Geolocalización y mapas

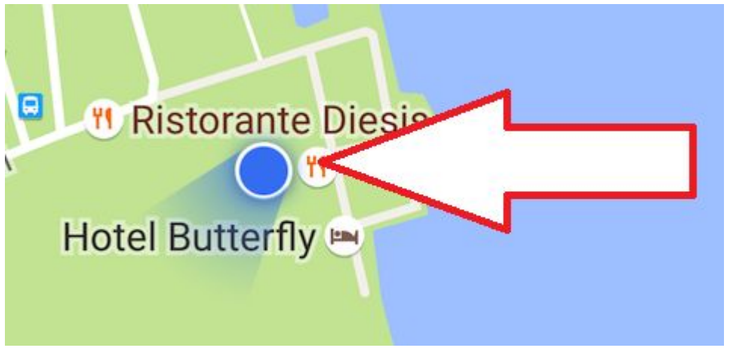
Indice

- Obtener geolocalización
- LocationServices
- Google Maps
- Dibujar en el mapa: marker, polyline, polygon
- Añadir información a los puntos en el mapa: InfoWindow

Obtener geolocalización

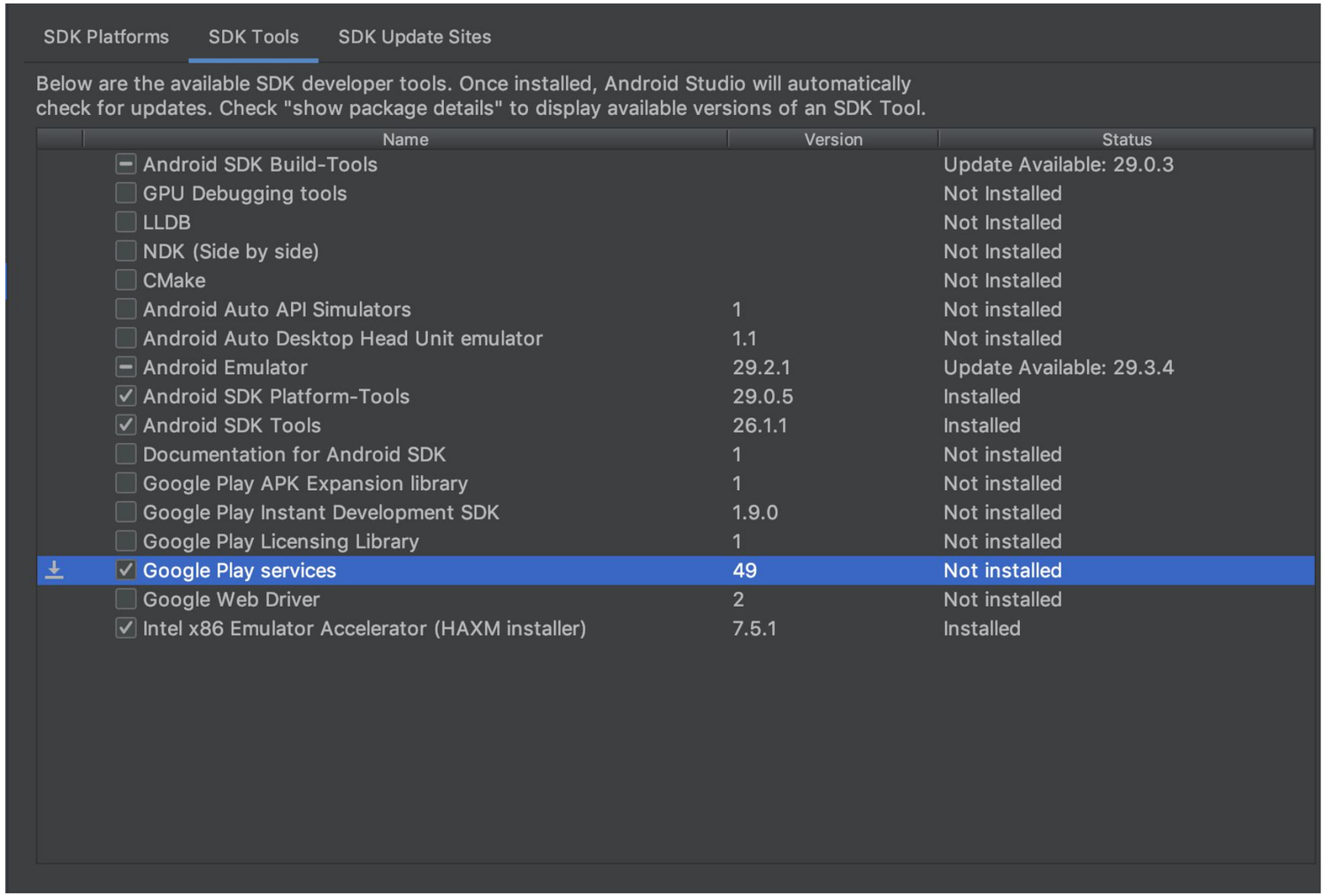
Conceptos:

- Latitud
- Longitud
- Altura
- Bearing
- Velocity



Obtener geolocalización

Para poder utilizar geolocalización y mapas en nuestras apps, debemos descargarnos el SDK de **Google Play services** con el SDK Manager en Android Studio



Obtener geolocalización

Por otro lado, deberemos también incluir en nuestros archivos build.gradle

- ***build.gradle*** top level

```
allprojects {
    repositories {
        google()
        jcenter()

    }
}
```

- ***build.gradle*** del módulo en el que vamos a usar geolocalización y mapas

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation"org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.core:core-ktx:1.2.0'
    implementation 'com.google.android.gms:play-services-maps:16.1.0'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
}
```

Permisos necesarios

Hay que añadir en el gradle uno de los siguientes permisos:

- ACCESS_COARSE_LOCATION
- ACCESS_FINE_LOCATION
- ACCESS_BACKGROUND_LOCATION (API 29)

Location

Clase que representa una localización geográfica

Propiedades:

- latitude
- logitude
- timestamp
- bearing
- altitude
- velocity

Todas las locations obtenidas vía LocationManager garantizan tener una latitud, longitud y timestamp válidos

Obtener última localización

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.google.android.gms.location.sample.basiclocationsample" >

    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
</manifest>
```

```
val fusedLocationClient = LocationServices.getFusedLocationProviderClient(activity)
fusedLocationClient.lastLocation
    .addOnSuccessListener { location : Location? ->
        //Obtener la última localización (puede ser null)
    }
```

Preferencias LocationServices

LocationRequest

setInterval() -> setea cada cuanto tiempo vamos a solicitar la localización

setFastestInterval() -> setea el intervalo al intervalo más corto en el que el dispositivo pueda obtener la localización

setPriority() -> establece prioridad del request

PRIORITY_BALANCED_POWER_ACCURACY -> Precision de aproximadamente 100 metros, utilizando en mayor medida Wifi y red móvil. Bajo consumo de batería.

PRIORITY_HIGH_ACCURACY -> La mayor precisión posible. Los LocationServices usarán el GPS en la medida de lo posibe

PRIORITY_LOW_POWER -> Precisión aproximada de 10 kilómetros. Bajo consumo de batería.

PRIORITY_NO_POWER -> Nuestra app no solicitará ninguna actualización de la localización (no impactará en la batería del usuario, ni los recursos utilizados) pero recibirá la localización cada vez que otra aplicación la solicite.

Preferencias LocationServices

```
fun createLocationRequest() {
    val locationRequest = LocationRequest.create()?.apply {
        interval = 10000
        fastestInterval = 5000
        priority = LocationRequest.PRIORITY_HIGH_ACCURACY
    }
}

val builder = LocationSettingsRequest.Builder()
    .addLocationRequest(createLocationRequest())
```

Consultar preferencias

```
val client: SettingsClient = LocationServices.getSettingsClient(this)
val task: Task<LocationSettingsResponse> = client.checkLocationSettings(builder.build())
task.addOnSuccessListener {
   it.locationSettingsStates
}
```

Obtener localizaciones periódicas

```
private fun startLocationUpdates() {
val fusedLocationClient = LocationServices.getFusedLocationProviderClient(activity)
   fusedLocationClient.requestLocationUpdates(locationRequest,
       locationCallback,
       Looper.getMainLooper())
}

val locationCallback = object : LocationCallback() {
   override fun onLocationResult(locationResult: LocationResult?) {
       locationResult ?: return
       for (location in locationResult.locations){
           // Procesar locations recibidas
       }
   }
}

private fun stopLocationUpdates() {
   fusedLocationClient.removeLocationUpdates(locationCallback)
}
```

Obtener una dirección (reverse geocoding)

- ***Geocoder*** nos permite tanto obtener coordenadas a partir de una dirección como obtener la dirección a partir de las coordenadas (reverse geocoding)

- La sugerencia de Google para usar ***Geocoder*** es crear un servicio que extienda de ***IntentService*** (ya que es una task asíncrona que puede tomar mucho tiempo)

