

# CoreLocation

**Giftbot**

# Core Location Framework

기기의 지리적 위치와 방향을 얻기 위한 프레임워크

- 기기의 지리적 위치, 고도, 방향 또는 iBeacon 주변의 상대적 위치를 결정하는 서비스를 제공
- 다음과 같은 모든 사용 가능한 온보드 하드웨어를 이용해 데이터를 수집  
**Wi-Fi, GPS, Bluetooth, Magnetometer (자력계), Barometer (기압계), Cellular Hardware 등**
- CLLocationManager 를 이용해 대부분의 서비스를 시작하고 연결된 Delegate 를 통해 응답을 수신
- 위치 정보를 얻기 위해 반드시 유저로부터 권한을 얻어야 함
- 크게 위치 업데이트 / 지역 모니터링 / iBeacon / 장치 방향 / 좌표 변환 등의 역할 수행

# CoreLocation Framework

---

```
import CoreLocation.CLAvailability
import CoreLocation.CLBeaconRegion
import CoreLocation.CLCircularRegion
import CoreLocation.CLError
import CoreLocation.CLErrorDomain
import CoreLocation.CLGeocoder
import CoreLocation.CLHeading
import CoreLocation.CLLocation
import CoreLocation.CLLocationManager
import CoreLocation.CLLocationManagerDelegate
import CoreLocation.CLLocationManager_CLVisitExtensions
import CoreLocation.CLPlacemark
import CoreLocation.CLRegion
import CoreLocation.CLVisit
import CoreLocation
import Foundation
```

# Request Authorization

## Requesting Authorization for Location Services

```
func requestWhenInUseAuthorization()
```

Requests permission to use location services while the app is in the foreground.

```
func requestAlwaysAuthorization()
```

Requests permission to use location services whenever the app is running.

```
enum CLAuthorizationStatus
```

Constants indicating whether the app is authorized to use location services.

# Request Authorization

## When-in-use authorization

- : 앱이 **Foreground**에서 동작 중일 때만 위치 서비스 사용
- : 앱을 자동으로 재실행하는 서비스는 사용 불가
- : 굳이 **Always**를 써야 하는 경우가 아니면 이 방식을 권장. 동시에 한 가지 방식으로만 설정 가능

## Always authorization

- : **Foreground** 나 **Background** 모두에서 필요할 때 위치 서비스 사용
- : 앱이 실행 중이지 않을 때 위치 기반 이벤트가 발생하면 시스템이 앱을 실행하고 이벤트를 전달

```
let locationManager = CLLocationManager()  
locationManager.requestWhenInUseAuthorization()  
locationManager.requestAlwaysAuthorization()
```

