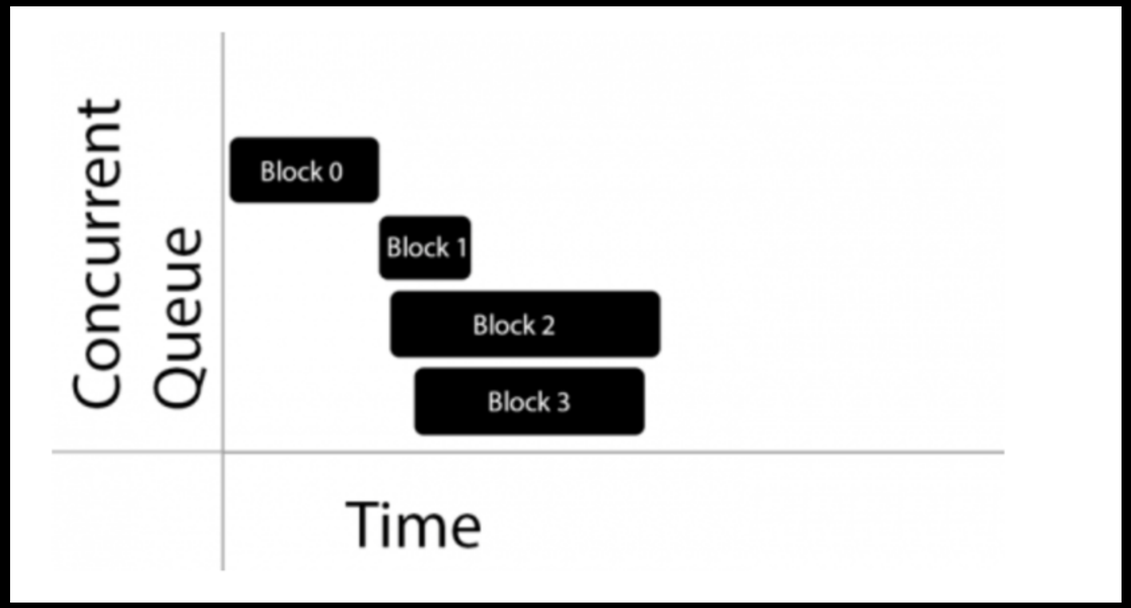
Grand Central Dispatch

- “Grand Central Dispatch (GCD) comprises language features, runtime libraries, and system enhancements that provide systemic, comprehensive improvements to the support of concurrent code execution on multicore hardware on iOS and OS X"
- GCD is a system you can use for concurrency
- GCD sits above manually creating threads and locks, and below NSOperationQueue
- GCD feels a lot like NSOperationQueue, so it’s not a drastic transition going from NSOperationQueue to GCD
- GCD is a C-style API, so you don’t call the methods on classes, they are just global functions. The functions will look_like_this

- Tasks executed serially are executed on at a time



- Tasks executed concurrently might be executed at the same time or different times



Async vs synchronous

- A synchronous function returns only after the completion of tasks that it orders
- An asynchronous function returns immediately, ordering the task to be done but does not wait for it (does not block for it)

Concurrency and structure

- Even when using a higher level API like NSOperationQueue, you need to think about and structure your app with concurrency in mind. Specifically you need to design with these things in mind:
  - **Race Conditions:** The behavior of your app depends on the timing of certain events that are being executed in an uncontrolled concurrent manner
  - **Dead Locks:** Two or more threads are deadlocked if one thread is waiting for another thread to finish, while the other thread is waiting for the first thread to finish
  - **Thread safety:** Thread safe code is safe to call from multiple threads or concurrent tasks. Code that is not thread safe must be only accessed in one context at a time. An immutable array is thread safe because it cannot be changed, while a mutable array is not thread safe since one thread could be changing it while another is trying to read it.
  - **Critical Sections:** Any piece of code that must not be executed from more than one thread or task. For example manipulating a shared resource that isn't thread safe.

