# App-Entwicklung für iOS und OS X

SS 2016

Stephan Gimbel

# Core Location

- Framework zum managen von “Location” und “Heading”
  Betrifft nicht das User-Interface.

- Objekt ist `CLLocation`
  **Properties**: `coordinate`, `altitude`, `horizontal/versionalAccuracy`, `timestamp`, `speed`, `course`.

- Wo (ungefähr) ist diese Location?

```
var coordinate: CLLocationCoordinate2D
struct {
    CLLocationDegrees latitude  // Double
    CLLocationDegrees longitude // Double
}

var altitude: CLLocationDistance // Meter
```

Negativer Wert bedeutet “unter Meeresspiegel”

# Core Location

- Wie genau ist die Location (latitude/longitude) zur tatsächlichen Position?

```
var horizontalAccuracy: CLLocationAccuracy // Meter
var verticalAccuracy: CLLocationAccuracy   // Meter
```

Negativer Wert bedeutet, dass die Koordinate (Coordinate) oder Höhe (Altitude) invalid ist.

Weitere Werte…

```
kCLLocationAccuracyBestForNavigation  // Stromversorgung?!?
kCLLocationAccuracyBest
kCLLocationAccuracyNearestTenMeters
kCLLocationAccuracyHundredMeters
kCLLocationAccuracyKilometer
kCLLocationAccuracyThreeKilometer
```

- Je genauer wir arbeiten wollen, desto höher ist der Stromverbrauch

Das Device “tut sein bestes” in Abhängigkeit von der Genauigkeit, die wir haben möchten.

Mobilfunkmast Triangulation (nicht sehr genau, aber geringer Energieverbrauch).

WiFi Accesspoint Datenbank (genauer, Stromverbrauch höher).

GPS (sehr genau, hoher Stromverbrauch).

# Core Location

- Speed

  `var speed: CLLocationSpeed // Meter/Sekunde`

  Momentane Geschwindigkeit (nicht Durchschnittsgeschwindigkeit).

  Generell nützlich als Hinweis/Information wenn wir in einem Fahrzeug sind.

  Negativer Wert bedeutet “invalide Geschwindigkeit”.

- Course

  `var course: CLLocationDirection // Grad, 0 ist Norden, CW (im Uhrzeigersinn)`

  Nicht alle Devices können diese Information liefern.

  Negativer Wert bedeutet “invalider Kurs”

- Time Stamp

  `var timestamp: NSDate`

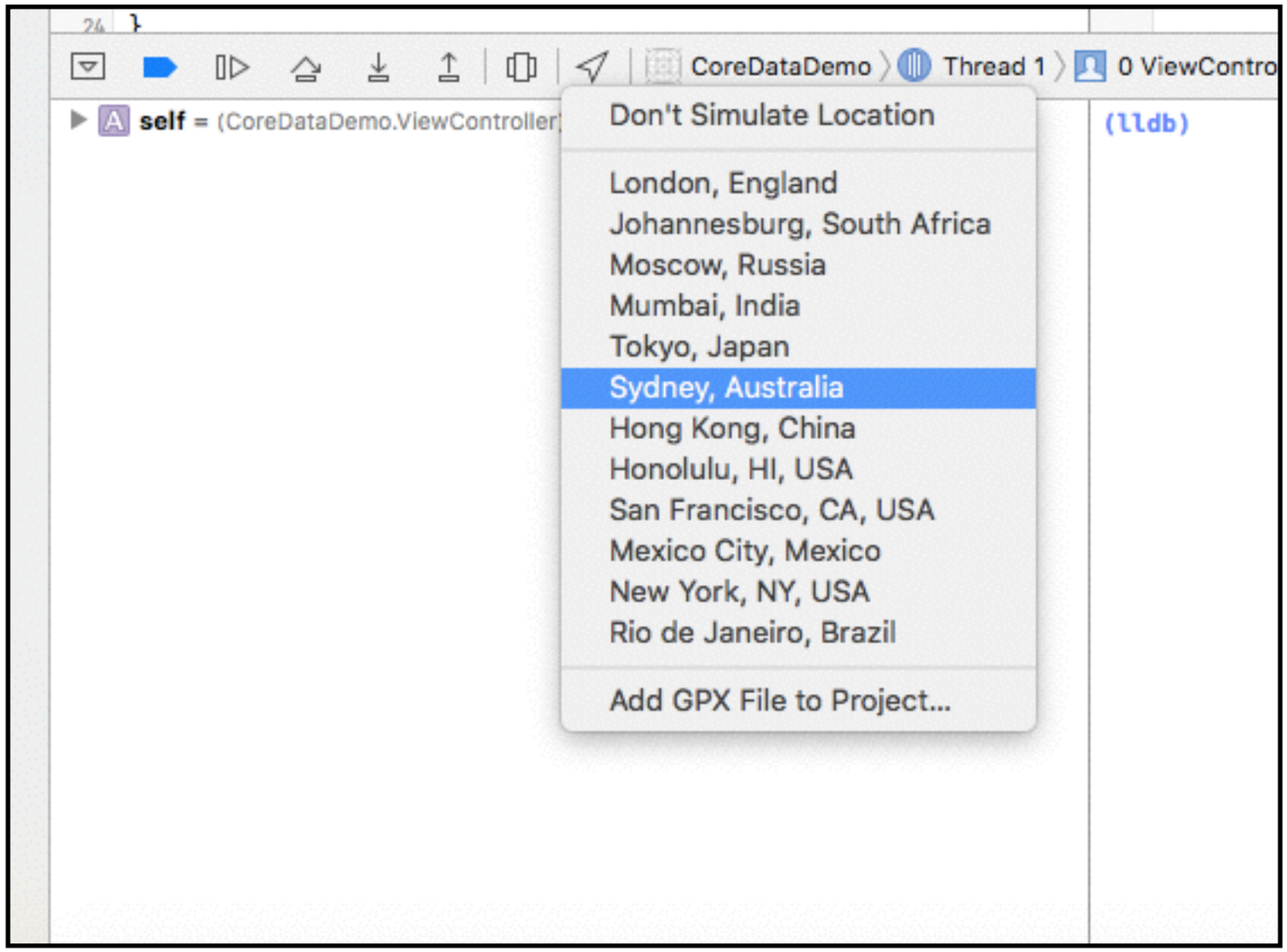
  Wichtig, sollten wir also beachten, da Locations in unregelmäßigen Abständen geliefert werden.

