AND231

Location Services

Download class materials from
university.xamarin.com

Microsoft

Xamarin University

Objectives

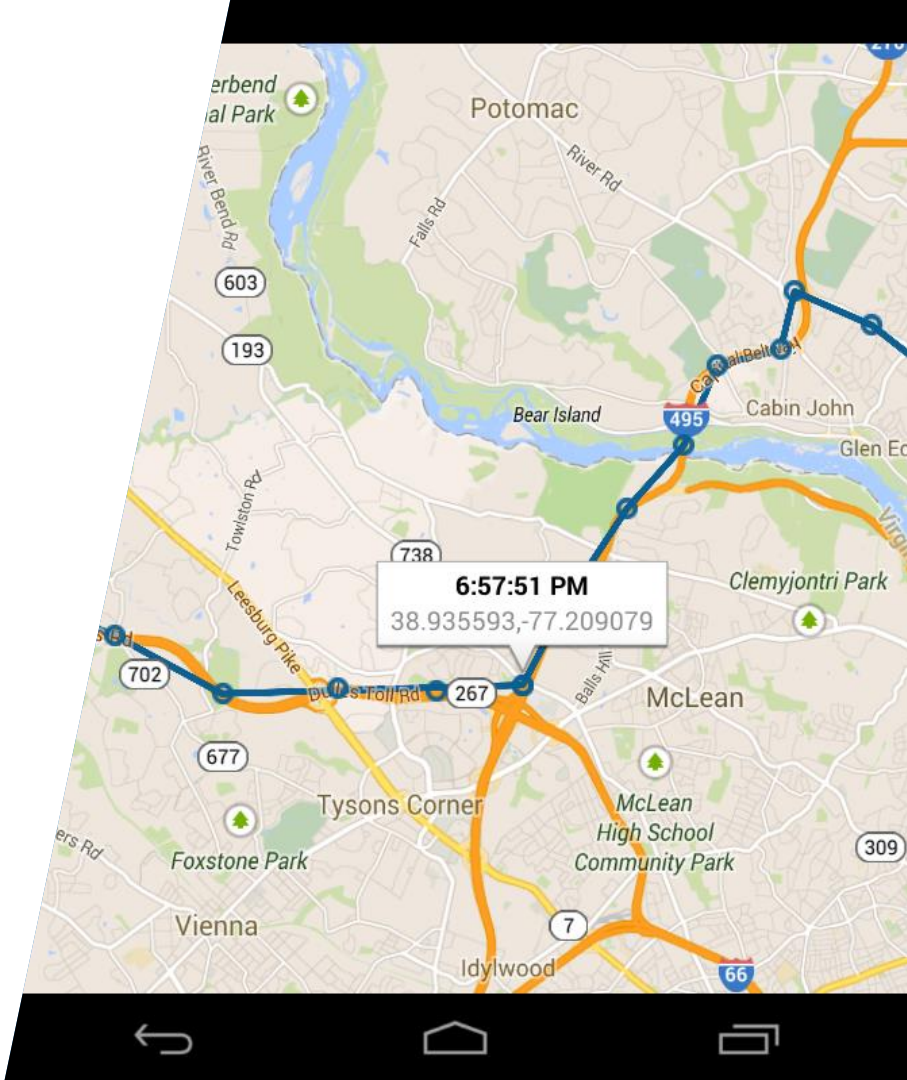
1. Determine the current location
2. Search for an address

Determine the current location

Xamarin University

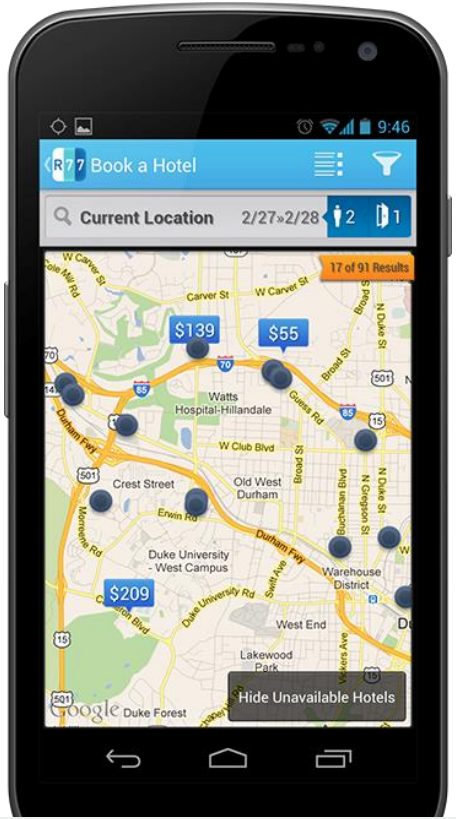
Tasks

1. Receive location update notifications
2. Select a location provider
3. Process location updates efficiently



Recall: Maps in Android

- Current location support can enhance applications in a variety of ways
  - points-of-interest around me
  - point-to-point navigation
  - interactive tour guidance
  - "Where Am I?"
  - …



This class will use Google Maps API V2 and should be considered a continuation of the **AND230 – Intro to Maps in Android** course

Recall: Google Play Services

- Google Play Services has significant setup requirements
  - Uses keystore SHA1 fingerprint combined with app package id
  - Create an App ID in Google Play developer console

    console.developers.google.com/
  - Requires permissions in manifest + `[MetaData()]` for app id

Group Exercise

Setup the Google Maps API key

Xamarin University

Location strategies

- Mobile devices often have several built-in sensors which can be used to determine the device's location
  - GPS
  - WiFi
  - Cellular

- Each strategy has different accuracy and reliability characteristics

Where am I?

