iOS Dev Accelerator
Week 5 Day 2

- Blocks
- CoreLocation
- Map Annotations
- Map Overlay
- Encapsulation

Blocks

- Introduced in ObjC 2.0
- Serve the exact same purpose as Closures in Swift
- Used extensively in Cocoa APIs that accept completion blocks
- Can be anonymous or stored local variables or properties, same as closures
- Very easy to introduce retain cycle, if you’re not careful.
- Syntax not easy to remember: http://goshdarnblocksyntax.com

Blocks

- Block are denoted with the caret operator ^
- Declare a return type; typically void on completion blocks
- Include types of parameters. Do not need to be named in method prototype

```
- (void)loadRoutes:(void (^)(NSArray *, NSError *))success;
```

```
[[RFRoutesModel sharedModel] loadRoutes:^(NSArray *routes, NSDictionary *err) {
    //
}];
```

Weak self and blocks

- When referring to self inside a block (or closure), best practice is to create a weak reference in self before the block and use that reference instead of self.
- Since blocks and closures capture strong references to all objects accessed in their bodies, if the object represented by self were to get a strong reference to the block or closure, then we would have a retain cycle.
- We can use the __weak modifier in Objective-C to avoid this

Demo

CoreLocation

Core Location

- Core Location provides several services that you can use to get and monitor the devices current location:
  - The significant-change location service provides a low power way to get the current location and be notified when significant changes occur .
  - The standard location service offers a highly configurable way to get current location and track changes.
  - Region monitoring lets you monitor the boundary crossing defined by geographical regions and bluetooth low energy beacon regions (iBeacons!).

Location services availability

- Situations where location services might not be available
  - The user disables location services in the settings app or system preferences
  - the user denies location services for a specific app
  - the device is in airplane mode and unable to power up the necessary hardware
- You should always call locationServicesEnabled class method on CLLocationManager before attempting to start services. If it returns NO and you attempt to start the services anyway, the user will be prompted to turn on the services, which may be annoying after one time.

CoreLocation Authorization

- CLLocationManager has a class method for retrieving the authorization status: authorizationStatus()
- Also has a delegate method when the authorization status changes: locationManager:didChangeAuthorizationStatus:
- You can request authorization for always running (foreground+background) or just foreground.
- Upon requesting, the user will be presented with the text you have stored in your NSLocationAlwaysUsageDescription or NSLocationWhenInUseDescription keys in your info.plist. **You must have these keys for the system to ask the user for permission.**

Standard Location Service

- Most common way to get a user's current location since its available on all devices.
  - Before activating, you specify the desired accuracy and the distance that must be traveled before reporting a new location.
  - Once you start the service, it uses those parameters to determine which hardware to use.
  - Create an instance of CLLocationManager class to get your services setup.
  - You will need authorization before activating it.

CLLocationManager

- Configure the desiredAccuracy and distanceFilter properties before starting the services.
- The distance filter is used for movement. When the location manager has detected movement that is larger than the distance filter, an update is delivered. Not setting it means you want updates for any small movement, which will be a lot.
- Desired accuracy tells the location manager how accurate readings should be that are delivered to the delegate. The more accurate you want, the more battery power your app will eat up. Use the absolute minimum your app can get away with.
- To start the services, make sure you have a delegate set and then call startUpdatingLocation method.
- The delegate will now be updated as location data becomes available.
- Call stopUpdatingLocation to stop the updates.

Significant-Change Location Service

- Provides accuracy that is good enough for most apps, and saves significantly more power than regular location services.
- This service uses Wi-Fi to determine the user's location and report changes in that location.
- To begin monitoring significant changes, setup your CLLocationManager with a delegate, and then call startMonitoringSignificantLocationChanges method.

Demo

Receiving Location Data

- The way you receive location events is the same whether you use the standard or the significant-change location services.
- The location manager reports to the delegate by sending locationManager:didUpdateLocations: method.
- If theres an error with the location, it sends locationManager:didFailWithError

CLLocation

- The delegate method sends an array of CLLocation objects.
- CLLocation represents location data.
- Incorporates geographical coordinates and altitude of the devices location along with accuracy measurements and when the measures were taken.
- In iOS, this object also has data about the speed and heading of the device.

