



# Mapas





## Contenidos

Google Maps Android API .....	3
Proyecto P_93_Mapas .....	4
Tareas por hacer ( <i>TODO</i> ).....	4
Tareas facilitadas por el asistente .....	4
Clases GoogleMap y SupportMapFragment .....	5
Tipos de mapas.....	6
Mover el mapa a una determinada posición .....	6
Movimientos básicos.....	6
Movimientos simultáneos.....	7
Obtener posición de la cámara .....	7
Características de la interfaz de usuario en el mapa .....	7
Gestos en el mapa .....	7
Controles rápidos .....	8
Zoom .....	8
Brújula .....	8
Botón herramientas de Google .....	8
Botón “Mi ubicación” .....	8
Fijar propiedades por layout .....	8
Eventos en el mapa: onMapClick(LatLng) .....	9
Eventos en el mapa: onMapLongClick(LatLng) .....	9
Eventos en el mapa por movimiento de cámara .....	9
Eventos en POI's.....	9
Marcadores .....	9
Eventos en marcadores.....	10
Eventos en la ventana de información.....	10
Borrar un marcador.....	10
No mostrar ventana de información.....	10
Dibujar formas en el mapa.....	11
Dibujar líneas.....	11
Dibujar polígonos .....	11
Dibujar círculos.....	11
Mapas personalizados.....	11
Funciones avanzadas.....	12
Web Services de Google Maps.....	13
Prácticas .....	13



## Google Maps Android API

Google Developers

Google Maps Android API

Productos > API de Google Maps > Para Android > Google Maps Android API

Google Maps Android API

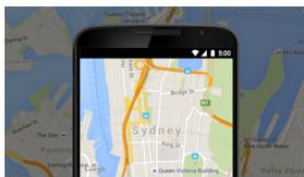
Agrega Google Maps a tu aplicación de Android.

OBTENER UNA CLAVE VER PLANES

PÁGINA DE INICIO GUÍAS REFERENCIA MUESTRAS ASISTENCIA ENVIAR COMENTARIOS

### Lo mejor de Google Maps para cada aplicación de Android

Escribe un mapa personalizado para tu aplicación de Android con edificios en 3D, planos para pisos de interiores y más.



#### Mapas

Integra mapas básicos, edificios en 3D, planos para pisos de interiores y el modo básico.



#### Imágenes

Agrega imágenes de Street View e imágenes satelitales.



#### Personalización

Agrega marcadores personalizados, ventanas de información y polilíneas.

### Descubre lo que puedes hacer con las API de Google Maps



#### Crea aplicaciones basadas en la ubicación

Utiliza las herramientas y los servicios de Google para crear innovadoras aplicaciones basadas en la ubicación.



#### Crea mapas para aplicaciones móviles

Crea aplicaciones de alto rendimiento que funcionen en distintos dispositivos móviles.



#### Visualiza datos geoespaciales

Crea imágenes en 3D con el API de Google Earth, mapas de calor en Google Fusion Tables y mucho más.



#### Personaliza tus mapas

Crea mapas personalizados que destaquen tus datos, tus imágenes y tu marca.



#### Licencia del API de Google Maps

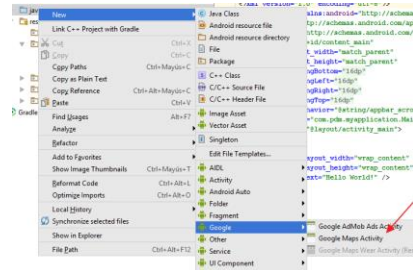
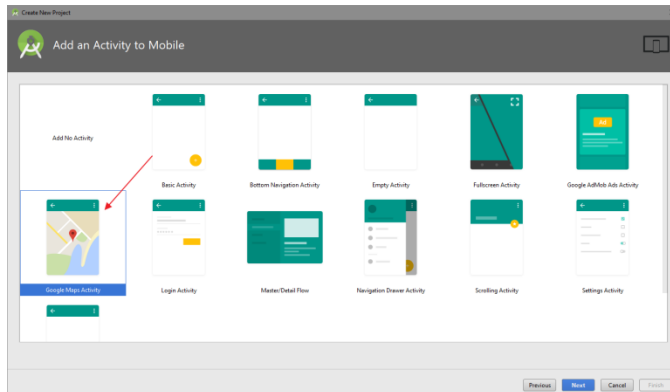
Para obtener más información sobre los precios y las condiciones, haz clic [aquí](#).

**Nota importante:** como ocurre con otros servicios de Google Play, es necesario obtener la API Key de la aplicación, pero Android Studio nos facilita la tarea!.