- `LocationManager` is a built-in Android service which provides access to all the location providers available on the device

- Must have location permissions in the manifest to use the providers
  - `AccessFineLocation`
  - `AccessCoarseLocation`

LocationManager
Class
Object

Properties
- AllProviders : IList<string>

Methods
- FromContext() : LocationManager
- GetBestProvider() : string
- GetGpsStatus() : GpsStatus
- GetLastKnownLocation() : Location
- GetProvider() : LocationProvider
- GetProviders() : IList<string> (+ 1 overload)
- IsProviderEnabled() : bool
- RemoveUpdates() : void (+ 1 overload)
- RequestLocationUpdates() : void (+ 4 overloads)
- RequestSingleUpdate() : void (+ 3 overloads)

Using LocationManager

- Can retrieve the **`LocationManager`** singleton using **`GetSystemService`** or the static **`FromContext`** method

```
LocationManager locManager = this.GetSystemService(
                Context.LocationService) as LocationManager;
```

```
LocationManager locManager = LocationManager.FromContext(this);
if (locManager != null) {
   // Make sure to always test the result to see if
   // Location Services are supported!
}
```

Requesting location updates

- Start listening for location updates by calling `RequestLocationUpdates`

```
LocationManager locManager;

const string provider = LocationManager.GpsProvider;

locManager.RequestLocationUpdates(provider, 2000, 1f, this);
```

Must indicate where you want location data from – this is called the *location provider*

Requesting location updates

- Start listening for location updates by calling `RequestLocationUpdates`

```
LocationManager locManager;

const string provider = LocationManager.GpsProvider;

locManager.RequestLocationUpdates(provider, 2000, 1f, this);
```

Minimum update interval in milliseconds

Requesting location updates

- Start listening for location updates by calling `RequestLocationUpdates`

```
LocationManager locManager;

const string provider = LocationManager.GpsProvider;

locManager.RequestLocationUpdates(provider, 2000, 1f, this);
```

Minimum distance change in meters to issue an update

Requesting location updates

- Start listening for location updates by calling **RequestLocationUpdates**

```
LocationManager locManager;

const string provider = LocationManager.GpsProvider;

locManager.RequestLocationUpdates(provider, 2000, 1f, this);
```

Location update processor – this can be an **ILocationListener** implementation (app callback, often implemented on the **Activity**) or a pending intent mapped to a broadcast receiver (system callback)

What is ILocationListener?

- `ILocationListener` receives location and provider updates from `LocationManager` - must provide an implementation when requesting location updates

ILocationListener
Interface
IJavaObject
IDisposable

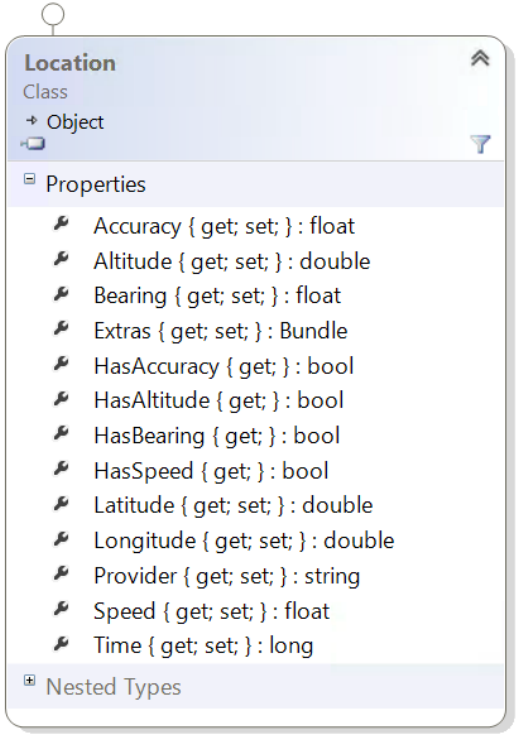
Methods

OnLocationChanged(Location location) : void
OnProviderDisabled(string provider) : void
OnProviderEnabled(string provider) : void
OnStatusChanged(string provider, Availability status, Bundle extras) : void

Working with the location data

- **`Location`** object contains Longitude/Latitude and (possibly) a plethora of other information such as altitude, bearing and speed

- Should check the **`HasXXX`** properties to determine which values are valid



Receiving location updates

- Implement the **`OnLocationChanged`** method to receive location updates – can then animate the camera for the map to the new location

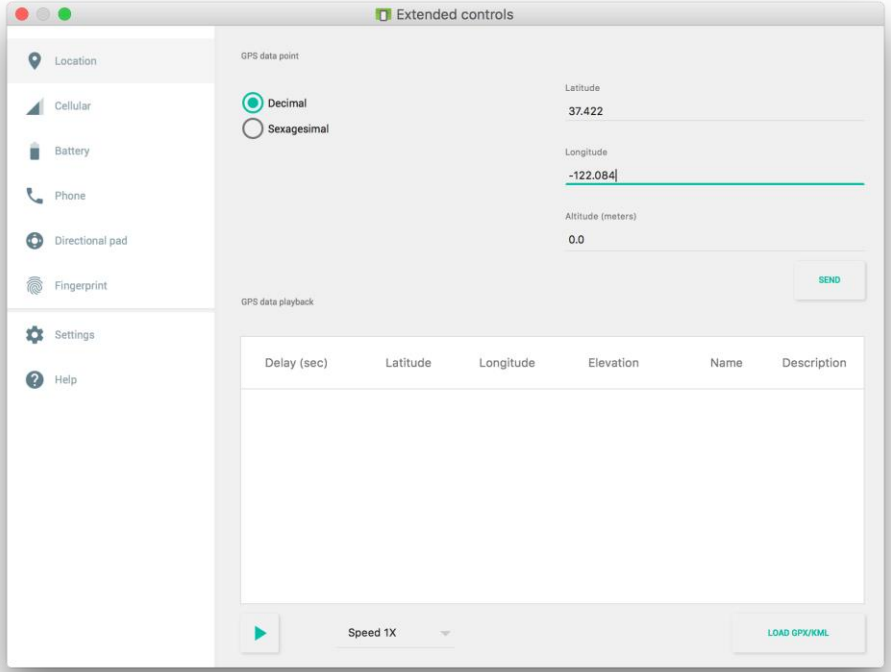
```
GoogleMap map = ...;

void ILocationListener.OnLocationChanged(Location loc)
{
   var coord = new LatLng(loc.Latitude, loc.Longitude);
   var update = CameraUpdateFactory.NewLatLngZoom(coord, 17);
   map.AnimateCamera(update);
}
```

Testing your location changes

- Many emulators support simulation of location information – can change to a given coordinate and check your application logic

- Google AVDs' locations can be set in the Extended controls panel



Individual Exercise

Working with the Location Manager

Xamarin University

What are Location Providers?

