

Software para Dispositivos Móviles
Grado en Ingeniería Informática del Software
Escuela de Ingeniería Informática – Universidad de Oviedo

Geolocalización en Android

Juan Ramón Pérez Pérez
Departamento de Informática
jrpp@uniovi.es

Experiencia basada en el contexto

- Una de las características únicas de las aplicaciones móviles es el **conocimiento de la ubicación**.
- Los usuarios llevan su **dispositivo móvil a todos sitios**
- Para ofrecer una experiencia contextual podemos añadir la **conciencia de la ubicación** en las aplicaciones que desarrollemos.

La capa "My Location"

```
// Activa la capa de  
geolocalización
```

```
mapa.setMyLocationEnabled(  
true);
```

- Es necesario disponer de permisos
- No permite recuperar datos por programa
- Sólo capa visual con el punto indicando dónde nos encontramos



Alternativas uso geolocalización en Android

- **Android framework location APIs**, package android.location
- **Google Play Services Location API**, parte de Google Play Services
- Google aconseja el uso de la segunda alternativa:
 - Ofrece un framework de más alto nivel y más potente.
 - Automatiza tareas de elección de proveedor de localización y gestión de energía
 - Incorpora detección de la actividad



Google Play Services Location API

Enlazar Google Play Services (Android Studio)

- Forma parte de Google Play Services
- Mismo Proceso previo que la API Google Maps

1. build.gradle (Module: app)

```
compile 'com.google.android.gms:play-services-  
location:11.0.4'
```

3. Sync Project with Gradle Files



PBL: Actualiza Localización

- Sin mapas para demostrar que es independiente
- Empezar Act(ualizar), primera posición y actualización periódica.
- Parar Act(ualizar), parar actualización periódica.
- Busca dirección, sobre la localización actual convierte dirección postal



Permisos para Servicios de localización (I)

- Permisos que necesitamos incorporar en el AndroidManifest.xml

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

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

- Cada permiso está relacionado con un *location provider*:
 - **ACCESS_FINE_LOCATION** : Localización más exacta (GPS)
 - **ACCESS_COARSE_LOCATION** : Localización menos exacta (Ej. Red, triangulación de la posición,..)

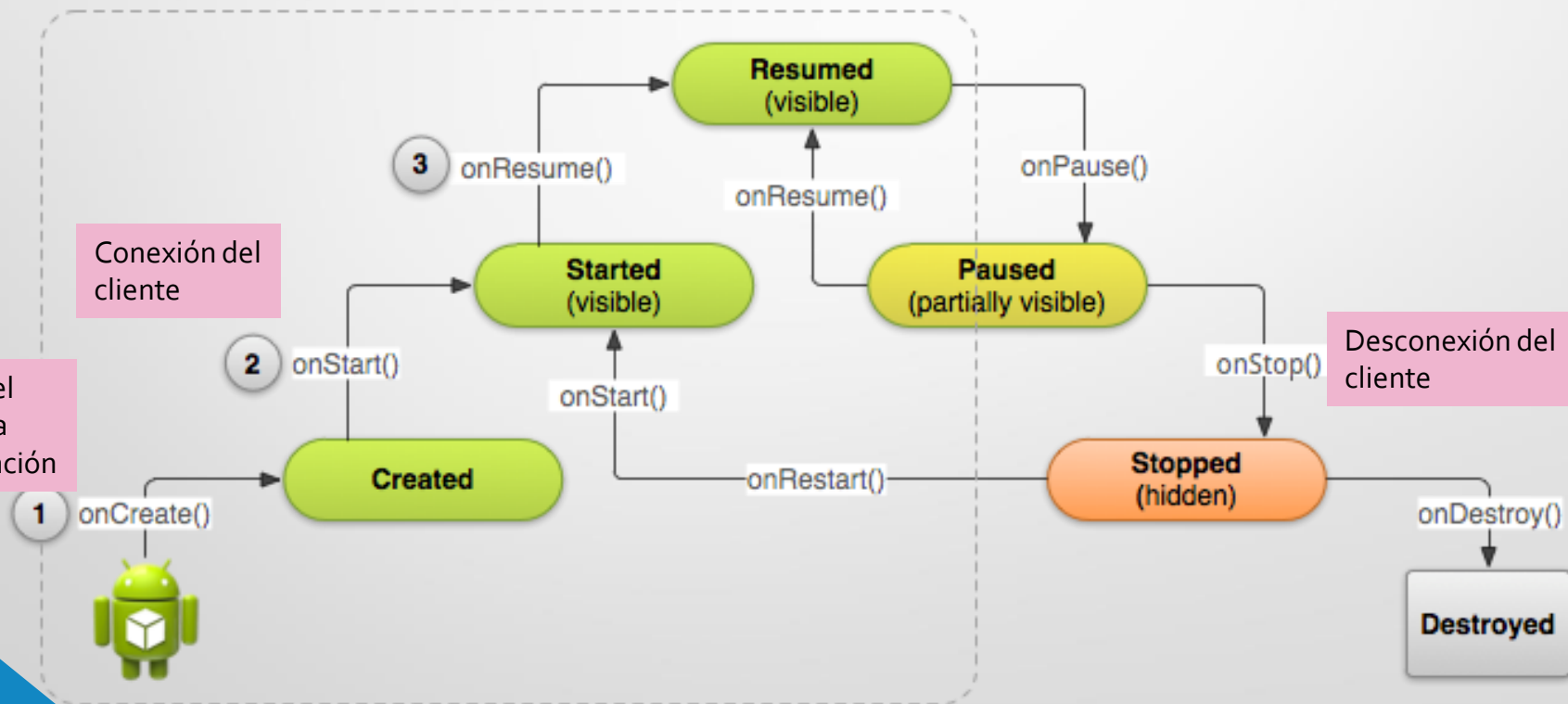
Permisos para Servicios de localización (II)

