IOS231

Customizing Maps

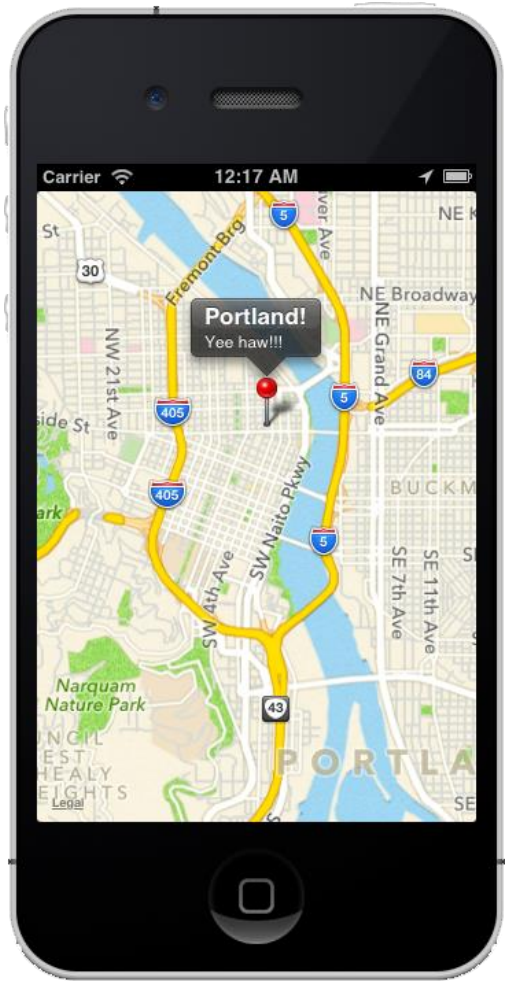
Download class materials from
university.xamarin.com

Microsoft

Xamarin University

Objectives

1. Create custom annotations
2. Group annotations with clusters
3. Interact with annotations
4. Search for points of interest
5. Add overlays and directions

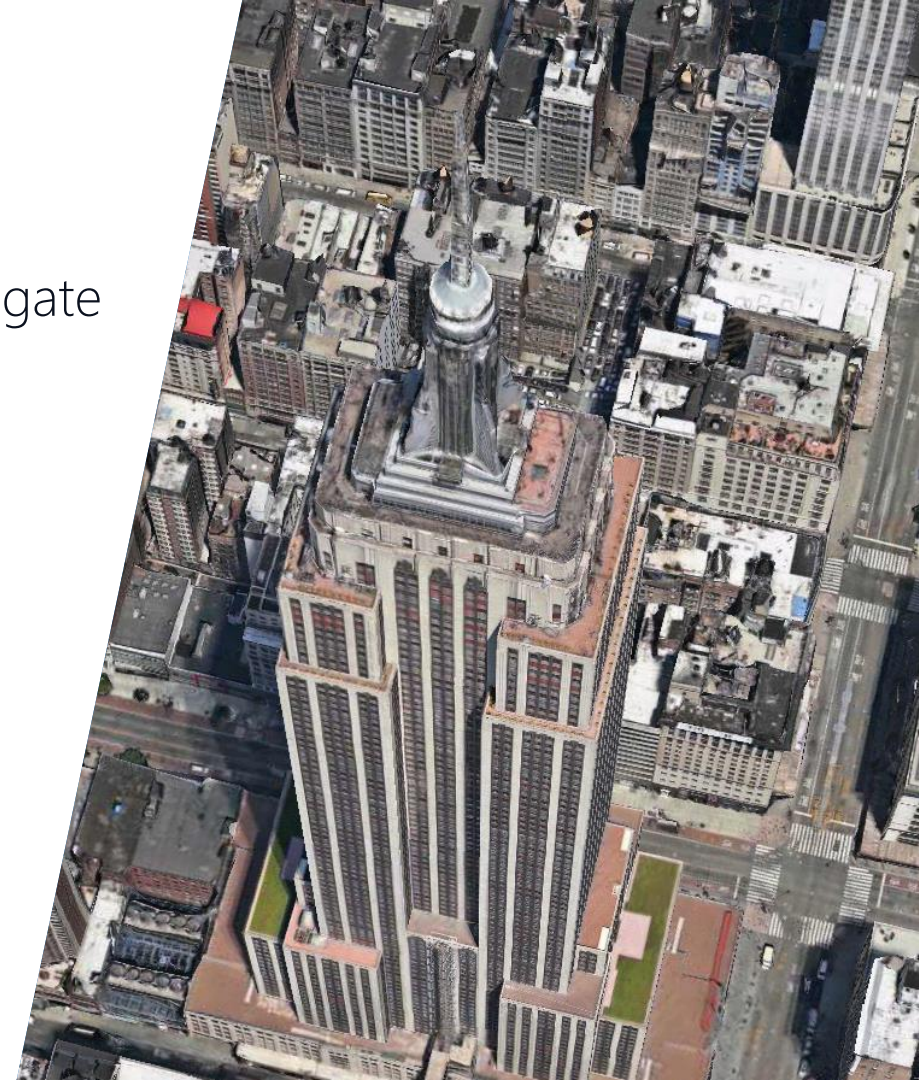


Create custom annotations

Xamarin University

Tasks

1. Identify and create a Map View Delegate
2. Customize the default annotation
3. Create custom annotations
4. Use custom images on annotations
5. Virtualize and reuse annotations



Review: annotations

- An annotation is a visualization of a single location on the map and has several related objects

MKAnnotation

The base class for all annotations

MKAnnotationView

Used to visualize the annotation

MKPointAnnotation

A model containing the title and position of a point of interest on the map

MKMarkerAnnotationView

The default visualization of the location on the map

Event Properties

- **MKMapView** provides public event properties to allow you to respond to map notifications including position changes, user location, change, and annotation rendering

```
MKMapView myMapView = new MKMapView();

myMapView.
```

```
DidAddOverlayRenderers
DidAddOverlayViews
DidChangeUserTrackingMode
DidDeselectAnnotationView
DidFailToLocateUser
DidFinishRenderingMap
DidSelectAnnotationView
```

What is the MKMapViewDelegate?

