

6. Programación multimedia.

Contenido

1. Introducción.....	1
2. Visualización de imágenes	1
2. Animaciones.....	5
3. Sonido y música.....	15
4. Vídeo	19
5. Geolocalización.....	20

1. Introducción

Casi cualquier aplicación actual para dispositivos móviles que quiera ser atractiva para los usuarios, aprovechar el hardware y dar la máxima información con una pantalla reducida, debe incorporar información en uno o más sentidos para el usuario: imágenes, vídeo, sonido, etc. Los dispositivos móviles también llevan un conjunto de sensores (de orientación, de posición geográfica, etc.) que facilitan mucho la interacción de los usuarios y las aplicaciones.

Para construir aplicaciones móviles valiosas hay que saber emplear todos estos elementos. Por suerte Android nos facilita mucho su inclusión.

2. Visualización de imágenes

Existen diferentes técnicas disponibles para visualizar imágenes en las aplicaciones Android. Comenzaremos con la técnica más sencilla para visualizar una imagen fija y a continuación veremos cómo cambiar la imagen dinámicamente, es decir, como respuesta a una acción del usuario. Finalmente, iremos más allá y visualizaremos todo un conjunto de imágenes utilizando una galería de imágenes, que nos permitirá navegar entre todas las imágenes de una colección.

1.1. Imágenes estáticas

Con el fin de utilizar imágenes en sus aplicaciones, es necesario que primero las agregue como recursos de la aplicación. Pero antes es conveniente prepararlas para que después no te den ningún tipo de problemas. Concretamente, lo que hay que hacer es lo siguiente:

- Se recomienda utilizar imágenes en formato PNG.
- El nombre de los archivos de imagen solo puede contener letras minúsculas y números, sin espacios, signos, letras mayúsculas ni con acento, cedilla u otros caracteres no ingleses.
- Al utilizar muchas imágenes a una aplicación hay que fijarse bien en cuál es su

resolución (puntos de ancho y de alto). En general, es conveniente que todas las imágenes tengan un tamaño similar.

El formato PNG

El formato Portable Network Graphics (PNG) es un formato de imagen libre (sin patentes) y sin pérdida de calidad que, además, permite transparencia. Por estas razones es el formato de imagen recomendado para las aplicaciones Android, frente a otros formatos populares como el JPEG.

Una vez preparada la imagen hay que incorporarla a su aplicación. En primer lugar, debe crear un nuevo proyecto Android (puede llamarlo multimedia1 y dejar el resto por defecto). A continuación, cambie la vista del proyecto de Android Project y despliegue la carpeta de su proyecto. Acceda a continuación en la subcarpeta `/app/src/main/res` (*res* viene de *resources*, 'recursos', es decir, los datos y ficheros adicionales tales como imágenes o sonidos de una aplicación) y, dentro de él, en la subcarpeta *drawable*.

Cuando haya encontrado esta subcarpeta, debe arrastrar la imagen que desea ver para que pase a formar parte de la aplicación. Aparecerá un diálogo para modificar el nombre de la imagen y su ruta; es necesario que acepte.

De este modo, la imagen ya está incorporada a su aplicación, pero esto no implica que sea visible. La forma más sencilla de visualizar una imagen en una aplicación Android es mediante el control `ImageView` que se encuentra en la sección *widgets* de la paleta del editor del *layout* (abre la carpeta de la aplicación y vaya a `/res/layout/activity_main.xml`).

Añádase el `ImageView` a su aplicación en la esquina superior izquierda, el `ImageView` no ocupará toda la pantalla, para que lo haga tendremos que hacer clic en los botones *layout_width* y asignar el valor *match_parent* y en *layout_height* el valor *match_parent* o cambie al XML las propiedades `android:layout_width` y `android:layout_height` a *match_parent* haciendo doble clic sobre el `ImageView` aparecerá un diálogo con la propiedad *src*, seleccione la imagen que ha copiado antes desde el directorio *Drawable* en la pestaña *Project*.



Para acabar de ajustar la aplicación, hay que añadir una cadena explicando lo que hay en

la imagen (el archivo strings. Xml), con el nombre descripcion_imagen, por ejemplo, y que regrese al main. Xml y agregue la línea del contentDescription (la última línea del siguiente código) de forma que la parte del ImageView quede de la siguiente manera:

```
<ImageView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/imageView"
    android:layout_alignParentTop="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:src="@drawable/ou"
    android:contentDescription="@string/descripcion_imagen"/>
```

1.2. Selección dinámica de imágenes

Acaba de ver cómo mostrar una imagen seleccionando el archivo de imagen con el editor de layouts de Android. Esta es la manera más sencilla y directa de mostrar una imagen al usuario. ¿Pero cómo se puede cambiar esta imagen dinámicamente, es decir, en respuesta a una acción del usuario?

En este ejemplo modificarás la aplicación anterior para que la imagen visualizada cambie cuando el usuario la toque. Para que la nueva aplicación tenga un nombre diferente, vaya al archivo strings. Xml y cambie la cadena app_name.

A continuación, añada una nueva imagen en la carpeta /res/drawable. De momento, si ejecuta la aplicación sólo verá la imagen inicial. ¿Cómo puede hacer que el programa reaccione y muestre otra imagen cuando la toque?

Abra el código del programa (archivo MainActivity. Java) para añadir el código de respuesta al clic en la imagen. En primer lugar, añadir la modificación tal como sigue a la declaración de la clase para que pueda detectar clics:

```
public class MainActivity extends ActionBarActivity implements View.OnClickListener {
```

Debe añadir lo siguiente a las importaciones que hace Java para que pueda trabajar con OnClickListener y poder acceder a la ImageView

```
import android.view.View;
import android.widget.ImageView;
```

Lo primero que necesita hacer es asociar el clic en la imagen con una respuesta por parte de la aplicación añadiendo el siguiente código al método onCreate()

```
ImageView visorImagen = (ImageView) findViewById(R.id.ImageView);  
visorImatge.setOnClickListener(this);
```

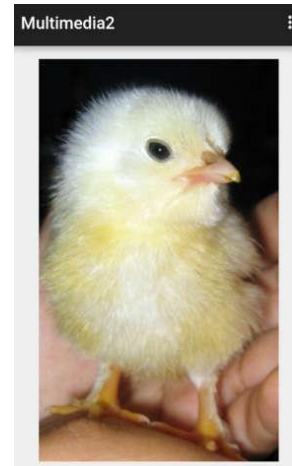
A continuación, hay que crear el método de respuesta al clic, que lo que hará es acceder al ImageView y cambiar el recurso al que está asociado:

```
@Override  
public void onClick (View v) {  
    // Cambiamos la imagen asignándole otro recurso  
    ImageView visorImagen = (ImageView) findViewById(R.id.ImageView);  
    visorImagen.setImageResource(R.drawable.pollo);  
}
```

Como puede ver, por cada recurso que añadimos al proyecto se crea un identificador (en realidad, un entero) que nos permite identificar los recursos y trabajar fácilmente. En este caso es R.drawable.pollo

Con las cadenas de texto y los controles del *layout* pasa lo mismo. Puede mirar (pero no cambiar) los archivos R.java que se encuentran en la carpeta */app/build/source/r/*, accesible desde la vista *project* del explorador del proyecto.