- API 23 (6.0) o superior → pedir permisos en tiempo de ejecución
 - Mayor control sobre la funcionalidad: elegir y revocar permisos
- Permisos normales y críticos (peligrosos)
- Comprobar si existen permisos
- Solicitar permisos
- Actuar en función de la respuesta del usuario

Cómo obtener la ubicación actual

- **Location services guarda** automáticamente la **ubicación** actual del usuario.
- Las aplicaciones pueden solicitarla cuando la necesiten
- Las APIs de localización de Google Play Services utilizan un proveedor de localización **“combinado”** que utiliza información de distintas fuentes.
- El proveedor de localización funciona como un **servicio** y las aplicaciones como **clientes**

Crear y conectar el Location Client en función del ciclo de vida de la actividad



Creación de una instancia de GoogleApiClient

- Google Play Services tiene una forma común de crear clientes para utilizar las distintas APIs
- Es a través de la clase: **GoogleApiClient**
- Especificando en la API: **LocationServices.API**

```
private void crearClienteLoc() {  
    if (clienteLocalizacion == null) {  
        clienteLocalizacion = new GoogleApiClient.Builder(this)  
            .addConnectionCallbacks(this)  
            .addApi(LocationServices.API)  
            .build();  
    }  
}
```

Interfaces que implementamos para GoogleApiClient

- Google Play Services utiliza interfaces para comunicarse con nuestra aplicación, a través de callbacks:
 - *ConnectionCallbacks*
 - Especifica los métodos que llama Google Play Services cuando el cliente de localización (GoogleApiClient) se conecta o se desconecta.
 - *onConnected()*
 - *onConnectionSuspended()*
 - *OnConnectionFailedListener*
 - Especifica un método que llama Google Play Services si ocurre un error cuando se intenta la conexión
 - *onConnectionFailed()*

Puesta en funcionamiento de Google Play services API client para localización

- Creamos un campo del tipo GoogleApiClient
- onCreate
 - **Creación de una instancia** de GoogleApiClient
- onStart
 - Invocamos al **método connect()** de GoogleApiClient
- onStop
 - Invocamos al **método disconnect()** de GoogleApiClient

Obtener la ubicación actual

- Invocar el método:

LocationServices.FusedLocationApi.*getLastLocation*(clienteLocalizacion)