CLLocation Properties

- coordinate : CLLocationCoordinate2D - struct with two values: latitude and longitude, both in degrees. Positive values north of equator and negative south of the equator.
- altitude : CLLocationDistance - A double, positive values indicate above the sea level, negative below.
- course : CLLocationDirection - The direction in which the device is traveling. Measured in degrees starting at due north and continuing clockwise. North is 0 degrees, east is 90,etc. Negative means invalid.
- horizontalAccuracy : CLLocationAccuracy - A double, the lat and long identify the center of the circle, this value indicates the radius of that circle.
- timeStamp : NSDate - can be used to figure out how ‘stale’ the location update is. Use timeIntervalSinceNow.

CLLocationDistance

- CLLocation has an instance method that takes in another cllocation and returns a CLLocationDistance.
- CLLocationDistance is a double and is in meters.

Demo

Map View Annotations

MKMapView Annotation

- “Annotations display content that can be defined by a single coordinate point”
- User’s current location, a specific address, single points of interest, etc.
- Remain fixed to the map.
- “Map Kit separates the data associated with an annotation from its visual presentation on the map” (allows for great performance, even with hundreds of annotations)

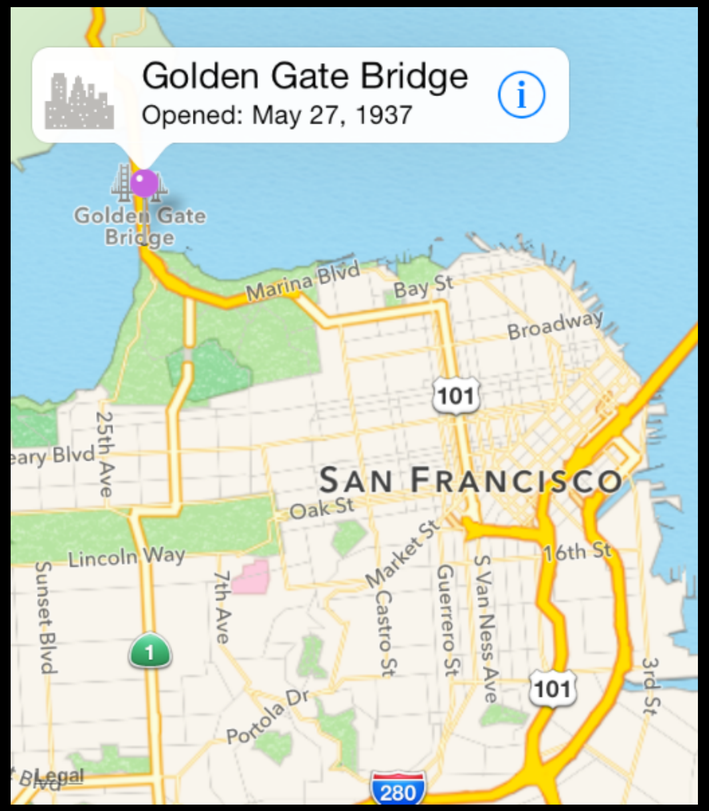
Adding Annotations

- To display an annotation, you need two objects:
- An annotation object, which is an object that conforms to the MKAnnotation protocol and manages the data for the annotation. Typically pretty small object.
- An annotation view, which is a view used to draw the visual representation of the annotation on the map.

Adding Annotations

1. Instantiate an MKPointAnnotation instance, or you own custom class that conforms to MKAnnotation protocol.
2. Give that annotation instance data (coordinates, optional title and subtitle for your custom class)
3. Call addAnnotation: or addAnnotations: on your map view.
4. **Optional:** Implement mapView:ViewForAnnotation delegate method on your map view's delegate to return an AnnotationView for a given annotation. If you don't implement this method, you will get default behavior of a plain red pin with no custom behavior.

Annotation Callouts



- “A callout is a standard or custom view that can appear with an annotation view”
- Standard callout displays the title, and can display additional info or images.
- The annotation object **needs a value set for its title property or the callout wont show.**
- Just set the canShowCallout property to true on the annotation view.
- Has left and right accessoryView properties that can be set to buttons, images, etc.
- mapView:calloutAccessoryControlTapped: tells you which annotation and callout was tapped.

Demo

Map Overlays

Map Overlays

- “Overlays offer a way to layer content over an arbitrary region of the map”
- Overlays are usually defined by multiple coordinates, which is different from the single point of an annotation.
- Overlays can be contiguous or noncontiguous lines, rectangles, circles, and other shapes.
- Those shapes can then be filled and stroked with color.

