Lab 6 - Location Sensing

This lab we are going to introduce the localisation of a mobile user. You will create an application that uses Location API (from Google Play Services) to infer a user's location, and show the result back to the user.

NOTE: In 2017 Google Play Services 11.0.0 introduced a new way to access LocationServices. This simplifies the way to get a device's location and reduces chances for null exceptions. In this lab we will concentrate on this new method. You might find different ways to access location in online tutorials.

Showing a Map

Google Play Services are a powerful Android library provided by Google that helps with tasks related to Google Maps, Google Drive, and many other purposes. To use the services, you need to add them to your Android Studio (SDK manager -> SDK tools -> Google Play Services) and then to the gradle file (`implementation 'com.google.android.gms:play-services-maps:18.0.2'`, or whatever version you have installed, to your dependencies). To simplify this, and since we are going to use a map in this assignment, let's just start a new project with a brand **new Google Maps Activity**. Android Studio creates boilerplate code for us. Copy-paste the link that's shown to you in Android Studio (**google_maps_api.xml**) to obtain the key you need to paste in, for Google to allow your app to access its services.

If you haven't modified the defaults, you have **activity_maps** layout, so open it and let's edit it. At the moment you have a single **SupportMapFragment** here. Create a new **LinearLayoutCompat** and put it around the **Fragment**. In addition, add two **TextViews** and a **Button** after the fragment, so that your layout looks something like this:

```
<androidx.appcompat.widget.LinearLayoutCompat...
  <Fragment...
  <TextView...
  <TextView…
  <Button…
</androidx.appcompat.widget.LinearLayoutCompat>
```

The end goal is to have a screen with a map, and two TextViews that will show a user's latitude and longitude, and a button that will query for the user's location. To fit all these into one screen we need to modify the layout parameters. First, we need to set the LinearLayoutCompat's `android:orientation` to "`vertical`". Then we will set `android:layout_width` of the Fragment, TextViews and Button to "`match_parent`", extending to fit the screen. We will set `android:layout_height` of the TextViews and the Button to "`wrap_content`" so they are as tall as

the letters are. Finally, we will set `android:layout_height` of the Fragment to `"0dp"`, and its `android:layout_weight` to `"1"`. This will tell the View System to span the Fragment height to whatever is left of the screen after the TextViews and the Button are drawn. Needless to say, give your Views nice descriptive IDs and set Button text to a String `"Get location"`

This is a good moment to stop and check if everything compiles and runs on your phone/emulator. The most common errors stem from the incompatibility between Google Play Services version installed on the phone/emulator and the one used for compilation. You can check your Google Play services version on the phone/emulator in Settings->Apps->Google Play Services.

Getting Location Information

Google play services location API is the preferred way of obtaining a user's location. Not only do these services abstract the tedious work of turning different sensors on and off, but they also ensure that the energy spent for getting the location is minimized. The services manage location querying across applications, thus if one application requested a user's location, another application, requesting the same info immediately after, will be served the already queried result, and therefore save time and energy for querying the sensors again.

To use the services, just like with the Google Maps API, you need to add them to your Android Studio (SDK manager -> SDK tools -> Google Play Services) and then to the gradle file (`implementation 'com.google.android.gms:play-services-location:21.0.1'`, or whatever version you have installed, to your dependencies).

We are going to access the location info by connecting to the services in our MapsActivity. Open the activity and add:

```
private lateinit var fusedLocationClient: FusedLocationProviderClient
```

This is the key object that will help us interact with Google Play Location Services. Instantiate it in `onCreate()` method:

```
fusedLocationClient = LocationServices.getFusedLocationProviderClient(this)
```

We want to show the last known location on the map. For this, create a new function `showLastKnownLocation()`. FusedLocationProviderClient's `lastLocation` method returns a Task that you can use to get a Location object with the latitude and longitude coordinates of a geographic location, the best most recent location currently available. A really easy way for asynchronous location querying! Add it to `showLastKnownLocation()`:

```
fusedLocationClient.lastLocation
        .addOnSuccessListener { location : Location? ->
            // Got last known location. In some rare situations this can be null.
        }
```

The resulting Location object contains the latitude and longitude of the sensed location. Once the result is ready, we can show it as a marker on the map, together with the exact time when the location was taken. After making sure that location is not null, do the following:

```
mMap.addMarker(
        MarkerOptions()
            .position(LatLng(location.latitude, location.longitude))
```

```
.title(LocalDateTime.now().format(DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT, FormatStyle.SHORT)))
    )
```

Also here, you should center the map to the newly added maker, with a chosen zoom level:

```
val zoom_level = 15.0F

mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(LatLng(location.latitude, location.longitude), zoom_level))
```

Finally, set the latitude and longitude TextViews to the values from the `location` object.