Si intenta ejecutar el programa, comprobaréis como al principio se ve la imagen inicial, pero cuando la toque cambia por la segunda imagen.



2. Animaciones

Un aspecto fundamental para dar una apariencia más dinámica a las aplicaciones con fuerte contenido multimedia son las animaciones, que con respecto al desarrollo para Android podemos clasificar en dos tipos: animaciones por interpolación (*tween*) y animaciones por fotogramas (*frames*). Verá ejemplos de ambos tipos.

Por cierto, una vez que haya experimentado con los dos tipos de animación verá que es posible combinarlos para crear efectos aún más especiales.

2.1. Animaciones por interpolación

En primer lugar, definiremos qué quiere decir eso de interpolación, para entender cómo funcionan este tipo de animaciones.

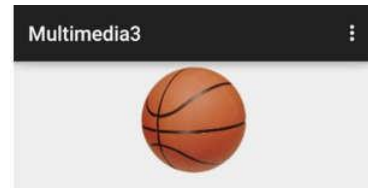
De manera intuitiva, podemos definir interpolación como la acción de encontrar los valores que tiene una función entre dos puntos conocidos. Nosotros hacemos interpolaciones cuando, por ejemplo, unimos dos puntos con una línea, o tres puntos con una curva.

Tween en inglés significa lo mismo que between, es decir, entre (entre dos puntos, por ejemplo). Se llaman animaciones tween porque encuentran lo que se debe hacer entre dos puntos determinados por el programador.

Para crear una animación por interpolación sólo debe definir el estado inicial de una imagen (de hecho, se puede aplicar a cualquier otra vista, como un texto) y el estado final deseado, y cuánto debe tardar la imagen en pasar del estado inicial al final, y el interpolador ya hace el resto. Los parámetros que puede cambiar de nuestra imagen son la escala, la posición y la rotación. Es decir, podemos hacer que la imagen vaya cambiando de tamaño, que se vaya moviendo o que vaya girando. O que se produzca más de un efecto al mismo tiempo, como verá enseguida.

También se puede cambiar la transparencia de la imagen, lo que permite hacer que las imágenes aparezcan o desaparezcan de forma gradual.

Para poder experimentar con las animaciones, hay que crear un nuevo proyecto Android (la llamaremos Multimedia3), añadiéndole una imagen y un `ImageView` que nos la muestre. Le recomendamos que utilice una imagen en formato PNG con un motivo claro (una pelota, un coche, un animal ...) y con el fondo transparente, de modo que no se vea más que el motivo principal. En general será más fácil si se trata de un dibujo que de una foto. Centráis la imagen en la parte superior *del* layout, como se ve en la figura.



Para apreciar mejor las animaciones, hará que se inicien cuando hacemos clic en la imagen. Hay que añadir un método como el siguiente en la clase `MainActivity` que llenaréis más adelante con las instrucciones pertinentes:

```
public void onClick (View v) {  
    }  
}
```

Y a continuación, debe buscar la propiedad de la imagen `onClick`, y escribir `onClick` para que al hacer clic en la imagen se llame el método que acabamos de escribir. Esto genera un atributo en la definición del elemento en el `layout.xml` que hace que cuando el usuario pulse el `ImageView` se llame el método `onClick()`

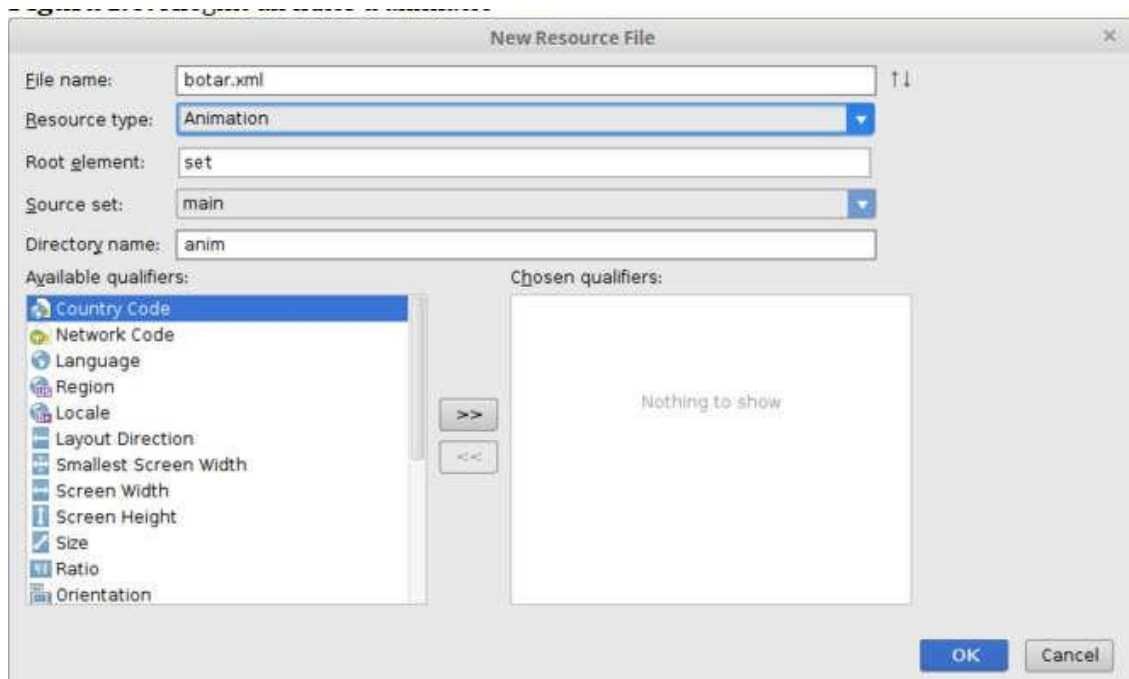
```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
  
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"  
    android:layout_height="match_parent" tools:context=".MainActivity">  
  
    <ImageView  
        android:id="@+id/imageView1"  
        android:layout_width="match_parent"  
        android:layout_height="116dp"  
        android:onClick="onClick"  
        android:src="@drawable/basketball"/>  
</RelativeLayout>
```

Haremos cuatro animaciones diferentes:

- Botar la pelota.
- Magia: una pelota que aparece.
- Hacer rodar la pelota.
- Hacer venir de lejos la pelota.

