ATLS 4120/5120: Mobile Application Development
Week 14: Google Location

Location Updates

Along with displaying a map you can also get the user's location, get location updates, and display locations on the map. https://developer.android.com/training/location/request-updates
FusedLocationProviderClient (Google Play services version 11.6.0 or higher)
The Fused Location Provider Client is one of the location APIs in Google Play services. It manages the underlying location technology such as analyzing GPS, Cellular and Wi-Fi network location data in order to provide the highest accuracy data. It uses different device sensors to define if a user is walking, riding a bicycle, driving a car or just standing in order to adjust the frequency of location updates. It also optimizes the device's use of battery power.
Using the FusedLocationProviderClient you can get access to the Location object to request location updates.
https://developers.google.com/android/reference/com/google/android/gms/location/LocationRequest
You can get the last location using getLastLocation() or request regular updates using requestLocationUpdates().

Permissions

https://developer.android.com/training/location/permissions
https://developer.android.com/training/permissions/requesting.html
Historically Android's permission system has been one of the biggest security concerns. Originally, permissions are asked for at install time. Once installed, the application would be able to access all of areas granted without any user's acknowledgement of what exactly the application does with that access. In Android 6.0 Marshmallow (API 23) the permission system was redesigned so apps are not granted any permission at installation time. Instead, apps have to ask users for a permission one-by-one at runtime.
Every Android app runs in a limited-access sandbox. If an app needs to use resources or information outside of its own sandbox, the app has to request the appropriate permission. You declare that your app needs a permission by listing the permission in the app manifest using a <uses-permission> element. You then must request that the user approve each permission at runtime. This permission request dialog will not launch automatically, you must call it programmatically. In the case that an app tries to call some function that requires a permission which user has not yet granted, the function will suddenly throw an exception which will lead to the application crashing.
The basic principles for requesting permissions at runtime are as follows:

- Ask for permissions in context, when the user starts to interact with the feature that requires it.
- Don't block the user. Always provide the option to cancel.
- You app should handle the case of the user denying or revoking a permission

Determine if the app was already granted permission by calling ContextCompay.checkSelfPermission()
    - PERMISSION_DENIED
    - PERMISSION_GRANTED

If permission is denied call shouldShowRequestPermissionRationale() to see if the user has previously denied the request,

- returns true if the user has not denied the request
    - present the user a UI explaining the need for this permission

- returns false if a user has denied a permission and selected the "Don't ask again" option in the permission request dialog, or if a device policy prohibits the permission.

Call requestPermissions() to request the appropriate permission for your app

- onRequestPermissionsResult() is called after the user responds to the permissions dialog and is passed the user response.
    - If permission was granted the app may use that feature
    - If permission was not granted the app should handle it gracefully
        - Don't block the UI
        - Provide information on what features will be unavailable
        - Show clear messages where the results or data would have appeared

Users are able to revoke the granted permission anytime through a device's settings so you always need to check to see if they've granted permission and request again if needed.

Devices on pre-API23 will use the old behavior.

Map

Adding location access permissions and updates to the Map app.

In your app go into Gradle Scripts and open build.gradle(Module: app) and under dependencies add the following and sync. To run on your device you might need the same version you used for maps if you changed that to match the version of Google Play Services.

implementation `'com.google.android.gms:play-services-location:17.1.0'`

Now let's update onMapReady() so we don't have the hard coded marker for Sydney and instead can show our location.

First let's change the map type to Hybrid.

```
mMap.mapType=GoogleMap.MAP_TYPE_HYBRID
```

Then comment out the Sydney marker.

Before we can access the user's location we need to ask permission for accessing their location data.

Permissions

Starting in Android 6.0 Marshmallow (API 23) the apps have to ask user for permissions at runtime.

Add the strings that we'll use in our permission explanations to strings.xml.

```
<string name="dialogTitle">Location access</string>
<string name="dialogMessage">Location access needed for this app to
show your current location</string>
<string name="dialogOK">OK</string>
<string name="dialogCancel">Cancel</string>
```

In MapsActivity.kt add a constant for our location permission and requesting permission. Can be any integer >0.

```
private val REQUEST_LOCATION_PERMISSION = 1
private val REQUEST_CHECK_SETTINGS = 2
```

Create a method to enable location by checking permissions.

```
private fun enableMyLocation() {
    if (ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) ==
PackageManager.PERMISSION_GRANTED) {
        mMap.isMyLocationEnabled = true
    }
    else {
        //permission not granted
        if (ActivityCompat.shouldShowRequestPermissionRationale(this,
android.Manifest.permission.ACCESS_FINE_LOCATION)) {
            // returns true if the app has requested this permission
previously and the user denied the request

            // Show an explanation to the user *asynchronously* --
don't block
            // this thread waiting for the user's response! After the
user
            // sees the explanation, try again to request the
permission.

            val alertDialog = AlertDialog.Builder(this)

alertDialog.setTitle(R.string.dialogTitle).setMessage(R.string.dialog
Message)
            alertDialog.apply {
                setPositiveButton(R.string.dialogOK,
DialogInterface.OnClickListener { dialog, which ->

ActivityCompat.requestPermissions(this@MapsActivity,
arrayOf(Manifest.permission.ACCESS_FINE_LOCATION),
REQUEST_LOCATION_PERMISSION)
                })
                setNegativeButton(R.string.dialogCancel,
DialogInterface.OnClickListener {dialog, which ->
                    dialog.dismiss()
                })
            }
            alertDialog.create().show()
        } else {
            ActivityCompat.requestPermissions(this,
arrayOf(Manifest.permission.ACCESS_FINE_LOCATION),
REQUEST_LOCATION_PERMISSION)
        }
    }
}
```

requestPermissions() will then pass the result with the user's response and your request code to the onRequestPermissionsResult() callback.

```
override fun onRequestPermissionsResult(requestCode: Int,
permissions: Array<String>, grantResults: IntArray) {
    when(requestCode){
        REQUEST_LOCATION_PERMISSION -> {
            if ((grantResults.isNotEmpty() && grantResults[0] ==
PackageManager.PERMISSION_GRANTED)) {
                // Permission is granted
                enableMyLocation()
            } else {
                Snackbar.make(findViewById(R.id.map), "Location
access denied", Snackbar.LENGTH_SHORT).show()
            }
        }
    //add other statements to handle other permissions your app might
request
    }
}
```