Overlay overview

- Getting an overlay onscreen involves two different objects working together:
  - An overlay object, which is an instance of a class that conforms to the MKOverlay protocol. This object manages the data points of the overlay.
  - An overlay renderer, which is a subclass of MKOverlayRenderer that does the actual drawing of the overlay onto the map.

Overlay workflow

1. Create an overlay object and add it to the map view
2. return an overlay renderer in the delegate method mapView:rendererForOverlay:

Overlay objects

- Overlay objects are typically small data objects that simply store the points that define what the overlay should represent and other attributes.
- MapKit defines several concrete objects you can use for your overlay objects if you want to display a standard shaped overlay.
- Otherwise, you can use any class as an overlay object as long as it conforms to the MKOverlay protocol.
- The map view keeps a reference to all overlay objects and uses the data contained in those objects to figure out when it should be displaying those overlays.

Overlay objects

- Concrete objects: MKCircle, MKPolygon, MKPolyline
- MKTileOverlay: use if your overlay can be represented by a series of bitmap tiles
- Subclass MKShape or MKMultiPoint for custom shapes
- Use your own objects with the MKOverlay protocol

OverlayRenderer

- MapKit provides many standard overlay renderers for standard shapes.
- You don’t add the renderer directly to the map, instead the delegate object provides an overlay renderer when the map view asks for one.

OverlayRenderer Objects

- Standard shapes: MKCircleRenderer, MKPolygonRenderer, MKPolylineRenderer.
- Tiled: MKTileOverlayRenderer
- Custom shapes : MKOverlayPathRenderer
- For the most customization, subclass MKOverlayRenderer and implement your custom drawing code.

Demo

Encapsulation

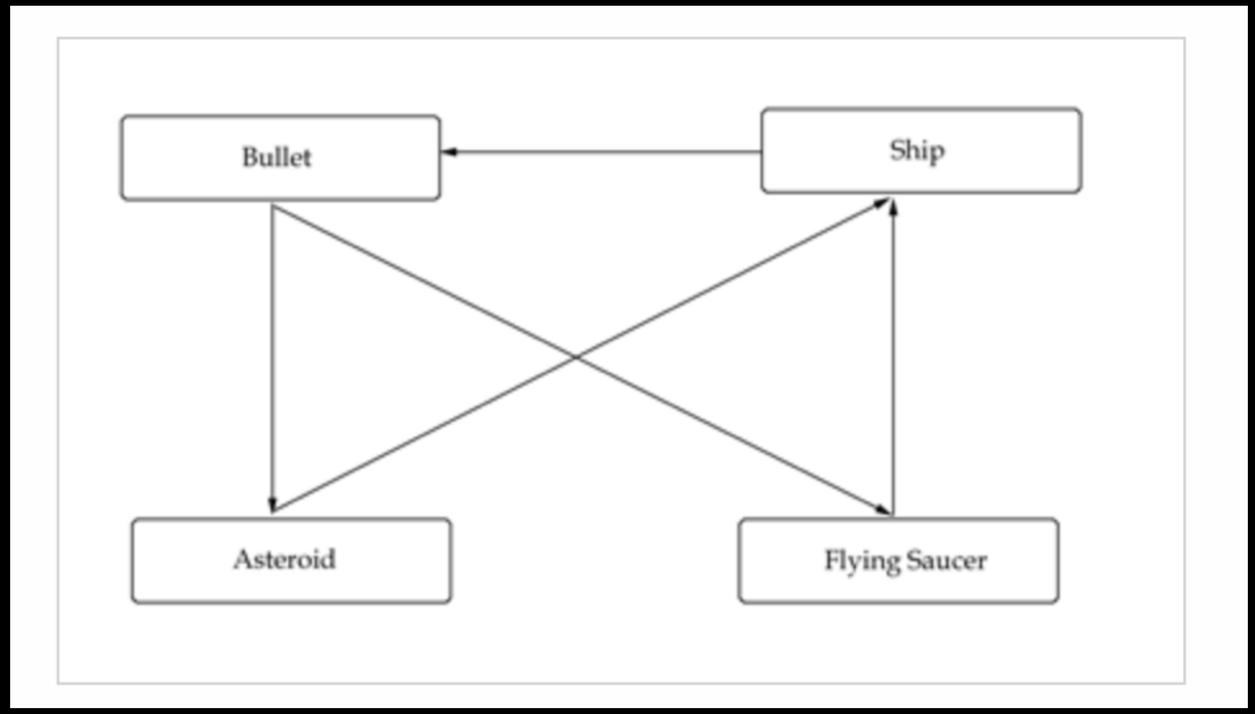
- In computer science and object oriented programming, encapsulation means that the internal representation of an object is generally hidden from view outside of the object's definition.
- Usually only the object's own methods can directly inspect or manipulate its data.
- When a programming language has features to help you with encapsulation, it is usually referred to as access control.
- In Objective-C, the separation of .h and .m does most of the work for us.