Hacer botar la pelota

La primera animación que haréis consiste en hacer botar el balón. Para añadir una animación a su aplicación, en primer lugar, hay que crear una nueva carpeta llamada *anim/* dentro de la carpeta *res/* (pulsando el botón derecho sobre *res/* y *New/Android resource directory* y escogiendo de tipo *anim*). A continuación, hay que añadir un nuevo archivo XML que contendrá las especificaciones de la animación. Pulse el botón derecho sobre la carpeta *res/* y vaya a *New / New resource file*. Llámelo, por ejemplo, **botar.xml**. El tipo de recurso debe ser **Animation**, y el **Root Element** puede ser **set**, como se puede ver en la figura. Enseguida veremos qué significan estas opciones, de momento es suficiente saber que son los diferentes tipos de animaciones disponibles.



Recuerde: los nombres de todos los archivos (imágenes, XML, etc.) que agregue a sus proyectos deben ser en minúsculas y sin espacios ni signos (sólo el guión bajo).

El contenido del archivo botar.xml debe ser el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
  <translate
    android:interpolator="@android:anim/accelerate_interpolator"
    android:repeatCount="infinite"
    android:repeatMode="reverse"
    android:fromYDelta="0%p"
    android:toYDelta="80%p"
    android:duration="1000"/>
</set>
```

Vemos rápidamente qué significan estas líneas:

- set permite definir una animación compuesta de más de una transformación. En este caso no se aplica, pero de esta manera se puede ampliar fácilmente más adelante.
- xmlns:android define el espacio de nombres android para que las líneas siguientes puedan usarlo como prefijo para el resto de opciones.
- translate indica que comienza una transformación de posición, queremos mover la imagen.
- interpolator es la función que se utilizará para calcular los valores intermedios. Si es linear quiere decir que los valores cambian a ritmo constante, y si es accelerate quiere decir que los valores cambian cada vez más rápido. En este caso es el que nos interesa, porque la pelota caiga cada vez más rápido.
- repeatCount define cuántas veces se debe repetir el efecto. En este caso, queremos que se repita para siempre.
- repeatMode tiene dos posibles valores: restart haría que al caer la pelota esta volviera directamente al comienzo, mientras que reverse haría que el balón pudiera hacer el recorrido inverso al que ha hecho para caer.
- fromYDelta establece en qué punto (de la y, es decir, la coordenada vertical) comienza la animación. Se puede poner un valor absoluto en píxeles ("140"), un valor relativo a la vista

("40%") o a su pariente, en el *layout* de la aplicación, indicando un valor tipo "50% p".

- `toYDelta` establece en qué punto de la coordenada vertical termina la animación. El formato es el mismo que para `fromYDelta`
- `duration` establece la duración de la animación en milisegundos.

Para las animaciones de tipo `translate` también están las opciones `fromXDelta` y `toXDelta` equivalentes a los de la Y que acabamos de ver.

Para que la animación se visualice, hay que cargarla, asociarla a la imagen y ponerla en marcha. Vaya al método `onClick()` que ha escrito antes y añada lo siguiente:

```
ImageView imagen = (ImageView)findViewById(R.id.imageView1);
Animation animacionPelota = AnimationUtils.loadAnimation(this, R.anim.botar);
imagen.startAnimation(animacionPelota);
```

Tenga en cuenta que se ha creado un identificador para la animación con el nombre del archivo donde la ha definido.

Magia: una pelota que aparece

Menudo utilizará animaciones para hacer aparecer imágenes y otros elementos de forma gradual, para dar un aspecto más interesante a su aplicación. Hacerlo es muy sencillo, basta con que cree otro archivo llamado `res/anim/aparecer.xml` y que le deis el siguiente contenido:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <alpha
        android:fromAlpha="0"
        android:toAlpha="1"
        android:duration="2000"/>

</set>
```

La estructura es muy similar a la del archivo de animación anterior, pero en este caso nos encontramos algunos elementos nuevos:

- alpha se refiere al grado de transparencia de una imagen, o sea que las animaciones de este tipo lo que harán es cambiar la transparencia con el tiempo.
- fromAlpha es el valor inicial de transparencia; 0 significa transparente del todo, es decir, invisible.
- toAlpha es el valor final de transparencia; 1 significa opaco del todo.

Para que su aplicación utilice esta nueva animación, hay que editar el método onClick() y hacer que el orden loadAnimation cargue R.anim.apararecer que es la nueva animación. Intente ejecutar la aplicación para notar el efecto. Como veis, en este caso la animación no se repite si no vuelva a hacer clic en la imagen.