# Core Location

- Wie erhalten wie eine `CLLocation`?

  Meistens über einen `CLLocationManager` (geschickt über dessen `delegate`).

  Kann im Simulator in Xcode getestet werden.



# Core Location

- Wie erhalten wie eine `CLLocation`?

  Meistens über einen `CLLocationManager` (geschickt über dessen `delegate`)

  Kann im Simulator in Xcode getestet werden.

- `CLLocationManager`

  Generelle Vorgehensweise bei der Nutzung:

  1. Prüfen ob die Hardware auf der wir laufen und der User das Location Update welches wir nutzen wollen unterstützt.
  2. Erstellen einer `CLLocationManager` Instanz und setzen des `delegate` um Updates zu erhalten.
  3. Konfiguration des Managers bzgl. der Art der Location Updates die wir erhalten wollen.
  4. Start des Manager Monitorings für Location Änderungen.

- Arten von Location Monitoring

  Accuracy-based kontinuierliche Updates.

  Updates nur wenn signifikante Änderungen der Location auftreten (Significant Changes).

  Region-based Updates.

  Heading Monitoring.

# Core Location

- Anfrage an `CLLocationManager`, was die Hardware unterstützt

```
class func authorizationStatus() -> CLAuthorizationStatus  // Authorized, Denied or Restricted
class func locationServericesEnabled() -> Bool             // enabled/disabled for app
class func significantLocationChangeMonitoringAvailable() -> Bool
class func isMonitoringAvailableForClass(AnyClass!) -> Bool // CLBeacon/CLCircularRegion
class func isRangingAvailble() -> Bool // Distance from beacons
```

  Weitere Tests für andere Location Anfragemöglichkeiten.

- Erhalten von Location Informationen vom `CLLocationManager`

  Wir können den `CLLocationManager` einfach nach Location oder Heading fragen (polling), machen wir aber normalerweise nicht.

  Stattdessen lassen wir uns updaten wenn sich die Location (genug) ändern via dessen `delegate`...

# Core Location

- Authorization für Location Info

  Mittels...

```
func requestWhenInUseAuthorization()  // nur wenn aktiv
func requestAlwaysAuthorization()     // auch nicht nicht aktiv
```

  Dies erhält Authorization von User für uns (asynchron).

  Ob dies gewährt wurde, lässt sich über eine delegate Methode herausfinden.

  Bis dies geschehen ist, ist der `authorizationStatus` `NotDetermined`.

