# Stanford CS193p

## Developing Applications for iOS

## Fall 2013-14

# Today

- `UIApplication`
  - What's that?
- Network Activity Indicator
  - An application wide activity spinner for network activity only
- Demo Followup
  - A couple of things to note about last week's demo
- Demo
  - More Photomania (iPad version with popover)
- Maps
  - Showing whether things are on earth
  - We'll get as far as we can, then continue on Wednesday (along with a demo)

# UIApplication

- UIApplication

  There is a shared instance of a `UIApplication` object in your application.

  This is different from your Application Delegate (the thing that handles all those message from iOS).

  You almost never need it, but it can give you some interesting (very global) information.

  ```
  UIApplication *myApplication = [UIApplication sharedApplication];
  ```

  Check out its documentation.

# Network Activity Indicator

- Network Activity Indicator
  - This property in UIApplication is interesting …
  - `@property (nonatomic, getter=is…) networkActivityIndicatorVisible;`
  - When this is set to YES, a little spinner will appear in the status bar. NO means turn it off.
  - This spinner is ONLY for network activity (but you should spin it for ALL network activity you do).
- It can be somewhat difficult to use this property correctly
  - Because it is global and is a boolean.
  - What if you have multiple, overlapping threads using the network at the same time?
  - You are required to layer mechanism for that on top of this property yourself.

# Demo Followup

- We forgot to set our minimum background fetch interval

```
[[UIApplication sharedApplication] setMinimumBackgroundFetchInterval:(NSTimeInterval)interval];
```

  The default is `UIApplicationBackgroundFetchIntervalNever`, so set it or you get none!
  Minimum you can set it to is `UIApplicationBackgroundFetchIntervalMinimum` (often want this).
  Usually you would set this in `application:didFinishLaunchingWithOptions:`.

  Also, the user can turn off your application's ability to run in the background entirely!

```
@property UIBackgroundRefreshStatus backgroundRefreshStatus;
```

- Fetching when given the opportunity

  When we are given the opportunity to fetch in the background, we should do a normal fetch.
  In other words, do a normal, ephemeral URL session fetch, not a background session URL fetch.
  Background session URL fetches are discretionary (meaning iOS can refuse if in background).
  The posted code from last week does this.
  Doing a normal fetch also makes it easier to call the completion handler with the `NewData` option!

# Demo

- More Photomania!

  Flesh out Photomania on iPad & add the table of photos by the photographer and an image VC.
  Then we'll add a popover to show the URL of the photo we're looking at.

# Core Location

- Framework for managing location and heading

  No user-interface.

- Basic object is `CLLocation`

  `@propertys: coordinate, altitude, horizontal/verticalAccuracy, timestamp, speed, course`

- Where (approximately) is this location?

```
@property (readonly) CLLocationCoordinate2D coordinate;
typedef {
    CLLocationDegrees latitude;   // a double
    CLLocationDegrees longitude;  // a double
} CLLocationCoordinate2D;

@property (readonly) CLLocationDistance altitude;  // meters
```

  A negative value means “below sea level.”

# Core Location

- How close to that latitude/longitude is the actual location?

```
@property (readonly) CLLocationAccuracy horizontalAccuracy;   // in meters
@property (readonly) CLLocationAccuracy verticalAccuracy;     // in meters
```

A negative value means the coordinate or altitude (respectively) is invalid.

```
kCLLocationAccuracyBestForNavigation  // phone should be plugged in to power source
kCLLocationAccuracyBest
kCLLocationAccuracyNearestTenMeters
kCLLocationAccuracyHundredMeters
kCLLocationAccuracyKilometer
kCLLocationAccuracyThreeKilometers
```

- The more accuracy you request, the more battery will be used
  - Device "does its best" given a specified accuracy request
  - Cellular tower triangulation (not very accurate, but low power)
  - WiFi node database lookup (more accurate, more power)
  - GPS (very accurate, lots of power)

# Core Location

- Speed

```
@property (readonly) CLLocationSpeed speed;          // in meters/second
```

  Note that the speed is instantaneous (not average speed).
  Generally it's useful as "advisory information" when you are in a vehicle.
  A negative value means "speed is invalid."