Servicio Geocoder

```
class FetchAddressIntentService : IntentService("FetchAddress") {
    private var receiver: ResultReceiver? = null

    override fun onHandleIntent(intent: Intent?) {
        var errorMessage = ""
        receiver = intent?.getParcelableExtra(Constants.RECEIVER)
        val location =
intent.getParcelableExtra<Location>(Constants.LOCATION_DATA_EXTRA)
        val geocoder = Geocoder(this, Locale.getDefault())
        var addresses: List<Address> = emptyList()
        try {
            addresses = geocoder.getFromLocation(location.latitude,
location.longitude, 1)
        } catch (ioException: IOException) {
            errorMessage = getString(R.string.service_not_available)
        } catch (illegalArgumentException: IllegalArgumentException) {
            errorMessage = getString(R.string.invalid_lat_long_used)
        }
```

```
        if (addresses.isEmpty()) {
            if (errorMessage.isEmpty()) {
                errorMessage = getString(R.string.no_address_found)
                Log.e(TAG, errorMessage)
            }
            deliverResultToReceiver(Constants.FAILURE_RESULT, errorMessage)
        } else {
            val address = addresses[0]
            val addressFragments = with(address) {
                (0..maxAddressLineIndex).map { getAddressLine(it) }
            }
            deliverResultToReceiver(Constants.SUCCESS_RESULT,
                addressFragments.joinToString(separator = "\n"))
        }
    }

    private fun deliverResultToReceiver(resultCode: Int, message: String) {
        val bundle = Bundle().apply { putString(Constants.RESULT_DATA_KEY, message) }
        receiver?.send(resultCode, bundle)
    }
}
```

Lanzar servicio y procesar resultado

```
private inner class AddressResultReceiver internal constructor(
    handler: Handler
) : ResultReceiver(handler) {

    override fun onReceiveResult(resultCode: Int, resultData: Bundle) {
        addressOutput = resultData.getString(Constants.RESULT_DATA_KEY)
        displayAddressOutput()
        if (resultCode == Constants.SUCCESS_RESULT) {
            //Tenemos resultado!
        }
    }

}
```

```
val resultReceiver = AddressResultReceiver(Handler())
fun startIntentService() {
    val intent = Intent(this, FetchAddressIntentService::class.java).apply { \
        putExtra(Constants.RECEIVER, resultReceiver)
        putExtra(Constants.LOCATION_DATA_EXTRA, lastLocation)
    }
    startService(intent)
}
```

Mapas

Necesitaremos usar Maps SDK para Android

Para ello, necesitamos obtener una API key (instrucciones completas: https://developers.google.com/maps/documentation/android-sdk/get-api-key)

API keys

Name	Creation date	Restrictions	Key	Usage with all services (last 30 days)
API key 1	15 Feb 2020	Android apps	AIzaSyBrlA...mlJ8BD47hY	0

Maps SDK

Una vez obtenida, debe ser incluida en el Manifest:

```
<meta-data
   android:name="com.google.android.geo.API_KEY"
   android:value="YOUR_API_KEY"/>
```

Componentes

Se pueden utilizar dos componentes para crear un mapa:

- MapFragment (subclase de Fragment)
- MapView (subclase de View)

Una vez creados (tanto el fragment como la vista) se puede obtener el objeto Map

```
mapFragment.getMapAsync(callback: OnMapReadyCallback)
```

Propiedades Map

MapType: Tipo de mapa (Normal, Hybrid, Satellite, Terrain, None)

TrafficEnabled: Muestra el tráfico

MapStyle: Utiliza un objeto de la clase MapStyleOptions para definir el estilo del mapa (se puede crear un estilo con: https://mapstyle.withgoogle.com/)

SetMyLocationEnabled: Muestra el botón my location

SetPadding: Establece el área en la que se puede interactuar con el mapa

Preferencias UI

```
val uiSettings = mMap.uiSettings
uiSettings.isZoomControlsEnabled = true
uiSettings.isCompassEnabled = true
uiSettings.isMyLocationButtonEnabled = true
uiSettings.isScrollGesturesEnabled = true
uiSettings.isZoomGesturesEnabled = true
uiSettings.isTiltGesturesEnabled = true
uiSettings.isRotateGesturesEnabled = true
```

Inicializar mapa

```
class MapsActivity : AppCompatActivity(), OnMapReadyCallback {

    private lateinit var mMap: GoogleMap

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_maps)
        val mapFragment = supportFragmentManager
            .findFragmentById(R.id.map) as SupportMapFragment
        mapFragment.getMapAsync(this)
    }

    override fun onMapReady(googleMap: GoogleMap) {
        mMap = googleMap
        //Ya podemos trabajar con el mapa!
    }
}
```

Añadir marker

```
val specialPlace = LatLng(48.858315, 2.294398)
mMap.addMarker(MarkerOptions().position(specialPlace).title("Lugar especial"))
```

```
mMap.moveCamera(CameraUpdateFactory.newLatLng(specialPlace))
```