- **`MKMapViewDelegate`** is a protocol defined by iOS to extend and influence the **`MKMapView`** class

- This is an optional protocol – if you don't define and assign it, you get the default behavior and visualization

When using a **`MKMapViewDelegate`**, it's not possible to subscribe to the Event Properties on the same **`MKMapView`** instance

Create an MKMapViewDelegate

- To use the map delegate, create a new class that derives from **`MKMapViewDelegate`**

```
public class MyMapDelegate : MKMapViewDelegate
{
    ...
}
```

Override the delegate methods

- **`MKMapViewDelegate`** provides methods we can override to customize the appearance and behavior of the **`MKMapView`**

```
public override |
    CalloutAccessoryControlTapped(MKMapView mapView, MKAnnotationView view, UICon…
    ChangedDragState(MKMapView mapView, MKAnnotationView annotationView, MKAnnota…
    DidAddAnnotationViews(MKMapView mapView, MKAnnotationView[] views)
    DidAddOverlayRenderers(MKMapView mapView, MKOverlayRenderer[] renderers)
    DidAddOverlayViews(MKMapView mapView, MKOverlayView overlayViews)
    DidChangeUserTrackingMode(MKMapView mapView, MKUserTrackingMode mode, bool an…
    DidDeselectAnnotationView(MKMapView mapView, MKAnnotationView view)
```

Assign the MKMapViewDelegate

- Must then assign a delegate instance to the map through the **`MKMapView.Delegate`** property

```
var map = new MKMapView(...);

map.Delegate = new MyMapDelegate();
```

GetViewForAnnotation

- Override the **`GetViewForAnnotation`** method in the map delegate to customize the appearance of annotations

```
public class MyMapDelegate : MKMapViewDelegate
{
   ...
   public override MKAnnotationView GetViewForAnnotation(
           MKMapView mapView, IMKAnnotation annotation)
   {
      MKAnnotationView view = new MKMarkerAnnotationView(annotation, "pin");
      ...
      return view;
   }
}
```

Pass the annotation model we are visualizing

Must return the annotation view to render

MKMarkerAnnotationView

- `MKMarkerAnnotationView` provides the default marker visualization of a map annotation

- Renders a red marker image by default

Change the marker's color

- You can select any color for your markers by setting the **`MarkerTintColor`** property

```
var view = new MKMarkerAnnotationView (annotation, "pin");

view.MarkerTintColor = UIColor.Yellow;
```

```
view.MarkerTintColor = UIColor.FromRGB (0xFF,0x98,0x03);
```

Customize the marker's content

- There are multiple properties on **MKMarkerAnnotationView** that allow you to change the content of the marker

Change the text of the marker

```
var view = new MKMarkerAnnotationView (annotation, "pin");

view.GlyphText = " A+ ";

View.GlyphImage = UIImage.FromBundle (...);

view.SelectedGlyphImage = UIImage.FromBundle (...);
```

Use an image

Use a different image when selected

Going beyond colors

- Annotations are composed of two related objects, both of which may be customized to provide custom annotations

`MKPointAnnotation`
The data

`MKAnnotationView`
The visual

Custom MKPointAnnotation

- To associate additional information with a map annotation, create a class that inherits from **`MKPointAnnotation`**

```
public class HouseAnnotation : MKPointAnnotation
{
   public double Price { get; set; }
   public double Stories { get; set; }
   public int YearBuilt { get; set; }

   public  HouseAnnotation  (double price, double stories, int year)
   {
      ...
   }
}
```

Adding a custom annotation to the map

- Custom annotations can be added using the the **MKMapView**'s add annotation method

```
var houseAnnotation = new HouseAnnotation (200000, 2, 1990);

myMap.AddAnnotation (houseAnnotation);
```

No cast necessary since we conform to the proper protocol (**IMKAnnotation**) by deriving from **MKPointAnnotation**

Accessing custom annotation data

- The map delegate stores the annotation as an **IMKAnnotation**, cast to your custom class to access it's unique properties and/or methods

```
public override MKAnnotationView GetViewForAnnotation(
   MKMapView mapView, IMKAnnotation annotation) {
   ...
   var house = annotation as HouseAnnotation;

   if (house != null && house.Price < 50000)
         annotationView.MarkerTintColor = UIColor.Blue;

   ...
}
```

Individual Exercise

Change the marker color

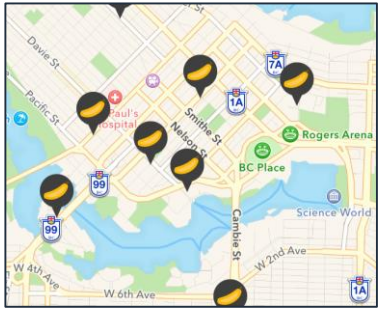
Xamarin University

What if I want a banana pin?

- We can create custom annotations when we want to display something other than the standard pin

Custom annotation visualizations

- We can tweak the **`MKAnnotationView`** in two ways:



Customize instances of `MKAnnotationView`

Place a custom image on an annotation view

Subclass `MKAnnotationView`

Allows for more control of the visualization

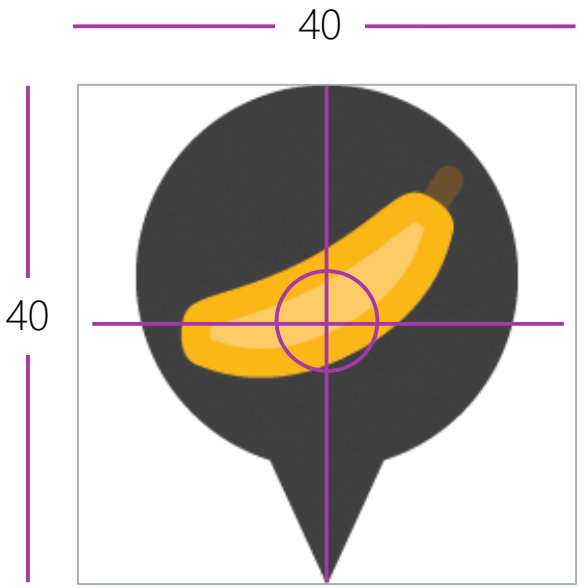
Replace the default annotation image

- In **`GetViewForAnnotation`** – create instances of **`MKAnnotationView`** and set the image property to your custom image

```
public override MKAnnotationView GetViewForAnnotation(
     MKMapView mapView, IMKAnnotation annotation)
{

   var view = new MKAnnotationView (annotation, "pin");
   view.Image = UIImage.FromBundle ("banana_pin.png");

   ...

   return view;
}
```

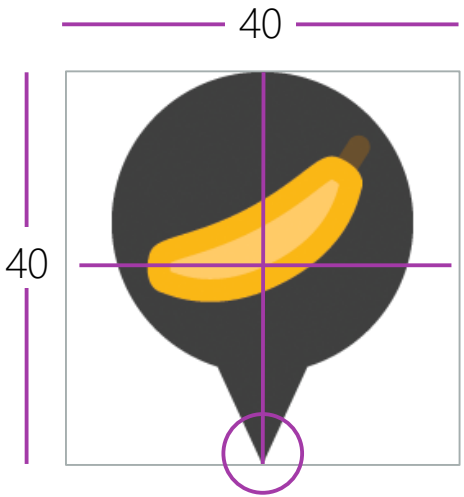
Align pin visualization to their locations