- Course

```
@property (readonly) CLLocationDirection course;    // in degrees, 0 is north, clockwise
```

  Not all devices can deliver this information.
  A negative value means "course is invalid."

- Time stamp

```
@property (readonly) NSDate *timestamp;
```

  Pay attention to these since locations will be delivered on an inconsistent time basis.

- Distance between CLLocations

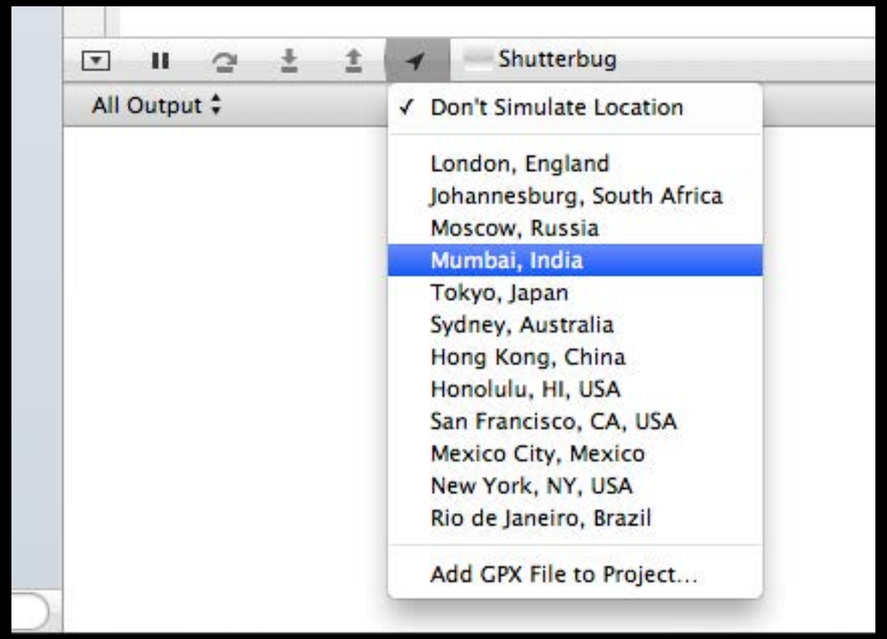
```
- (CLLocationDistance)distanceFromLocation:(CLLocation *)otherLocation; // in meters
```

# Core Location

- How do you get a `CLLocation`?

  Almost always from a `CLLocationManager` (sent to you via its `delegate`).

  Can be tested in the simulator from Xcode.



# Core Location

- How do you get a `CLLocation`?

  Almost always from a `CLLocationManager` (sent to you via its `delegate`).

  Can be tested in the simulator from Xcode.

- `CLLocationManager`

  General approach to using it:

  1. Check to see if the hardware you are on/user supports the kind of location updating you want.
  2. Create a `CLLocationManager` instance and set the delegate to receive updates.
  3. Configure the manager according to what kind of location updating you want.
  4. Start the manager monitoring for location changes.

# Core Location

- Kinds of location monitoring
  - Accuracy-based continual updates.
  - Updates only when "significant" changes in location occur.
  - Region-based updates.
  - Heading monitoring.

# Core Location

- Checking to see what your hardware can do

```
+ (CLAuthorizationStatus)authorizationStatus; // Authorized, Denied or Restricted (parental, enterprise)
+ (BOOL)locationServicesEnabled; // user has enabled (or not) location services for your application
+ (BOOL)significantLocationChangeMonitoringAvailable;
+ (BOOL)isMonitoringAvailableForClass:(Class)regionClass;  // [CLBeacon/CLCircularRegion class]
+ (BOOL)isRangingAvailable;   // device can tell how far it is from beacons
```

  Other tests for other location capabilities too.

- Getting the information from the `CLLocationManager`

  You can just ask (poll) the `CLLocationManager` for the location or heading, but usually we don't.