- Esto se hace dentro del método *onConnected()*
- Devuelve resultado de tipo *Location*
- De un objeto [Location](#), entre otras cosas, podemos saber:
 - Posición : Latitud y Longitud
 - Fecha de la última localización


Datos de localización simulados con el emulador (Android Studio)

- Controles del emulador > Opción más controles
- Permite controlar diversas condiciones del dispositivo
 - Llamadas, batería, señal y sensores
- Location, permite controlar valores del GPS:
 - como coordenadas individuales latitud / longitud,
- Como secuencia de coordenadas
 - con un fichero GPX para reproducir la ruta o
 - un fichero KML para introducir múltiples puntos



Pruebas en dispositivos reales simulando ubicaciones

- Los dispositivos Android se pueden configurar para sus fuentes de localización proporcionen datos simulados
- Una vez configurado el dispositivo podemos desarrollar aplicaciones que generen estos datos
- Un ejemplo de aplicación es:
 - Fake GPS location



Actualización periódica de la localización

Recibir actualizaciones de la ubicación

- Uso de las actualizaciones periódicas de la ubicación:
 - Aplicaciones que hacen navegación o tracking
 - Obtener la ubicación del usuario periódicamente
 - Pedir actualizaciones periódicas de Location Services
- Planteamiento para actualizaciones de localización:
 - Callback de un método
 - Ante un cambio de ubicación, realiza una llamada a este método con la nueva ubicación

Preparativos recibir actualizaciones de ubicación

- Mismos preparativos que para obtener la ubicación actual
 - Definir los permisos para los dos tipos de localización
 - Crear un cliente de Google Play Service APIs
- Configuración específica
 - Establecer parámetros de uso del servicio
 - Comprobar disponibilidad del servicio

Especificar los parámetros de actualización

- **LocationRequest**
 - Se envía para empezar a recibir actualizaciones
 - Permite establecer el intervalo de actualizaciones y la precisión
- Dos métodos para establecer el intervalo
 - ***setInterval()*** – intervalo en milisegundos para recibir la posición
 - ***setFastestInterval()*** – intervalo mínimo
 - ***setPriority()*** – prioridad entre precisión y utilización de batería

Ejemplo de LocationRequest

- Este código se inserta
 - Después de haber conectado con el servicio
 - Antes de empezar a recibir actualizaciones de localización

```
protected void crearPeticiónLoc() {  
    peticiónLoc = new LocationRequest();  
    peticiónLoc.setInterval(10000);  
    peticiónLoc.setPriority(  
        LocationRequest.PRIORITY_HIGH_ACCURACY);  
}
```


Definir el callback para actualizaciones

- Las actualizaciones se activan con la siguiente llamada:

```
LocationServices.FusedLocationApi.requestLocationUpdates(  
    clienteLocalizacion, petitionLoc, this);
```

- Implementar la interfaz **LocationListener** en la actividad que de soporte a la localización
- Location service invoca al método
 - **onLocationChanged(Location)**
- La función recibe un argumento de tipo Location
- Desactivar actualizaciones:

```
LocationServices.FusedLocationApi.removeLocationUpdates  
(clienteLocalizacion, this);
```



Obtener dirección:
Servicio Geocodificación
mediante tarea asíncrona

Servicios de Google Maps:

Geocoder

- Dos funciones:
 - Convertir direcciones a coordenadas geográficas (Geocodificación)
 - Convertir coordenadas geográficas a direcciones postales (Geocodificación inversa)

<https://maps.googleapis.com/maps/api/geocode/json?latlng=43.354971,-5.851552>

- Android encapsula la comunicación con el servicio en la **clase Geocoder** (android.location.Geocoder)

Clase Geocoder

- Geocoding: dirección → coordenadas geográficas
 - `List<Address> getFromLocationName(String locationName, int maxResults, double lowerLeftLatitude, double lowerLeftLongitude, double upperRightLatitude, double upperRightLongitude)`
 - `List<Address> getFromLocationName(String locationName, int maxResults)`
- Geocoding inverso: coordenadas → dirección
 - `List<Address> getFromLocation(double latitude, double longitude, int maxResults)`

Llamada al servicio geocoding

- **getFromLocation()**
- Es una llamada **síncrona**, hace la petición y queda bloqueado hasta que recibe respuesta
- Podría **no ser inmediata** dejando bloqueada la interfaz.
- **No se debe llamar desde el mismo hilo de la interfaz de usuario** de la aplicación.