- By default Apple aligns your annotation to the center of the image



Align pin visualization to their locations

- Use the **`CenterOffset`** property on an **`MKAnnotationView`** to offset the center position




```
view = new MKAnnotationView (annotation, "pin");
view.Image = UIImage.FromBundle ("banana_pin.png");
view.CenterOffset = new CoreGraphics.CGPoint (0, -20);
...
```

MKAnnotationView

- Subclass **`MKAnnotationView`** for more complex annotation customizations

```
public class HousePin : MKAnnotationView
{
    UILabel lblPrice = new UILabel();
    double price;

    public  HousePin (IMKAnnotation annotation, string reuseID, double price)
    : base (annotation, reuseID)
    {
        this.Image = UIImage.FromBundle ("price_pin.png");
        this.price = price;
        AlignViews();
    }

    public void OnTapped () { ... }
}
```

This approach is often used to improve architecture when visualizing complex data

Recycling annotations

- Much like Table View cells, Map View allows you to reuse annotations to improve memory usage and performance

```
public override MKAnnotationView GetViewForAnnotation(
      MKMapView mapView, IMKAnnotation annotation)
{
   MKAnnotationView view = mapView.DequeueReusableAnnotation ("pin");

   if (view == null)
   {
      view = new MKAnnotationView (annotation, "pin");
      ...
   }
   return view;
}
```

Reuse identifier

Update recycled pin data

- Must update the **`Annotation`** property of the **`MKAnnotationView`** when using "recycled" pins to assign the annotation data

```
MKAnnotationView view = mapView.DequeueReusableAnnotation("pin");

if (view == null) {
   view = new MKAnnotationView (annotation, "pin");
}
else {
   view.Annotation = annotation;
}
```

Individual Exercise

Create a custom annotation

Xamarin University

Summary

1. Identify and create a Map View Delegate
2. Customize the default annotation
3. Create custom annotations
4. Use custom images on annotations
5. Virtualize and reuse annotations

Group annotations with clusters

Xamarin University

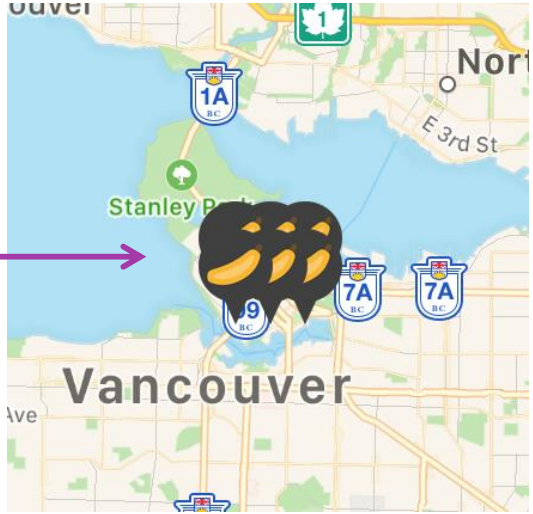
Tasks

1. Enable cluster support
2. Create clusters with MKMarkerAnnotationView

Motivation

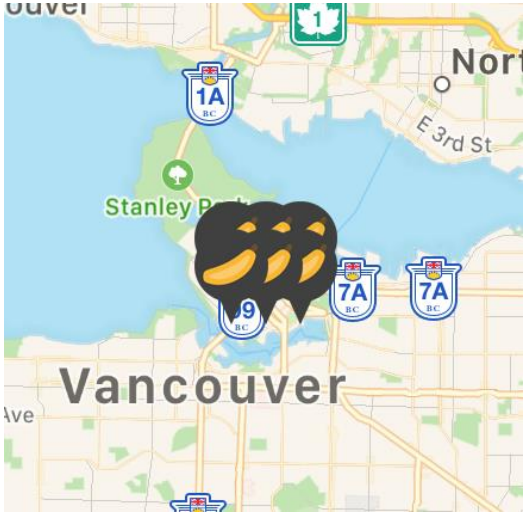
- An **MKMapView** with multiple annotations close together can be difficult to read

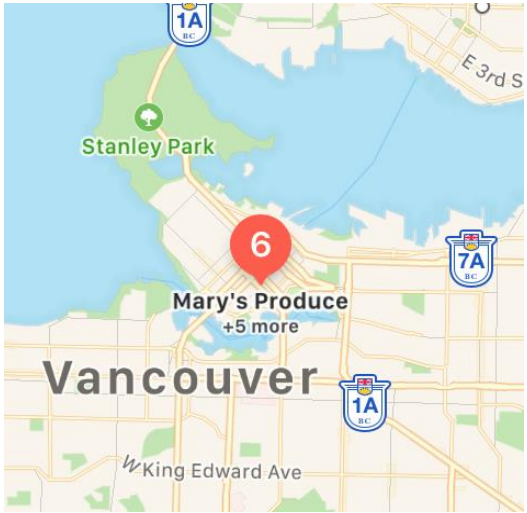
Annotations overlap and obscure the map



What is clustering?

- *Clustering* replaces overlapping annotations with a single annotation

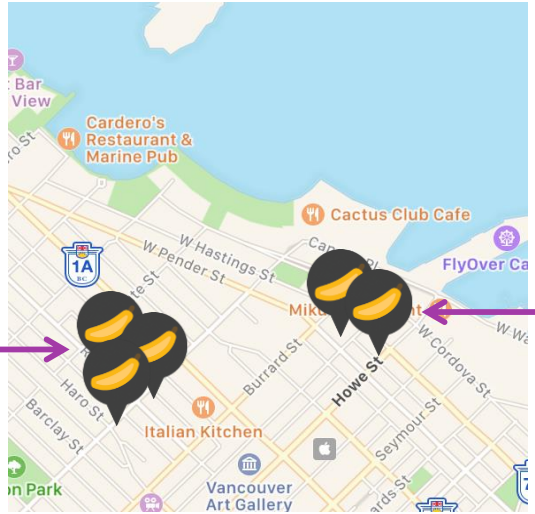




Cluster detection

- **MKMapView** does cluster detection based on hit testing of annotation views; annotations that overlap are eligible to be grouped into a cluster