# Info.plist - Usage Description

Always authorization 이용 시

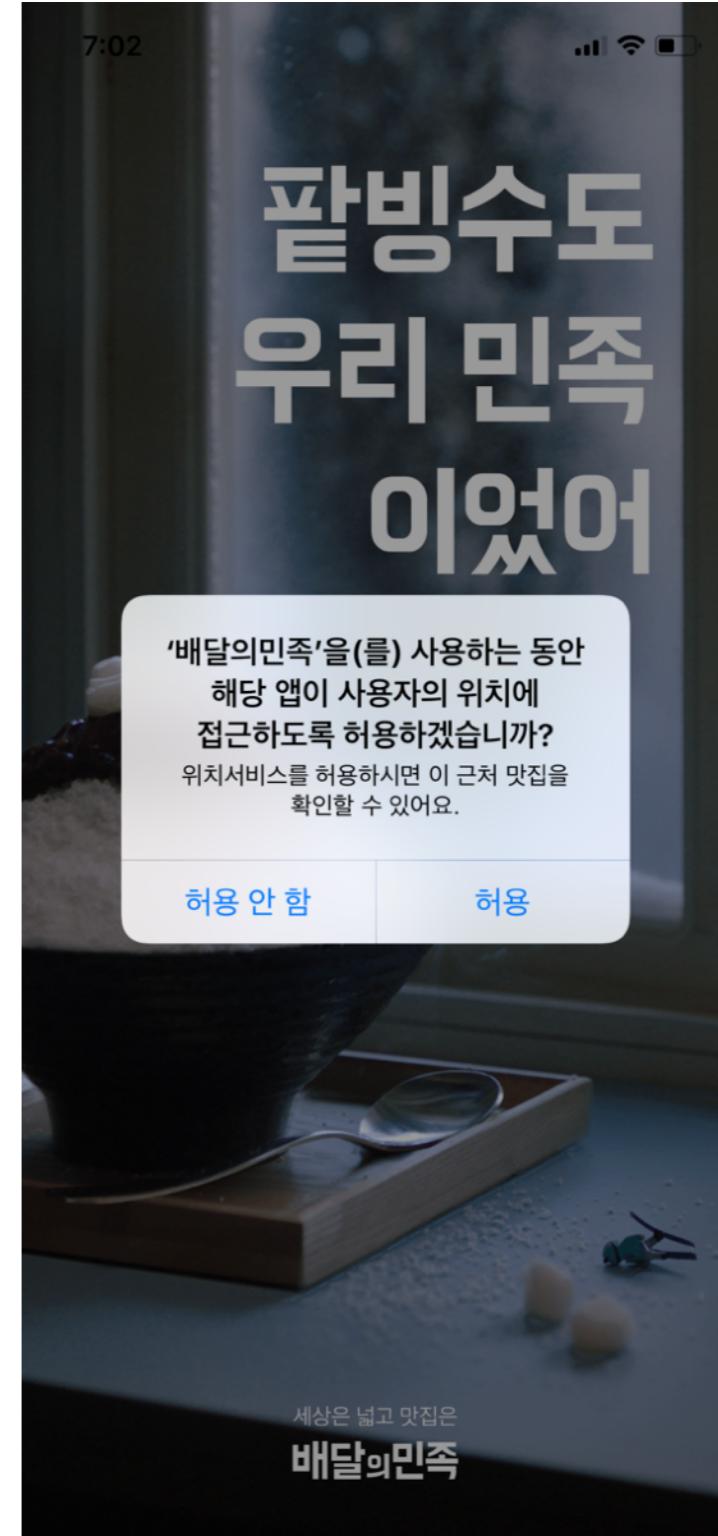
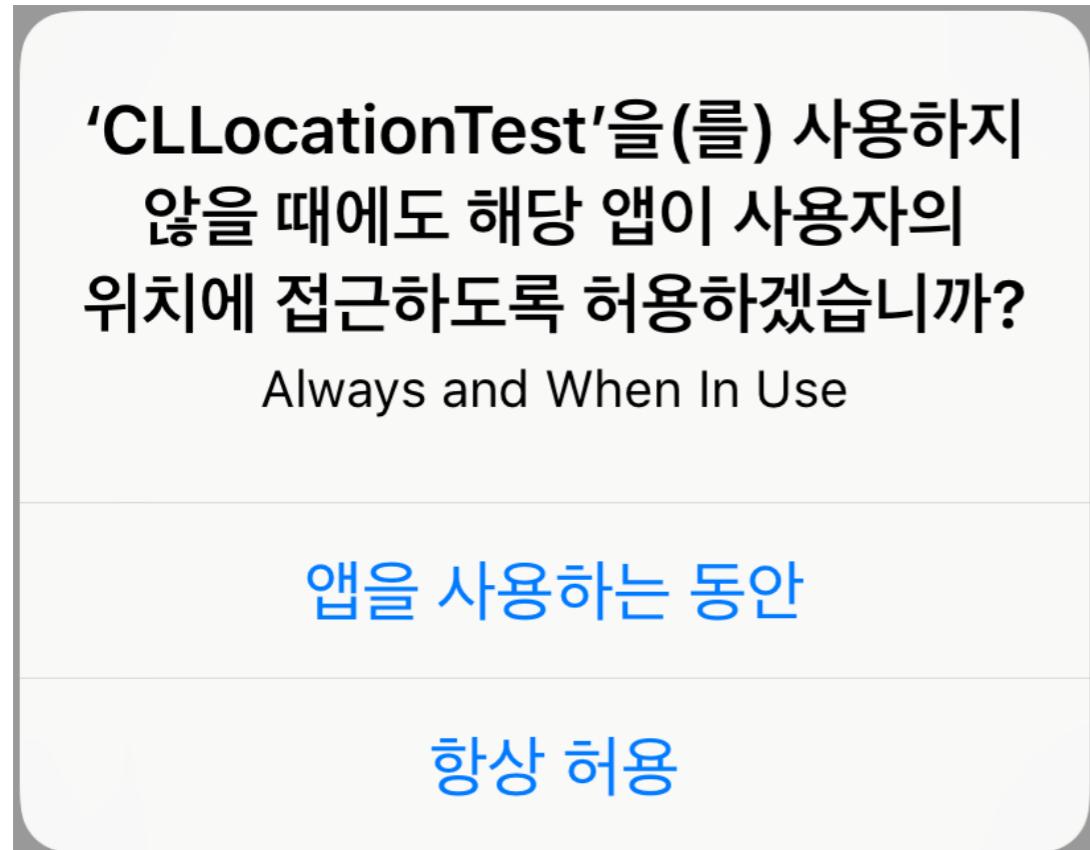
iOS 11 이상 - NSLocationAlwaysAndWhenInUseUsageDescription 키 등록

iOS 10 이하 - NSLocationAlwaysUsageDescription 키 등록

When-in-use authorization - NSLocationWhenInUseUsageDescription 키 등록

Key	Type	Value
▼ Information Property List	Dictionary	(17 items)
Privacy - Location Always and When In Use Usage Description	String	Always and When In Use
Privacy - Location Always Usage Description	String	Always
Privacy - Location When In Use Usage Description	String	When In Use

# Request Authorization



Service	When-in-use	Always
Standard location service	Supported	Supported
Significant-change location service	Not available	Supported
Visits service	Not available	Supported
Region monitoring	Not available	Supported
iBeacon ranging	Supported	Supported
Heading service	Supported	Supported
Geocoding services	Supported	Supported

---

Service	Launches app
---------	--------------

---

Standard location service	No
---------------------------	----

---

Significant-change location service	Yes
-------------------------------------	-----

---

Visits service	Yes
----------------	-----

---

Region monitoring	Yes
-------------------	-----

---

iBeacon ranging	No
-----------------	----

---

Heading service	No
-----------------	----

---

# CLAuthorizationStatus

---

```
switch CLLocationManager.authorizationStatus() {  
    case .notDetermined:  
        locationManager.requestWhenInUseAuthorization()  
    case .restricted, .denied:  
        // Disable location features  
        disableMyLocationBasedFeatures()  
    case .authorizedWhenInUse:  
        // Enable basic location features  
        enableMyWhenInUseFeatures()  
    case .authorizedAlways:  
        // Enable any of your app's location features  
        enableMyAlwaysFeatures()  
}
```