Subprocesos de trabajo para llamar al servicio

- Si el proceso no es instantáneo se podría bloquear la interfaz
- Para evitarlo ejecución en un subproceso de trabajo
- No acceder a la IU desde fuera del subproceso de IU
- AsyncTask
 - Permite realizar trabajo asíncrono en la interfaz de usuario.
 - Publica los resultados en el subproceso de IU

Cómo se utiliza AsyncTask

- Crear una subclase de la AsyncTask
- Implementar el método de callback doInBackground(), que se ejecuta en un grupo de subprocesos en segundo plano.
- Para actualizar la IU, debes implementar onPostExecute(),
- que recoge el resultado de doInBackground() y se ejecuta en el subproceso de IU.
- Luego, puedes ejecutar la tarea llamando a execute() desde el subproceso de IU.


Declaración de AsyncTask

- Tres tipos genéricos utilizado por AsyncTask
 - Params, tipo de los parámetros enviado a la tarea desde el método *execute*.
 - Progress, tipo de las unidades de progreso publicadas durante la ejecución en segundo plano. Se utilizan en el método *onProgressUpdate*.
 - Result, tipo del resultado de la ejecución en segundo plano. Utilizado en *onPostExecute*.
- Si alguno no se utiliza podemos definirlo como *Void*.

AsyncTask aplicado a este caso

```
public void buscadirButtonHandler(View view) {  
    // lanza servicio para recuperar dirección  
    new BuscaDireccion().execute(localizacionActual);  
}
```

```
private class BuscaDireccion extends AsyncTask<Location, Void, String> {  
    /**  
     * El sistema llama este método para realizar tareas en un hilo worker  
     * Recoge los parámetros que le pasa AsyncTask.execute() */  
    @Override  
    protected String doInBackground(Location... locations) {  
        return recuperarDireccion(locations[0]);  
    }  
  
    /**  
     * El sistema llama este método para realizar tareas en el hilo UI  
     * Recoge el resultado de doInBackground() */  
    @Override  
    protected void onPostExecute(String direccion) {  
        mostrarDireccion(direccion);  
    }  
}
```