Will become a cluster of 3

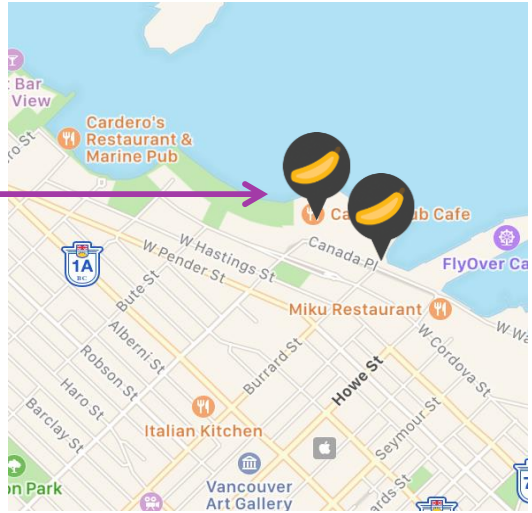


Will become a cluster of 2

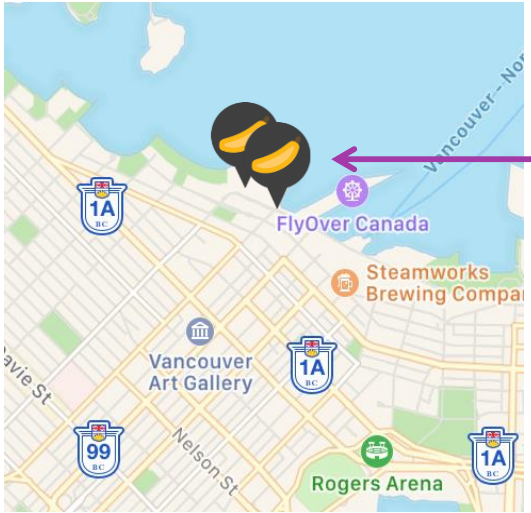
Clustering and zoom level

- Zoom level impacts placement of annotations; changing the zoom level may change which annotations overlap

No overlap



Zoom level: 15x



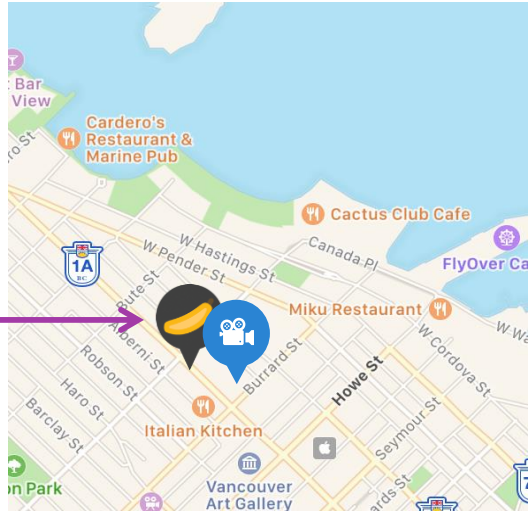
Overlapping

Zoom level: 10x

Related annotations

- Only annotations that are related should be grouped into a cluster

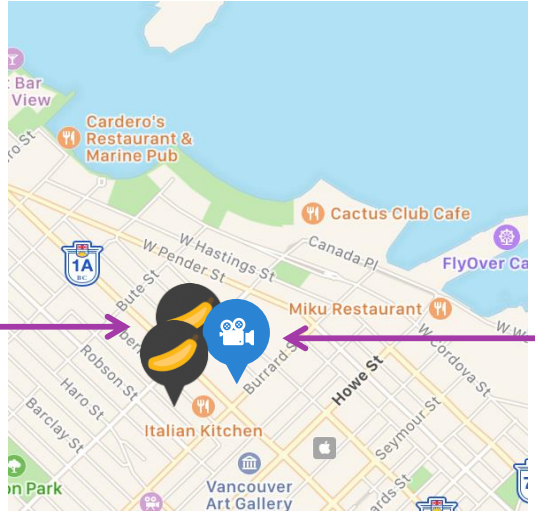
No cluster should be created



What is a cluster identifier?

- A cluster identifier is a string used by **MKMapView** to identify annotations that are eligible to be grouped

Same cluster identifier:
`"groceryStore"`



Different cluster identifier:
`"movieTheater"`

Enable clustering

- To enable clustering you assign the **`ClusteringIdentifier`** property to a string

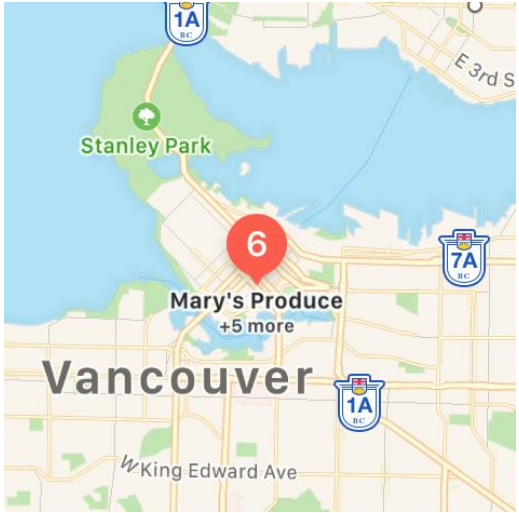
```
public override MKAnnotationView GetViewForAnnotation(
     MKMapView mapView, IMKAnnotation annotation)
{
   var view = new MKAnnotationView (annotation, "pin");
   view.ClusteringIdentifier = "banana";
   ...
   return view;
}
```

This enables clustering and sets the identifier

MKMarkerAnnotationView support

- `MKMarkerAnnotationView` supports clustering and will display the amount of replaced annotations automatically