# Delegate - didChangeAuthorization

---

```
func locationManager(  
    _ manager: CLLocationManager,  
    didChangeAuthorization status: CLAuthorizationStatus  
) {  
  
    switch status {  
    case .authorizedWhenInUse, .authorizedAlways:  
        print("Authorized")  
    default:  
        print("Unauthorized")  
    }  
}
```

# Determining the Availability

## 위치 서비스를 사용할 수 없는 상황

- 기기에 이 기능을 지원하는 데 필요한 하드웨어가 없음
- 사용자가 시스템 설정에서 위치 서비스 기능을 끔
- 사용자가 앱의 위치 서비스에 대한 접근을 거부
- 기기가 비행기 모드로 설정되었을 때
- 백그라운드 갱신 기능을 사용할 수 없고 필요한 기능을 우선 순위가 높은 다른 서비스가 이미 사용 중일 때

## 가용성 체크

```
func locationServicesEnabled() -> Bool
```

```
func headingAvailable() -> Bool
```

```
func significantLocationChangeMonitoringAvailable() -> Bool
```

```
func isMonitoringAvailable(for regionClass: Swift.AnyClass) -> Bool
```

```
func isRangingAvailable() -> Bool
```

# Getting the User's Location Data

## Standard location service (When-in-use or Always authorization)

- 사용자 위치를 실시간으로 파악하기 위한 범용 솔루션
- 다른 서비스에 비해 더 많은 전력을 쓰지만 가장 정확하고 즉각적인 정보를 제공

## Significant-change location service (Always authorization)

- 전력 소모를 줄이기 위한 것으로 업데이트가 자주 필요하지 않고 GPS 정밀도가 낮아도 되는 경우 사용
- 사용자 위치를 대폭 변경한 경우에만 업데이트를 제공
- 사용자가 걷고 있을 때 주변의 관심 장소(POI)에 대한 추천 정보를 제안해주는 등의 서비스를 제공 가능

## Visits location service (Always authorization)

- 가장 효율적으로 전력을 사용하지만 다른 서비스에 비해 업데이트 횟수가 적은 방법
- 유저가 한 장소에 머물러 시간을 보내다가 이동할 때 업데이트 알림 발생 (위치 및 시간 정보)
- 사용자의 행동 패턴을 파악하고 그 지식을 앱의 다른 부분에 적용하기 위한 서비스로 활용

# Improve Battery Life

---

- 필요한 기능에 맞는 적절한 위치 서비스 또는 정밀도 사용
- 모니터링을 시작(start) 한 뒤 필요 없을 때는 즉시 중단(stop)
- `pausesLocationUpdatesAutomatically` (기본값 true) 활성화 시, 사용자가 움직이지 않을 때 Location 하드웨어를 비활성화해 전원 절약. 업데이트 품질에는 거의 영향 없음
- `allowDeferredLocationUpdates(untilTraveled:timeout)`로 조건 충족시까지 업데이트 지연

# Standard Location Service

지정한 설정 값에 따라 정기적인 위치 업데이트를 수행

다른 위치 서비스와 달리 When-In-Use 권한에서도 사용 가능

위치 정보를 세밀하게 파악해야 하고 자주 업데이트 해야 하는 앱에서 사용

높은 정밀도를 필요로 하지 않을 경우 시스템이 알아서 GPS 대신 Wi-Fi 를 이용하는 등의 작업 수행

```
func startUpdatingLocation() {  
    let status = CLLocationManager.authorizationStatus()  
    guard status == .authorizedAlways || status == .authorizedWhenInUse,  
        CLLocationManager.locationServicesEnabled()  
    else { return }  
    locationManager.desiredAccuracy = kCLLocationAccuracyNearestTenMeters  
    locationManager.distanceFilter = 10.0  
    locationManager.startUpdatingLocation()  
}
```

# desiredAccuracy

위치 정보의 정밀도 값으로, 반드시 보장되지는 않지만 가능한 한 설정에 가까운 정보를 취합  
정밀도가 높을수록 더 많은 전력 소비와 연산 시간이 필요하므로 높은 것이 꼭 좋은 것은 아님  
높은 정밀도의 위치 정보 요청 시, 우선 가능한 빨리 초기 정보를 받은 뒤 더 정확한 위치 정보를 계속 파악해  
더 정확한 정보를 갱신하는 형태로 전달받게 됨

기본값 - AccuracyBest (iOS, macOS) / AccuracyHundredMeters (watchOS)

```
locationManager.desiredAccuracy = kCLLocationAccuracyBest
```

```
typealias CLLocationAccuracy = Double
let kCLLocationAccuracyBestForNavigation: CLLocationAccuracy
let kCLLocationAccuracyBest: CLLocationAccuracy
let kCLLocationAccuracyNearestTenMeters: CLLocationAccuracy
let kCLLocationAccuracyHundredMeters: CLLocationAccuracy
let kCLLocationAccuracyKilometer: CLLocationAccuracy
let kCLLocationAccuracyThreeKilometers: CLLocationAccuracy
```