- Android uses an extensible set of *location providers* (identified by **string** name) to provide the underlying location data to the Location Manager

```
foreach (string provider in locManager.AllProviders)
   Console.WriteLine(provider);
```

These strings are passed into **RequestLocationUpdates** to select the provider →

```
passive
gps
network
```

Location Provider properties

Provider	Accuracy	Power Usage	Notes
gps	20-200ft	Medium > High	▪ Requires GPS on the device ▪ Requires line-of-sight to satellites ▪ Slow to get a fix but very accurate
network	5300ft	Low > Medium	▪ Fairly accurate ▪ Depends on carrier support ▪ Can use assisted GPS
passive	~ 1mile	Low	▪ Very fast, GPS can be turned off ▪ No extra power required ▪ Generally low accuracy

Location Provider status

- Location providers can be turned on and off based on various network settings, battery status and user preferences; can check to see if a specific provider is enabled before using it

```
const string provider = LocationManager.GpsProvider;
if (locManager.IsProviderEnabled(provider)) {
   locManager.RequestLocationUpdates(provider, 2000, 1f, this);
}
```

- Can also implement other methods of **`ILocationListener`** to monitor the provider status while your application runs

Selecting a provider

- Android can tell you the proper location provider based on *criteria* you specify such as cost, accuracy and power requirements

```
Criteria locationCriteria = new Criteria {
     Accuracy = Accuracy.Medium,
     CostAllowed = false,
     AltitudeRequired = false,
     BearingRequired = true,
     BearingAccuracy = Accuracy.Coarse,
     PowerRequirement = Power.Low,
};
...
var provider = locManager.GetBestProvider(
                    locationCriteria, true);
```

Can specify to only consider enabled providers

Selecting a provider

- Can also pass **`Criteria`** right into to **`RequestLocationUpdates`**

```
Criteria locationCriteria = new Criteria { ... };
...
locManager.RequestLocationUpdates(
          2000, 1f, locationCriteria, this, null);
```

Individual Exercise

Identifying a provider based on criteria

Xamarin University

Thinking about battery life

- Location services are *expensive* – should decide when they are necessary and turn them on and off as needed to conserve battery
- Can increase the distance or time interval to reduce impact, but this will reduce the accuracy
- Select the proper provider based on accuracy vs. power

Turning off location updates

- Turn off location updates when you are no longer interested in the location, or when the application is no longer active

```
LocationManager locManager;

protected void override OnStop()
{
    base.OnStop();
    ...
    locManager.RemoveUpdates(this);
}
```

This will unregister all notifications to the passed **ILocationListener**

When should the app start monitoring?

- Must decide when to start listening for location updates and how long to monitor the location changes

Decide between battery consumption vs. location accuracy

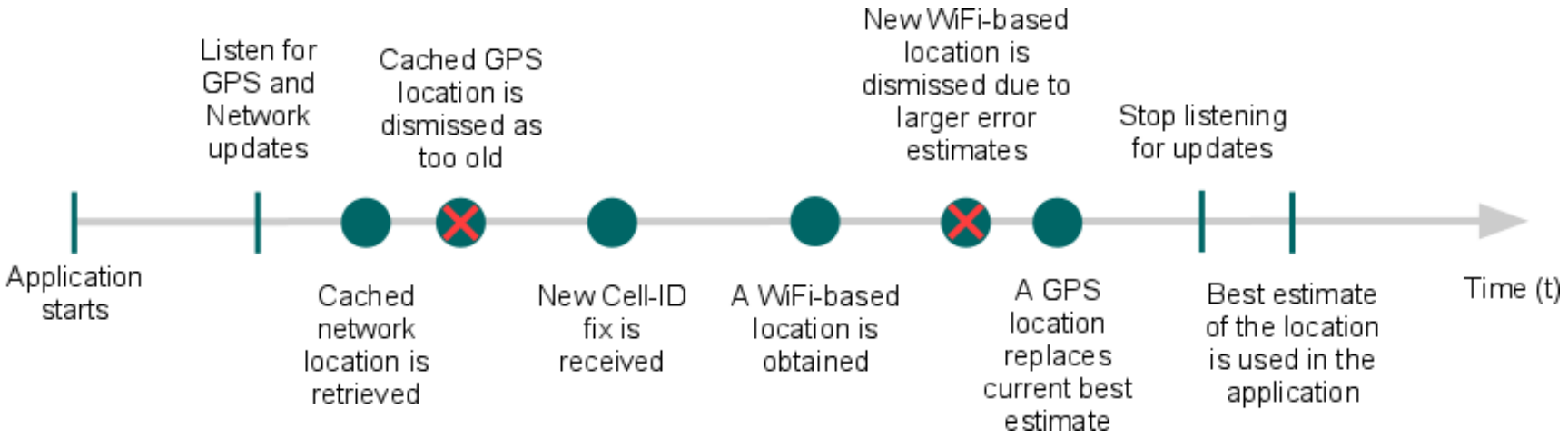
Use cached location for quick fixes

- Providers cache off the last known location – can use this information without turning on location updates, or to provide an initial position while providers get a location fix

```
LocationManager locManager;
const string provider = ...;
...
Location lastLoc = locManager.GetLastKnownLocation(locationProvider);
if (lastLoc != null) {
   LatLng coord = new LatLng(lastLoc.Latitude, lastLoc.Longitude);
   ...
}
```

Getting location updates

- Common to use location data from *multiple* providers – and then decide which location you want to use based on provider + time