Hacer rodar el balón

Si lo que desea ver es el balón rodando de un lado a otro, hay que añadir un nuevo archivo de animación llamado rodar.xml con el siguiente contenido:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
  <rotate
    android:interpolator="@android:anim/linear_interpolator"
    android:repeatCount="infinite"
    android:repeatMode="reverse"
    android:fromDegrees="0"
    android:toDegrees="360"
    android:pivotX="50%"
    android:pivotY="50%"
    android:duration="4000"/>
  <translate
    android:interpolator="@android:anim/linear_interpolator"
    android:repeatCount="infinite"
    android:repeatMode="reverse"
    android:fromXDelta="-50%p"
    android:toXDelta="50%p"
    android:duration="4000"/>
</set>
```

La novedad, en este caso, es que hay dos transformaciones: la pelota gira (rotate) pero también se desliza. Por eso hay que escribir dos entradas. En la sección rotate hay algunas opciones nuevas:

El orden en el que se escriben las diferentes transformaciones en un fichero de animaciones es fundamental, porque es el orden en el que se aplican. Pruebe a cambiar de orden el `rotate` y el `translate` y comprobaréis la diferencia.

- `fromDegrees` en qué ángulo (medido en grados) comienza la rotación.
- `toDegrees` en qué ángulo termina la rotación. En este ejemplo termina a los 360 ° para que haga la vuelta completa. Si quieres que dé más vueltas puede poner un valor mayor que 360, por ejemplo 720 para que dé dos vueltas cada vez.
- `pivotX` y `pivotY` indican el punto de la imagen respecto al cual estagirá: no es lo mismo girar respecto del centro, como en este caso, que girar respecto de un lado.

Por cierto, recordad que debéis editar el `onClick()` para que cargue esta animación y no otra.

Hacer venir de lejos el balón

Finalmente, creará una animación que muestre la imagen que viene de lejos y al mismo tiempo girando. Esta animación quedaría muy bonita con la imagen de una portada de periódico, como en las películas antiguas donde se acostumbraba a hacer efecto cuando salía alguna noticia en los periódicos.

Agregue otro archivo de animación que puede llamar **venir.xml** y escriba el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
<rotate
    android:interpolator="@android:anim/linear_interpolator"
    android:duration="500"
    android:repeatCount="8"
    android:pivotX="40%"
    android:pivotY="40%"
    android:fromDegrees="0"
    android:toDegrees="360"/>

<scale
    android:interpolator="@android:anim/linear_interpolator"
    android:repeatCount="0"
    android:duration="4000"
    android:pivotX="50%"
    android:pivotY="50%"
    android:fromXScale="0"
    android:toXScale="1"
    android:fromYScale="0"
```

```
android:toYScale="1"/>
</set>
```

Ahora, en el **rotate** hemos cambiado el punto de pivote para que la imagen gire un poco más y no sólo dé vueltas sobre sí misma.

Reproducción de animaciones

Desde la versión de Android Honeycomb (3.x) se puede añadir una opción en la etiqueta set para reproducir las operaciones de animación una detrás de la otra y no todas a la vez: <set android:ordering = "sequentially">

En cuanto a la transformación de escala scale sus parámetros son bastante evidentes después de haber trabajado con las otras transformaciones. Básicamente hay que indicar la escala inicial y final en X e Y.

Podemos asociar una misma animación a varios visores al mismo tiempo, para dar un estilo uniforme en nuestra aplicación.

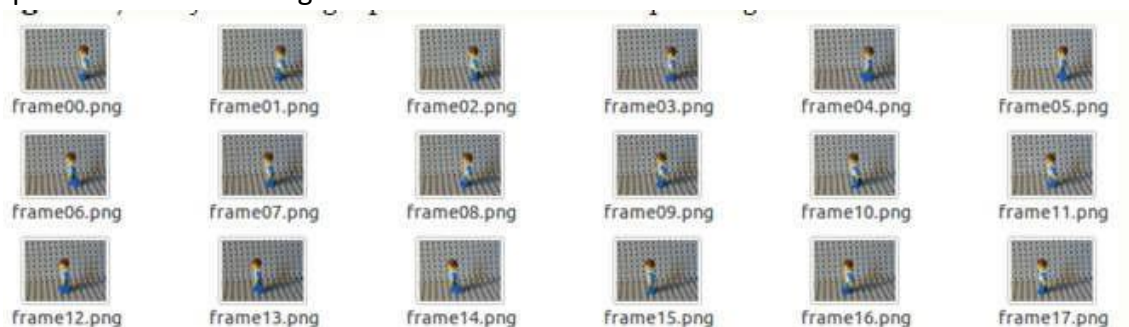
2.2. Animaciones por fotogramas

A menudo, las transformaciones que nos proporciona Android no nos sirven para conseguir la animación que necesitamos en nuestra aplicación y es necesario generarlas con programas específicos de animación. En este punto tenemos diferentes posibilidades, como generar un vídeo o también, si la animación no es demasiado larga, podemos generar un conjunto de imágenes con los diferentes pasos de la animación, de modo que pasando rápidamente las imágenes del usuario tenga la sensación de presenciar movimiento.

Fotogramas a alta velocidad

Cada una de las imágenes de la secuencia recibe el nombre de fotograma, y al fin y al cabo el cine, la televisión y los videojuegos, lo que hacen es mostrar fotogramas a alta velocidad (como mínimo, 25 por segundo) para darnos la impresión de que estamos viendo movimiento continuo.

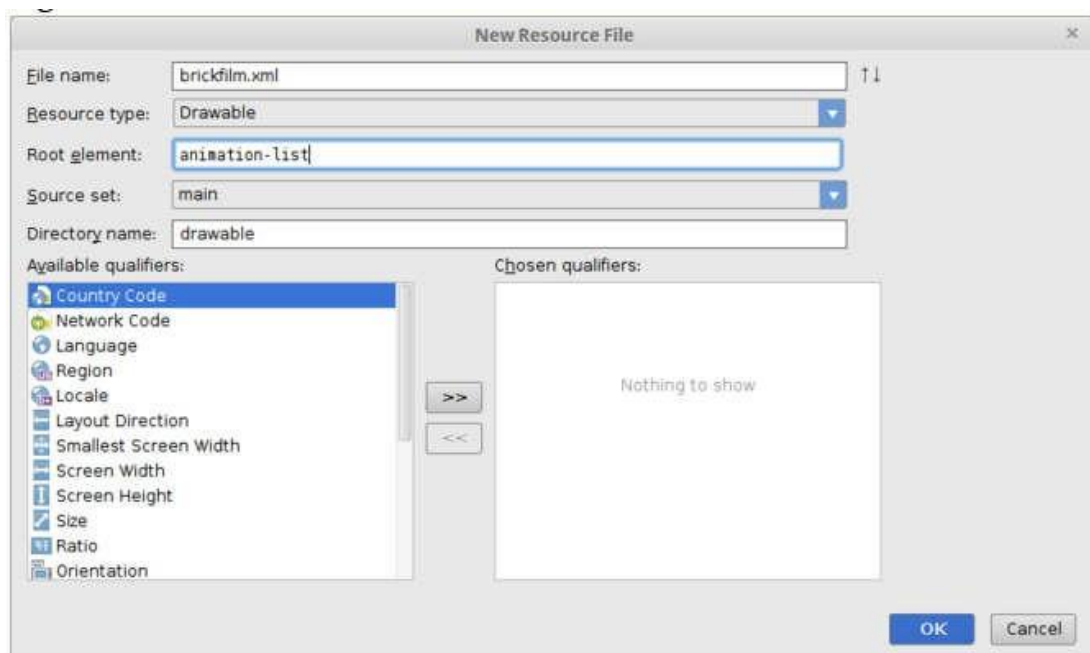
Comience un nuevo proyecto Android (en el ejemplo lo llamaremos Multimedia4). Encuentre un conjunto de imágenes utilizables como animación, como por ejemplo el que se muestra en la figura.