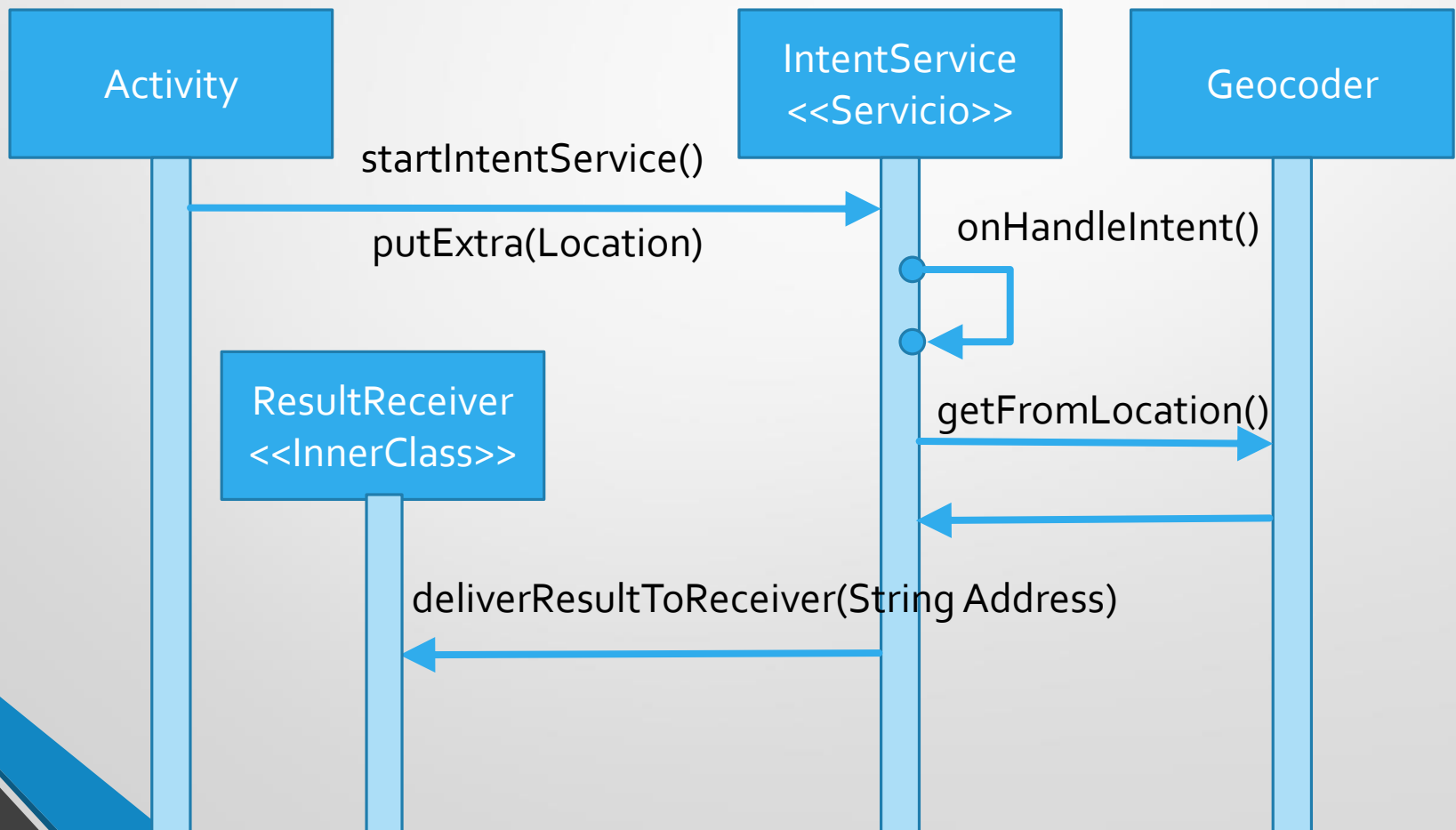


Utilización de servicios en vez de AsyncTask

IntentService para ejecutar tareas en segundo plano

- La clase *IntentService* permite ejecutar una tarea en un hilo en segundo plano (*worker thread*)
- *AsyncTask*, también permite esto; pero está pensado para operaciones de poca duración en el tiempo.
- AsyncTask necesita una referencia a la activity desde la que se lanza, si tiene que ser recreada puede haber problemas
- IntentService no tiene ese problema porque no está ligada a la activity.

¿Cómo usar IntentService?



Implementación del servicio: parámetros y procesamiento


- En **onHandleIntent**(Intent)
 - Recogemos el dato Location con `getParcelableExtra()`
 - Llamamos a `geocoder.getFromLocation()`
 - Recuperamos la dirección con: `address.getAddressLine()`
 - Devolvemos el resultado con `bundle.putString()`

Lanzar el IntentService

- En método estático: **startIntentService**
 - Creamos un Intent con la clase IntentService
 - Pasamos los parámetros con putExtra()
 - Clase receptora
 - Location
- Lanzamos el servicio con **startService(intent);**

Recepción de resultados

- *Inner Class* **ResultReceiver**
 - Método onReceiveResult
 - Para recuperar resultados bundle.getString()
- Crear una instancia en el onCreate y pasarla la invocar al servicio