  Instead, we let it update us when the location changes (enough) via its `delegate` …

# Core Location

- Error reporting to the delegate

```
- (void)locationManager:(CLLocationManager *)manager
       didFailWithError:(NSError *)error;
```

Not always a fatal thing, so pay attention to this delegate method. Some examples …

```
kCLErrorLocationUnknown   // likely temporary, keep waiting (for a while at least)
kCLErrorDenied            // user refused to allow your application to receive updates
kCLErrorHeadingFailure    // too much local magnetic interference, keep waiting
```

# Core Location

- Accuracy-based continuous location monitoring

```
@property CLLocationAccuracy desiredAccuracy;  // always set this as low as will work for you
```

You can also limit updates to only occurring if the change in location exceeds a certain distance …

```
@property CLLocationDistance distanceFilter;
```

- Starting and stopping normal position monitoring

```
- (void)startUpdatingLocation;
- (void)stopUpdatingLocation;
```

Be sure to turn updating off when your application is not going to consume the changes!

- Get notified via the CLLocationManager's delegate

```
- (void)locationManager:(CLLocationManager *)manager
    didUpdateLocations:(NSArray *)locations; // of CLLocation
```

- Similar API for heading (CLHeading, et. al.)

# Core Location

- Background

  It is possible to receive these kinds of updates in the background.
  Apps that do this have to be very careful (because these updates can be power hungry).
  There are very cool ways to, for example, coalesce and defer location update reporting.
  Have to enable backgrounding (in the same area of your project settings as background fetch).

  But there are 2 ways to get location notifications (on a coarser scale) without doing that ...

# Core Location

- Significant location change monitoring in CLLocationManager
  - "Significant" is not strictly defined. Think vehicles, not walking. Likely uses cell towers.
  - `- (void)startMonitoringSignificantLocationChanges;`
  - `- (void)stopMonitoringSignificantLocationChanges;`
  - Be sure to turn updating off when your application is not going to consume the changes!
- Get notified via the CLLocationManager's delegate
  - Same as for accuracy-based updating if your application is running.
- And this works even if your application is not running!
  - (Or is in the background.)
  - You will get launched and your Application Delegate will receive the message `application:didFinishLaunchingWithOptions:` with an options dictionary that will contain `UIApplicationLaunchOptionsLocationKey`
  - Create a CLLocationManager (if you don't have one), then get the latest location via `@property (readonly) CLLocation *location;`
  - If you are running in the background, don't take too long (a few seconds)!

# Core Location

- Region-based location monitoring in CLLocationManager
  - `- (void)startMonitoringForRegion:(CLRegion *)region; // CLCircularRegion/CLBeaconRegion`
  - `- (void)stopMonitoringForRegion:(CLRegion *)region;`
  - Alloc and initWithCenter:radius:identifier: a CLCircularRegion to monitor an area.
  - Beacons are for detecting when you are near another device. New in iOS 7.
- Get notified via the CLLocationManager's delegate
  - `- (void)locationManager:(CLLocationManager *)manager didEnterRegion:(CLRegion *)region;`
  - `- (void)locationManager:(CLLocationManager *)manager didExitRegion:(CLRegion *)region;`
  - `- (void)locationManager:(CLLocationManager *)manager monitoringDidFailForRegion:(CLRegion *)region withError:(NSError *)error;`
- Works even if your application is not running!
  - In exactly the same way as "significant location change" monitoring.
  - The set of monitored regions persists across application termination/launch.
  - `@property (readonly) NSSet *monitoredRegions;` // property on CLLocationManager

# Core Location

- CLRegions are tracked by name
  Because they survive application termination/relaunch.
- Circular region monitoring size limit
  `@property (readonly) CLLocationDistance maximumRegionMonitoringDistance;`
  Attempting to monitor a region larger than this (radius in meters) will generate an error
  (which will be sent via the `delegate` method mentioned on previous slide).
  If this property returns a negative value, then region monitoring is not working.
- Beacon regions can also detect range from a beacon
  `- (void)startRangingBeaconsInRegion:(CLBeaconRegion *)beaconRegion;`
  Delegate method `locationManager:didRangeBeacons:inRegion:` gives you `CLBeacon` objects.
  `CLBeacon` objects will tell you proximity (e.g. `CLProximityImmediate`/`Near`/`Far`).
- To be a beacon is a bit more involved
  Beacons are identified by a globally unique UUID (that you generate).
  Check out `CBPeripheralManager` (Core Bluetooth Framework).