Con ocho o diez imágenes es suficiente para apreciar el efecto, no debe utilizar muchas más imágenes.

Aunque el efecto es más evidente si las imágenes forman parte de una secuencia, también puede utilizar imágenes independientes.

En primer lugar, asegúrese de que los archivos de imagen tienen un nombre adecuado para añadirlos a los proyectos: nombres en minúscula y sin espacios y sin signos que no sean guiones bajos. A continuación, arrastre las imágenes en la carpeta `res/drawable`. A continuación, añadir un nuevo archivo XML de tipo (Resource Type) `drawable` y escriba en Root Elemento: `animation-list` que es una lista de animación. Tal como muestra la figura, darle un nombre válido en el archivo y continúe.

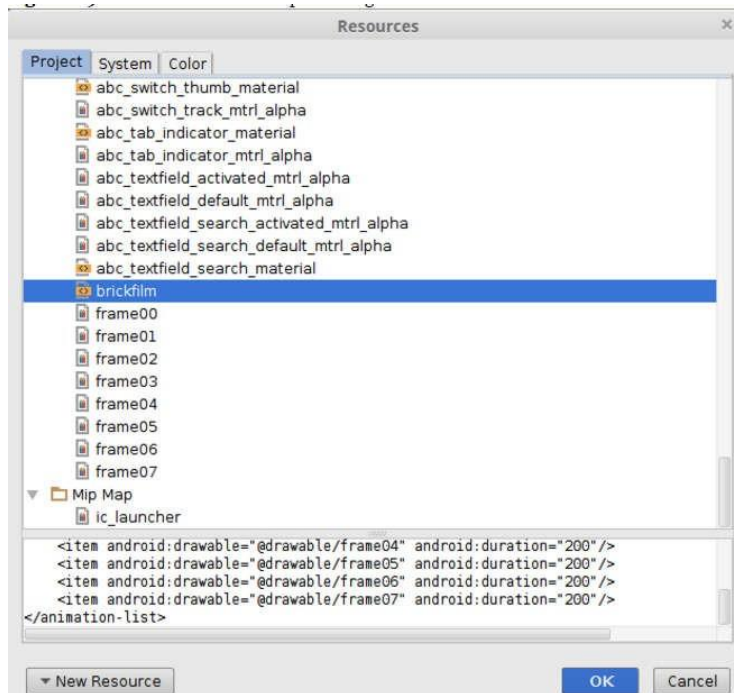


A continuación, edite el archivo XML para que quede de la siguiente manera (en vez de "frameXX", deberá escribir el nombre de las imágenes que ha agregado previamente, sin la extensión):

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:drawable="@drawable/frame00" android:duration="200"/>
  <item android:drawable="@drawable/frame01" android:duration="200"/>
  <item android:drawable="@drawable/frame02" android:duration="200"/>
  <item android:drawable="@drawable/frame03" android:duration="200"/>
  <item android:drawable="@drawable/frame04" android:duration="200"/>
  <item android:drawable="@drawable/frame05" android:duration="200"/>
  <item android:drawable="@drawable/frame06" android:duration="200"/>
  <item android:drawable="@drawable/frame07" android:duration="200"/>
</animation-list>
```

Con este archivo se define cuál es la secuencia de imágenes que formarán la animación y cuál es la duración de cada fotograma, en milisegundos.

A continuación, hay que crear un *ImageView* en el *layout* y, a la propiedad *src*, asociarle la animación que acaba de crear (en este ejemplo se llama *brickfilm*), como se muestra en la figura.



Si intenta ejecutar la aplicación sólo verá el primer fotograma; hay que decirle a la animación que se ejecute. Para ello, vaya al *MainActivity.java* y añada las siguientes instrucciones al final del método *onCreate()*:

```
ImageView imatge = (ImageView) findViewById(R.id.imageView);
AnimationDrawable animacio = (AnimationDrawable) imatge.getDrawable();
animacio.start();
```

Compruebe que se han añadido los *imports* y ejecute la aplicación. Básicamente lo que ha hecho con estas instrucciones es acceder a la imagen, obtener lo que está dibujando con `getDrawable()` (en este caso, la animación, porque así se lo ha indicado), y decirle a esta animación que empiece a ejecutarse.

Como veis, la animación se repite de forma indefinida. Existe la opción de decirle que se detenga al terminar, si desea probarla vaya al archivo que define la animación (`brickfilm.xml` en este ejemplo) y añada la siguiente opción en la etiqueta `animation-list` para que quede:

```
<animation-list xmlns:android="http://schemas.android.com/apk/res/android" android:oneshot="true">
```

Las animaciones se pueden aplicar a botones y otros controles. Esto se suele utilizar en juegos y aplicaciones similares, para darles un aspecto más lúdico y dinámico.

Las animaciones por fotogramas se pueden aplicar tanto a la *view* que estamos usando como a su fondo (*background*). Piense que, como las imágenes en formato PNG pueden tener transparencia, sería posible tener una imagen de fondo y una animación encima, o incluso una animación encima de la otra.

3. Sonido y música

A continuación, verá cómo añadir archivos de sonido o música en sus aplicaciones y cómo reproducirlos de manera más o menos sofisticada. También se pueden reproducir los archivos de audio que hay almacenados en el dispositivo, o incluso acceder a audio *online*.

Los formatos de audio soportados por Android son, principalmente: mp3, ogg, flac y wav.

3.1. Reproducción básica

Si sólo quiere que su aplicación reproduzca una pista de audio cuando se pone en marcha, los pasos a seguir son sencillos.

En primer lugar, se debe añadir un archivo de audio con un formato compatible con Android y un nombre de archivo compatible con Java. Hay que crear una nueva carpeta en *res/* que se llame *raw/* y arrastrar el archivo de audio a esta carpeta.

Para escuchar audio no hay que añadir ningún control en el *layout*, creará dinámicamente un `MediaPlayer` que es el objeto encargado de reproducir audio y vídeo. Para ello sólo hay que añadir las siguientes líneas al final del `onCreate()` de la *activity* principal nombrada que es el nombre del archivo de audio que hemos añadido,

sin la extensión):