This happens automatically; no custom code is required



Individual Exercise

Enable cluster support

Xamarin University

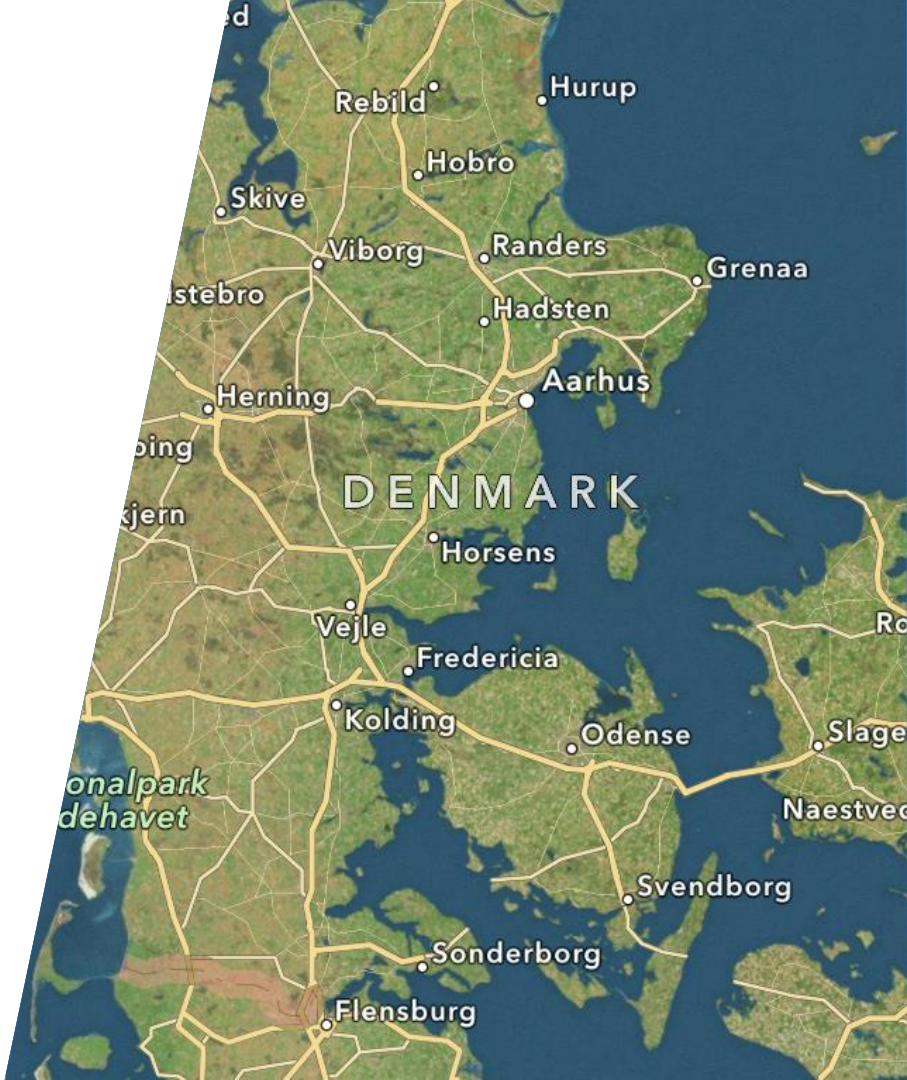
Summary

1. Enable cluster support
2. Create clusters with MKMarkerAnnotationView

Interact with annotations

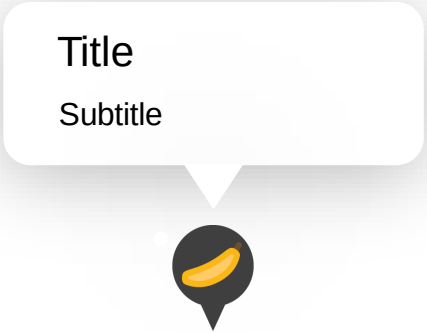
Tasks

1. Customize a callout
2. Detect touch events on the callout



Show the Callout View

- The callout for a specific location can be enabled or disabled by setting the **`CanShowCallout`** property of the **`MKAnnotationView`** class



The callout is displayed when a user taps a marker

```
view = new MKAnnotationView (annotation, "pin");
pinView.CanShowCallout = true;
...
```

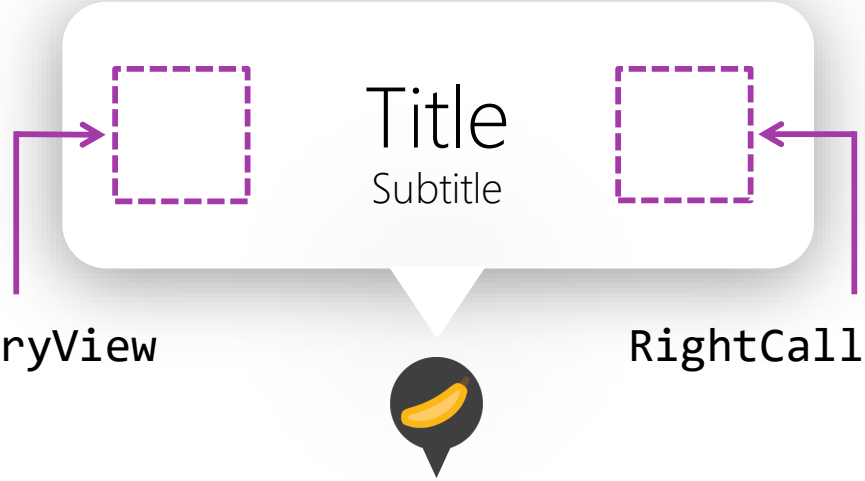
Why customize callouts?

- When we want to display more information in our callout than just a title and subtitle we have to implement some customization

VS

Callout accessory views

- The callout has two accessory views that can be utilized to display additional visual information



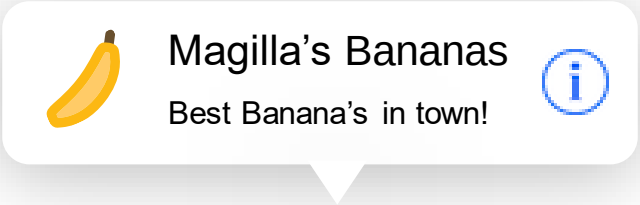
`LeftCalloutAccessoryView`

`RightCalloutAccessoryView`

Customize Accessory View

- Customize the accessory views by setting the **`RightCalloutAccessoryView`** or **`LeftCalloutAccessoryView`** properties of a **`MKAnnotationView`**

```
pinView.RightCalloutAccessoryView =
    UIButton.FromType(UIButtonType.DetailDisclosure);

pinView.LeftCalloutAccessoryView =
    new UIImageView(UIImage.FromFile("banana.png"));
```



Interact with the callout

- Override **`CalloutAccessoryControlTapped`** in the map delegate to respond to the taps on the callout

```
public override void CalloutAccessoryControlTapped (MKMapView mapView,
                                MKAnnotationView view, UIControl control)
{
   var annotation = view.Annotation as HouseAnnotation;
   ...

   var msg = String.Format ("Price:{0} Stories {1}",
                            annotation.Price, annotation.Stories);

   new UIAlertView (annotation.Title, msg, null, "OK", null).Show();
}
```

`CalloutAccessoryControlTapped` is only raised if a button has been added as an accessory view

Respond to taps

- Override **`DidSelectAnnotationView`** in the map delegate class to get notified when the user taps on an annotation

```
public class MyMapDelegate : MKMapViewDelegate
{
   public override void DidSelectAnnotationView (MKMapView mapView,
                                                 MKAnnotationView view)
   {
      var tappedAnnotation = view.Annotation;
      ...
   }
}
```