Get an initial value

- Start by getting the last known location from the active providers

```
LocationManager locManager;

var l1 = locManager.GetLastKnownLocation(LocationManager.GpsProvider);
var l2 = locManager.GetLastKnownLocation(LocationManager.NetworkProvider);

Location currentLocation  = (l1 != null && l2 != null)
        ? l1.Time >= l2.Time
             ? l1 : l2
        : l1 ?? L2;
if (currentLocation != null)
   OnLocationChanged(currentLocation);
```

Registering for location updates

- Register to listen for GPS and Network (WiFi+Cell) updates

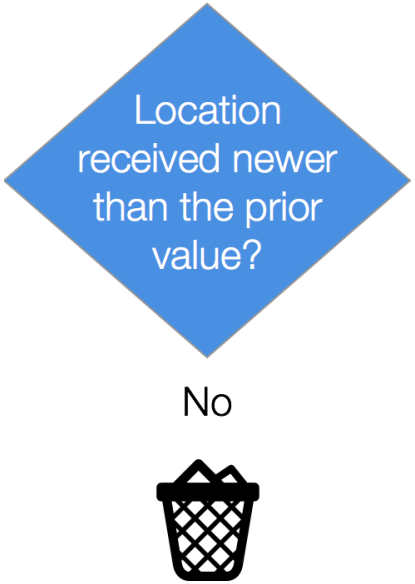
```
LocationManager locManager;
Location currentLocation;

locManager.RequestLocationUpdates(LocationManager.GpsProvider,
                                  5000, 100f, this);
locManager.RequestLocationUpdates(LocationManager.NetworkProvider,
                                  5000, 100f, this);
```

Use the same **`ILocationListener`** for both providers

Location updates

- Recent location fixes may not always be the most accurate, it depends on the provider and accuracy of the data



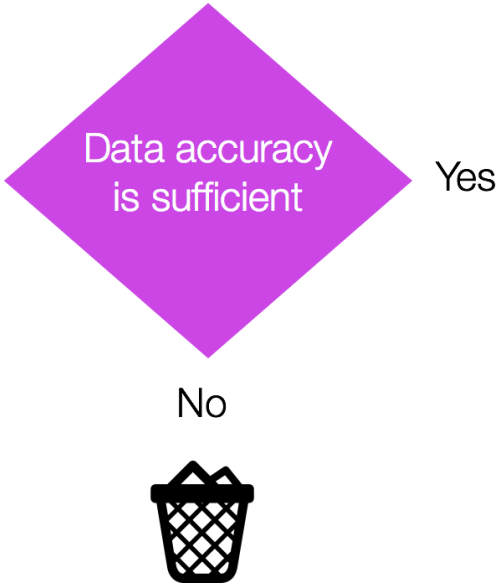

```
Location currentLocation;

public void OnLocationChanged(Location location)
{
    if (location.Time > currentLocation.Time) {
        ...
    }
}
```

Time is represented in milliseconds since Jan 1 1970 (UTC)

Location updates

- Recent location fixes may not always be the most accurate, it depends on the provider and accuracy of the data



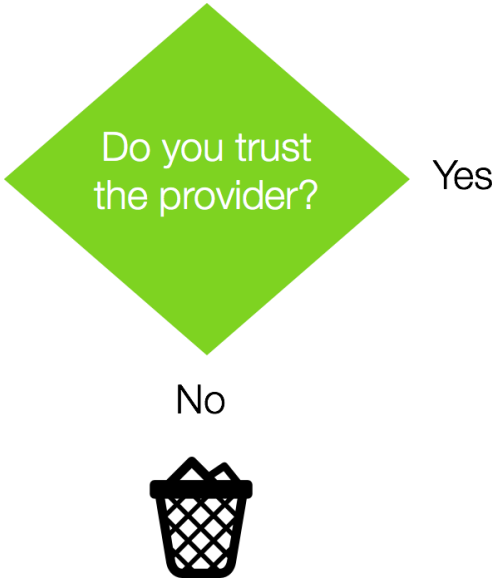

```
Location currentLocation;

public void OnLocationChanged(Location location)
{
    ...
    if (location.Accuracy < 5000
        && location.Accuracy
            <= currentLocation.Accuracy) {
        ...
    }
}
```

Accuracy is reported in meters

Location updates

- Recent location fixes may not always be the most accurate, it depends on the provider and accuracy of the data




```
Location currentLocation;

public void OnLocationChanged(Location location)
{
    ...
    if (location.Provider == currentLocation.Provider
    || location.Provider == LocationManager.GpsProvider)
    {
      currentLocation = location;
      // .. Use new location
    }
}
```

Flash Quiz

Xamarin University

Flash Quiz

① You must have both **AccessCoarseLocation** and **AccessFineLocation** permissions in the manifest to use GPS and network-based location

a) True

b) False

Flash Quiz

① You must have both **AccessCoarseLocation** and **AccessFineLocation** permissions in the manifest to use GPS and network-based location

a) True

b) False

Flash Quiz

② What class do you use to request location updates?

a) LocationManager
b) RequestLocationUpdates
c) ILocationListener
d) None of the Above

Flash Quiz

② What class do you use to request location updates?