- Damit dies funktioniert muss ein Eintrag in `Info.plist` hinzugefügt werden...

```
NSLocationWhenInUseUsageDescription
NSLocationAlwaysUsageDescription
```

  Recht-Klick in `Info.plist` dann “Add Row”

- Accuracy-based kontinuierliches Location Monitoring

```
var desiredAccuracy: CLLocationAccuracy // immer so niedrig wie möglich
                                        // setzen
```

  Updates können auch limitiert werden bzgl. der Distanz einer Änderung der Location...

```
var distanceFilter: CLLocationDistance
```

# Core Location

- Starten und Stoppen des Normal Position (nicht significant) Monitoring
  ```
  func startUpdatingLocation()
  func stopUpdatingLocation()
  ```
  Sicherstellen, dass wir Updating abschalten, wenn die App keine Änderungen verarbeitet!

- Benachrichtigen lassen via `CLLocationManager`'s `delegate`
  ```
  func locationManager(CLLocationManager, didUpdateLocations: [CLLocation])
  ```

- Ähnliche API für Heading (`CLHeading`, et.al.)

# Core Location

- Fehlermeldungen an das `delegate`

```
func locationManager()(CLLocationManager, didFailWithError: NSError)
```

  Nicht immer fatal, also diese `delegate` Methode beachten.

  Beispiele…

```
kCLErrorLocationUnknown // meist temporär, eine Weile warten
kCLErrorDenied          // User verhindert dass die App Updates erhält
kCLErrorHeadingFailure  // zu viele lokale magnetische Störung, warten
```

- Background

  Es ist möglich diese Arten von Updates zu empfangen während die App im Hintergrund läuft.

  Es gibt Wege Update Reporting mit Verzögerung zu verbinden.

  Backgrounding muss dafür aktiviert sein (in den Project-Settings, ähnlich wie Background Fetch).

  Es gibt zwei Möglichkeiten (grobe) Location Notifications zu erhalten ohne dies zu tun…

# Core Location

- Significant Location Change Monitoring im `CLLocationManager`
  "Significant" ist nicht strikt definiert. Fahrzeuge, nicht laufen. Nutzt wahrscheinlich Mobilfunk-Mäste.
  ```
  func startMonitoringSignificantLocationChanges()
  func stopMonitoringSignificantLocationChanges()
  ```
  Sicherstellen dass wir Updating abschalten, wenn die App die Änderungen nicht verarbeitet.

- Benachrichtigung via `CLLocationManager`'s `delegate`
  Identisch zu Accuracy-based Updating, wenn die App läuft.

- Dies funktioniert auch, wenn die App gar nicht läuft!
  (Oder im Background ist)
  Wird gestartet und das App Delegate's
  ```
  func application(UIApplication, didFinishLaunchingWithOptions: [NSObject,AnyObject])
  ```
  hat einen Dictionary Eintrag für `UIApplicationLaunchOptionsLocationKey`.
  Erstellen eines `CLLocationManagers` (sofern wir keinen haben), danach die letzte Location erhalten via...
  ```
  var location: CLLocation
  ```
  Wenn wir im Hintergrund laufen, nicht zu lange Zeit nehmen dafür (nur ein paar Sekunden)!

# Core Location

- Region-based Location Monitoring im `CLLocationManager`

```
func startMonitoringForRegion(CLRegion) // CLCircularRegion/CLBeaconRegion
func stopMonitoringForRegion(CLRegion)
let cr = CLCircularRegion(center: CLLocationCoordinate2D,
                           radius: CLLocationDistance,
                       identifier: String)
```

  … um einen Bereich zu überwachen.

  `CLBeaconRegion` dient zur Erkennung ob wir in der Nähe eines anderen Devices sind.

- Dann werden wir ebenfalls benachrichtigt via `CLLocationManager`'s `delegate`

```
func locationManager(CLLocationManager, didEnterRegion: CLRegion)
func locationManager(CLLocationManager, didExitRegion: CLRegion)
func locationManager(CLLocationManager, monitoringDidFailForRegion: CLRegion,
                                                          withError: NSError)
```

# Core Location

- Region-Monitoring funktioniert auch, wenn die App nicht läuft

  Auf genau die gleiche Art und Weise wie "significant location change" Monitoring.

  Die Regions die beobachtet werden sind persistent über App Termination/Launch.

  ```
  var monitoredRegions: NSSet // von Strings (Property in CLLocationManager)
  ```

- `CLRegions` werden identifiziert durch Name

  Da sie App Termininationn/Relaunch überleben.

- Circular Region Size Limit

  ```
  var maximumRegionMonitoringDistance: CLLocationStance { get }
  ```

  Der Versuch eine Region größer also dies (Radius in Meter) zu überwachen, generiert einen Error (welcher via der delegate Methode auf dem letzen Slide geschickt wird).

  Wenn das Property einen negativen wert zurück gibt, dann funktioniert Region Monitoring nicht.