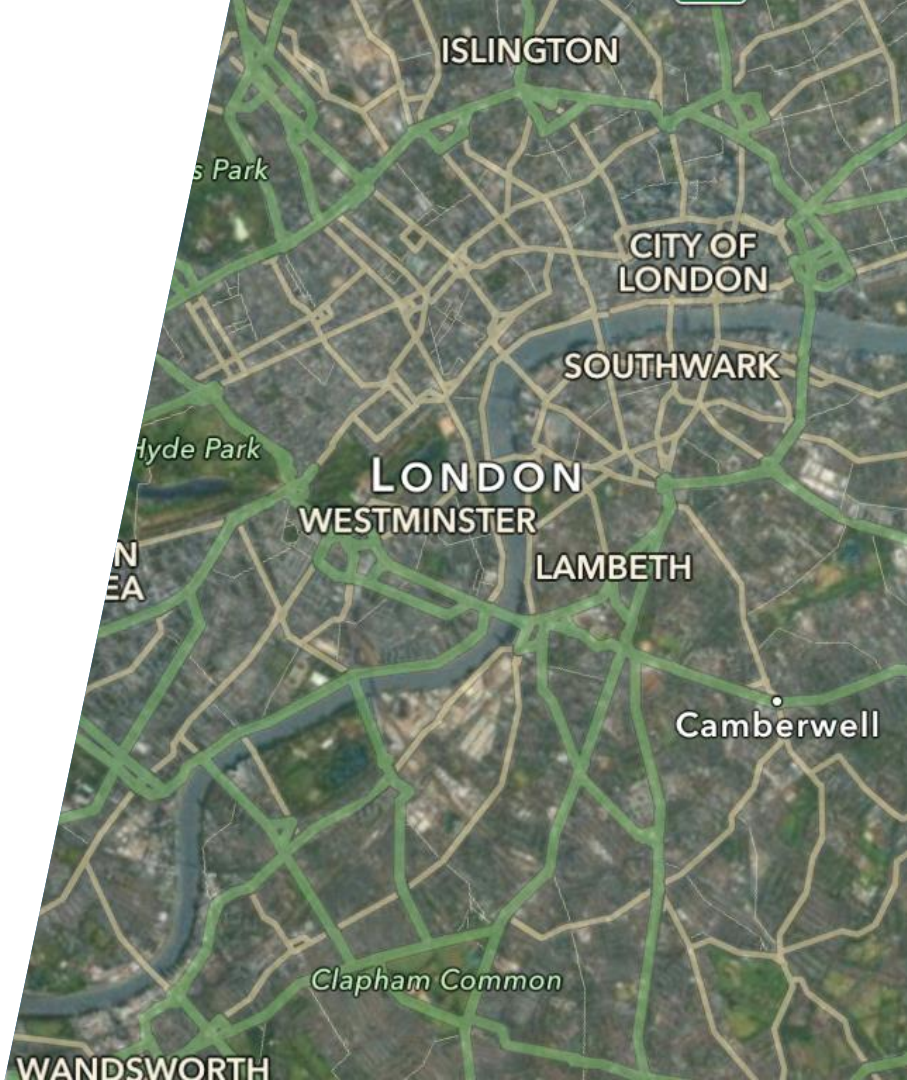
Individual Exercise

Add behavior to the annotations

Xamarin University

Summary

1. Customize a callout
2. Detect touch events on the callout

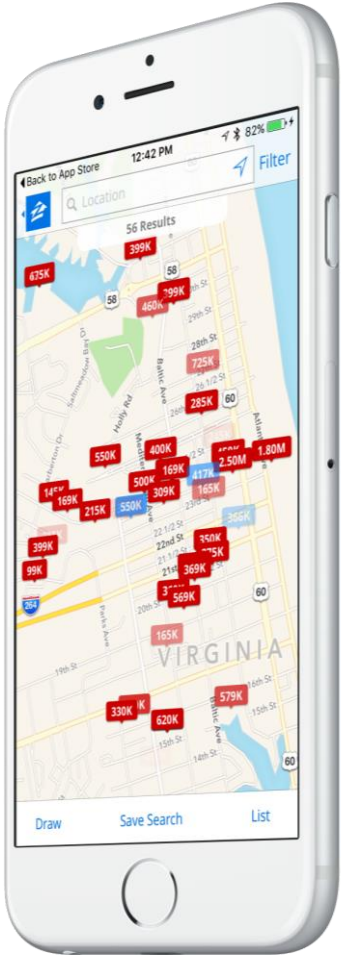


Search for points of interest

Xamarin University

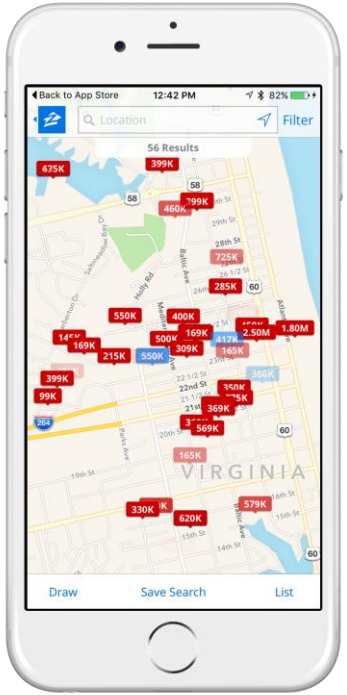
Tasks

1. Perform a search for points of interest
2. Display search results on the map



Searching for addresses and places

- **`MapKit`** has built in support for searching for addresses, names of locations and points of interest
- API is exposed through **`MKLocalSearch`** class



MKLocalSearchRequest

- **`MKLocalSearchRequest`** is used to provide search parameters

Set a natural language search term

```
var request = new MKLocalSearchRequest ();
   request.NaturalLanguageQuery = "Coffee Shops";
   request.Region = map.Region;
```

Optionally restrict searches to a specific region

Performing a search

- To perform a search, instantiate a new **MKLocalSearch** passing in a **MKLocalSearchRequest** and then call the **StartAsync** method

```
var request = new MKLocalSearchRequest ();
request.NaturalLanguageQuery = "Coffee shops";
request.Region = map.Region;

var local = new MKLocalSearch(request);

var response = await local.StartAsync();
```

MKLocalSearchResponse

- A **MKLocalSearchResponse** object is returned when **StartAsync** completes containing the search results

```
var local = new MKLocalSearch(request);

MKLocalSearchResponse response = await local.StartAsync ();
```

Returns a **Task** – can use **async**/**await** to easily consume API

Search Results

- **MKLocalSearchResponse** provides the search results as an array of **MKMapItem**s on its **MapItems** property

```
var local = new MKLocalSearch(request);

MKLocalSearchResponse response = await local.StartAsync ();

if (response != null && response.MapItems.Length > 0)
{
   foreach (var item in response.MapItems)
   {
      string result = item.Name + ": "+ item.Placemark.Title;
   }
}
```

MKMapItem

- **`MKMapItem`** is an object that encapsulates information about a specific location including:

Name

Phone Number

Url

Placemark

MKPlacemark

- A **`MKPlacemark`** object stores placemark data for a particular location including longitude, latitude, address and available geographically related information

CoreLocation.CLPlacemark

- AddressDictionary
- AdministrativeArea
- AreasOfInterest
- ClassHandle
- Country
- InlandWater
- IsoCountryCode
- Locality
- Location
- Name
- Ocean
- PostalCode
- Region
- SubAdministrativeArea
- SubLocality
- SubThoroughfare
- Thoroughfare
- TimeZone