## Proyecto P\_93\_Mapas

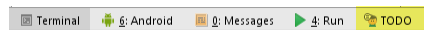
El IDE ofrece un asistente a la hora de crear la actividad que usa un GoogleMap que ahorra muchas tareas.



## Tareas por hacer (TODO)

El fichero `/res/values/google_maps_api.xml` nos recuerda con "TODO" que debemos fijar la API Key y nos señala la URL para obtenerla. Seguimos las instrucciones, borramos el citado TODO y reconstruimos el proyecto.

Observación: si vuelves a abrir la ventana TODO desde



aparece que aún queda por actualizar la versión "release" necesaria para publicar en el "market" Google Play. También tendrías que firmar.

## Tareas facilitadas por el asistente

- Añadir la librería:  
`implementation 'com.google.android.gms:play-services-maps:17.0.0'`
- Fijar los permisos en el Manifest (observa los comentarios)  
`<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />`
- Crear el layout con el fragmento para mapas  
El Layout no puede mostrar una vista previa!
- Plantilla de la Activity:

Ejecuta la aplicación:



Obseva que no aparece la toolbar. Si en tu aplicación necesitas que aparezca (porque, por ejemplo, haya un menú de opciones), cambia

```
public class MapsActivity extends FragmentActivity implements OnMapReadyCallback {
    por
    public class MapsActivity extends AppCompatActivity implements OnMapReadyCallback {
```



## Clases GoogleMap y SupportMapFragment

```
public class MapsActivity extends FragmentActivity implements OnMapReadyCallback {

    private GoogleMap mMap;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);

        // Obtain the SupportMapFragment and get notified when the map is ready to
        // be used.
        SupportMapFragment mapFragment = (SupportMapFragment)
            getSupportFragmentManager()
                .findFragmentById(R.id.map);

        mapFragment.getMapAsync(this);
    }

    /**
     * Manipulates the map once available.
     * This callback is triggered when the map is ready to be used.
     * This is where we can add markers or lines, add listeners or move the camera.
     * In this case,
     * we just add a marker near Sydney, Australia.
     * If Google Play services is not installed on the device, the user will be
     * prompted to install
     * it inside the SupportMapFragment. This method will only be triggered once the
     * user has
     * installed Google Play services and returned to the app.
     */
    @Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;

        // Add a marker in Sydney and move the camera
        LatLng sydney = new LatLng(-34, 151);
        mMap.addMarker(new MarkerOptions().position(sydney).title("Marker in
        Sydney"));
        mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
    }
}
```

La actividad implementa la interfaz OnMapReadyCallback

Obtendremos una referencia al MapFragment que hemos añadido a nuestro layout a través del fragment manager

Llamaremos a su método `getMapAsync()` pasándole un objeto que implemente la interfaz `OnMapReadyCallback`, en nuestro caso la propia actividad principal. Con esto conseguimos que en cuanto esté disponible la referencia al mapa (de tipo `GoogleMap`) incluido dentro de nuestro `MapFragment` se llame automáticamente al método `onMapReady()` de la interfaz.

Por el momento la implementación de este método va a ser muy sencilla, limitándonos a guardar el objeto `GoogleMap` recibido como parámetro en nuestro objeto `mMap` y fijando unos atributos iniciales (añadir un marcador y mover la "cámara" a Sydney)

El objeto `mMap` es la representación interna del propio mapa. La clase [GoogleMap](#) administra las siguientes operaciones de manera automática:

- conexión al servicio de Google Maps;
- descarga de mosaicos de mapas;
- visualización de mosaicos en la pantalla del dispositivo;
- visualización de varios controles, como el desplazamiento y el zoom.
- respuesta a gestos de desplazamiento y zoom a través del movimiento del mapa y su acercamiento o alejamiento.

Además de estas operaciones automáticas, puedes controlar el comportamiento de los mapas con objetos y métodos de la API. Por ejemplo, [GoogleMap](#) tiene métodos callback que responden a la pulsación de teclas y a gestos táctiles en el mapa. También puedes configurar iconos de marcadores en tu mapa y agregarle diseños a través de objetos que proporciones a [GoogleMap](#).

[MapFragment](#) es una subclase de la clase `Fragment` de Android, que permite disponer un mapa en un fragmento de Android. Los objetos `MapFragment` actúan como contenedores del mapa y brindan acceso al objeto `GoogleMap`.

Posible mejora: añadir la prueba que comprueba que el dispositivo es válido (dispone de Google Play Services y Google Play está actualizado) tal y como vimos en la unidad 46.