GCD and Queuing Tasks for Dispatch

- GCD provides and manage FIFO queues to which your app can submit tasks in form of blocks (or closures in swift)
- Blocks submitted on dispatch queues are executed on a pool of threads that are managed by the system
- 3 types of queues:
  - Main: tasks execute serially on your app's main thread
  - Concurrent: tasks are dequeued in FIFO order, but run concurrently
  - Serial: tasks execute one at a time in FIFO order

Queuing a task for the main queue

```
NSArray *results = [Question questionsFromJSON:data];
dispatch_async(dispatch_get_main_queue(), ^{
  completionHandler(results,nil);
});
```

- dispatch_async is a function that takes in two parameters:

  1. The queue you want to execute a task on

  2. a block that represents the task you want to execute

Using a concurrent global queue

- Global Queues are created by the system for your app and are available to use without any setup
- To access a global queue you use the function dispatch_get_global_queue
- dispatch_get_global_queue has 2 parameters:
  - identifier: how you tell GCD which queue you want to enqueue tasks on
  - flags: ‘will be used in the future’ just use 0

global queue identifiers

These are the newest set of QOS identifiers to help you choose which global queue to use:

- QOS_CLASS_USER_INTERACTIVE: highest priority since the user experience relies on it
- QOS_CLASS_USER_INITIATED: should be used when the user is waiting on results to continue interaction
- QOS_CLASS_UTILITY: long running tasks, designed to be energy efficient
- QOS_CLASS_BACKGROUND: anything the user isn't directly aware of.

global queue identifiers

These are the old, and less confusing ones

- DISPATCH_QUEUE_PRIORITY_HIGH
- DISPATCH_QUEUE_PRIORITY_DEFAULT
- DISPATCH_QUEUE_PRIORITY_LOW
- DISPATCH_QUEUE_PRIORITY_BACKGROUND

You can use the old or new ones, just know that both sets exist

Using a concurrent global queue

```
  dispatch_queue_t imageQueue = dispatch_get_global_queue
(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

dispatch_async(imageQueue, ^{

  NSURL *url = [NSURL URLWithString:avatarURL];
  NSData *data = [[NSData alloc] initWithContentsOfURL:url];
  UIImage *image = [UIImage imageWithData:data];

  dispatch_async(dispatch_get_main_queue(), ^{
    completionHandler(image,nil);
  });
});
```

Creating a private queue

- You can create a private queue if you want to run tasks on a serial or concurrent queue that isn't a global queue or the main queue
- The only structural difference between using a private queue and a global queue is that a private queue can be a serial queue.
- To create a private queue use the function dispatch_queue_create which takes in 2 parameters:
  - label - string to attach to the queue to uniquely identity what process is launching these queues. Use reverse DNS, or pass in NULL
  - attr - specify DISPATCH_QUEUE_SERIAL or DISPATCH_QUEUE_CONCURRENT or NULL (which sets it to serial)

Queuing up your tasks

- The two primary ways of enqueuing your tasks for execution are:
  - dispatch_sync
  - dispatch_async

there is also dispatch_once which is great for singletons

dispatch_async

- Dispatches a task to to run asynchronously and returns immediately
- This is by bar the most common way to enqueue a task on a dispatch queue
- Even if adding tasks to a serial queue, still use dispatch_async

dispatch_sync

- Submits a block to execute on a dispatch queue and waits until that block completes before returning
- Rarely will use this

dispatch_async vs sync

```
dispatch_queue_t myQueue = dispatch_get_global_queue
  (QOS_CLASS_BACKGROUND, 0);

NSLog(@"1");

dispatch_async(myQueue, ^{
  NSLog(@"2");
});

NSLog(@"3");

dispatch_async(myQueue, ^{
  NSLog(@"4");
});
```