# distanceFilter

위치 업데이트 이벤트를 발생시키기 위해 필요한 기기의 최소 이동 거리 ( Meter 단위 ) 설정

기본값 - kCLDistanceFilterNone (모든 이동 감지시마다 업데이트)

Standard Location Service 를 사용할 때는 desiredAccuracy 와 distanceFilter 를 설정 필요

```
locationManager.distanceFilter = kCLDistanceFilterNone
```

```
typealias CLLocationDistance = Double
```

```
let kCLDistanceFilterNone: CLLocationDistance
```

```
let CLLocationDistanceMax: CLLocationDistance
```

# Significant-change Location Service

사용자 위치 파악을 위해 저전력 정보 사용 (Wi-Fi / cellular 정보 등)

사용자 위치가 500 meters 이상과 같이 크게 변경됐을 때만 위치 업데이트를 앱에 전달

distanceFilter , desiredAccuracy 속성은 무시하므로 설정 불필요

```
func monitorSignificantLocationChange() {  
    guard CLLocationManager.authorizationStatus() == .authorizedAlways,  
        CLLocationManager.significantLocationChangeMonitoringAvailable()  
    else { return }  
  
    locationManager.startMonitoringSignificantLocationChanges()  
}
```

# Visits Location Service

위치 데이터를 수집하는 가장 전력 효율적인 방법

실시간 활동이나 Navigation 을 위한 용도보다는 사용자의 행동 패턴을 식별하기 위함

일반적인 운동 시간에 체육관에 도착하면 자동으로 음악을 실행하는 작업 등을 수행

distanceFilter , desiredAccuracy 속성은 설정 불필요

```
func monitorVisitsLocation() {  
    guard CLLocationManager.authorizationStatus() == .authorizedAlways,  
        CLLocationManager.locationServicesEnabled()  
    else { return }  
    locationManager.startMonitoringVisits()  
}
```

# Location Update

---

```
func locationManager(  
    _ manager: CLLocationManager,  
    didUpdateLocations locations: [CLLocation]  
) {  
  
    let current = locations.last! // 항상 하나 이상의 위치 포함  
  
    // 캐시된 위치 정보가 오래된 경우 업데이트 미처리  
    if (abs(current.timestamp.timeIntervalSinceNow) < 10) {  
        // 위치 정보 업데이트  
    }  
}
```

# Handling Location-Related Errors

---

```
func locationManager(  
    _ manager: CLLocationManager,  
    didFailWithError error: Error  
) {  
  
    if let error = error as? CLError, error.code == .denied {  
        // Location updates are not authorized.  
        // Stop Requesting Location Service  
        return  
    }  
  
    // Notify the user of any errors.  
}
```

# Represent Location Data

class [CLLocation](#)

The latitude, longitude, and course information reported by the system.

struct [CLLocationCoordinate2D](#)

The latitude and longitude associated with a location, specified using the WGS 84 reference frame.

class [CLFloor](#)

The floor of a building on which the user's device is located.

class [CLVisit](#)

Information about the user's location during a specific period of time.

# CLLocation

시스템으로부터 전달되는 위도, 경도, 고도 및 코스 정보 등을 담고 있는 객체

그대로 사용하도록 설계되었으므로 서브클래싱 하지 않아야 하며, 일반적으로 직접 생성하지 않음

커스텀 위치 정보를 캐시하고 싶거나 두 지점 사이의 거리를 구하는 경우 등에는 직접 생성할 수도 있음

```
func locationManager(  
    _ manager: CLLocationManager,  
    didUpdateLocations locations: [CLLocation]  
) {  
    let lastLocation = locations.last!  
    let location = CLLocation(latitude: latitude, longitude: longitude)  
    let distance = lastLocation.distance(from: location)  
    print(distance)  
}
```

# CLLocation

---

## [ 위치 ]

```
var coordinate: CLLocationCoordinate2D { get }
var altitude: CLLocationDistance { get }
var horizontalAccuracy: CLLocationAccuracy { get }
var verticalAccuracy: CLLocationAccuracy { get }
var floor: CLFloor? { get }
var timestamp: Date { get }
```