## Tipos de mapas

La Google Maps Android API ofrece cuatro clases de mapas y una opción para que no se muestre ninguno:

- **Normal:** Mapa de carreteras típico. En él se muestran carreteras, elementos naturales importantes y elementos artificiales, como los ríos. También se ven etiquetas de carreteras y elementos.
- **Híbrido:** Datos de fotos satelitales con mapas de carreteras agregados. También se ven etiquetas de carreteras y elementos.
- **Satélite:** Datos de fotos satelitales. No se ven etiquetas de carreteras y elementos.
- **Tierra:** Datos topográficos. En el mapa se incluyen colores, líneas de contornos y etiquetas, y sombreados de perspectiva. También se pueden ver carreteras y etiquetas.
- **Ninguno:** Ausencia de mosaicos. El mapa se representa como una cuadrícula vacía sin mosaicos cargados.



Para cambiar el tipo de mapa:

```
mMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
```

Habilitar información de carreteras

```
mMap.setTrafficEnabled(true);
```

## Mover el mapa a una determinada posición

Podremos mover libremente nuestro punto de vista (o cámara, como lo han llamado los desarrolladores de Android) por un espacio 3D.

De esta forma, ya no sólo podremos hablar de latitud-longitud (target) y zoom, sino también de orientación (bearing) y ángulo de visión (tilt).

El movimiento de la cámara se va a realizar siempre mediante la construcción de un objeto **CameraUpdate** con los parámetros necesarios.

### Movimientos básicos

La clase **CameraUpdateFactory** y sus métodos permiten actualización de la latitud y longitud y/o el nivel de zoom.

Para cambiar sólo el nivel de zoom se usan los métodos **zoomIn()**, **zoomOut()** y **zoomTo()**:

- **CameraUpdateFactory.zoomIn()**: Aumenta en 1 el nivel de zoom.
- **CameraUpdateFactory.zoomOut()**: Disminuye en 1 el nivel de zoom.
- **CameraUpdateFactory.zoomTo(nivel\_de\_zoom)**: Establece el nivel de zoom.

Para actualizar sólo la latitud-longitud de la cámara, el método **newLatLng()**:

- **CameraUpdateFactory.newLatLng(lat, long)**: Establece la lat-lng expresadas en grados.

Si queremos modificar los dos parámetros anteriores de forma conjunta, método **newLatLngZoom()**:

- **CameraUpdateFactory.newLatLngZoom(lat\_long, zoom)**: Establece la lat-lng y el zoom.

Para movernos lateralmente por el mapa (*panning*), método **scrollBy()**:

- **CameraUpdateFactory.scrollBy(scrollHorizontal, scrollVertical)**: Scroll expresado en píxeles.

Tras construir el objeto **CameraUpdate** con los parámetros de posición tendremos que llamar a los métodos **moveCamera()** o **animateCamera()** de nuestro objeto **GoogleMap**, dependiendo de si queremos que la actualización de la vista se muestre directamente o de forma animada. Si quisiéramos por ejemplo centrar la vista en España con un zoom de 5 podríamos hacer lo siguiente:

```
CameraUpdate camUpd1 = CameraUpdateFactory.newLatLngZoom(new LatLng(40.41, -3.69), 5);  
mMap.moveCamera(camUpd1);
```



## Movimientos simultáneos

Si queremos modificar los demás parámetros de la cámara o varios de ellos simultáneamente tendremos disponible el método más general **CameraUpdateFactory.newCameraPosition()** que recibe como parámetro un objeto de tipo **CameraPosition**.

Este objeto lo construiremos indicando todos los parámetros de la posición de la cámara a través de su método **Builder()** de la siguiente forma:

```
LatLng ies = new LatLng(41.644521, -0.863994);
CameraPosition camPos = new CameraPosition.Builder()
    .target(ies)           //Centramos el mapa en Instituto
    .zoom(18)             //Establecemos el zoom en 18
    .bearing(45)          //Establecemos la orientación con el noreste arriba
    .tilt(70)             //Bajamos el punto de vista de la cámara 70 grados
    .build();
CameraUpdate camUpd = CameraUpdateFactory.newCameraPosition(camPos);
mMap.animateCamera(camUpd);
```

## Obtener posición de la cámara

Mediante el método **getCameraPosition()** que nos devuelve un objeto **CameraPosition**.