# Core Location

- Beacon Regions können auch die Entfernung von einem Beacon detektieren
  `func startRangingBeaconsInRegion(CLBeaconRegion)`
  Delegate Methode `locationManager(didRangeBeacons:inRegion:)` gibt uns `CLBeacon` Objekte.
  `CLBeacon` Objekte geben uns die Nähe (z.B. `CLProximityImmediate/Near/Far`).

- Selbst als Beacon zu fungieren ist etwas aufwendiger
  Beacons werden über eine global eindeutige UUID (die wir generieren) identifiziert.
  Siehe `CBPeripheralManager` (Core Bluetooth Framework).

# Map Kit

- `MKMapView` zeigt eine Karte (Map)

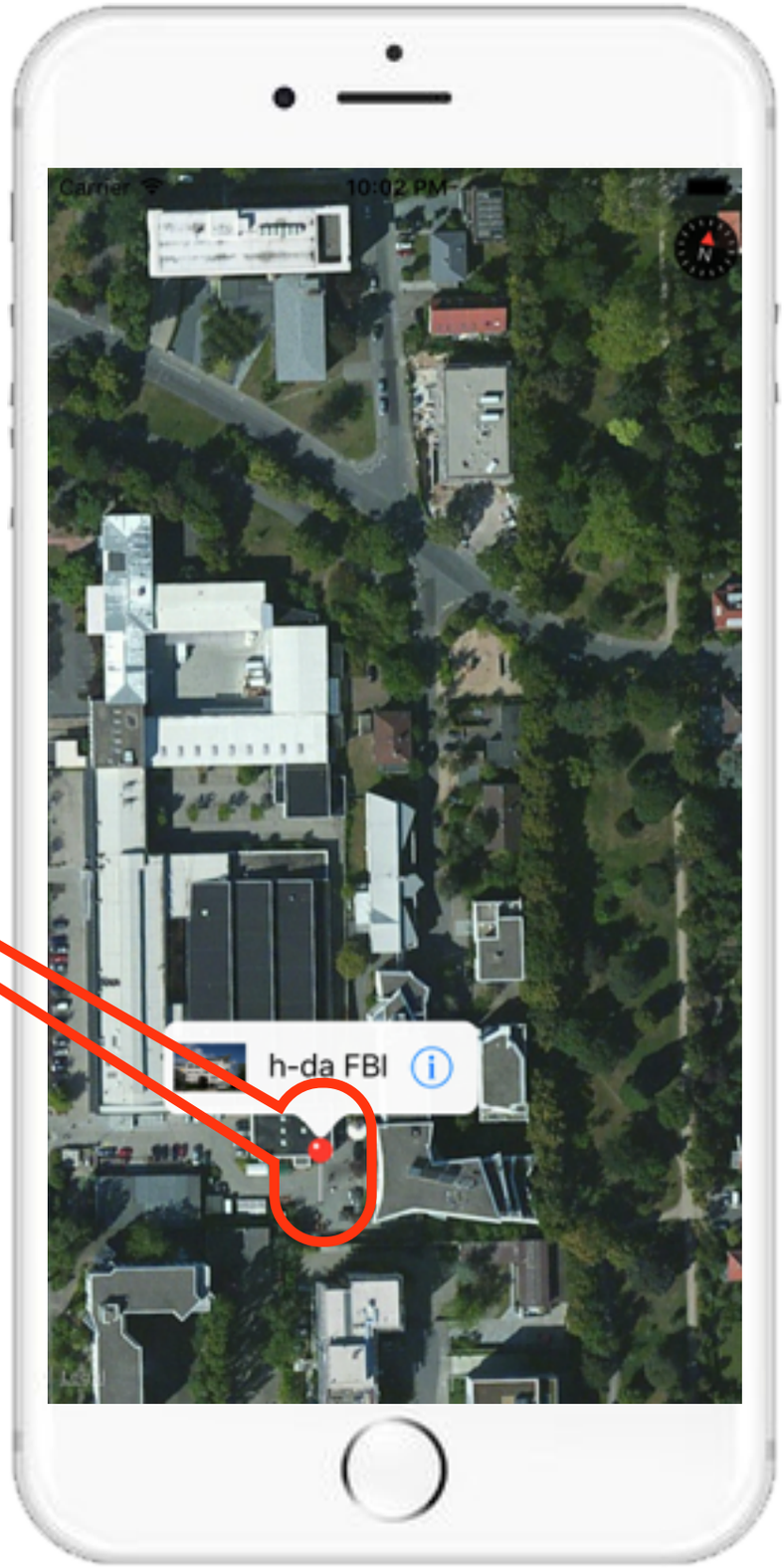
# Map Kit

- `MKMapView` zeigt eine Karte (Map)

- Eine Karte kann Annotations haben

  Jede Annotation ist eine `coordinate`, ein `title` und `subtitle`.