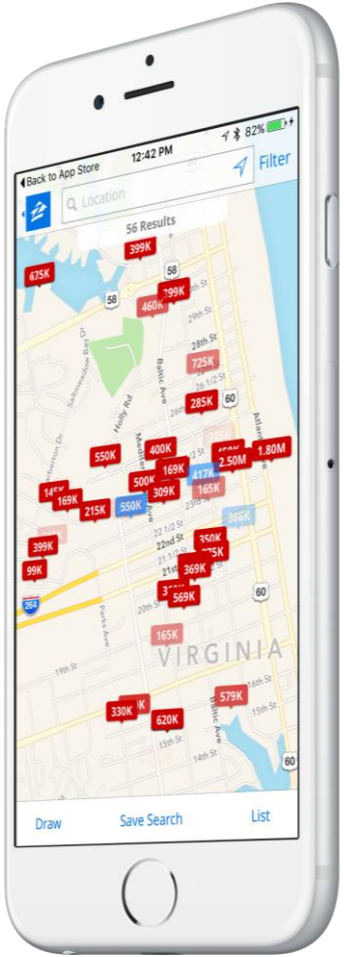
Individual Exercise

Search for points of interest

Xamarin University

Tasks

1. Perform a search for points of interest
2. Display search results on the map

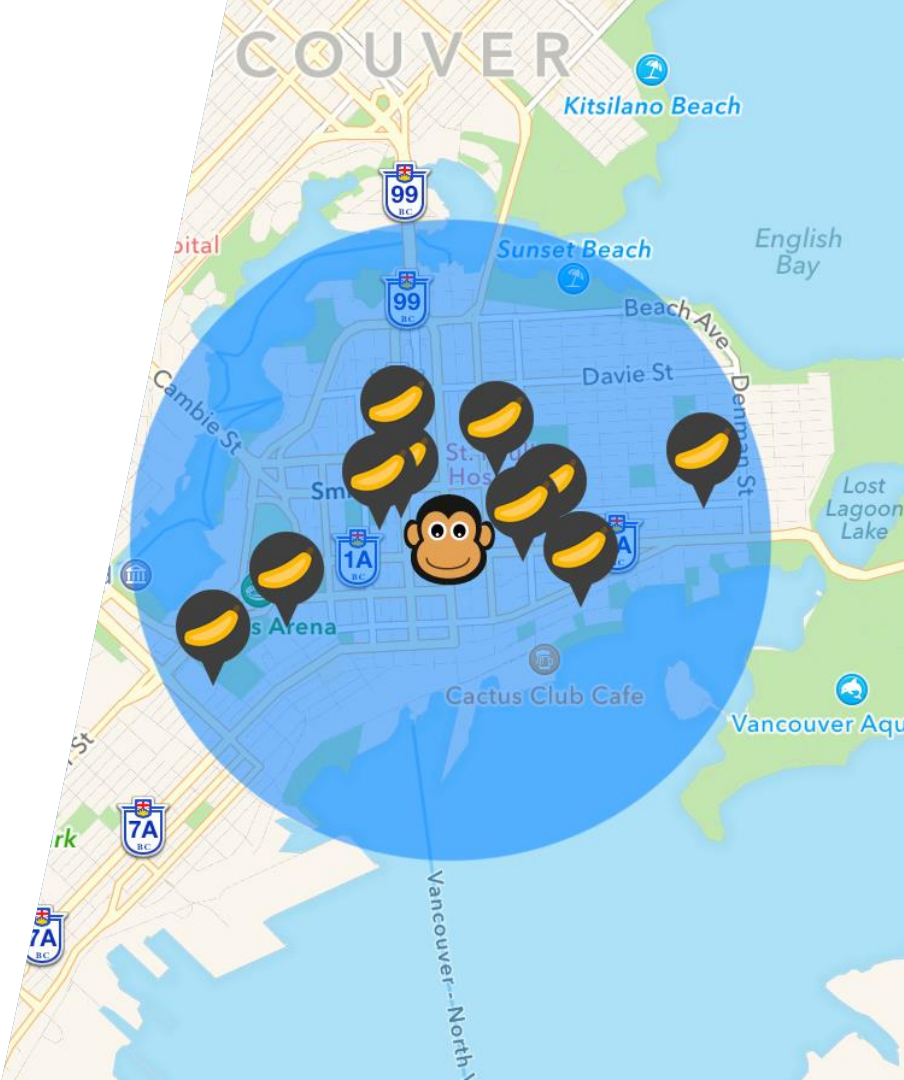


Add overlays and directions

Xamarin University

Tasks

1. Create overlays
2. Render overlays on a map
3. Calculate directions



Types of Overlays

- iOS provides support for several geometric types of overlays

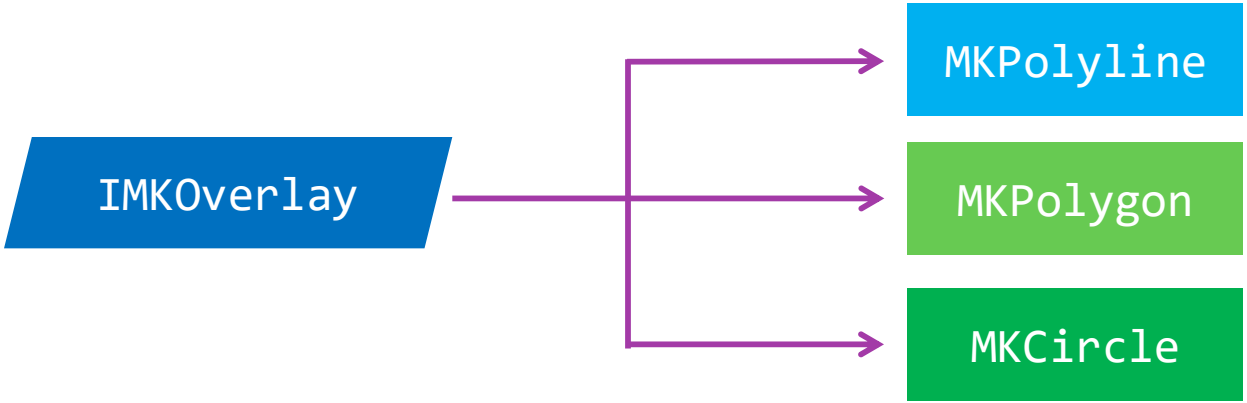
Circles – used to show a range

Polygons – used to highlight an area

Polylines – used to show routes

IMKOverlay

- **`IMKOverlay`** exposes the **`MKOverlay`** protocol, which defines a type of annotation that represents a location and an area on the map



Several implementations provided to describe specific shapes

Creating overlays

- Overlays are created using **static factory methods** that accept size and position details

MKPolygon creates a closed polygon from points or coordinates


```
var mapPoints = new CLCoordinate2D[] { ... };
MKPolygon polygonOverlay = MKPolygon.FromCoordinates(mapPoints);
```

Add Overlays to a map

- To add an overlay to the map, use the **`MKAddOverlay`** method on a **`MKMapView`**

```
var circle = MKCircle.Circle ();
circle.SetCoordinate (new CLLocationCoordinate2D (49, -123));
Circle.SetRadius(1000000);
mapView.AddOverlay(circle);
```