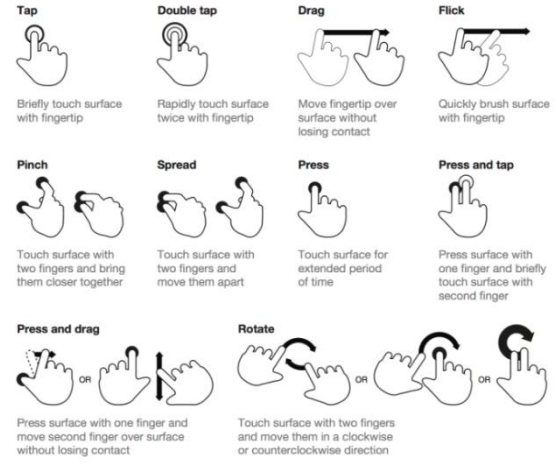

Accediendo a los distintos métodos y propiedades de este objeto podemos conocer con exactitud la posición de la cámara, la orientación y el nivel de zoom.

```
CameraPosition camPosVer = mMap.getCameraPosition();
LatLng coordenadas = camPosVer.target;
double latitud = coordenadas.latitude;
double longitud = coordenadas.longitude;
float zoom = camPosVer.zoom;
float orientacion = camPosVer.bearing;
float angulo = camPosVer.tilt;
```

## Características de la interfaz de usuario en el mapa

Se pueden fijar accediendo a las características de la interfaz de usuario mediante el método **getUiSettings()**

### Gestos en el mapa

 <p><b>Tap</b> Briefly touch surface with fingertip</p> <p><b>Double tap</b> Rapidly touch surface twice with fingertip</p> <p><b>Drag</b> Move fingertip over surface without losing contact</p> <p><b>Flick</b> Quickly brush surface with fingertip</p> <p><b>Pinch</b> Touch surface with two fingers and bring them closer together</p> <p><b>Spread</b> Touch surface with two fingers and move them apart</p> <p><b>Press</b> Touch surface for extended period of time</p> <p><b>Press and tap</b> Press surface with one finger and briefly touch surface with second finger</p> <p><b>Press and drag</b> Press surface with one finger and move second finger over surface without losing contact</p> <p><b>Rotate</b> Touch surface with two fingers and move them in a clockwise or counterclockwise direction</p>	<p><b>Zoom:</b></p> <p><i>Double tap</i> para acercar  <i>Press and Tap</i> para alejar  <i>Press and drag</i> para acercar/alejar  <i>Spread</i> para acercar  <i>Pinch</i> para alejar</p> <p><b>Desplazamiento:</b></p> <p><i>Drag</i> para desplazar continuamente el mapa de forma omnidireccional  <i>Flick</i> para desplazar en pequeñas porciones el mapa de forma omnidireccional</p> <p><b>Inclinación:</b></p> <p><i>Drag x 2</i> dedos de forma vertical. Hacia arriba inclina hasta 90° y hacia abajo declina a 0°</p> <p><b>Rotación:</b></p> <p><i>Rotate</i> para rotar el mapa a favor o en contra de las manecillas del reloj.</p>
<p><b>Multi-touch en Genymotion</b></p> <ul style="list-style-type: none"> <li>• Zoom in: right click + move mouse left</li> <li>• Zoom out: right click + move mouse right</li> <li>• Tilt forth: right click + move mouse up</li> <li>• Tilt back: right click + move mouse down</li> <li>• Clockwise rotation: Shift + right click + move mouse right</li> <li>• Counterclockwise rotation: Shift + right click + move mouse left</li> </ul> <p> If you use Mac OS X, replace right click with ctrl + click.</p>	<p><b>Multi-touch en AVDs</b></p> <p><i>Double tap</i>: Doble click</p> <p><i>Drag/Flick</i>: Click izquierdo sostenido</p> <p><i>Pinch/Spread</i>: Ctrl+ click izquierdo sostenido de adentro hacia a fuera o viceversa</p> <p><i>Rotate</i>: Ctrl + click izquierdo sostenido siguiendo una circunferencia</p> <p><i>Press and tap</i>: Ctrl + click derecho sostenido con movimiento vertical.</p>





Modificar la disponibilidad de los gestos de rotación. Usar false para deshabilitarlos.

```
mMap.getUiSettings().setRotateGesturesEnabled(true);
```

Modificar la disponibilidad de los gestos de desplazamiento. Usar false para deshabilitarlos.

```
mMap.getUiSettings().setScrollGesturesEnabled(true);
```

Modificar la disponibilidad de los gestos de inclinación. Usar false para deshabilitarlos.

```
mMap.getUiSettings().setTiltGesturesEnabled(true);
```

Modificar la disponibilidad de los gestos de zoom. Usar false para deshabilitarlos.

```
mMap.getUiSettings().setZoomGesturesEnabled(true);
```