a) LocationManager

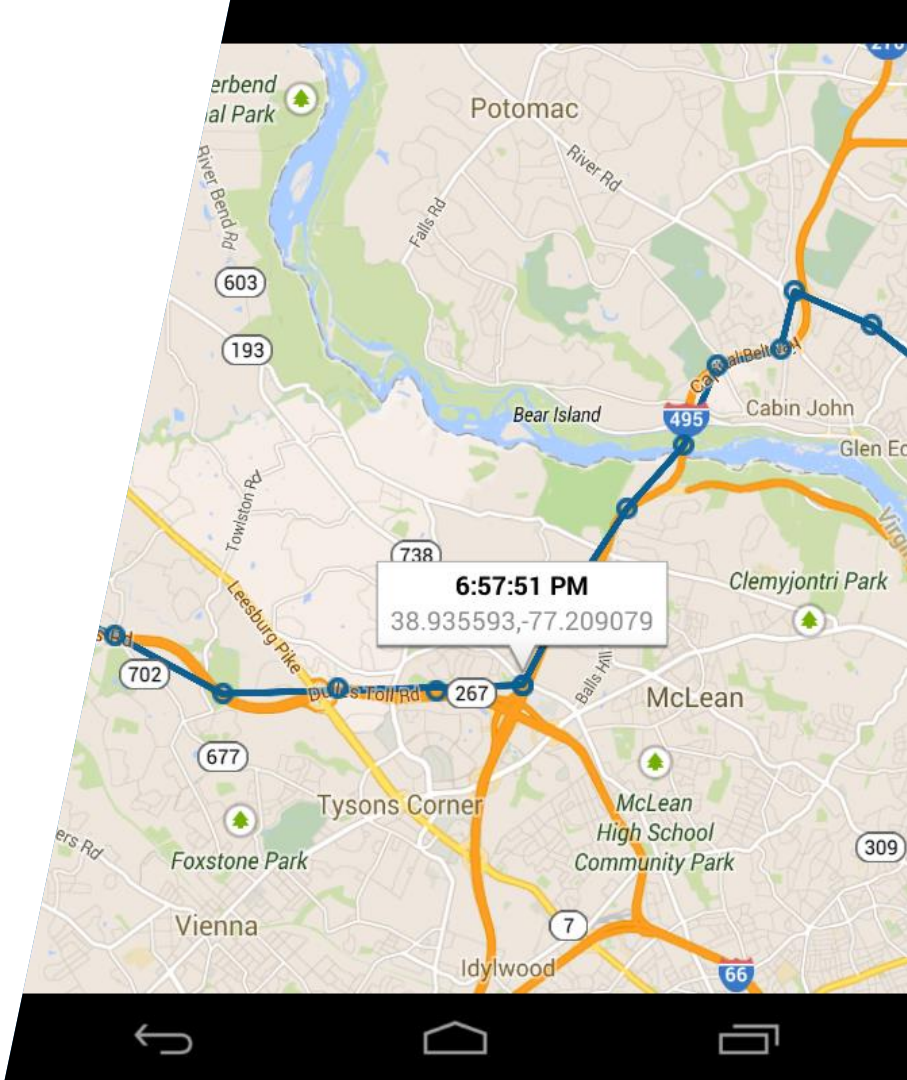
b) RequestLocationUpdates

c) ILocationListener

d) None of the Above

Summary

1. Receive location update notifications
2. Select a location provider
3. Process location updates efficiently



Using Google Play Services Location APIs

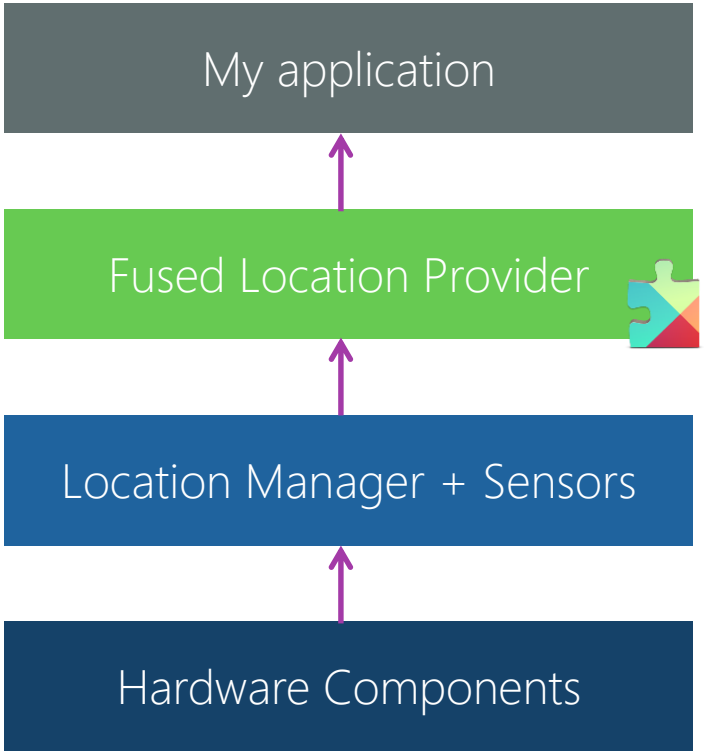
Xamarin University

Tasks

1. Connect to the Google API
2. Initialize the fused provider

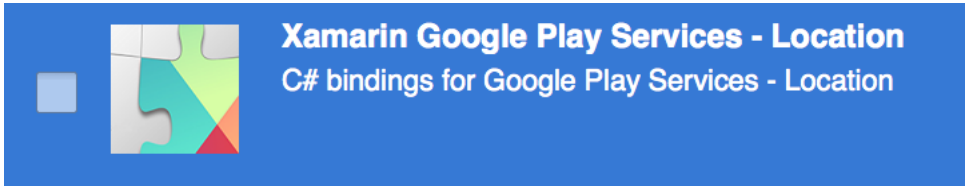
Fused Location Provider

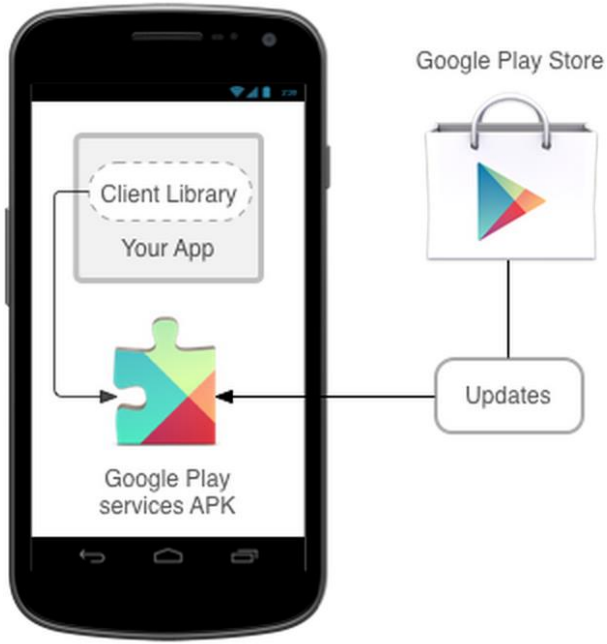
- Google Play services implements a new API which consolidates input from cell, WiFi, GPS and sensors to provide location
- Uses different API includes in the Google Play Services SDK
- Faster and more power-friendly than `LocationManager`