## [ 좌표 간 거리 ]

```
func distance(from location: CLLocation) -> CLLocationDistance
```

## [ 속도 및 코스 정보 ]

```
var course: CLLocationDirection { get }
var speed: CLLocationSpeed { get }
```

# CLLocationCoordinate2D

WGS 84 참조 프레임을 사용하여 지정된 위도, 경도에 대한 위치 정보

※ WGS : World Geodetic System. 1984년에 제정된 범 지구적 측위 시스템

## [ 좌표 ]

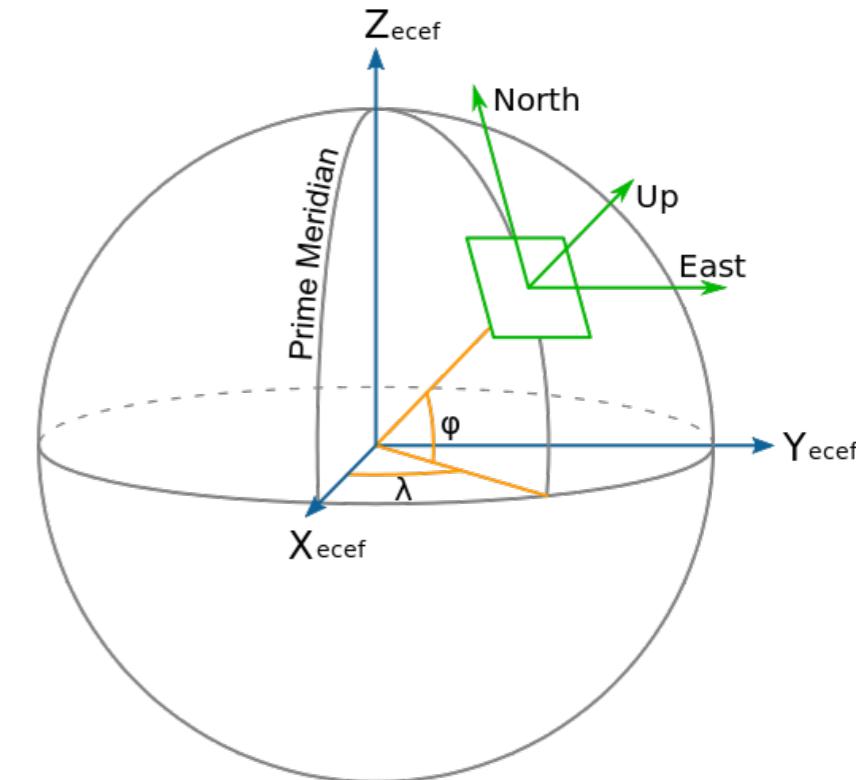
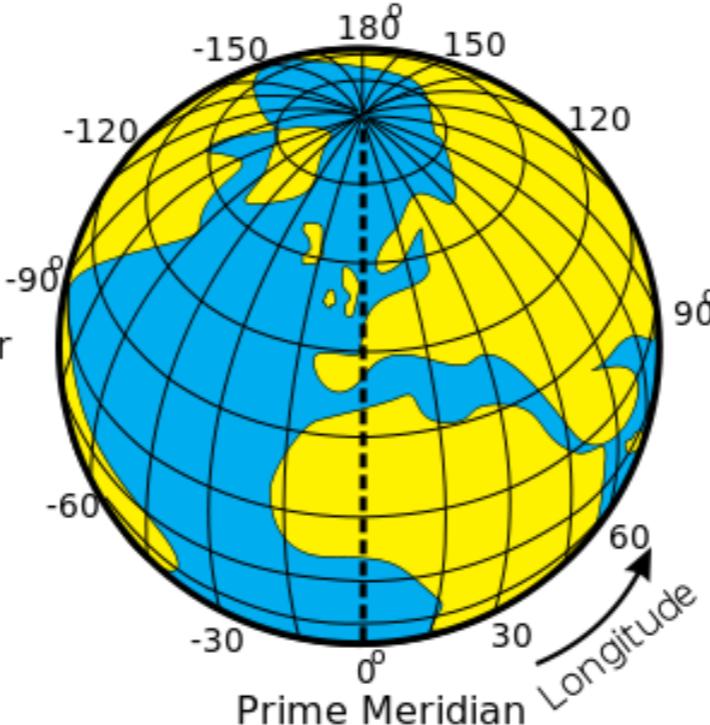
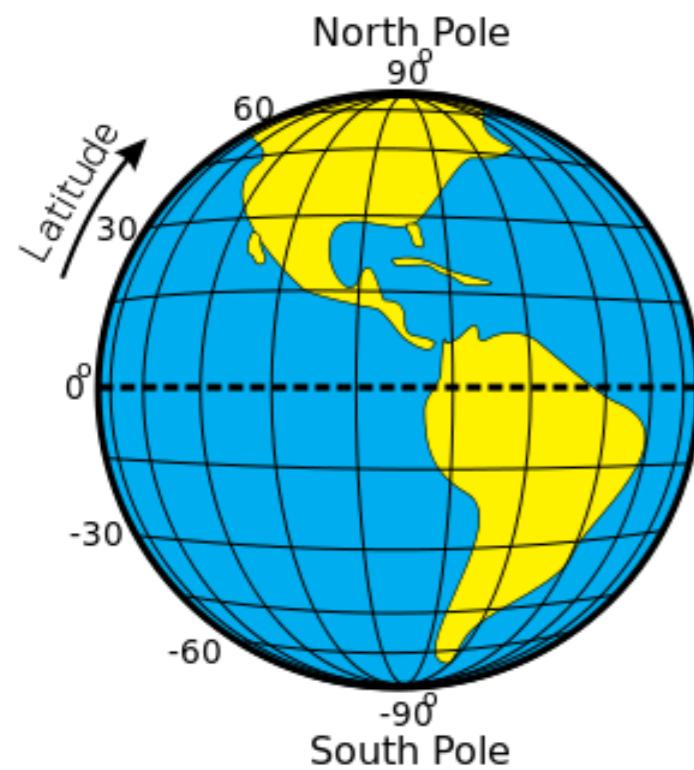
```
var latitude: CLLocationDegrees  
var longitude: CLLocationDegrees
```

## [ 관련 함수 ] - CLLocation.h 에 정의

```
func CLLocationCoordinate2DMake(  
    _ latitude: CLLocationDegrees, _ longitude: CLLocationDegrees  
) -> CLLocationCoordinate2D
```

```
func CLLocationCoordinate2DIsValid(  
    _ coord: CLLocationCoordinate2D  
) -> Bool
```