Modificar la disponibilidad de todos los gestos. Usar false para deshabilitarlos.

```
mMap.getUiSettings().setAllGesturesEnabled(true);
```

## Controles rápidos

En general el valor por defecto es true (excepto en Zoom)

### Zoom

```
mMap.getUiSettings().setZoomControlsEnabled(true);
```

Con altos niveles de zoom, en el mapa se mostrarán planos de pisos para espacios interiores, como en aeropuertos, centros comerciales,... Incluso sus diferentes alturas.

```
mMap.setIndoorEnabled(true);
```

```
mMap.getUiSettings().setIndoorLevelPickerEnabled(true);
```

### Brújula

Aparece cuando la orientación e inclinación no son 0°. Si se presiona la cámara se moverá a su posición estándar.

```
mMap.getUiSettings().setCompassEnabled(true);
```

### Botón herramientas de Google

Dos accesos que se muestran al tocar un marcador en la parte inferior derecha. Su función es dirigirte a la app de Google Maps.

```
mMap.getUiSettings().setMapToolbarEnabled(true);
```

### Botón “Mi ubicación”

Al ser presionado mueve la cámara a la ubicación actual (punto azul) que marque el dispositivo si es que hay una (es decir hay que conseguirla usando la clase FusedLocationProviderClient vista en la unidad de Geolocalización).

```
mMap.setMyLocationEnabled(true);
```

O

```
mMap.getUiSettings().setMyLocationButtonEnabled(true);
```

Pero desde Android 6+ necesita tratamiento de permiso peligroso y declaración en el manifest (ya lo ha hecho el asistente)

## Fijar propiedades por layout

Muchas de las características anteriores que hemos fijado programando código java, pueden fijarse desde XML en el layout para el mapa que se carga inicialmente:

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:map="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/map"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.pdm.p_113_mapas.MapsActivity"
    map:cameraBearing="112.5"
    map:cameraTargetLat="-33.796923"
    map:cameraTargetLng="150.922433"
    map:cameraTilt="30"
    map:cameraZoom="13"
    map:mapType="normal"
    map:uiCompass="false"
    map:uiRotateGestures="true"
    map:uiScrollGestures="false"
    map:uiTiltGestures="true"
    map:uiZoomControls="false"
    map:uiZoomGestures="true" />
```





## Eventos en el mapa: onMapClick(LatLng)

Este método recibe como parámetro, en forma de objeto LatLng, las coordenadas de latitud y longitud sobre las que ha pulsado el usuario.

Si quisiéramos traducir estas coordenadas geográficas en coordenadas en pantalla podríamos utilizar un objeto **Projection**, obteniéndolo a partir del mapa a través del método **getProjection()** y posteriormente llamando a **toScreenLocation()** para obtener las coordenadas (x,y) de pantalla donde el usuario pulsó:

```

mMap.setOnMapClickListener(new GoogleMap.OnMapClickListener() {
    @Override
    public void onMapClick(LatLng latLng) {
        mMap.animateCamera(CameraUpdateFactory.newLatLng(latLng));
        Projection proj = mMap.getProjection();
        Point coord = proj.toScreenLocation(latLng);
        String texto= "Click\n" + "Lat: " + latLng.latitude + "\n" + "Lng: " + latLng.longitude +
"\n" + "X: " + coord.x + " - Y: " + coord.y;
        Toast.makeText(getApplicationContext(), texto, Toast.LENGTH_SHORT).show();
    }
});

```

## Eventos en el mapa: onMapLongClick(LatLng)

Similar pero para click largo

(Nota: si quisierámos deshabilitarlos, desde el objeto mapFragment, llamando a sus métodos: mapFragment.getView().setClickable(false); )

## Eventos en el mapa por movimiento de cámara

GoogleMap.OnCameraMoveStartedListener, GoogleMap.OnCameraMoveListener y GoogleMap.OnCameraIdleListener.

## Eventos en POI's

POI (puntos de interés: parques, hospitales, etc.). Suelen visualizarse por defecto junto con sus iconos en mapas de tipo normal. Pueden controlarse sus eventos con onPoiClick(PointOfInterest poi)

## Marcadores

Agregar un marcador es llamar al método addMarker():

```

mMap.addMarker(new MarkerOptions()
    .position(new LatLng(40.417325, -3.683081))
    .title("Pais: España"));

```

Otra manera es instanciando un objeto de tipo Marker:

```

Marker madrid=mMap.addMarker(new MarkerOptions()
    .position(new LatLng(40.417325, -3.683081))
    .title("Pais: España"));

```

Basta con llamar al método addMarker() pasando como parámetro un nuevo objeto MarkerOptions sobre el que establecemos la posición del marcador (método position()) y el texto a incluir en la ventana de información del marcador (métodos title() para el título y snippet() para el resto del texto).

Los marcadores pueden personalizarse a través de los siguientes métodos:

- **position** (obligatorio): El valor LatLng de la posición del marcador en el mapa. Esta es la única propiedad obligatoria para un objeto Marker.
- **anchor**: El punto de la imagen que se dispondrá en la posición de LatLng del marcador. Su valor predeterminado es el centro de la parte inferior de la imagen. Dicho punto se representa por coordenadas de textura (u,v). El punto A de la ilustración muestra la ubicación por defecto.
- **alpha**: Establece la opacidad del marcador con valores de 0 (transparente) a 1 (opaco). Valor predeterminado es 1.0.
- **title**: Una cadena que se muestra en la ventana de información cuando el usuario toca el marcador.
- **snippet**: Texto adicional que aparece debajo del título.





- **icon:** Un bitmap que se muestra en lugar de la imagen de marcador predeterminada.  
`icon(BitmapDescriptorFactory.fromResource(R.drawable.marker_icon_pointer))`  
También permite cambiar el color por defecto  
`icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_BLUE))`
- **draggable:** Debe fijarse en el valor true si deseas permitir que el usuario mueva el marcador. El valor predeterminado es false.
- **visible:** Debe fijarse en el valor false para que el marcador sea invisible. El valor predeterminado es true.
- **flat:** De manera predeterminada, los marcadores se orientan contra la pantalla y no giran ni se inclinan con la cámara.
- **rotation:** La orientación del marcador, especificada en grados en sentido horario.

## Eventos en marcadores

- De click:

```
mMap.setOnMarkerClickListener(new GoogleMap.OnMarkerClickListener() {
    public boolean onMarkerClick(Marker marker) {
        Toast.makeText(getApplicationContext(), "Marcador pulsado:\n"
+marker.getTitle(), Toast.LENGTH_SHORT).show();
        return false;
    }
});
```

No tienen que ser siempre Toast, podemos por ejemplo lanzar otra actividad si el marcador clickado es uno determinado.

- De arrastre:

Es obligatorio que el marcador tenga el atributo de draggable(true)

```
Marker madrid=mMap.addMarker(new MarkerOptions()
    .position(new LatLng(40.417325, -3.683081))
    .title("Pais: España")
    .draggable(true));

mMap.moveCamera(CameraUpdateFactory.newLatLng(new LatLng(40.417325, -3.683081)));
mMap.setOnMarkerDragListener(new GoogleMap.OnMarkerDragListener() {
    @Override
    public void onMarkerDragStart(Marker marker) {
        Toast.makeText(MapsActivity.this, "Empieza " + marker.getId() ,
        Toast.LENGTH_SHORT).show();
    }
    @Override
    public void onMarkerDrag(Marker marker) {
        Toast.makeText(MapsActivity.this, "Se mueve " + marker.getId() + " a " +
        marker.getPosition(), Toast.LENGTH_SHORT).show();
    }
    @Override
    public void onMarkerDragEnd(Marker marker) {
        Toast.makeText(MapsActivity.this, "Acaba " + marker.getId() ,
        Toast.LENGTH_SHORT).show();
    }
});
```

## Eventos en la ventana de información

```
mMap.setOnInfoWindowClickListener(new GoogleMap.OnInfoWindowClickListener() {
    @Override
    public void onInfoWindowClick(Marker marker) {
        //Lo que haya que hacer
    }
});
```

## Borrar un marcador

Si el objeto ha sido instanciado:

```
madrid.remove();
```

## No mostrar ventana de información

El método onMarkerClick() debe devolver true.



## Dibujar formas en el mapa

Dibujar líneas, círculos y polígonos sobre el mapa son elementos muy utilizados para trazar rutas o delimitar zonas del mapa.

- Una Polyline es una serie de segmentos de líneas conectados que pueden formar cualquier forma y pueden emplearse para marcar trazados y rutas en el mapa.
- Un Polygon es una forma delimitada que puede emplearse para marcar áreas del mapa.
- Un Circle es una proyección geográficamente precisa de un círculo de la superficie terrestre dibujado en el mapa.

### Dibujar líneas

Para dibujar una línea lo primero que tendremos que hacer será crear un nuevo objeto **PolylineOptions** sobre el que añadiremos utilizando su método **add()** las coordenadas (latitud-longitud) de todos los puntos que conformen la línea.

Tras esto estableceremos el grosor y color de la línea llamando a los métodos **width()** y **color()** respectivamente, y por último añadiremos la línea al mapa mediante su método **addPolyline()** pasándole el objeto **PolylineOptions** recién creado.