Permissions

By now you have probably noticed that we haven't asked the user for a permission, yet. Beginning with Android 6.0 (API 23) users grant permissions when the app is running, before that, they would grant the permission only at the time the app is installed. This gives more control to the user to decide which permissions to grant. Unfortunately, this requires a bit more effort from the developer's side.

First, make sure that both ACCESS_FINE_LOCATION and ACCESS_COARSE_LOCATION are listed as required in the Manifest.

We are going to handle the permission requests as follow:

- In showLastKnownLocation() check whether the user has already given a permission
    - If not, check whether a user should be shown a rationale
        - If yes, show the rationale and ask for the permission if the user agrees to the rationale
        - If not, just ask for the permission
    - If yes, continue with location sensing
- When the request results are received, check whether the answer is "PERMISSION_GRANTED" and if so, call showLastKnownLocation() again

Let's go step by step. At the beginning of `showLastKnownLocation()` put a check, to see if permissions for fine and coarse location are not given yet, and if so, request them from the user:

```
if (ActivityCompat.checkSelfPermission(
        this,
        Manifest.permission.ACCESS_FINE_LOCATION
    ) != PackageManager.PERMISSION_GRANTED || ActivityCompat.checkSelfPermission(
        this,
        Manifest.permission.ACCESS_COARSE_LOCATION
    ) != PackageManager.PERMISSION_GRANTED
)
{
```

If the user needs to be shown an explanation of why a permission is necessary, display a Snackbar with the description (something like "To show map markers, this app needs your location information").
Note, you have to compile the Material Design library in order to use the Snackbar. In the build.gradle file add: `implementation 'com.google.android.material:material:1.3.0'` (or whatever version you're using).

```
if (ActivityCompat.shouldShowRequestPermissionRationale(this, Manifest.permission
.ACCESS_COARSE_LOCATION) || ActivityCompat.shouldShowRequestPermissionRationale (this,
Manifest.permission.ACCESS_FINE_LOCATION)) {
   Snackbar.make(
       binding.root,
       R.string.permission_location_rationale,
       Snackbar.LENGTH_INDEFINITE
   )
       .setAction(R.string.ok) { // If the user agrees with the Snackbar, proceed
with asking for the permissions:
           ActivityCompat.requestPermissions(
               this@MapsActivity, arrayOf(
                   Manifest.permission.ACCESS_COARSE_LOCATION,
                   Manifest.permission.ACCESS_FINE_LOCATION
               ),
               REQUEST_ID_LOCATION_PERMISSIONS
           )
       }.show()
}
```

`REQUEST_ID_LOCATION_PERMISSIONS` is an internal code (companion object private val) we use to discern among different requests.

The permission request result will be available through a `onRequestPermissionResult` method of `ActivityCompat.OnRequestPermissionsResultCallback` interface. So, ensure that your MapsActivity class implements the interface and overrides the method. If the user has granted the permission (you will see that `PackageManager.PERMISSION_GRANTED` is in `grantResults`), go ahead and call `showLastKnownLocation()` again. Finally, don't forget to connect the button with the `showLastKnownLocation()` function.

Time to test your app! First, make sure your emulator device is "wiped clean". Install the app and play with the location reported by the emulator ("..." icon on the emulator right-hand side). Click on "Get location" in the app - Does the app get the new location?

Remember, we are using the `fusedLocationClient.lastLocation` method. This method provides a simplified way to get location, without incurring any additional "costs" - basically, it re-uses the latest previously sensed location. It is particularly well suited for applications that do not require an accurate location and that do not want to maintain extra logic for location updates. Hence, this method might return a null location, if the device never recorded its location previously, which could be the case of a new device or a device that has been restored to factory settings.

Next, let's open the Google Maps app on the emulator, and use it to find the device's current location (click on the "My location" button to center the map on the current location). Now, switch to your app and click "Get location". You should be able to see the newly added marker at the current location. Play some more with the app and emulator location controls, to add several markers.

Click on the markers to see the update times. You might realize that it makes sense to tweak it a bit, for example, add map zoom controls. More info here:
https://developers.google.com/maps/documentation/android-sdk/controls

If you did not attend the lab at FRI you can commit your solution to a private repository named **PBD2023-LAB-6** in your Bitbucket account. User **pbdfrita** must be added as a read only member of this repository. The code must be committed by Sunday (April 16th) 23:59.

Happy coding!