iOS Dev Accelerator Week5 Day1

- MapKit
- Map Annotations
- CoreLocation
- RegionMonitoring
- Tab Bar Controller

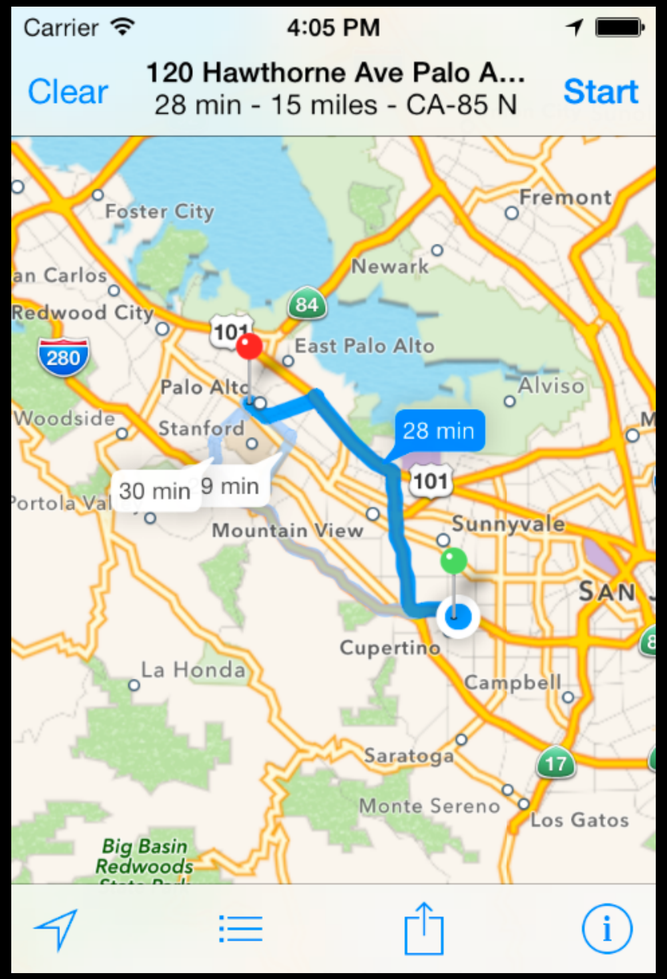
Locations and Maps

- “Location-based information consists of two pieces: location services and maps”
- Location services are provided by Core location framework
- Maps are provided by the MapKit framework
- Both frameworks available on iOS and OSX

MapKit

MapKit

- “The Map Kit framework lets you embed a fully functional map interface into your app”
- Provides many of the same features that the maps app on iOS and OSX does.
- Street level map info, satellite imagery, etc.
- Programmatically zoom, pan, and pinch, display 3d buildings, and show customized annotations, in addition to regular touch controls.



MapKit Info

- To enable Maps in your app, turn on the Maps capability in your Xcode project.
- Any map view in your app represents a flattened earth sphere.
- Uses a Mercator map projection to project the round earth to a flat view.
- Uses Prime Meridian as its central meridian.

MapKit data points

- Map Kit supports 3 basic ways to specify map data points:
  - **Map Coordinate**: is a lat/long pairing using the CLLocationCoordinate2D struct. Also used to specify areas with MKCoordinateSpan and MKCoordinateRegion.
  - **Map Point**: An x and y value on the map projection. Mainly used for custom overlays and designating their shape and location on your view.
  - **A Point**: Regular CGPoint, used in CGSize and CGRect.
- The data types you use is usually chosen for you by the API you are implementing. When saving locations in files or inside your app, coordinates are the most precise.

Getting Map view on screen

- “MKMapView is a self contained interface for presenting map data in your app”
- Displays map data, managers user interactions, and hosts custom content provided by your app.
- Never subclass MKMapView.
- Has a delegate object which the map view reports all relevant interactions to.
- Drag onto your storyboard, or programmatically initialize an instance of MKMapView.

MKMapView Specifics

- Keep in mind an MKMapView is still a view, so you can manipulate its size and position, and add subviews to it just like any other view.
- Subviews added to it will not scroll when the user pans the map, its like an overlay.
- If you want something to stay stationary relative to a specific map coordinate, you just use an annotation.
- By default a new map view is configured for user interaction and to display map data. Uses a 3D perspective by enabling pitch (not available on simulator!).
- Manipulate the pitch and rotation by creating an MKMapCamera Object (more on this later).
- Change its Type attribute to go from satellite to map data or a mix.
- You can limit user interaction by changing values in rotateEnabed, pitchEnabled, zoomEnabled, etc properties.
- Use the delegate to respond to user actions.