prints 1,3,2,4

```
dispatch_queue_t myQueue = dispatch_get_global_queue
  (QOS_CLASS_BACKGROUND, 0);

NSLog(@"1");

dispatch_sync(myQueue, ^{
  NSLog(@"2");
});

NSLog(@"3");

dispatch_sync(myQueue, ^{
  NSLog(@"4");
});
```

prints 1,2,3,4

dispatch_sync & deadlock

- If you ever call dispatch_sync on a queue and pass in that queue as the queue to dispatch to, you will get a deadlock.
- This happens because:
  - dispatch_sync blocks the queue you called it from
  - this now blocked queue will never be able to perform the task that was passed into dispatch_sync because the queue is already blocked

dispatch_sync & deadlock

```
dispatch_queue_t myQueue = dispatch_queue_create(NULL,
  DISPATCH_QUEUE_SERIAL);

dispatch_async(myQueue, ^{
  //do something expensive on this background queue

  //do another thing
  //this queue is now dead locked
  dispatch_sync(myQueue, ^{
    NSLog(@"Brad");
  });
  //will never reach this line

});
```

Demo

GCD vs NSOperationQueue

- Sometimes NSOperationQueue is a better choice than GCD. The advantages of NSOperationQueue:
  - Canceling operations is possible in NSOperationQueue, GCD is fire and forget
  - KVO compatibility (more on this later in the week)
  - Operation priorities make it easy to modify how operations are scheduled

Dispatch Groups

Dispatch Groups

- Dispatch Groups are a GCD feature that allows you to easily group tasks
- You can then wait on that set of tasks to finish or be notified by a callback when those tasks finish
- This is awesome when you are performing multiple tasks concurrently and need to be notified when they are all complete.
- A dispatch group is created with this function:
- dispatch_group_t myGroup = dispatch_group_create()

Dispatch Groups

- Dispatch Groups are a GCD feature that allows you to easily group tasks
- You can then wait on that set of tasks to finish or be notified by a callback when those tasks finish
- This is awesome when you are performing multiple tasks concurrently and need to be notified when they are all complete.
- A dispatch group is created with this function:
- dispatch_group_t myGroup = dispatch_group_create()

Associating tasks with groups

- Two ways to do this:
  - use this function dispatch_group_async(group, queue, block)
    - This looks like normal async dispatch, except it takes in a group as well.
  - use these two functions: dispatch_group_enter(group) and dispatch_group_leave(group)
    - This causes the number of tasks the group think are currently running to increase(enter) or decrease(leave). Its a simple counter. This means for every enter there must be a leave.

Waiting on a dispatch group to finish

- Two ways to be notified when a dispatch group is finished, one blocking, one not blocking:
  - dispatch_group_wait(group, timeout) ->long : This function takes the group to wait on and a time out value. This function blocks the current queue you are on, until either the group is finished or the timeout thresh hold is hit. You can pass in DISPATCH_TIME_FOREVER if you dont want a timeout
  - dispatch_group_notify(group, queue,block) : this function takes in the group, a queue to run the block, and a block. This is different than the wait because it does not block. Its Asynchronous.

Demo

Cocoa Pods

Cocoa Pods

- Cocoa Pods is a popular application that allows you to easily integrate third party code into your apps
- Check out CocoaPods.org, the official website of Cocoa pods to read up on it, see how it works, why it works, etc.
- And then check out cocoacontrols.com for a list of cool third party controls you can use.