  Sie werden mittels eines `MKAnnotationViews` dargestellt (hier ein `MKPinAnnotationView`).



# Map Kit

- `MKMapView` zeigt eine Karte (Map)

- Eine Karte kann Annotations haben

  Jede Annotation ist eine `coordinate`, ein `title` und `subtitle`.

  Sie werden mittels eines `MKAnnotationViews` dargestellt (hier ein `MKPinAnnotationView`).

- Annotations können einen `Callout` haben

  Taucht auf, wenn der Annotation View angeklickt wird.

  Per Default wird nur der `title` und `subtitle` gezeigt.

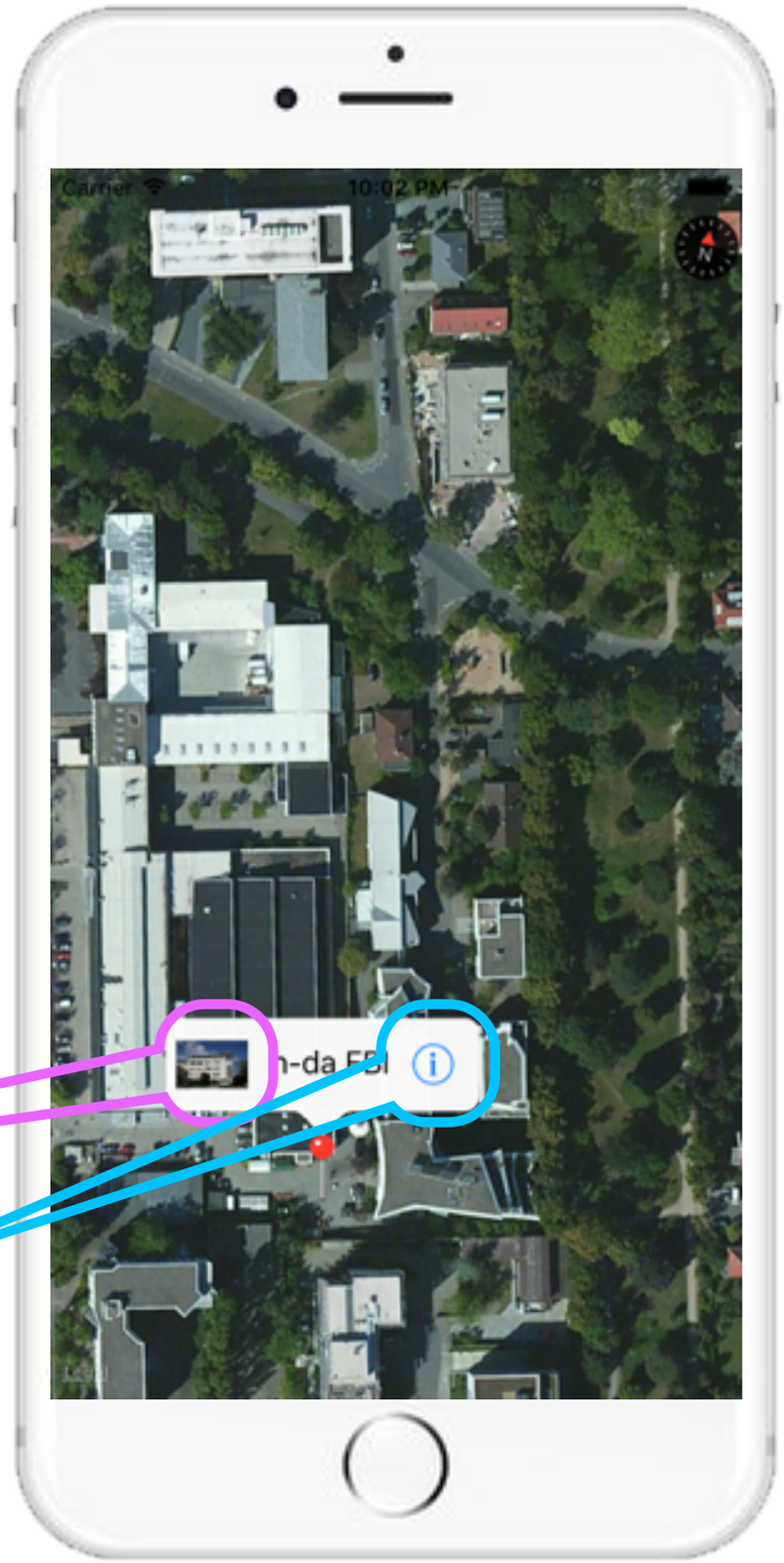
# Map Kit

- `MKMapView` zeigt eine Karte (Map)

- Eine Karte kann Annotations haben
  Jede Annotation ist eine `coordinate`, ein `title` und `subtitle`.
  Sie werden mittels eines `MKAnnotationViews` dargestellt (hier ein `MKPinAnnotationView`).