Encapsulation Benefits

- Encapsulation provides a number of benefits:
  - **Hides complexity:** Think of a car. A car has super complex internals. But we as users only interface with simple controls, like the wheel and ignition. Imagine if you had to actually know exactly how a car works to use it. We can consider those wheel and ignition controls its public interface.
  - **It helps us achieve loose coupling:** Encapsulation helps us decouple modules that comprise our app, allowing them to be developed, used, tested, and understood in isolation.
- Lets look at examples of these concepts.

Tight Coupling

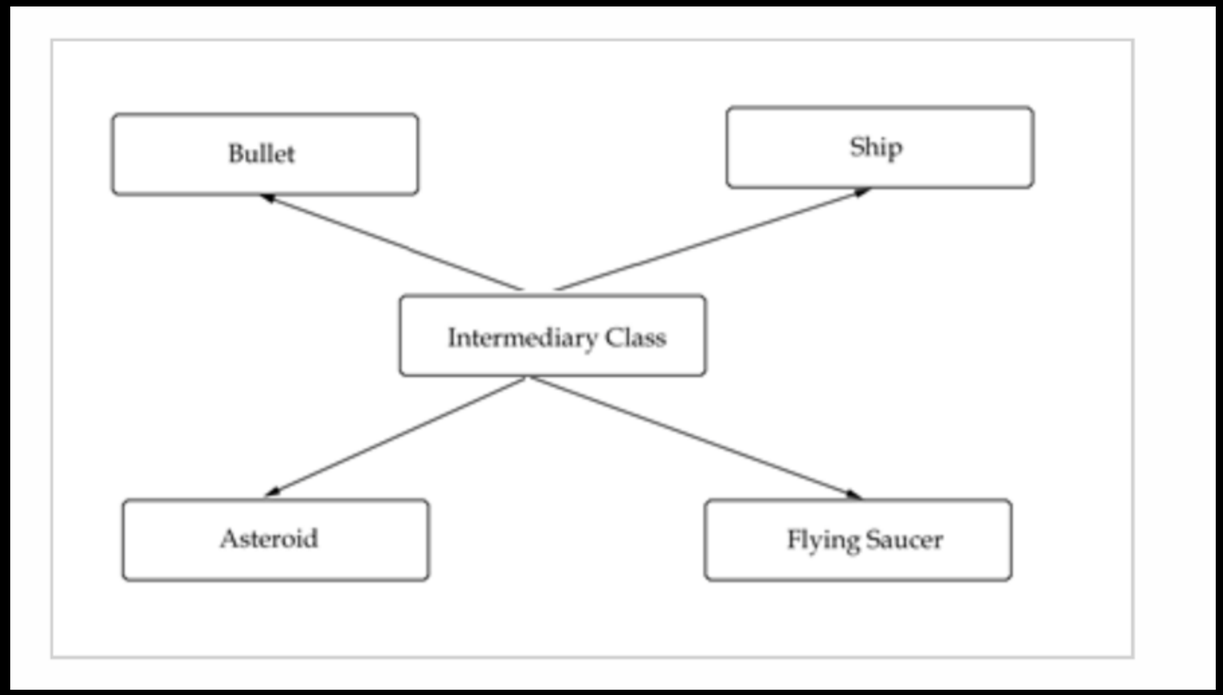
- Lets take a look at a game like Asteroids for example
- Asteroids has 4 main classes : Bullet, Ship, Asteroid, and Enemy
- With tight coupling, the ship would fire a bullet, keep track and update the bullets location, check if it hit an asteroid, and if it did destroy the asteroid, or destroy an enemy.



The issue with this is if later on down the road, for example, if we change how bullets work, or change the bullet to a lazer, we will have to update all the other classes appropriately.

Loose Coupling

- Instead lets design a system where the ship fires a bullet, and then doesn't have to worry about it after that point.
- We will let another class, maybe a class called GameLogic, handle that stuff
- With this GameLogic class able to work as an intermediary class, our diagram looks much cleaner:



Now if we change how bullets work, we only have to update our intermediary class. Bullet and Ship are loosely coupled.

Hiding Complexity Demo