Práctica 5:
Servicios basados en la localización

Introducción y Objetivos

El objetivo de la práctica es implementar una app móvil cuyo funcionamiento dependa de la localización del usuario. En concreto, se creará un cliente que pueda *(i)* descargar y mostrar fotos de Flick próximas a la localización del usuario y *(ii)* mostrar en un Google Map dicha localización.

La interfaz de la de app ha de disponer (al menos) de un botón, una imagen y un Google Map. Cuando el usuario pulse el botón, se ha de obtener la localización actual, y con la misma buscar en Flickr fotos recientes próximas a la localización. La imagen de la interfaz mostrará una de ellas (por ejemplo, la más reciente) y se mostrará también por pantalla información adicional sobre la localización (latitud y longitud) y, opcionalmente, otra información (como por ejemplo el título). Por último, se actualizará el mapa centrándolo en la ubicación del usuario y añadiendo un marcador. El mapa puede complicarse tanto como se quiere mostrando también la ubicación de la imagen.

Obtención de localización y *Google Play Services*

Android proporciona una API para localización, en el paquete `android.location`. Esta API permite obtener datos de localización de una serie de fuentes distintas: en general, se pueden obtener datos precisos de localización del GPS, o datos (menos precisos) de conexiones WiFi o de las estaciones de la red de teléfono. Sin embargo, a pesar de que se puede utilizar esta API para obtener la localización, tiene un inconveniente: conforme el usuario se mueve, y/o su contexto cambia, la fuente de datos sobre localización a consultar cambia. Así, si el usuario está al aire libre el GPS puede ser la mejor opción, pero si está dentro de un edificio o no hay señal GPS, puede que la torre telefónica sea mejor, o incluso si no hay ninguna señal disponible el acelerómetro y/o el giróscopo del teléfono proporcionarán información, aunque limitada, sobre localización.

La programación de la app para que vaya cambiando de fuente de datos según la situación puede ser compleja y tediosa. Alternativamente, y por medio de *Google Play Services*, podemos obtener la localización sin preocuparnos por gestionar el origen de la misma, en concreto mediante la API *Fused Location Provider*, que integra las diferentes fuentes de información mencionadas para devolvernos la localización más precisa y con un menor consumo de energía.

Google Play Services: configuración y permisos. Las instrucciones y descripciones que siguen están testeadas en la versión 7.3.0. Aunque re-utilizaremos código de otras prácticas, se recomienda crear un nuevo proyecto desde cero para esta práctica. El primer paso es añadir a las dependencias del proyecto la API de localización de Google Play Services. Puedes hacerlo en `File -> Project Structure` o editando el fichero `build.gradle`. Añade: `com.google.android.gms:play-services-location:núm_version`. También tendremos que añadir permisos para poder obtener la localización (`ACCESS_FINE_LOCATION` y `ACCESS_COARSE_LOCATION`) y acceder a Internet.

Creación de un cliente. Para utilizar *Google Play Services* necesitamos un cliente, una instancia de `FusedLocationProviderClient`. Para crearlo, típicamente en `onCreate`, se invoca a `LocationServices.getFusedLocationProviderClient` pasando como parámetro la actividad o el contexto:

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

Siguiendo la recomendación de Google, conectaremos al cliente en `onStart()`, invocando `connect()`, y lo desconectaremos en `onStop()`, utilizando `disconnect()`. Otro método de interés de `GoogleApiClient` es `isConnected()`, ya que solo tendrá sentido intentar obtener una localización al pulsar el botón si el cliente está conectado al servicio.

Opcionalmente (no es necesario realizarlo para esta práctica), Google recomienda verificar que *Play Services* está disponible. Un modo de hacerlo es crear un `OnConnectionFailedListener` y asociarlo a nuestro cliente, e implementar `onConnectionFailed()`, que es el método al que se llama si falla la conexión al cliente. Alternativamente, se puede utilizar, dentro de `onResume()`, el método `isGooglePlayServicesAvailable()` de `GoogleApiAvailability` y obtener un `Dialog` de error si el resultado es diferente de SUCCESS. Para más información, visitad http://developers.google.com/android/guides/setup#ensure_devices_have_the_google_play_services_apk y http://developers.google.com/android/guides/api-client#handle_connection_failures.

Creación de una solicitud para obtener una posición. Una vez que tenemos el cliente, para obtener una posición (objeto de clase `Location`) podemos crear una solicitud a través del método `getLastLocation`:

```
fusedLocationClient.getLastLocation().addOnSuccessListener(this,
      new OnSuccessListener<Location>() {
            @Override
            public void onSuccess(Location location) {
                  if (location != null) {
                        Double latit = location.getLatitude();
                        Double longi = location.getLongitude();
                  }
            }
      }
);
```