Rendering Overlays

- To control the visualization of an overlay, we use one of the three available renderers – each tied to an overlay type

`MKCircleRenderer`

`MKPolygonRenderer`

`MKPolylineRenderer`

OverlayRenderer

- To provide a customized renderer for an overlay, override the **`OverlayRenderer`** method in the map view delegate class

```
public override MKOverlayRenderer OverlayRenderer(
   MKMapView mapView, IMKOverlay overlay)
{
   ...
}
```

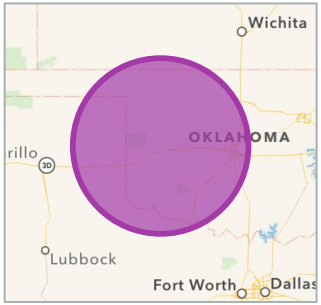
MKOverlayRenderer

- Create a renderer based on the overlay type and set the visualization properties


```
public override MKOverlayRenderer OverlayRenderer(
    MKMapView mapView, IMKOverlay overlay)
{
   if (overlay is MKCircle) {
      var renderer = new MKCircleRenderer ((MKCircle)overlay)
      { FillColor = UIColor.FromRGBA (0.0f, 0.5f, 1.0f, 0.1f) };
      return renderer;
   }
   ...
}
```

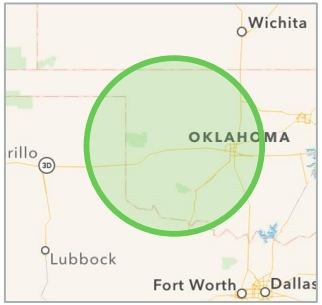
Properties on a renderer

- Customize the overlay apperance using properties inherited from `MKOverlayPathRenderer`



FillColor

StrokeColor



LineWidth

Overlays + Directions

- A common use of overlays is to plot out directions and points of interest
- MapKit has built in support for directions to get from point "A" to point "B"
- Exposed through the `MKDirections` API

MKDirectionsRequest

- **`MKDirectionsRequest`** object holds the source and destination information when requesting routing information from the **`MapKit`** APIs

```
var source = new MKMapItem (...);
var destination = new MKMapItem (...);

var request = new MKDirectionsRequest()
{
   Destination = mapItem,
   Source = source,
   RequestsAlternateRoutes = false
};
```

MKDirections

- Request turn-by-turn directions by calling **`CalculateDirections`** method on an instance of **`MKDirections`**

```
var directions = new MKDirections (request);

directions.CalculateDirections (
  (MKDirectionsResponse response, NSError e) =>
  {
     ...
  });
```

Must pass a delegate callback which will process the asynchronous response

Directions Response

- If succesful, the returned **MKDirectionsResponse** object has a **Routes** property that contains an array of **MKRoute** objects

```
var directions = new MKDirections (request);

directions.CalculateDirections (
  (MKDirectionsResponse response, NSError e) =>
  {
     if (e == null) {
        MKRoute[] routes = response.Routes;
        ...
     }
  });
```

MKRoute

- The **`MKRoute`** class defines a single route between 2 locations

Contains route data including geometry, name, and estimated travel time

Polyline is of type **`MKPolyline`** can be added to a **`MKMapView`** as an overlay

MapKit.MKRoute

- AdvisoryNotices
- ClassHandle
- Distance
- ExpectedTravelTime
- Name
- Polyline
- Steps
- TransportType

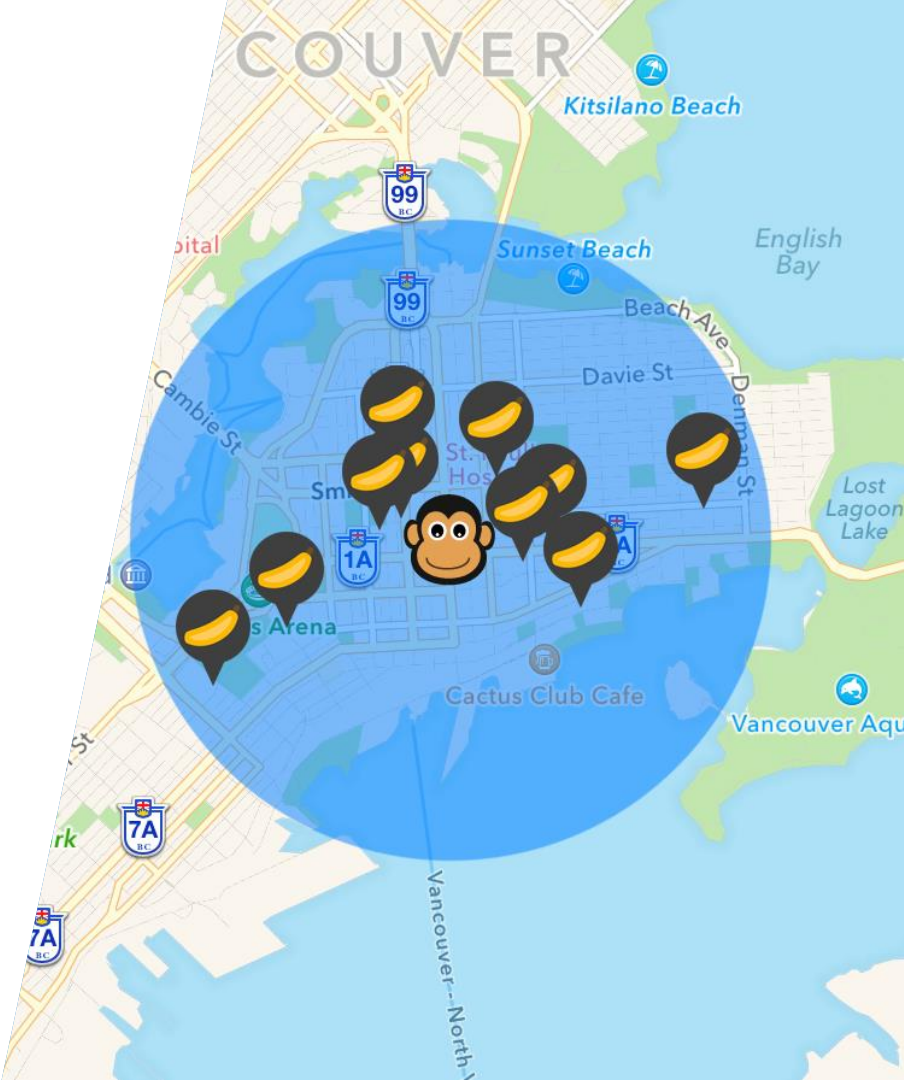
Individual Exercise

Add overlays and display directions

Xamarin University

Summary

1. Create overlays
2. Render overlays on a map
3. Calculate directions



Thank You!

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

Microsoft