To test permissions first deny access, you should see the permissions denied Snackbar.
Then run the app again and you should get the AlertDialog (because access was previously denied). Test both cancel and OK.
Then to allow access either choose OK or uninstall the app and then install it again and approve the permission. Once you've granted permissions you won't be prompted again unless you go into Settings | Apps | Your App | Permissions and turn off location permission (or uninstall and reinstall the app on the device).

Remember that requestPermissions() will only be called in Android 6 or later.

Location Request

Define some class level variables for our location information.

```
// FusedLocationProviderClient - Main class for receiving location
updates
private lateinit var fusedLocationClient: FusedLocationProviderClient

// LocationRequest - Requirements for location updates
private lateinit var locationRequest: LocationRequest

// LocationCallback - Called when FusedLocationProviderClient has a
new Location
private lateinit var locationCallback: LocationCallback

//last known location
private var currentLocation: Location? = null

//marker
private var marker: Marker? = null
```

Create the following method to create a location request.

```
private fun createLocationRequest(){
    // initialize LocationRequest object
    locationRequest = LocationRequest()
    //set the desired interval for location updates, in milliseconds
    locationRequest.interval = 10000
    //set the fastest interval for location updates, in milliseconds
    locationRequest.fastestInterval = 500
    //set priority of the request
    locationRequest.priority = LocationRequest.PRIORITY_HIGH_ACCURACY

    // create LocationSettingsRequest object using location request
object
    val builder =
LocationSettingsRequest.Builder().addLocationRequest(locationRequest)

    // create a settings client and a task to check location
settings.
    val settingsClient = LocationServices.getSettingsClient(this)
    val task = settingsClient.checkLocationSettings(builder.build())

    //start location updates if successful
    task.addOnSuccessListener {
        startLocationUpdates()
    }
    task.addOnFailureListener { e ->
        // failure means the location settings have some issues
        if (e is ResolvableApiException) {
            // Location settings are not satisfied
            // prompt user to enable location in Settings
            try {
                // prompts user to enable location permission
                e.startResolutionForResult(this,
REQUEST_CHECK_SETTINGS)
            } catch (sendEx: IntentSender.SendIntentException) {
                // Ignore the error
            }
        }
    }
}
```

startLocationUpdates() will give you an error because we haven't created it yet.

We will call createLocationRequest() from enableMyLocation() after we enable location.

```
private fun enableMyLocation() {
    if (ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) ==
PackageManager.PERMISSION_GRANTED) {
        mMap.isMyLocationEnabled = true
```

```
        createLocationRequest()
    }
    else ...
```

You can stop monitoring for location changes by calling removeLocationUpdates(locationCallback).

Location Changes

Now let's create startLocationUpdates() where we'll handle location changes.

```
private fun startLocationUpdates(){
    if (mMap.isMyLocationEnabled) {
        //initialize the FusedLocationProviderClient
        fusedLocationClient =
LocationServices.getFusedLocationProviderClient(this)
        //define location callback
        locationCallback = object : LocationCallback() {
            override fun onLocationResult(locationResult:
LocationResult) {
                super.onLocationResult(locationResult)
                onLocationChanged(locationResult.lastLocation)
            }
        }
        //request location updates
        fusedLocationClient.requestLocationUpdates(locationRequest,
locationCallback, Looper.getMainLooper())
    }
}
```

This will give you a warning about asking permission but we already have so you can suppress that warning.

onLocationChanged() will give you an error because we haven't created it yet so let's do that next.

onLocationChanged() will be called every time there is a location update and onLocationResult gets called.

```
private fun onLocationChanged(location: Location){
    if (location != null) {
        currentLocation = location
        //define an object of the Google LatLng class with location
coordinates
        val currentLatLng = LatLng(location.latitude,
location.longitude)
        //check to see if there's a current marker
        if (marker == null){
            //Create a MarkerOptions object
            var markerOptions = MarkerOptions()
            //set the position to the user's current location
            markerOptions.position(currentLatLng)
            //set the title
            markerOptions.title("you are here")
            //add marker to the map using the marker options and
```

```
assign to our marker
                marker = mMap.addMarker(markerOptions)
            } else {
                //change the position of the current marker
                marker!!.position = currentLatLng
            }
            //move the map camera to the new position and set the zoom

mMap.animateCamera(CameraUpdateFactory.newLatLngZoom(currentLatLng,
15f))
        }
}
```

Zoom level 0 corresponds to the fully zoomed-out world view. Most areas support zoom levels up to 20, while more remote areas only support zoom levels up to 13. A zoom level of 11 is a nice in-between value that shows enough detail without getting crazy-close.

You can also use custom pins or use the Geocoder class to show the location's address.
The marker can be configured as well as set up to be dragged or clicked.
We create only one marker and move it but you can have as many markers as you want by just creating new markers for each location.
You can remove markers by calling remove() or clear the whole map with clear()
https://developers.google.com/android/reference/com/google/android/gms/maps/model/Marker