# Geographic coordinate system



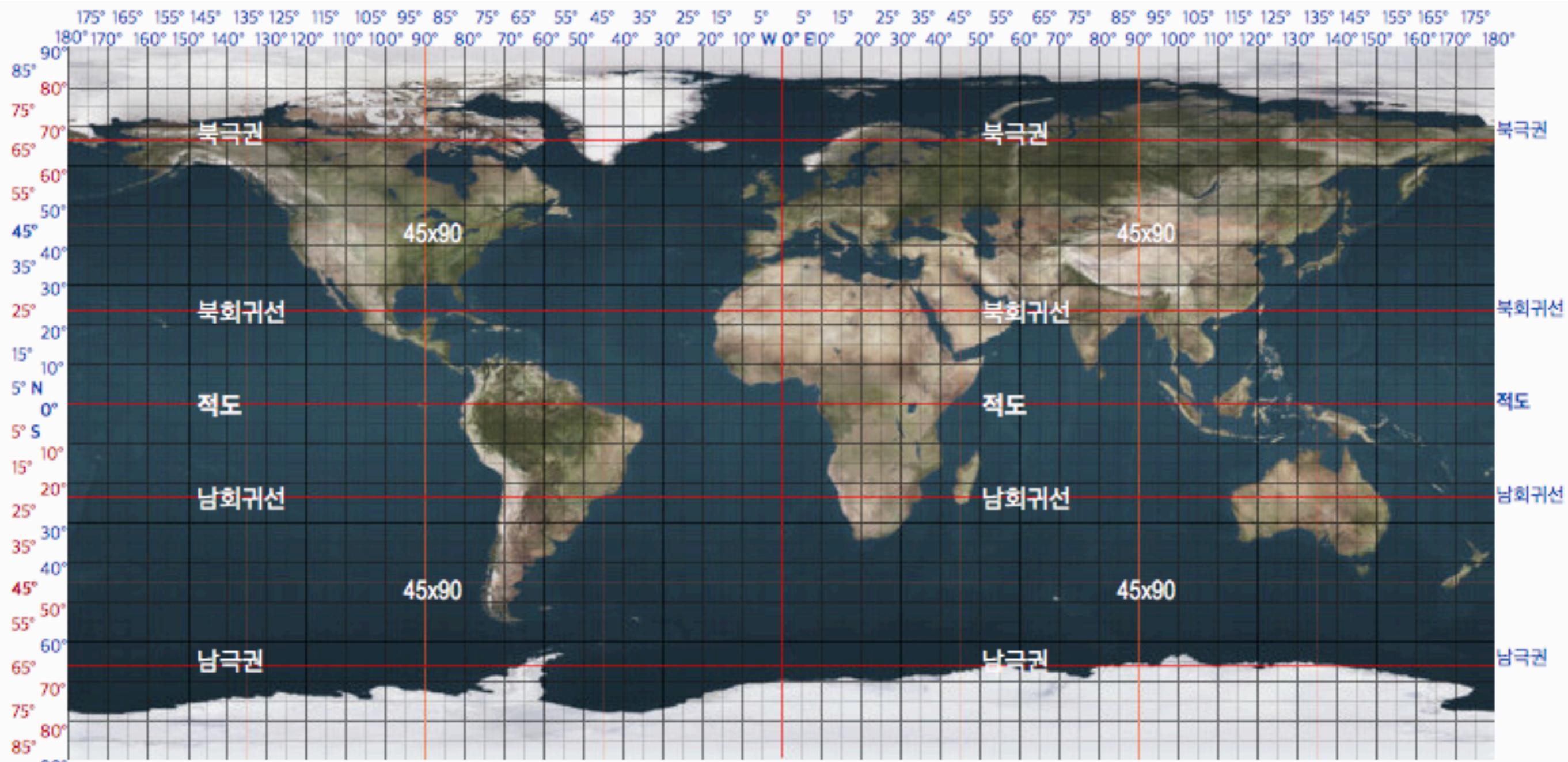
**latitude** [lætɪtu:d] : 위도

- 적도면과 표면 지점의 수직선(추선)이 이루는 각
- 범위  $90 \sim -90$  / 북위  $90$  (북반구) ~ 남위  $90$  (남반구)

**longitude** [la:ndʒətu:d] : 경도

- 북극부터 남극까지의 표면 지점을 그은 경선과 본초 자오선 (현재 그리니치 천문대 기준)이 이루는 각
- 범위  $180 \sim -180$ , 동경  $180$  (동반구) ~ 서경  $180$  (서반구)

# Geographic coordinate system



# Geographic coordinate system

서울 시청 - **37° 33' 58.87"N / 126° 58' 40.63"E**

## \* 읽는 법

- 37도 33분 58.87초
- thirty-seven degrees thity-three minutes fifty-eight dot eighty-seven seconds north

## \* 계산 방법 ( $1^\circ = 60'$ , $1' = 60''$ )

$$37.5663528 = 37^\circ 33' 58.87" N$$

e.g.