Google Play Services component

- Must add **Google Play Services - Location component (client library)** to your Xamarin.Android application to access the fusion provider





Connecting to the Google API

- Fused location provider requires a Google API Client which is typically created as part of **`OnCreate`** using a fluid API builder

```
GoogleApiClient apiClient;
protected override void OnCreate(Bundle bundle) {
   ...
   if (GoogleApiAvailability.Instance
         .IsGooglePlayServicesAvailable(this) == 0) {
      apiClient = new GoogleApiClient.Builder(this)
                        .AddConnectionCallbacks(this)
                        .AddOnConnectionFailedListener(this)
                        .AddApi(LocationServices.API)
                        .Build();
   }
```

Connecting to the Google API

- Once the API client has been constructed, must *connect* and *disconnect* from the service – this should be done in **`OnStart`** and **`OnStop`**

```
GoogleApiClient apiClient;
protected override void OnStart() {
   base.OnStart();
   if (apiClient != null)
      apiClient.Connect();
}
protected override void OnStop() {
   base.OnStop();
   if (apiClient != null)
      apiClient.Disconnect();
}
```

Google Client connection

- Google API Connection callbacks are often implemented by the **`Activity`**

```
public class MainActivity : Activity, ...
            GoogleApiClient.IConnectionCallbacks,
            GoogleApiClient.IOnConnectionFailedListener
```

```
public void OnConnected(Bundle connectionHint)
```

```
public void OnConnectionSuspended(int cause)
```

```
public void OnConnectionFailed(ConnectionResult result)
```

Recovering from connection failures

- When a connection cannot be established with Google Play services, the **`OnConnectionFailed`** method is invoked and passed a connection result which can be used to attempt recovery in certain cases

```
public void OnConnectionFailed(ConnectionResult result) {
   if (result.HasResolution) {
      ... // Attempt to recover
   }
}
```

Check the **`HasResolution`** flag to see if a user interaction, such as a login, can resolve the error and allow a connection

Recovering from connection failures

- Call **`StartResolutionForResult`** to display a recovery UI, must pass an **`Activity`** and unique request ID

```
const int ResolvePlayErrorRequestId = 1001;

public void OnConnectionFailed(ConnectionResult result) {
   if (result.HasResolution) {
      result.StartResolutionForResult(this,
            ResolvePlayErrorRequestId);
   }
}
```

Handling the recovery response

- When the recovery UI is dismissed, it will report the result back to the specified **`Activity`** through the **`OnActivityResult`** method

```
protected override void OnActivityResult(int requestCode,
        Result resultCode, Android.Content.Intent data)
{
   if (requestCode == ResolvePlayErrorRequestId) {
      if (resultCode == Result.Ok) {
         if (!apiClient.IsConnecting
             && !apiClient.IsConnected) {
            apiClient.Connect();
         }
      }
   }
}
```

If recovery is successful, then activity should restart the connection

Handling errors in your error handler!

- Should make sure only one UI is activated at a time, and catch exceptions coming from the activity launch and reconnect in response

```
bool resolvingError;  // Make sure to reset in OnActivityResult

public void OnConnectionFailed(ConnectionResult result) {
   if (result.HasResolution && !resolvingError) {
         resolvingError = true;
         try { /* StartResolutionForResult */ }
         catch (Android.Content.IntentSender.SendIntentException) {
            resolvingError = false;
            apiClient.Connect();
         }
    ...
```

Using the Fused provider

- Once a Google API connection is established, can request location updates from the fused provider using the **`LocationServices`** API

```
public void OnConnected(Bundle connectionHint)
{
   LocationRequest request = new LocationRequest()
          .SetInterval(5000)
          .SetPriority(LocationRequest.PriorityHighAccuracy);

   LocationServices.FusedLocationApi.RequestLocationUpdates(
                  apiClient, request, this);
}
```

Using the Fused provider

- Once a Google API connection is established, can request location updates from the fused provider using the **`LocationServices`** API

```
public void OnConnected(Bundle connectionHint)
{
   LocationRequest request = new LocationRequest()
          .SetInterval(5000)
          .SetPriority(LocationRequest.PriorityHighAccuracy);

   LocationServices.FusedLocationApi.RequestLocationUpdates(
                 apiClient, request, this);
}
```

Pass **`ILocationListener`** to receive updates

Google API ILocationListener

- Fused provider uses a *different* interface which defines only the **`OnLocationChanged`** method – *can share implementation*, but must mark with new interface type

```
public class MainActivity : Activity, ...
            GoogleApiClient.IConnectionCallbacks,
            GoogleApiClient.IOnConnectionFailedListener,
            Android.Locations.ILocationListener,
            Android.Gms.Location.ILocationListener
```

Has the same name, so must fully qualify both interfaces to be able to support both **`LocationServices`** and **`LocationManager`**

Group Exercise

Adding support for LocationServices

Xamarin University

Summary

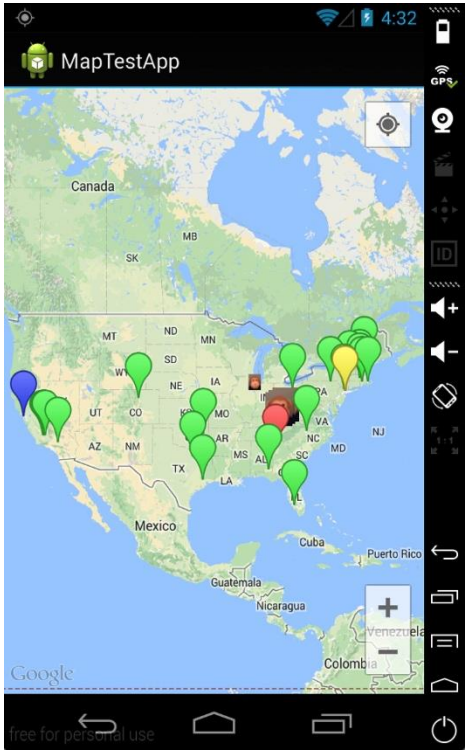
1. Connect to the Google API
2. Initialize the fused provider