Para obtener actualizaciones en la posición, se puede usar el método `requestLocationUpdates()`. En primer lugar, se crea una solicitud de información sobre la posición a través de un objeto de clase `LocationRequest`. Podemos configurar la prioridad (`setPriority()`, hay una serie de constantes para ello), el número de actualizaciones (`setNumUpdates()`) y el intervalo de las mismas (`setInterval()`). Dado que queremos una única posición, cuando pulsemos el botón el número de actualizaciones será uno y el intervalo cero:

```
LocationRequest locationRequest = new LocationRequest();
locationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
locationRequest.setNumUpdates(1);
locationRequest.setInterval(0);
fusedLocationClient.requestLocationUpdates(locationRequest,
  new LocationCallback() {
      @Override
      public void onLocationResult(LocationResult result) {
            if (result == null)
                  return;
            int lastIndex = result.getLocations().size() - 1;
            Location location = result.getLocations().get(lastIndex);
            ...
      }
  },
  Looper.getMainLooper()
);
```

Con esto, podemos tener un método que cree y configure una solicitud de posición (una `Location`), y que al obtenerla ejecute cierto código. En nuestro caso, obviamente, lo que ha de hacer es utilizar esa posición para hacer otra solicitud a Flickr para obtener una imagen; veremos a continuación cómo añadir restricciones de posición a las solicitudes a Flickr. Antes de seguir, sólo queda aclarar que el mencionado método lo habremos de invocar cuando se interactúe con la interfaz.

Solicitudes a Flickr con información de posición

Para obtener fotos que cumplan ciertos criterios de búsqueda Flickr provee el método `flickr.photos.search`. Si miráis la documentación del método veréis todos los posibles argumentos para especificar dichos criterios. A nosotros nos interesan `lat` y `lon`, utilizados para especificar latitud y longitud, respectivamente. Flickr recomienda pasar argumentos adicionales para limitar más la búsqueda; si no, busca por defecto "fotos añadidas en las últimas doce horas".

Para construir la URL de la solicitud utilizad la clase `Uri` e id añadiendo los diferentes parámetros a la URL base (http://api.flickr.com/services/rest/). Llamaremos primero a `Uri.parse()` pasándole el `String` del que queramos construir la `Uri`; con el `Uri` resultante de dicha llamada llamaremos a `buildUpon()` para obtener un `Uri.Builder` que nos ayuda a construir `Uri`s fácilmente.
Podemos utilizar `appendQueryParameter()` para añadir parámetros y `build()` para construir la `Uri` con los atributos que hayamos especificado, como por ejemplo:

```
Uri queryFlickr = Uri.parse("https://api.flickr.com/services/rest/").
              buildUpon().appendQueryParameter("api_key", API_KEY).build();
```

Por último, podéis construir las URLs por partes como se vio en prácticas anteriores o añadir un parámetro *"extra"* con valor *"url_s"* para que las devuelva. Para algunas fotos Flickr no devolverá la URL, así que ignoradlas o construid la URL manualmente.

Conexión para obtener las fotos según la posición. Igual que en prácticas anteriores, la solicitud a Flickr implica una conexión que se ha de realizar en una hebra diferente a la principal. Tanto `AsyncTask` como `HandlerThread` son posibilidades razonables aquí, dado que la conexión va a ser mucho más corta que en la práctica anterior donde se esperaba del usuario que hiciese *"scroll down"* y la app cargase fotos constantemente. Observad que todo se puede hacer desde la hebra principal salvo la conexión a Flickr para descargar las fotos. Si hacéis una `AsyncTask`, `doInBackground` tomará como entrada la posición y hará la conexión, obteniendo el JSON correspondiente. Debe ser, como siempre, `onPostExecute` el que tome la imagen descargada y la muestre en la interfaz. De entre todas las obtenidas, mostrad la primera, la más reciente o fijad otro criterio. Opcionalmente, podéis hacer que cada vez que se pulse el botón muestre una foto distinta, aunque siempre próximas a la posición.

Información de posición en el emulador

Para probar la app en el emulador, necesitamos algún tipo de proveedor de información de posición que el *Fused Location Provider* pueda utilizar. Para ello, podemos usar la app adicional *MockWalker.apk*. Descargadla de Moodle e instaladla en vuestro emulador, por ejemplo, con `adb`. Usad la **API 21** para el emulador (y por tanto para el proyecto), que es para la que está testeada la app. Para que la app proporcione posiciones basta con lanzarla, pulsar el botón *"Start"* y dejarla ejecutar en segundo plano (veréis que lo está porque aparece el icono en el área de notificaciones).

Acceso a Google Maps desde Android

Para usar la Google Maps Android API, se debe registrar el proyecto de la aplicación en la Google API Console y obtener una clave de API de Google. Las instrucciones detalladas pueden encontrarse en http://developers.google.com/maps/documentation/android-api/signup?hl=es-419. Una vez se haya obtenido la clave, debe editarse el fichero `AndroidManifest.xml` para añadir como hijo del elemento `<application>` (o sea, antes de la etiqueta de cierre `</application>`) lo siguiente:

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

Entrega

A través de Moodle hasta las 12 horas del **26 de abril**. Será un fichero `.zip` que incluya al menos

- la carpeta `src` del proyecto en Android Studio,
- la aplicación `.apk`, y
- una brevísima documentación en `pdf` (autores, aspectos no triviales …)