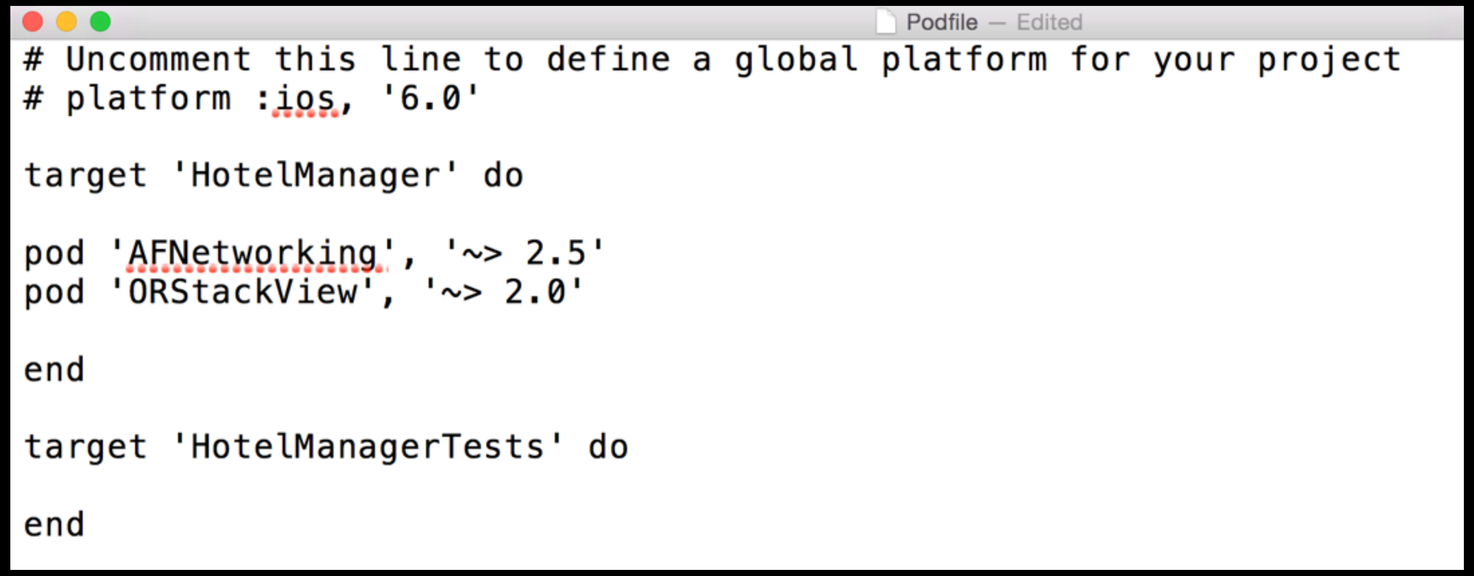
Cocoa Pods Install Workflow

- First you will need to install Cocoa Pods on your computer.
- Cocoa Pods is built with Ruby. Standard Ruby comes installed on OS X.
- Run this command in terminal to install cocoa pods :
- **sudo gem install cocoapods**

Cocoa Pods Workflow

- With cocoa pods installed, you can now use cocoa pods within any directory of your choosing.
- Once you have found a cool third party pod to use, go to the root directory of the xcode project you want to import the pod into.
- Create a text file named **Podfile**
- **Instead of manually creating it, just use the command pod init**

Podfile



- In this podfile, you list all the third party APIs you want to import into your project.

- pod ‘AFNetworking’
- would install the latest version of
- pod ‘AFNetworking’ ‘> 2.5’
- would install AFNetworking version 2.5 or higher

For full list of operators you can use on the pod line go to https://guides.cocoapods.org/using/the-podfile.html

Pod install and workspaces

- Once you have your Podfile ready to go, run this command:
- **pod install**
- This does 2 things:
  - Downloads the dependancies listed in your pod file
  - Creates a workspace files to manage both your project and the dependancies. **You must now only open this workspace file, not the project file.**

Demo

Ternary Operator

Ternary Operators

- Ternary Operators in Objective-C allow you to do if-else conditionals in a short hand syntax
- Ternary Operators take this basic form:
  - expression_to_evaluation ? expression_if_true : expression_if_false

```
NSInteger x = 10;
NSInteger y = 9;

x == y ? NSLog(@"they match") : NSLog(@"they dont match");
```

Ternary Operators

- Its common to see ternary operators used to set variables:

```
NSString *result = x < 11 ? @"less than" : @"greater than";
```

- You can nest ternary operators

```
NSString *result = i % 2 == 0 ? @"a" : i % 3 == 0 ? @"b" : i
  % 5 == 0 ? @"c" : i % 7 == 0 ? @"d" : @"e";
```

- Don’t do this, its stupid. One ternary operator is barely readable on its own.