- Annotations können einen `Callout` haben
  Taucht auf, wenn der Annotation View angeklickt wird.
  Per Default wird nur der `title` und `subtitle` gezeigt.

- Ein Callout kann auch Accessory Views haben
  Im Beispiel ist links ein UIImageView,
  der rechte ist ein UIButton (UIButtonTypeDetailDisclosure)



# MKMapView

- Erstellen mit einem Initializer oder Drag aus der Object Library in Xcode

```
import MapKit
let mapView = MKMapView()
```

# MKMapView

- Zeigt ein Array von Objekten welche `MKAnnotation` implementieren

```
var annotations: [MKAnnotation] { get }
```

- `MKAnnotation` Protocol

```
protocol MKAnnotation: NSObject {
    var coordinate: CLLocationCoordinate2D { get }
    var title: String! { get }
    var subtitle: String! { get }
}
```

Zur Erinnerung an `CoreLocation`...

```
struct CLLocationCoordinate2D {
    var latitude: CLLocationDegrees
    var longitude: CLLocationDegrees
}
```

# MKAnnotation

- Das Annotation Property ist readonly, also…

```
var annotations: [MKAnnotation] { get }
```

… müssen Annotations mittels dieser Methoden hinzugefügt/entfernt werden…

```
func addAnnotation(MKAnnotation)
func addAnnotations([MKAnnotation])
func removeAnnotation(MKAnnotation)
func removeAnnotations([MKAnnotation])
```

- Generell ist es eine gute Idee die Annotations vorher hinzuzufügen

  Erlaubt hohe Effizienz von `MKMapView` bzgl. Darstellung.

  Annotations sind “light-weight”, aber Annotation Views sind es nicht.

  `MKMapView` wiederverwendet Annotation Views, ähnlich wir ein `UITableView` Cells wiederverwendet.

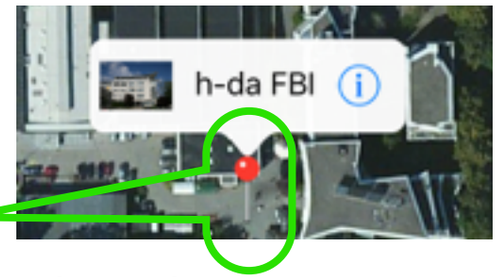
# MKAnnotation

- Wie sehen Annotations aus der Map aus?

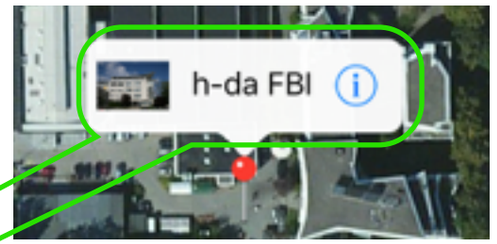
  Annotations werden gezeichnet mit einer MKAnnotationView Subclass.

  Der Default ist `MKPinAnnotationView` (deswegen sehen sie wie Pins aus).

  Können wir subclassen oder wir setzen Properties von bestehenden `MKAnnotationViews` um den Look zu modifizieren.



# MKAnnotation




- Wie sehen Annotations aus der Map aus?

  Annotations werden gezeichnet mit einer MKAnnotationView Subclass.

  Der Default ist `MKPinAnnotationView` (deswegen sehen sie wie Pins aus).

  Können wir subclassen oder wir setzen Properties von bestehenden `MKAnnotationViews` um den Look zu modifizieren.

- Was passiert, wenn wir eine Annotation anklicken?

  Hängt vom MKAnnotationView ab, der mit der Annotation assoziiert ist.

  Wenn `canShowCallout` `true` ist im AnnotationView, dann taucht eine kleine Box auf die `title` und `subtitle` der Annotation zeigt.

  Diese kleine Box (der Callout) dann durch `left`/`rightCalloutAccessoryViews` erweitert werden.