$$37 = 37^\circ$$

$$0.5663528 * 60 = 33.981168'$$

$$0.981168 * 60 = 58.87"$$

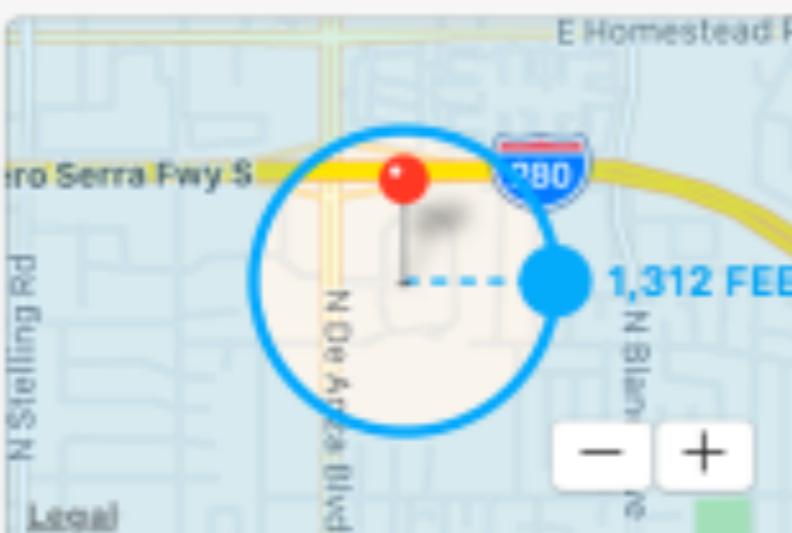
# Region Monitoring

Pick up dry cleaning

remind me  On a Day  At a Location

Work  
1 Infinite Loop MS 37-1DP Cupertino  
CA 95014 United States

Arriving  Leaving



priority None

note None

Done

# Region Monitoring

---

- 사용자가 정의한 지역을 시스템이 모니터링하여 경계를 넘나들 때 필요에 따라 앱 활성화/이벤트 처리
- 추상클래스인 **CLRegion** 을 상속받은 **CLCircularRegion** 클래스를 사용하며 특정 중심점을 기준으로 원형의 영역을 모니터링
- 코어 로케이션이 하나의 앱이 동시에 20개 이상의 지역을 모니터링하는 것을 방지
- 잘못된 호출을 방지하기 위해 사용자가 경계를 넘어 최소 거리를 이동하고 최소 20초 동안 경계의 같은 면에 있어야 함
- 가능한 현재 위치에 가까운 영역만 모니터링하고 이동 시 새 위치를 기반으로 목록 업데이트 권장
- 백그라운드에서 감지하고 동작할 수 있도록 백그라운드 모드의 **Location Updates** 활성화 필요

# Region Monitoring

---

```
func monitorRegionAtLocation(  
    center: CLLocationCoordinate2D,  
    identifier: String  
) {  
    guard CLLocationManager.authorizationStatus() == .authorizedAlways,  
        CLLocationManager.isMonitoringAvailable(for: CLCircularRegion.self)  
    else { return }  
    let maxDistance = locationManager.maximumRegionMonitoringDistance  
    let region = CLCircularRegion(  
        center: center, radius: maxDistance, identifier: identifier  
)  
    region.notifyOnEntry = true  
    region.notifyOnExit = false  
    locationManager.startMonitoring(for: region)  
}
```

# Compass Headings



# Heading and Course Information

---

## Heading

- 자성 (magnetic) 또는 진북 (true north)에 상대적인 기기의 방향
- 자력계 (magnetometer)가 내장된 기기에서만 사용 가능 (시뮬레이터에서 사용 불가)
- 이 정보를 토대로 기기의 현재 방향을 파악하여 나침반이나 AR에서의 직면 방향 결정
- 일반적으로 상단이 기준점이지만 headingOrientation 속성을 통해 별도로 설정 가능

## Course

- GPS 정보를 기반으로 기기가 움직이고 있는 방향 (기기의 실제 방향과 무관)
- 일반적으로 네비게이션 앱에서 사용되며 GPS가 내장된 기기 필요
- 충분한 위치 정보가 수집된 후 CLLocation 객체의 speed, course 속성에 자동으로 포함

# CLHeading

자성 또는 진북에 상대적인 기기의 방위각(방향)과 그 계산에 필요한 원시 값을 포함하는 객체  
일반적으로 이 클래스의 객체를 직접 생성하거나 서브클래싱 하지 않으며 Delegate 를 통해 처리  
CLHeading 객체의 True Heading 속성에 유효한 정보가 포함되게 하려면 위치 정보에 대한  
업데이트도 함께 수행해주어야 함