Search for points of interest around the device

Xamarin University

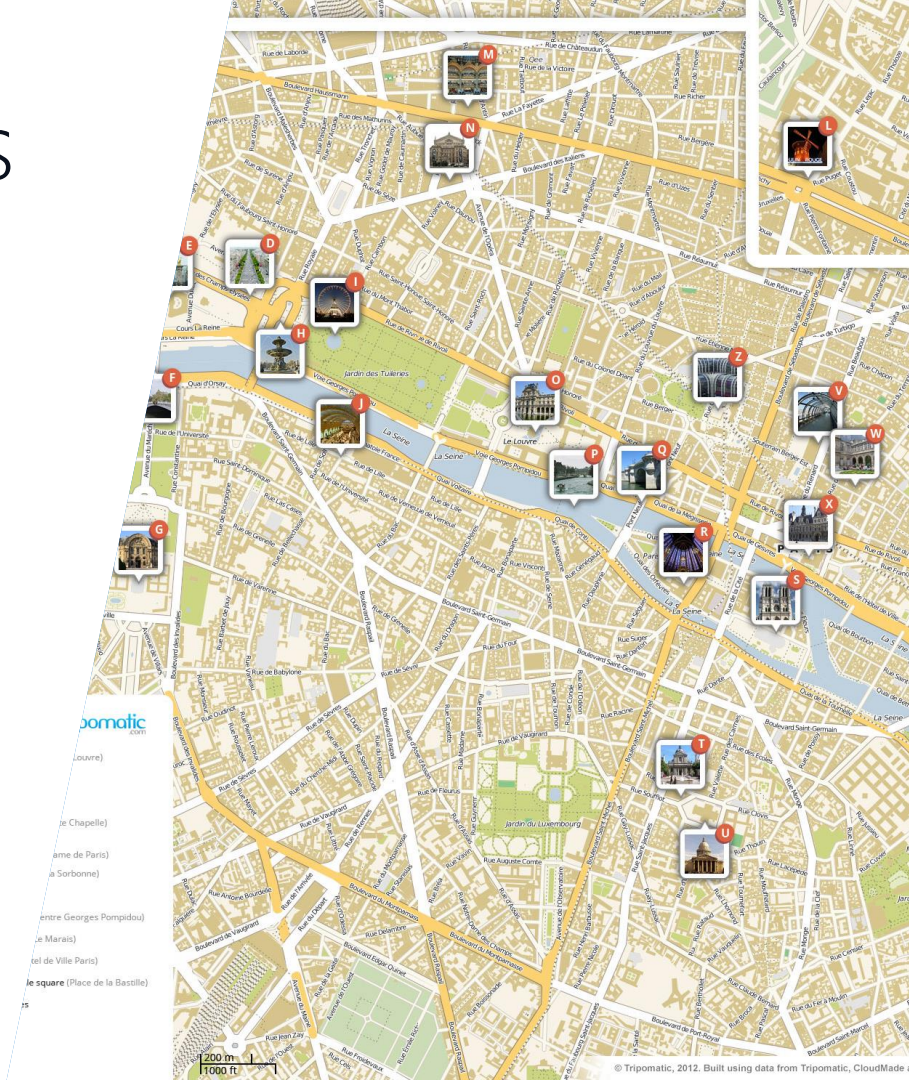
Tasks

1. Locate addresses from coordinates
2. Find locations by name



Searching for locations

- Google provides a web service API to search for locations by coordinates
- Can search for location of specific addresses or names



Searching for a location

- Can perform *reverse-geocoding* to lookup an address from a given coordinate (latitude and longitude) using the **Geocoder** class

```
if (Geocoder.IsPresent) {
   Geocoder geoCoder = new Geocoder(this);
   var addresses = await geoCoder.GetFromLocationAsync(
                                   37.42279, -122.08506, 1);
   if (addresses.Count > 0) {
      ... // Use located address
    }
}
```

Searching for a location

- Can perform *reverse-geocoding* to lookup an address from a given coordinate (latitude and longitude) using the **`Geocoder`** class

```
if (Geocoder.IsPresent) {
   Geocoder geoCoder = new Geoco
   var addresses = await geoCoder.Get
                                   37.422
   if (addresses.Count > 0) {
      ... // Use located address
    }
}
```

nc(
, 1);

Should verify that geo coder services are available

Searching for a location

- Can perform *reverse-geocoding* to lookup an address from a given coordinate (latitude and longitude) using the **`Geocoder`** class

```
if (Geocoder.IsPresent) {
   Geocoder geoCoder = new Geocoder(this);
   var addresses = await geoCoder.GetFromLocationAsync(
                                   37.4227    -122.08506, 1);
   if (addresses.Count > 0) {
      ... // Use located address
    }
}
```

Must pass **`Context`** to constructor

Searching for a location

- Can perform *reverse-geocoding* to lookup an address from a given coordinate (latitude and longitude) using the **`Geocoder`** class

```
if (Geocoder.IsPresent) {
   Geocoder geoCoder = new Geocoder(this);
   var addresses = await geoCoder.GetFromLocationAsync(
                                   37.42279, -122.08506, 1);
   if (addresses.Count > 0) {
      ... // Use located address
    }
}
```

Requested number of results – lower values (1-5) are recommended

Searching for a location

- Can perform *reverse-geocoding* to lookup an address from a given coordinate (latitude and longitude) using the **Geocoder** class

```
if (Geocoder.IsPresent) {
   Geocoder geoCoder = new Geocoder(this);
   var addresses = await geoCoder.GetFromLocationAsync(
                                   37.42279, -122.08506, 1);
   if (addresses.Count > 0) {
      ... // Use located address
    }
}
```