  Diese `MKMapViewDelegate` Methode wird ebenfalls aufgerufen...

```
func mapView(MKMapView, didSelectAnnotationView: MKAnnotationView)
```

  Dies ist ein idealer Ort für ein Lazy-Setup von Callout Accessory Views des `MKAnnotationViews`.
  Wir wollen wahrscheinlich bis zum Ausführen dieser Methode warten um ein Bild zu laden welches angezeigt wird.

# MKAnnotationView

- Wie werden `MKAnnotationViews` erstellt und mit Annotations assoziiert?

  Sehr ähnlich zu `UITableViewCells` in einem `UITableView`.

  Implementieren der folgenden `MKMapViewDelegate` Methode (wenn diese nicht implementiert ist, wird ein Pin View zurück gegeben).

```
func mapView(sender: MKMapView, viewForAnnotation: MKAnnotation) -> MKAnnotationView
{
    var view = sender.dequeReusableAnnotationViewWithIdentifier(someID)
    if !view {
        view = MKPinAnnotationView(annotation: annotation, reuseIdentifier: someID)
        view.canShowCallout = true or false
    }
    view.annotation = annotation // ja, passiert zwei mal wenn kein dequeue
    // Vorbereiten und befüllen (wenn nicht zu teuer) des Accessory View
    // oder resetten und warten bis mapView(didSelectAnnotationView:) um die Daten
    //zu laden
    return view
}
```

Es existiert kein Prototyp im Storyboard, wir müssen wie hier im Code erzeugen.

# MKAnnotationView

- Ein paar interessante Properties...

```
var annotation: MKAnnotation // die Annotation; behandeln wie readonly
var image: UIImage // statt dem Pin (kein Bild im Callout)
var leftCalloutAccessoryView: UIView // vielleicht ein UIImageView
var rightCalloutAccessoryView: UIView // vielleicht ein “disclosure” Button
var enabled: Bool // false heißt Touches ignorieren, keine Delegate Methode,
                  // ein Callout
var centerOffset: CGPoint // wo der “Kopf des Pins” ist, relativ zum Bild
var draggable: Bool // funktioniert nur, wenn Annotation’s coordinate
                    // property { get set } ist
```

- Wenn einer der Callout Accessory Views als `UIControl` gesetzt wird...

  z.B. `view.rightCalloutAccessoryView = UIButton.buttonWithType(UIButtonTypeDetailDisclosure)`

  Dann wird die folgende MKMapViewDelegate Methode aufgerufen, wenn der Callout View geklickt wird...

```
func mapView(MKMapView, annotationView: MKAnnotationView,
         calloutAccessoryControlTapped: UIControl)
```

  Vielleicht führen wir von dort z.B. ein Segue aus.

# MKAnnotationView

- Verwenden von `didSelectAnnotatiobView:` um Callout Accessories zu füllen/ laden

  Beispiel… Download eines Bildes in `leftCalloutAccessoryView` der ein `UIImageView` ist.

  In `mapView(viewForAnnotation:)`, `let view.leftCalloutAccessoryView = UIImageView()`

  Reset des `UIImageView`'s `image` zu `nil` (weil Wiederverwendung).

  Dann Laden des `image` on Demand in mapView(didSelectAnnotationView:)…

```
func mapView(MKMapView, didSelectAnnotationView aView: MKAnnotationView)
{
    if let imageView = aView.leftCalloutAccessoryView as? UIImageView {
        imageView.image = ... // Vorsicht bei Multithreading! Views werden
                              // wiederverwendet!
    }
}
```

# MKMapView

- Konfiguration des Map View's Display Type

```
var mapType: MKMapType // .Standard, .Satellite, .Hybrid
```

- Anzeigen der aktuellen Location des Users

```
var showsUserLocation: Bool
var isUserLocationVisible: Bool
var userLocation: MKUserLocation
```

`MKUserLocation` ist ein Objekt konform zu `MKAnnotation` welches die Location des Users beinhaltet.

- Einschränken der User Interaktion mit der Map

```
var zoomEnabled: Bool
var scrollEnabled: Bool
var pitchEnabled: Bool // 3D
var rotateEnabled: Bool
```

# MKMapCamera

- Setzen von wo der User die Map sieht (3D)

```
var camera: MKMapCamera // Property in MKMapView
```

- MKMapCamera

  Spezifizieren von `centerCoordinate`, `heading`, `pitch` und `altitude` der Kamera.