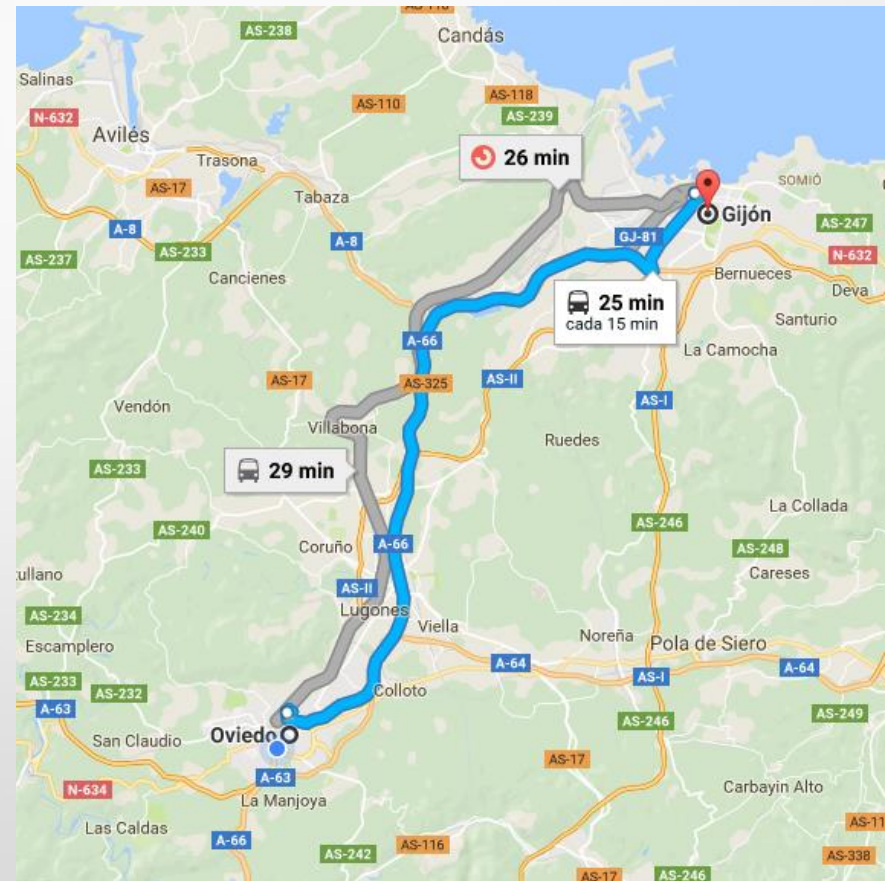
Otros servicios de información
geográfica

Otros servicios de información geográfica

- Distintos servicios REST relacionados con información geográfica
- Devuelven la respuesta en XML o JSON
- Google no proporciona un framework para encapsularlos
- Llamar al servicio REST → Procesar la respuesta JSON