```
PolylineOptions lineas = new PolylineOptions()
    .add(new LatLng(45.0, -12.0))
    .add(new LatLng(45.0, 5.0))
    .add(new LatLng(34.5, 5.0))
    .add(new LatLng(34.5, -12.0))
    .add(new LatLng(45.0, -12.0));
lineas.width(8);
lineas.color(Color.RED);
mMap.addPolyline(lineas);
```

### Dibujar polígonos

Crearíamos un nuevo objeto **PolygonOptions** y añadiremos las coordenadas de sus puntos en el sentido de las agujas del reloj. En este caso no es necesario cerrar el circuito (es decir, que la primera coordenada y la última fueran iguales) ya que se hace de forma automática.

Para polígonos el ancho y color de la línea los estableceríamos mediante los métodos **strokeWidth()** y **strokeColor()**. Además, el dibujo final del polígono sobre el mapa lo haríamos mediante **addPolygon()**.

```
PolygonOptions rectangulo = new PolygonOptions()
    .add(new LatLng(45.0, -12.0),
        new LatLng(45.0, 5.0),
        new LatLng(34.5, 5.0),
        new LatLng(34.5, -12.0),
        new LatLng(45.0, -12.0))
    .strokeWidth(8)
    .strokeColor(Color.BLUE);
mMap.addPolygon(rectangulo);
```

### Dibujar círculos

Para construir un círculo, debes especificar las dos propiedades siguientes:

- center como un LatLng;
- radius en metros.

```
CircleOptions circleOptions = new CircleOptions()
    .center(new LatLng(40.417325, -3.683081))
    .radius(1000000); // In meters
Circle circle = mMap.addCircle(circleOptions);
```

## Mapas personalizados

La [herramienta web](#) nos permite fijar el estilo que deseemos a nuestros mapas. Una vez definido el estilo que necesitamos podremos exportarlo a un fichero JSON que utilizaremos en nuestro proyecto añadiéndolo a la carpeta /res/raw. Aplicarlo es tan sencillo como llamar al método **setMapStyle()**, pasándole como parámetro el nuevo recurso que se carga usando el método **MapStyleOptions.loadRawResourceStyle()** con parámetro el ID del recurso:



```
mMap.setMapStyle(MapStyleOptions.loadRawResourceStyle(this, R.raw.estilomapa));
```



# Funciones avanzadas

Google ofrece una [biblioteca de utilidades](#)



## Importar GeoJSON a tu mapa

Con esta utilidad, puedes almacenar funciones en formato [GeoJSON](#) y representarlas como una capa sobre el mapa. Para agregar tus datos de GeoJSON al mapa, llama a `addLayer()`. También puedes agregar funciones individuales llamando a `addFeature()` y pasando un objeto `GeoJsonFeature`.

Para obtener información detallada, consulta la documentación sobre la [Utilidad GeoJSON para Google Maps en Android](#).



## Importar KML a tu mapa

Con esta utilidad, puedes convertir objetos [KML](#) en formas geográficas y representarlas como una capa sobre el mapa. Para agregar tu capa al mapa, llama a `addLayerToMap()`. Puedes acceder a propiedades de un objeto KML llamando a `getProperties()` en cualquier `Placemark`, `GroundOverlay`, `Document` o `Folder`.

Para obtener información detallada, consulta la documentación sobre la [Utilidad KML para Google Maps en Android](#).



## Agregar mapas de calor a tu mapa

Los mapas de calor permiten a los observadores comprender la distribución y la intensidad relativa de puntos de datos de un mapa de manera sencilla. En los mapas de calor, en lugar de disponerse un marcador en cada ubicación se usan colores y formas para representar la distribución de los datos. Crea una clase `HeatmapTileProvider` y pásale una recopilación de objetos `LatLng` que representen puntos de interés en el mapa. Luego crea una clase `TileOverlay`, pásale el proveedor de mosaicos de mapas de calor y agrega la superposición de mosaicos al mapa.

Para obtener información detallada, consulta la documentación sobre la [Utilidad de mapas de calor para Google Maps en Android](#).



## Personalizar marcadores mediante iconos de burbujas

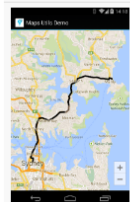
Agrega un objeto `IconGenerator` para mostrar fragmentos de información sobre tus marcadores. Esta utilidad ofrece una manera de hacer que tus iconos se asemejen a ventanas de información, en el sentido de que el propio marcador puede contener texto y otros elementos. La ventaja consiste en que puedes mantener abierto más de un marcador al mismo tiempo, mientras que en el caso de las ventanas de información solo puedes mantener una abierta a la vez. También puedes aplicar ajustes de estilo a los marcadores, modificar su orientación o contenido y cambiar su imagen de fondo o `NinePatch`.



## Administrar clústeres de marcadores

`ClusterManager` te permite administrar varios marcadores con diferentes niveles de zoom. Esto significa que puedes disponer muchos marcadores en un mapa sin que esto dificulte la lectura de este. Cuando un usuario visualiza el mapa con un alto nivel de zoom, los marcadores aparecen en él. Cuando el usuario aplica zoom de alejamiento, los marcadores se agrupan en clústeres para facilitar la visualización del mapa.

Para obtener información detallada, consulta la documentación sobre la [Utilidad de agrupación de marcadores en clústeres para Google Maps en Android](#).



## Codificar y decodificar polilíneas

La clase `PolyUtil` resulta útil para convertir polilíneas y polígonos codificados en coordenadas de latitud y longitud, y viceversa.

En Google Maps, las coordenadas de latitud y longitud que definen una polilínea o un polígono se almacenan como una cadena codificada. Consulta la explicación detallada sobre la [codificación de polilíneas](#). Puedes recibir esta cadena codificada en una respuesta de una API de Google, como la Google Maps Directions API.

Puedes usar `PolyUtil` en la biblioteca de utilidades de la Google Maps Android API para codificar una secuencia de coordenadas de latitud y longitud ('`LatLngs`') en una cadena de ruta de acceso codificada, y decodificar una cadena de ruta de acceso codificada en una secuencia de `LatLngs`. Esto garantizará la interoperabilidad con los servicios web de la Google Maps API.



## Calcular distancias, áreas y rumbos mediante geometría esférica

Con las utilidades de geometría esférica de `SphericalUtil`, puedes computar distancias, áreas y rumbos a partir de latitudes y longitudes. A continuación, se muestran algunos de los métodos disponibles en la utilidad:

- `computeDistanceBetween()`: devuelve la distancia, en metros, entre dos coordenadas de latitud y longitud.
- `computeHeading()`: devuelve el rumbo, en grados, entre dos coordenadas de latitud y longitud.
- `computeArea()`: devuelve el área, en metros cuadrados, de un trazado cerrado en el planeta.
- `interpolate()`: devuelve las coordenadas de latitud y longitud de un punto que se halla a una fracción de la distancia entre dos puntos determinados. Puedes usar esto para animar un marcador entre dos puntos, por ejemplo.



## Web Services de Google Maps

Si deseas mostrar ubicaciones con información más detallada, rastrear de forma precisa un elemento, calcular distancias y tiempos de viaje o trazar caminos por direcciones transitables, entonces investiga sobre los [Web Services de Google Maps](#).

Tu paquete de herramientas para el mundo real

Accede a interfaces HTTP, que proporcionan datos geográficos como geocodificación, indicaciones, elevación, sitio e información de zonas horarias.



### Google Maps Geocoding API

Realiza conversiones de direcciones a coordenadas geográficas.



### Google Places API Web Service

Implementa el autocompletado y agrega información actualizada sobre millones de ubicaciones a tu sitio o aplicación.



### Google Maps Elevation API

Datos de elevación para cualquier punto en el mundo.



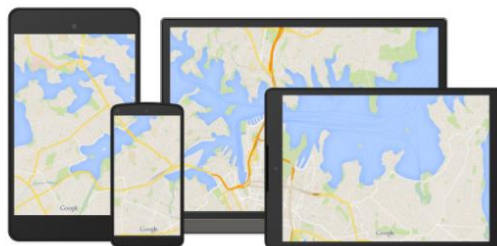
### Google Maps Roads API

Habilita la funcionalidad snap-to-road para rastrear de forma precisa rutas de navegación de GPS.



### Google Maps Geolocation API

Encuentra una ubicación según información de torres celulares y nodos WiFi.



### Google Maps Directions API

Calcula indicaciones entre varias ubicaciones.



### Google Maps Time Zone API

Proporciona datos de zonas horarias para cualquier lugar del mundo.



### Google Maps Distance Matrix API

Calcula el tiempo de viaje y la distancia para varios destinos.

## Prácticas

Realiza los ejercicios propuestos en el fichero Ejercicios\_21\_Geolocalización\_Mapas