  Oder verwenden des convenient Initializers von `MKMapCamera`...

```
let camera = MKMapCamera(lookingAtCenterCoordinate: CLLocationCoordinate2D,
                                     fromEyeCoordinate: CLLocationCoordinate2D,
                                           eyeAltitude: CLLocationDistance)
```

# MKMapView

- Kontrolle der Region (Teil der Welt) die auf der Karte dargestellt wird

```
var region: MKCoordinateRegion

struct MKCoordinateRegion {
    var center: CLLocationCoordinate2D // lat/long
    car span: MKCoordinateSpan
}

struct MKCoordinateSpan {
    var latitudeDelta: CLLocationDegrees
    var longitudeDelta: CLLocationDegrees
}

func setRegion(MKCoordinateRegion: animated: Bool)
```

- Wir können auch nur den Center Point setzen oder die Annotations zeigen

```
var centerCoordinate: CLLocationCoordinate2D
func setCenterCoordinate(CLLocationCoordinate2D, animated: Bool)
func showAnnotations([MKAnnotation], animated: Bool)
```

# MKMapView

- Viele C Funktionen zum konvertieren von Points, Regions, Rects, etc.
  Siehe Doku, z.B. `MKMapRectContainsPoint`, `MKMapPointForCoordinate`, etc.

- Konvertieren zu/von Map Points/Rects von/zu View Koordinaten

```
func mapPointForPoint(CGPoint) -> MKMapPoint
func mapRectForRect(CGRect) -> MKMapRect
func pointForMapPoint(MKMapPoint) -> CGPoint
func rectForMapRect(MKMapRect) -> CGRect
```

  usw.

- Eine weitere `MKMapViewDelegate` Methode...

```
func mapView(MKMapView, didChangeRegionAnimated: Bool)
```

  Dies ist ein guter Ort um Animationen zu verknüpfen (Animation Chaining).

  Wenn wir einen neuen Bereich auf der Karte anzeigen der weit weg ist, heraus zoomen, dann wieder herein zoomen.

  Die Methode benachrichtigt wenn das heraus zoomen beendet ist, so dass wir hinein zoomen können.

# MKLocalSearch

- Suche nach Orten auf der Welt

  Wir können mittels “natürlicher Sprache” nach Strings asynchron suchen (verwendet Netzwerkverbindung)...

```
var request = MKLocalSearchRequest()
request.naturalLanguageQuery = @“Starbucks”
request.region = ...
let search = MKLocalSearch(request: request)
search.startWithCompletionHandler { (MKLocalSearchResponse, NSError) -> Void in
    // Antwort ist ein Array von MKMapItems die ein MKPlacemark beinhalten
    // MKPlacemark beinhaltet die Location, Name der Location, PLZ, Region, etc.
}
```

- MKMapItem

  Wir können ein `MKMapItem` welches wir aus einer `MKLocalSearch` erhalten in der Maps App öffnen...

```
func openInMapsWithLaunchOptions([NSObject:AnyObject]) -> Bool
```

  Die Options können eine spezifische Region, Show Traffic, etc. beinhalten.

# MKDirections

- Erhalten von Directions von einem Ort zum anderen
  Sehr ähnliche API zum suchen.

  Spezifikation von Source und Destination `MKMapItem`.

  Asynchrone API um verschiedene Routen zu erhalten.

  `MKRoute` beinhaltet einen Namen für die Route, Turn-by-Turn Directions, erwartete Reisezeit, etc.

  Hat auch `MKPolyline` Descriptions der Routen welche als Overlay auf der Map angezeigt werden können…

# MKOverlays

- Overlays

  Hinzufügen von Overlays zu einem MKMapView fragt uns später nach einem Renderer zum zeichnen des Overlays.

  ```
  func addOverlay(MKOverlay, level: MKOverlayLevel)
  ```

  Level ist (aktuell) entweder `AboveRoads` oder `AboveLabels` (über alles außer Annotation Views).

  ```
  func removeOverlay(MKOverlay)
  ```

- `MKOverlay` Protocol

  Protocol welches `MKAnnotation` beinhaltet plus...

  ```
  var boundingMapRect: MKMapRect
  func intersectsMapRect(MKMapRect) -> Bool // optional, verwendet ansonsten
                                            // boundingMapRect
  ```

- Overlays sind assoziiert mit `MKOverlayRederers` via `delegate`

  Genau wie Annotations mit `MKAnnotationViews` assoziiert sind...

  ```
  func mapView(MKMapView, rendererForOverlay: MKOverlay) -> MKOverlayRenderer
  ```

# MKOverlayView

- Eingebaute Overlays und Renderer für verschiedene Formen…

```
MKCircleRenderer
MKPolylineRenderer
MKPolygonRenderer
MKTileOverlayRenderer // kann auch verwendet werden um die Map Daten von
                      // Apple zu ersetzen
```

  Es existieren eine ganze Menge von `MKShapes` und Subclasses zur Erkundung.