Polyline

```
val polyline: Polyline = mMap.addPolyline(
   PolylineOptions()
       .clickable(true)
       .add(
           LatLng(-35.016, 143.321),
           LatLng(-34.747, 145.592),
           LatLng(-34.364, 147.891),
           LatLng(-33.501, 150.217),
           LatLng(-32.306, 149.248),
           LatLng(-32.491, 147.309)
       )
)
```

Propiedades Polyline

Points: Vértices de la línea

Width: Ancho de la línea

Color: Color de la línea (ARGB)

Start/end cap: Forma que tendrán los extremos de la línea (ButtCap, SquareCap, RoundCap, CustomCap)

Stroke pattern: Solid (por defecto) o PatternItem que define la forma de la línea (Gap, Dash, Dot)

Z-Index: Cuando el mapa tiene mas de un elemento dibujado, Z-index indicara distancia en el eje Z

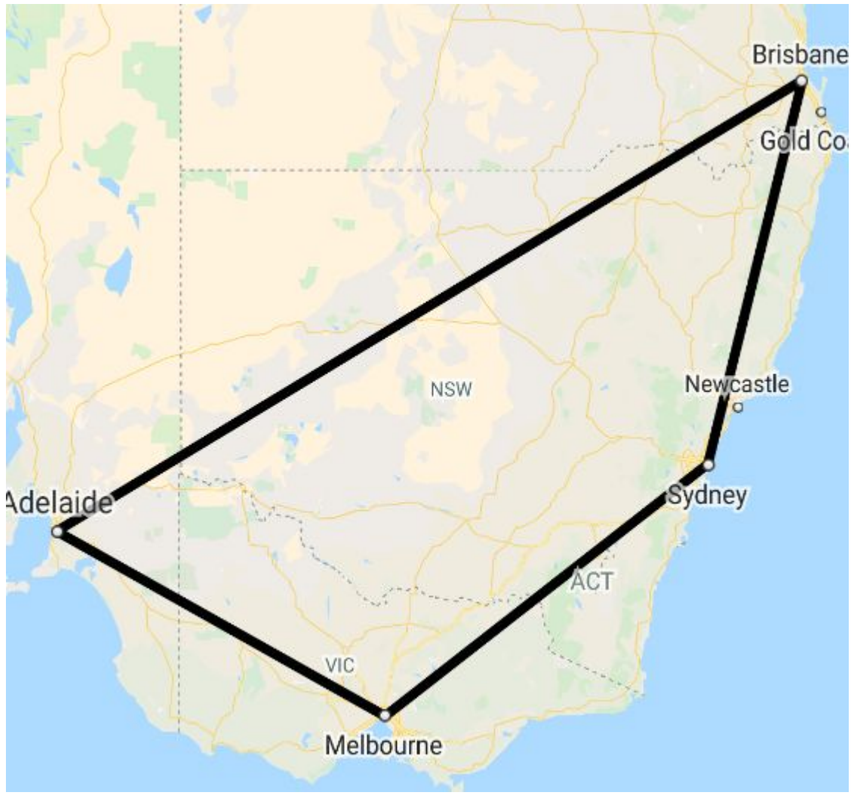
Visibility: Indica si la línea es visible o no

Clickability: Define si el polyline es clickable (se debe implementar y asignar OnPolylineClickListener)

Tag: Objeto asociado con el polyline para identificarlo

Polygon

```
val polygon: Polygon = mMap.addPolygon(
   PolygonOptions()
       .clickable(true)
       .add(
           LatLng(-27.457, 153.040),
           LatLng(-33.852, 151.211),
           LatLng(-37.813, 144.962),
           LatLng(-34.928, 138.599)
       )
)
```



Propiedades Polygon

Stroke Width: Ancho de la línea

Stroke color: Color de la línea (ARGB)

Stroke pattern: Solid (por defecto) o PatternItem que define la forma de la línea (Gap, Dash, Dot)

Fill Color: Color de relleno (ARGB)

Z-Index: Cuando el mapa tiene mas de un elemento dibujado, Z-index indicara distancia en el eje Z

Visibility: Indica si la línea es visible o no

Clickability: Define si el polyline es clickable (se debe implementar y asignar OnPolylineClickListener)

Tag: Objeto asociado con el polyline para identificarlo

InfoWindow

Permite ampliar la información mostrada por el marker

Por defecto, se puede personalizar el título (y existe otro campo llamado *snippet*, que podemos usar para lo que queramos)

También podemos usar una vista personalizada

GRACIAS

www.keepcoding.io