Servicios *Directions*

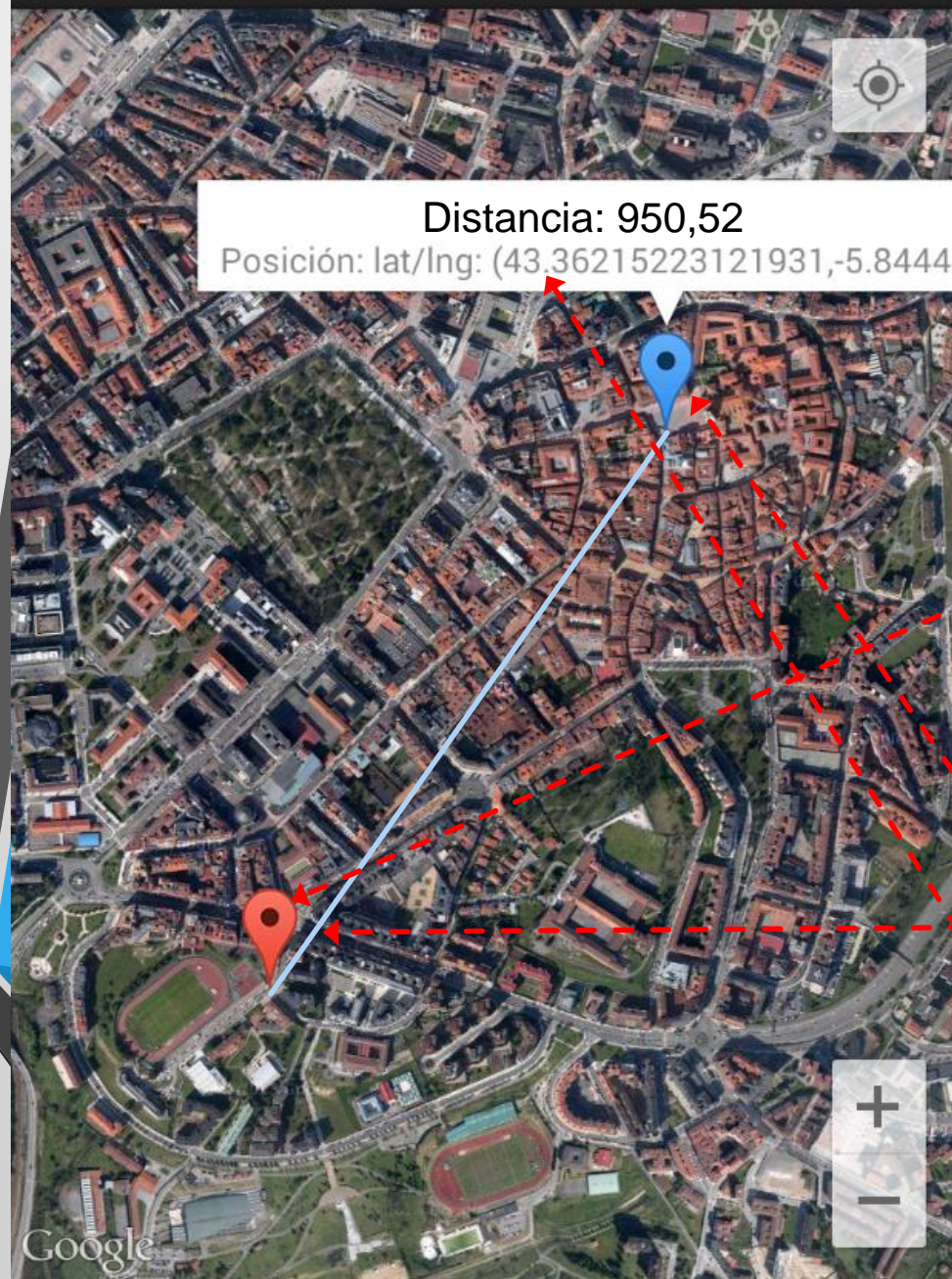
- Servicio que busca la mejor ruta entre dos puntos
- Gran versatilidad: caminando – en coche, puntos intermedios, rutas alternativas
- [Oviedo – Gijón](#)
- <https://maps.googleapis.com/maps/api/directions/json?origin=Oviedo+Asturias&destination=Gijon+Asturias>



Otros servicios

- Elevation
- Distance Matrix
- Roads
- Time Zone
- Places

<https://developers.google.com/maps/documentation/webservices/?hl=es>




Ejercicio: fusionar mapas y geolocalización

1. Insertar un marcador en el mapa correspondiente al punto donde nos encontramos.
2. Permitir al usuario que cree un nuevo marcador con una pulsación larga.
3. Trazar una polilínea entre los dos marcadores.
4. Calcular la distancia e incorporarla en el infowindow
5. Actualizar todo a medida que el usuario se mueva

¿Cómo medir la distancia en línea recta?

- La clase Location (paquete android.location) dispone de dos métodos que permiten medir distancia:
 - [distanceBetween\(\)](#), método estático que mide distancia entre dos puntos especificado por su latitud, longitud.
 - [distanceTo\(\)](#), mide distancia a otro objeto Location.



Detectar proximidad a un
punto

Geofencing:

Detectar proximidad a un punto

- Utiliza la conciencia de la localización actual del usuario combinada con la proximidad a un punto fijo.
- Área circular virtual establecida alrededor de un punto geográfico (*geofence*)
 - Se define por un punto central (Lat, Lng) y un radio
- Google Services permite definir y monitorizar estos perímetros

Características *geofences*

- Se pueden activar **múltiples** *geofences*
- Se pueden recoger **eventos** al **entrar y salir** en un geofence
- O establecer un **tiempo en el interior** de un geofence
- Además, se puede establecer un tiempo de duración, a partir del cual se eliminan automáticamente

<http://developer.android.com/intl/es/training/location/geofencing.html>



Referencias

- Guías sobre Google Maps API para Android:
 - <https://developers.google.com/maps/documentation/android/map>
- Tutorial Android Developer: Making Your App Location-Aware
 - <http://developer.android.com/training/location/index.html>