## [ 속성 ]

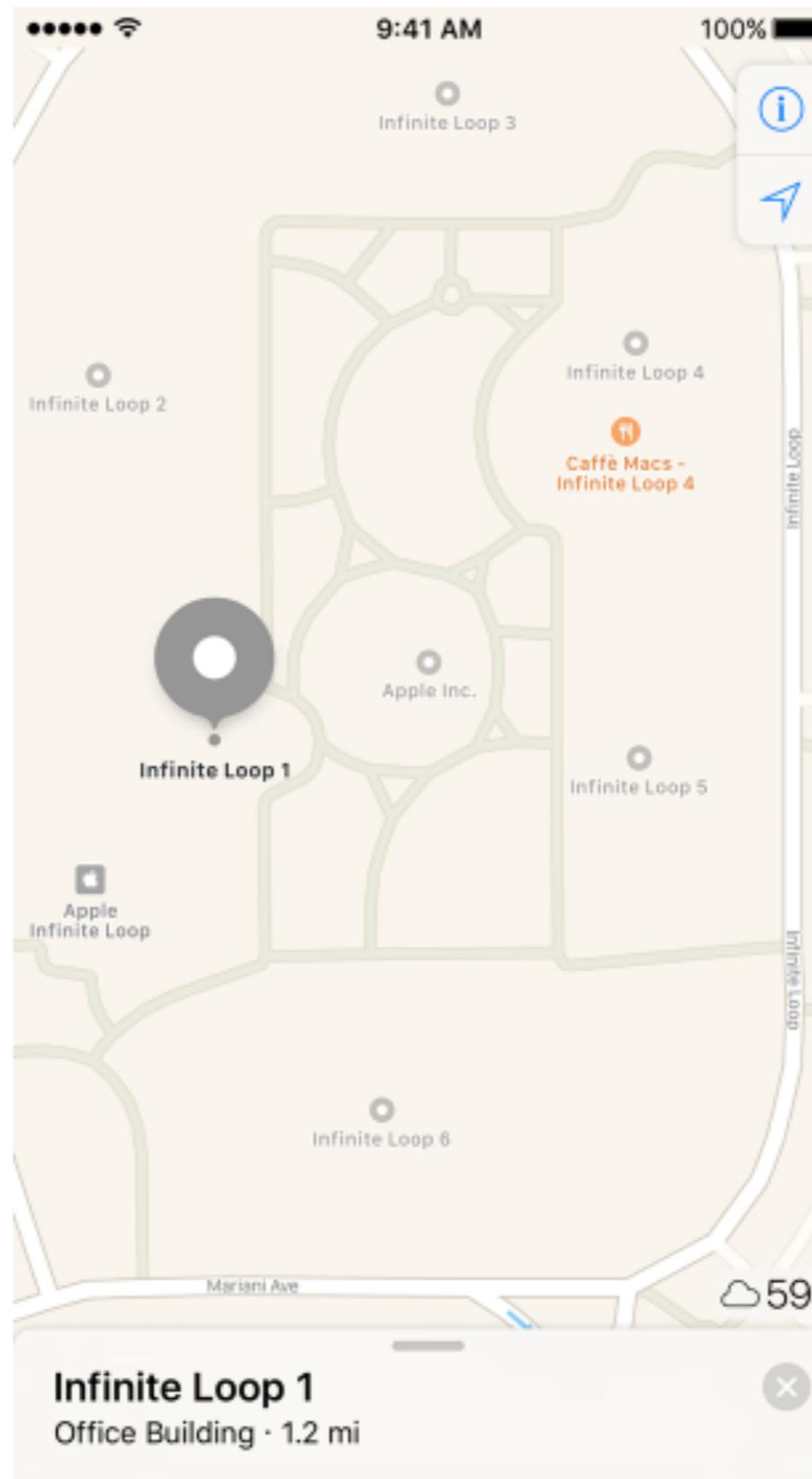
```
var magneticHeading: CLLocationDirection    // magnetic north
var trueHeading: CLLocationDirection        // true north
var headingAccuracy: CLLocationDirection   // Heading의 최대 편차
var timestamp: Date
```

## [ 원시 데이터 ]

```
var x, y, z: CLHeadingComponentValue      // 각 축에 대한 지자기 데이터
```

# Geocoding

---



# Geocoding

CLGeocoder 를 이용해 지리적 좌표(위도/경도)와 사용자에게 친숙한 지명 간 변환 처리  
장소에 대한 정보는 CLPlacemark 를 통해 제공

Geocoder 객체는 단일 객체(one-shot objects) 로서, 각 객체를 하나의 변환에만 사용  
여러 오브젝트를 생성해 다중 변환을 수행할 수 있으나 그 수에 제한이 있으며 그 이상 시도 시 실패 가능성

**Reverse Geocode : Coordinate → Placemark (좌표를 지명으로)**

**Geocode : Placemark → Coordinate (지명을 좌표로)**

[ Geocoding 을 효과적으로 사용하기 위한 팁 ]

한 사용자 액션에 대해 하나의 Geocoding 만 요청

동일 또는 근접 거리에서는 새로운 지오 코딩을 수행하는 대신 기존 요청 결과를 재활용

일반적인 상황에서는 분당 하나 이상의 지오 코딩 요청을 보내지 말 것

앱이 Foreground 상태여서 사용자가 결과를 즉시 볼 수 있는 상황에서만 Geocoding 수행

# Reverse Geocode Location

---

```
func lookUpCurrentLocation(  
    completionHandler: @escaping (CLPlacemark?) -> Void  
) {  
    guard let lastLocation = locationManager.location else {  
        return completionHandler(nil)  
    }  
    let geocoder = CLGeocoder()  
    geocoder.reverseGeocodeLocation(lastLocation) {  
        (placemarks, error) in  
        let firstLocation = placemarks?.first  
        completionHandler(firstLocation)  
    }  
}
```

# Geocode Location

---

```
func getCoordinate(  
    addressString: String,  
    completionHandler: @escaping(CLLocationCoordinate2D, Error?) -> Void  
) {  
    let geocoder = CLGeocoder()  
    geocoder.geocodeAddressString(addressString) {  
        (placemarks, error) in  
        guard error == nil else {  
            return completionHandler(kCLLocationCoordinate2DInvalid, error)  
        }  
        if let placemark = placemarks?.first {  
            completionHandler(placemark.location!.coordinate, nil)  
        }  
    }  
}
```