MKMapView Region

- “The region property of the MKMapView class controls the currently visible portion of the map”
- Typically when a map view is created, its region is set to the entire world.
- Change this by setting a new value to region, has a setRegion:animated method for smooth zooming.
- The properties type is MKCoordinateRegion struct, with two values:
  - center : CLLocationCoordinate2D
  - span : MKCoordinateSpan
- Span defines how much of the map at a given point should be visible. MKCoordinateSpan is measured in degrees, but you can create one with meters with MKCoordinateRegionMakeWithDistance.

Demo

MKMapView Zooming and Panning

- Zooming and panning can be done programmatically in addition to user interaction.
- To pan, but keep the same zoom level, change the value in the centerCoordinate property of the map view. Or call the map view's setCenterCoordinate:animated method to animate the change.
- To change the zoom level (and optionally pan the map), change the value in the region property of the map view. Or use the animated method.

MKMapView and User Location

- MKMapView comes with built in support for displaying the user's location.
- Just set the showsUserLocation property to true.
- Shows the user's location and adds an annotation on the users location.
- Interfacing with CoreLocation required to use this (more on CoreLocation later)

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)

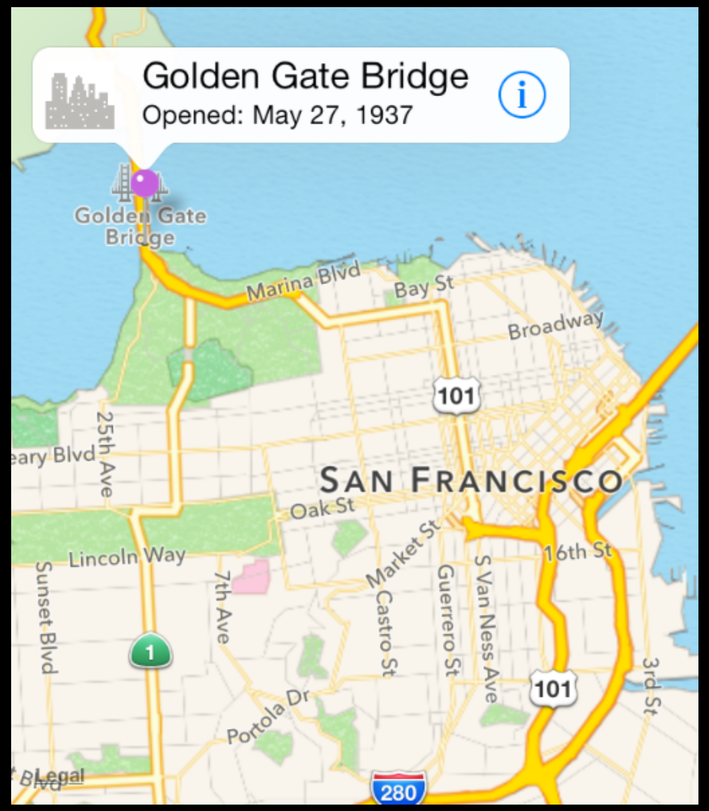
Adding Annotations

- 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.
- 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

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.

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.

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.**

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 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

Region Monitoring

Region Monitoring

- “A geographical region is an area defined by a circle of a specified radius around a known point on the Earth’s surface.”
- Apps can use region monitoring to be notified when a user crosses specific boundary.
- “In iOS, regions associated with your app are tracked at all times, included when the app isn't running”
- If it detects a region crossing, your app is relaunched in the background.
- Regions are considered a shared resource on the device, so any app can only have a maximum of 20 regions registered for monitoring at a time.

Region Monitoring Availability

- Reasons why region monitoring may not be available to the user:
  - hardware not available
  - user denied app authorization for region monitoring
  - user disabled location services in the settings app
  - the user disabled background app refresh in the settings app
  - the device is in airplane mode
- First ask the CLLocationManager class if region monitoring is available with the method isMonitoringAvailableForCass:
- If that returns true, then ask CLLocationManager if location services are enabled with authorizationStatus()

Setting up a Monitor

- In iOS7+ you define geographical regions using the CLCircularRegion class.
- Each region created should include both the data that defines the desired geographical area and a unique identifier string.
- To register a region, call startMonitoringForRegion: method on your CLLocationManager object.
- By default, every time a user's current location crosses a boundary region, the system generates an appropriate region event for your app.
- 2 methods: locationManager:didEnterRegion: and locationManager:didExitRegion:

Demo

Tab Bar Controller

UITabBarController

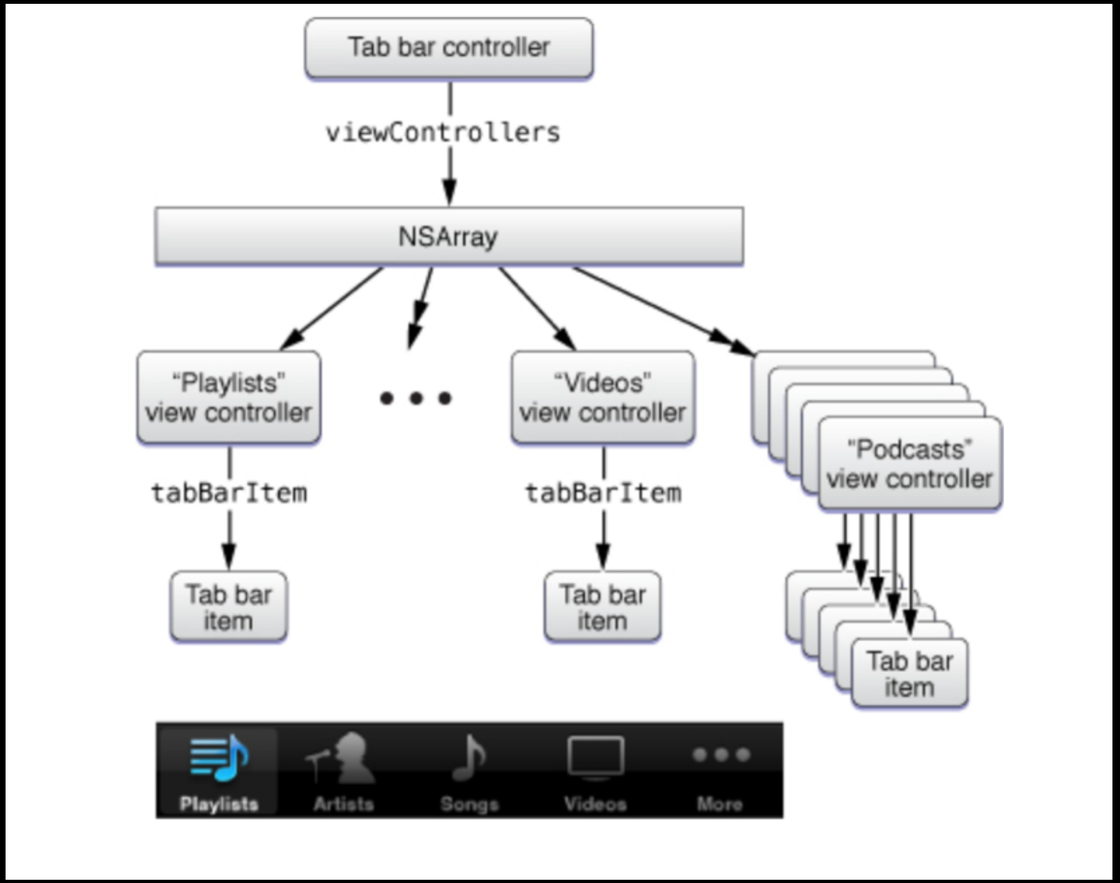
- “You use tab bar controller to organize your app into one more **distinct modes of operation**”
- UITabBarController is an example of a container view controller.
- “Each content view controller manages a distinct view hierarchy, and the tab bar controller coordinates the navigation between the view hierarchies”

Anatomy of the tab bar controller

1. The UITabBarController object
2. One content view controller object for each tab
3. an optional delegate

Tab Bar

- “The key component of a tab bar interface is the presence of a tab bar view along the bottom of the screen”
- You do not modify the tab bar view directly.
- The bar controller assemble the tabs by looking at the UITabBarItem object provided by each of its content view controllers.



Creating a tab bar controller

- 2 ways to get a tab bar controller in your app:
    1. Via storyboard: You can drag out a tab bar controller from the object library, or just Editor> Embed> Tab Bar Controller
    2. In code: Typically done in applicationDidFinishLaunching:
        1. Create a UITabBarController object
        2. Create one content View controller for each tab you want
        3. Add the view controllers to an array and set that array to the viewControllers property of the UITabController object.
        4. set the UITabBarController as the root view controller of your window.

Demo