Demo

Objective-C Constants

Using const

- 2 ways you will commonly see constants done in Objective-C:
  - Preprocessing Macros:

```
#define MAX_WIDTH 20
```

  - const:

```
const NSInteger MAX_HEIGHT = 30;
```

Preprocessing Macros

- The Objective-C Preprocessor is a separate step in the compilation process.
- It’s basically just a text substitution tool
- All preprocessor commands start with the # sign

Preprocessing Macros

- #define : substitutes a preprocessor macro
- #include : inserts a particular header from another file
- #import : improved version of include. ensures a header file is only ever included once
- #ifdef : returns true if the macro is defined
- #ifndef : returns true if the macro is not defined

#define

- When you define a macro using #define, the compiler will just substitute anywhere it sees the macro with the value you defined it with.

```
#define MAX_WIDTH 20
```

```
self.view.center = CGPointMake(MAX_WIDTH, 20);
```

At compile time, this is just replaced with the value 20

Demo

Using const

- Declare constants using the const keyword
- Any type can be a constant
- Value cannot change, once declared.
- Think of let keyword in Swift

```
NSString * const ReuseIdentifier = @“ReuseIdentifier";

int const limit = 100
```

Sharing const

- Expose constants in a header file to use from multiple points
- Define the actual value of the constants in an implementation file
- Useful technique to define constants and use across the app

Sharing const

- Create Constants.h and Constants.m
- In Constants.h, expose the const variables

```
extern NSString *const MyFirstConstant;
```

```
The extern keywords make this a globally available constant as long as you import constant.h
```

- Extern just lets the compiler know, and anyone reading your code that this variable is declared here, but defined somewhere else.
- In Constants.m, define the const variables value

```
NSString *const MyFirstConstant = @"FirstConstant";
```

Demo

Enums

Enums

- An enum, or enumerator, is just a list of predefined variables.
- Enums are a data type, just like ints and doubles.
- An enum variable can only hold values defined in the enum.
- To define an enum, you use the keyword enum
- Values in an enum are ints under the hood.

Enums

- Defining an enum is as simple as using the enum keyword, and then defining a list of comma separated options
- to create a variable with your enum type, also use the enum keyword
- same idea if you want to have it as a parameter

```
enum genre {
  comedy,
  romance,
  action,
  sci_fi
};
```

```
enum genre favoriteGenre = sci_fi;
```

```
-(void)findMoviesForGenre:(enum genre)genre {
  //find movies
}
```

Enums

- Enums default by starting at the value 0, but you can give each option a custom number.
- If you leave the option after a custom number without a custom number, it just takes the next number
- So sci_fi here is 13

```
enum genre {
    comedy,
    romance = 11,
    action = 12,
    sci_fi
};
```

Demo

Typedef

Typedefs

- Objective-C (and also Swift) provides a keyword called typedef, which lets you give types a new name.
- It can be something as simple as giving NSString a new nickname:

```
typedef NSString word;
```

- Or as complicated as a block:

```
typedef void (^downloadCallback)(NSArray *, NSError *);
```

refer to fuckingblocksyntax.com when you need to do this

Typedefs and enums

- Typedefs are great to use with enums.
- You can declare an anonymous enum, and then give that enum a typedef
- this way when you declare variables or want to use the enum as a parameter or return type, you dont have to use the keyword enum:

```
typedef enum {
  apple,
  samsung,
  HTC,
  Nokia
} brand;
```

Demo