Returns a list of **Address** objects, around or at given coordinate

Addresses

- **`Address`** class is used to represent a location as a set of strings; not all properties are available for every location

- Can identify just an address, or may contain additional details such as "Eiffel Tower", phone number and even URL for a public website

Address
Class
Object

Properties
- AdminArea { get; set; } : string
- CountryCode { get; set; } : string
- CountryName { get; set; } : string
- Extras { get; set; } : Bundle
- FeatureName { get; set; } : string
- HasLatitude { get; } : bool
- HasLongitude { get; } : bool
- Latitude { get; set; } : double
- Locale { get; } : Locale
- Locality { get; set; } : string
- Longitude { get; set; } : double
- MaxAddressLineIndex { get; } : int
- Phone { get; set; } : string
- PostalCode { get; set; } : string
- Premises { get; set; } : string
- SubAdminArea { get; set; } : string
- SubLocality { get; set; } : string
- SubThoroughfare { get; set; } : string
- Thoroughfare { get; set; } : string
- Url { get; set; } : string

Methods
- GetAddressLine(int index) : string

Nested Types

Common details in Address

- Street address can be retrieved using the **`GetAddressLine`** method, often includes at least two lines but should always check the index count

```
string GetFullAddress(Address found)
{
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < found.MaxAddressLineIndex; i++) {
        if (sb.Length > 0) sb.Append(", ");
        sb.Append(found.GetAddressLine(i));
    }
    return sb.ToString();
}
```

Individual Exercise

Using Geocoder to perform reverse geocoding

Xamarin University

Finding locations by name

- **Geocoder** supports looking up locations by name / address as well

```
Geocoder geoCoder = new Geocoder(this);
var addresses = await geoCoder.GetFromLocationNameAsync(
                    "123 Alien Ave., Roswell, New Mexico", 1);
```

```
Geocoder geoCoder = new Geocoder(this);
var addresses = await geoCoder.GetFromLocationNameAsync(
                    "Eiffel Tower", 1);
```

Other options

- Google also has a Places API which returns prominent points of interest
  - like Maps, it requires an API key
  - supports photos, query auto-complete and far more details than just address

- https://developers.google.com/places/documentation/

Flash Quiz

Xamarin University

Flash Quiz

① When would you want to use the `Geocoder` class?

a) To issue address queries to Google's servers

b) To receive a specific **Address** object

c) Both A and B

d) None of the Above

Flash Quiz

① When would you want to use the `Geocoder` class?

a) To issue address queries to Google's servers

b) To receive a specific **Address** object

c) Both A and B

d) None of the Above

Flash Quiz

② The `Address` class returns the street address in a string property

a) True

b) False

Flash Quiz

② The `Address` class returns the street address in a string property

a) True

b) False

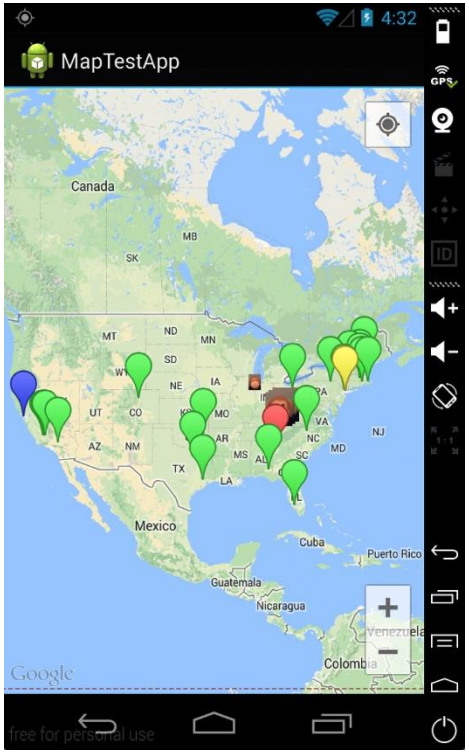
Individual Exercise

Using Geocoder to find specific address

Xamarin University

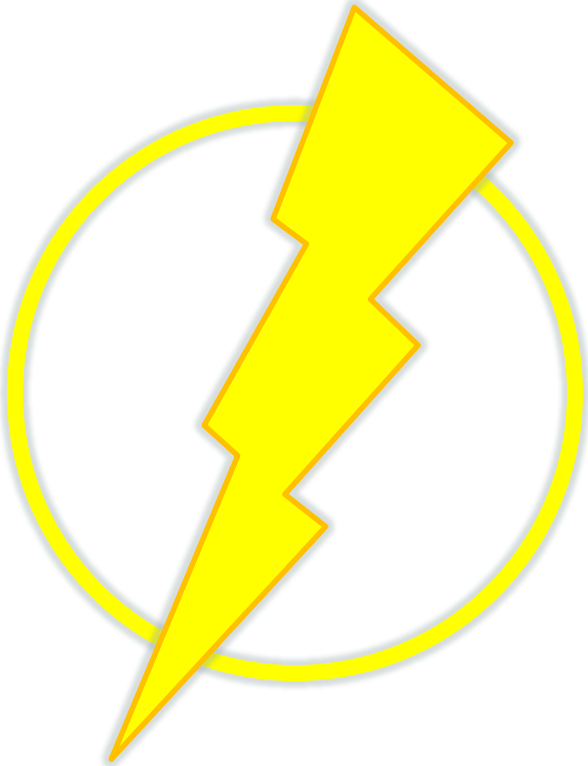
Summary

1. Locate addresses from coordinates
2. Find locations by name



Other Location classes

- Two **lightning lectures** are available to provide some practical examples of using location information:
  - Adding Geofencing into your application for location-aware notifications
  - Using the Google Directions API to draw routes on Android maps



Thank You!

Please complete the class survey in your profile:
university.xamarin.com/profile

Microsoft