```
MediaPlayer so = MediaPlayer.create(this, R.raw.nompista);  
so.start();
```

Puede comprobar que al abrir la aplicación se reproduce el archivo de sonido. Eso sí, sin opción a detenerlo ni nada parecido.

3.2. Controles de reproducción

Aunque es posible controlar el MediaPlayer con botones y otros elementos estándar, hay un *widget* específico para esta tarea que llamado MediaController. Este tipo de control se puede añadir desde el editor *del* layout, pero es más sencillo añadirlo desde el código directamente.

Primeramente, añade a la actividad el siguiente:

```
public class MainActivity extends ActionBarActivity implements MediaController.Media  
PlayerControl {
```

Con este código indica que su actividad llevará a cabo las funciones de control de sonido que ofrece MediaController. Hay que añadir *un* import con **android. Widget. MediaController**. Entonces, el Android Studio indicará un error a la actividad, diciendo que debe definir los métodos abstractos de MediaPlayerControl. La manera más sencilla de resolverlo es situarse sobre el error y hacer clic en la opción *Implement methods* de la bombilla de error (o pulsar Alt + Enter entonces añadirá automáticamente diez métodos (*canPause* hasta *start*), que iréis utilizando para añadir funcionalidad a su aplicación.

También hay que ir al archivo main. Xml y asignarle al RelativeLayout un ID, por ejemplo, *vista_controls_so*, como podéis ver a continuación:


```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:id="@+id/vista_controls_so" tools:context=".MainActivity">
```

A continuación, añadir los siguientes atributos a la clase de la actividad:

```
MediaPlayer sonido;
MediaController controles;
```

A continuación, vaya al `onCreate()` elimine las líneas que voy añadir para la primera versión del programa de reproducción de sonido y añada las siguientes líneas:

```
sonido= MediaPlayer.create(this, R.raw.bluestutorial);
controles = new MediaController(this);
controles.setMediaPlayer(this);
controles.setAnchorView(findViewById(R.id.vista_controls_so));
```

Con estas instrucciones, lo que está haciendo es crear *MediaPlayer* para reproducir la pista de audio, y luego crear *MediaController* para disponer de unos controles para el audio. A continuación, asociar el código del *MediaController* a la actividad actual, y finalmente defina a qué vista de la aplicación se debe dibujar el *MediaController*.

¿Que aún no se ve nada? Esto es porque, por defecto, el *MediaController* se esconde automáticamente. Es necesario que el obliguéis a mostrarse cuando toque la pantalla, añadiendo el siguiente método a su actividad:

```
@Override
public boolean onTouchEvent(MotionEvent event) {
    controles.show();
    return false;
}
```

Si al **show()** se le pasa un número entero n , los controles de audio permanecerán visibles durante n segundos; si le pasas un cero, los controles no se esconderán.

De momento, estos controles no hacen gran cosa porque tiene que rellenar sus métodos con código que haga lo que desee. Comience por lo más básico: hacer que el botón de reproducir (*play*) ponga en marcha la música. Por eso hay que ir al método `start` (creado automáticamente para implementar `MediaController.MediaPlayerControl`) y añadir:

```
sonido.start();
```

Con lo que le decís al reproductor de audio que comience a reproducir la pista de audio. También hay que ir al método `canPause()` y hacer que vuelva `true` para que el botón de reproducir sea activo. Con esto ya debería escuchar cuando toque el play.

Para que la interacción entre los controles y el reproductor de audio sea más completa, hay que ir llenando los métodos que se han generado para que queden como se ve a continuación:

```

@Override
public int getCurrentPosition() {
    // Devuelve la posición actual en la pista de audio
    return so.getCurrentPosition();
}

@Override
public int getDuration() {
    // Devuelve la duración de la pista de audio

    return so.getDuration();
}

@Override
public boolean isPlaying() {
    // cierto cuando se está reproduciendo audio

    return so.isPlaying();
}

@Override
public void pause() {
    // Cuando el usuario toca el botón de pausa
    so.pause();
}

@Override
public void seekTo(int pos) {
    // Para ir a una posición de la pista
    so.seekTo(pos);
}

```

La duración de las pistas multimedia y las posiciones se miden en milisegundos.

El resto de métodos se pueden dejar tal como están ahora mismo. Con esto ya tendrá un reproductor de audio con funciones de reproducción, pausa y búsqueda de una posición.

4. Vídeo

Reproducir vídeos a los dispositivos Android es muy sencillo, como veréis en este apartado. Para empezar, añadiremos los vídeos como recursos a su aplicación de ejemplo. También se pueden reproducir vídeos de Internet o almacenados en el dispositivo.

Los formatos de video más ampliamente soportados por los dispositivos Android son el MPEG-4 (.mp4) y el 3GPP (.3gp).

Para visualizar un vídeo en sus aplicaciones, es necesario que agregue el siguiente elemento en nuestra aplicación: dentro de la pestaña, al editor gráfico de *layouts* **Images & Media**, agregue un `VideoView`, que será la pantalla con la que se verá el

vídeo.

Para añadir un vídeo como recurso, debe crear una carpeta llamada *raw/* dentro de la carpeta *res/* y arrastrar un vídeo con el formato adecuado y con un nombre compatible con Java, como es habitual. A su ejemplo, este vídeo se llamará *castell.3gp*.

A continuación, hay que ir al código de la actividad y añadir lo siguiente:

```
VideoView visor = (VideoView)findViewById(R.id.videoView1);
Uri video = Uri.parse("android.resource://" + getPackageName() + "/"
    + R.raw.castell);
visor.setVideoURI(video);
visor.setMediaController(new MediaController(this));
visor.start();
```

Tenga en cuenta que el VideoView no acepta directamente un identificador de recurso como vídeo, por ello hay que acceder con la expresión que forma una dirección URI (Universal Resource Identifier, identificador universal de recursos) accediendo a los recursos empaquetados de la aplicación que está construyendo.

A continuación se asocia este recurso al VideoView, se crea un MediaController para tener controles de reproducción y, finalmente, se hace que el vídeo comience. Claro que para que todo esto funcione hay que añadir los *importes* adecuados.

Desgraciadamente, la reproducción de vídeo no siempre funciona en el emulador de Android, por lo que es posible que sólo escuche el sonido del vídeo. En este caso necesitará pasar la aplicación a un dispositivo real para poder visualizarlo.

5. Geolocalización

GPS significa Global Positioning System, es decir, sistema de posicionamiento global, y es un sistema que permite conocer la situación propia (longitud y latitud) mediante señales enviadas por satélites.

Una de las características más interesantes de los dispositivos móviles es su capacidad de detectar su posición geográfica para dar al usuario una serie de información, tales como mostrar su situación en un mapa, listar los recursos más cercanos (restaurantes, gasolineras ...), registrar su recorrido mientras marcha por la montaña, etc. El método más conocido de geolocalización, es decir, de obtención de la posición geográfica, es el sistema **GPS**.

Hay otros métodos de geolocalización, como detectar las redes Wi-Fi presentes y deducir a partir de las redes identificadas cuál es la posición geográfica. Este método no es tan preciso como el GPS, pero por otra parte puede funcionar bajo tierra o dentro de edificios, donde la señal GPS generalmente no llega.

En este apartado verá cómo obtener las coordenadas GPS y luego las utilizará para visualizar su posición en un mapa.

5.1. Coordenadas GPS

Primeramente, hay que dar permiso a su aplicación para acceder al sensor GPS. Para ello, después de crear una nueva aplicación Android, edite el archivo `AndroidManifest.xml` y añada la siguiente línea justo antes de la etiqueta `<application>`:

La posición del usuario del dispositivo es un aspecto más de su privacidad, por eso es necesario que la aplicación solicite permiso para acceder al GPS.

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

Para que su actividad pueda recibir las actualizaciones de la posición del dispositivo, es necesario que implemente `LocationListener`, entonces edite la cabecera de la actividad para que quede de la siguiente manera:

```
public class MainActivity extends ActionBarActivity implements LocationListener {
```

Añádase el `import android.location.LocationListener` si no lo ha hecho ya el Android Studio. Verá que nos marca como error el nombre de la clase que no implementa los métodos del `LocationListener`; al mensaje que aparece, haga clic en "Implemento methods" para que los añada. Concretamente, los métodos que debemos implementar son:

- **onLocationChanged:** es llamado cuando la localización GPS cambia, típicamente para que el dispositivo se desplaza a otra posición. Sirve para actualizar la posición a la aplicación.
- **onProviderDisabled:** se hace la llamada cuando el usuario ha deshabilitado el servicio GPS.
- **onProviderEnabled:** al contrario que el anterior, es llamado cuando el usuario ha habilitado el servicio.

- **onStatusChanged:** es llamado cuando el servicio cambia de estado, de activado a desactivado o viceversa. Nos sirve para saber cuándo el *provider* es incapaz de obtener la información de posición o si la recupera después de un tiempo de inactividad.

El parámetro status nos informa de este hecho, toma tres valores: OUT_OF_SERVICE, TEMPORARILY_UNAVAILABLE y AVAILABLE.

Antes de completar los métodos anteriores hay que activar el sistema de localización añadiendo el siguiente código al onCreate () de su actividad:

```
LocationManager gestorLoc =
    (LocationManager) getSystemService(Context.LOCATION_SERVICE);
gestorLoc.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1000, 1, this);
```

Como siempre, habrá que añadir los *importes* que les diga el Android Studio. Con estas instrucciones se accede al servicio GPS y se pide que las actualizaciones de posición se envíen a esta actividad. Los argumentos numéricos son, respectivamente, el tiempo aproximado (en milisegundos) entre actualizaciones y la distancia mínima (en metros) que debe cambiar la posición para recibir una actualización. En una aplicación real habrá que poner valores adecuados.

Lo más importante es el método **onLocationChanged ()**, que es llamado cada vez que la posición del dispositivo cambia. Edite el fin de que quede como sigue:

```
@Override
public void onLocationChanged(Location location) {

    String text = "Posicion actual:\n" +
        "Latitud = " + location.getLatitude() + "\n" +
        "Longitud = " + location.getLongitude();

    Toast.makeText(getApplicationContext(), text,
        Toast.LENGTH_LONG).show();

}
```

El método recibe la posición actual y lo que hace es formar un texto con la latitud y longitud, que son los parámetros que definen la posición geográfica.

Las toast (tostadas) son pequeños mensajes temporales que podemos utilizar para mostrar información al usuario.

También llenaréis los métodos siguientes para que la aplicación notifique al usuario cuando la cobertura GPS se pierde o se recupera:

```
@Override
public void onProviderDisabled(String provider) {
    Toast.makeText(getApplicationContext(),
        "GPS desactivado por el usuario",
        Toast.LENGTH_LONG).show();
}

@Override
public void onProviderEnabled(String provider) {
    Toast.makeText(getApplicationContext(),
        "GPS activado por el usuario",
        Toast.LENGTH_LONG).show();
}

@Override
public void onStatusChanged(String provider, int status, Bundle extras) {

    String missatge = "";
    switch (status) {
        case LocationProvider.OUT_OF_SERVICE:

            missatge = "GPS status: Out of service";
            break;
        case LocationProvider.TEMPORARILY_UNAVAILABLE:
            missatge = "GPS status: Temporarily unavailable";
            break;
        case LocationProvider.AVAILABLE:
            missatge = "GPS status: Available";
            break;
    }

    Toast.makeText(getApplicationContext(),
        missatge,
        Toast.LENGTH_LONG).show();
}
}
```

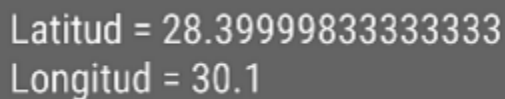
Ahora bien, ¿cómo puede probar esta aplicación? La primera opción y la más realista es ejecutarla en un dispositivo real, pero si esto no es posible, tiene dos opciones que no son demasiado realistas pero que le permitirán probarla.

En primer lugar, ejecute la aplicación en el emulador. Con la aplicación en marcha, abra una ventana del terminal y escriba lo siguiente:

```
telnet localhost 5554
geo fix 30.1 28.4
```

Con ello, lo que haga es conectarse al emulador y enviarle la longitud y la latitud. El número 5554 es el puerto por defecto por el que puede conectarse al emulador. El puerto específico que utiliza su emulador semuestra en su barra de aplicación. Si miráis la pantalla del emulador, verá como aparece la información que le ha enviado, similar a la *toast* de la figura .

La latitud es la distancia (angular) de un punto en el ecuador, es decir, en el eje sur-norte. La longitud es la distancia (angular) de un punto en el meridiano de Greenwich, es decir, en el eje este-oeste.



Latitud = 28.399998333333333
Longitud = 30.1

La otra opción, sólo disponible si utiliza el entorno Android Studio, es ir a *Tools/Android/Android Device Monitor*, concretamente en la pestaña *Emulator Control*, que contiene diferentes herramientas para controlar el emulador: para simular una llamada telefónica, la recepción de un SMS ... Si sigue bajando dentro de esta ventana encontrará una herramienta llamada *Location Controles* que permite enviar la posición en el emulador (pestaña GPS), y que se muestra en la figura.

Si introduce una coordenada y pulse el botón *Send* la enviarás al emulador. Pero esta herramienta le ofrece una manera mucho mejor de introducir coordenadas GPS: usar archivos **GPX**.

Los archivos GPX (GPS eXchange formato) son archivos XML que contienen recorridos expresados como secuencias de coordenadas GPS, con un tiempo de llegada asociado a cada punto.

Si abre la pestaña GPX puede cargar un archivo en este formato y enviarlo al emulador con el botón Play. También hay un control llamado Speed que le permite enviar las coordenadas más rápidamente si desea, sobre todo porque a menudo estos archivos se han registrado caminando o en bicicleta y, por tanto, se mueven lentamente por el mapa.

De esta manera se puede ver cómo se van actualizando las coordenadas a la aplicación, haciendo el recorrido del mapa.

5.2. Visualización de mapas

Apenas se ha visto cómo leer la posición geográfica del dispositivo mediante el servicio GPS, pero sólo lo ha podido utilizar para mostrar las coordenadas por pantalla con una simple toast. Su aplicación sería más atractiva si en vez de escribir unos números por pantalla dibujara el mapa con la situación actual y la fuera actualizando a medida que se mueve. Esto es lo que haréis en este apartado.

Como probablemente ya saben, la fuente más importante de mapas e imágenes por satélite es **Google Maps**, y dado que Google ha sido la creadora de Android lo ha puesto bastante fácil porque utilice sus mapas a las aplicaciones Android.

Para poder hacerlo, es necesario configurar su proyecto para que utilice las Google API y los Google Play services, es decir, el conjunto de bibliotecas y servicios para acceder a las aplicaciones y recursos de Google.

En primer lugar, configuraremos un emulador para trabajar con las google APIs. Tendremos que ir al Android SDK Manager y descargar la última versión de:

- **Google APIs Intel x86 Atom System Image** de la última versión del SDK.
- **Google APIs** de la última versión del SDK.
- **Google Play services**, que encontrará en la sección extras.

Con el fin de ejecutar un proyecto que utilice las **Google APIs** necesitamos disponer de un AVD con las librerías y aplicaciones necesarias. Así, compruebe que el *target* de su emulador sea "Google APIs".

Ahora que ya lo tenemos todo listo para poder crear el proyecto, en la selección de actividades elija **Google Maps Activity** y deje la configuración predeterminada.

En el manifest encontraremos un contenido similar al siguiente:

```
<resources>
<!--
TODO: Before you run your application, you need a Google Maps API key.

To get one, follow this link, follow the directions and press "Create" at the
end:

https://console.developers.google.com/flows/enableapi?apiid=maps_android_back
end&keyType=CLIENT_SIDE_ANDROID&r=FF:47:8C:B5:B3:8A:XX:13:79:XX:E5:DB:4F:XX:8
F:6A:A5:BD:FF:F5%3Bioc.xtec.cat.multimediamaps

You can also add your credentials to an existing key, using this line:
FF:47:8C:B5:B3:8A:XX:13:79:XX:E5:DB:4F:XX:8F:6A:A5:BD:FF:F5;ioc.xtec.cat.mult
imediamaps
```

Once you have your key (it starts with "AIza"), replace the "google_maps_key" string in this file.

```
-->
<string name="google_maps_key" translatable="false"
templateMergeStrategy="preserve">
    YOUR_KEY_HERE
</string>
</resources>
```

Acceda a la URL que se os propone desde un navegador web y, después de crear un nuevo proyecto, se le permitirá crear una clave para poder utilizar la API de Google Map. Para añadir más aplicaciones ligadas a una misma clave debemos añadir líneas con el SHA1, un ";" y el *package name* de la aplicación. Una vez generada, copie el valor del campo **CLAVE DE LA API** y péguelo en el XML como contenido de la etiqueta string (sustituiremos el texto YOUR_KEY_HERE).

Con estos pasos ya podremos ejecutar el emulador para comprobar el funcionamiento del mapa, pero antes observamos cuáles son las modificaciones que ha hecho el Android Studio para que todo funcione. Si abre el archivo **AndroidManifest.xml** podrá observar el siguiente contenido:

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="ioc.xtec.cat.multimediamaps" >

<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
<!--
The ACCESS_COARSE/FINE_LOCATION permissions are not required to use
Google Maps Android API v2, but are recommended.
-->
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme"
    >
    <meta-data
        android:name="com.google.android.gms.version"
        android:value="@integer/google_play_services_version" />
    <meta-data
        android:name="com.google.android.maps.v2.API_KEY"
        android:value="@string/google_maps_key" />

    <activity
        android:name=".MapsActivity"
        android:label="@string/title_activity_maps" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>

</manifest>

```

Podemos observar que añade los siguientes permisos:

- android.permission.INTERNET: para acceder a Internet, necesario para descargar los mapas.

- `android.permission.ACCESS_NETWORK_STATE`: para detectar cambios en la red.
- `android.permission.WRITE_EXTERNAL_STORAGE`: permite escribir el almacenamiento del teléfono, en este caso es necesario para descargar archivos para generar la caché de los mapas.
- `com.google.android.providers.gsf.permission.READ_GSERVICES`: permite al API acceder a los servicios de Google.
- `android.permission.ACCESS_COARSE_LOCATION`: acceso a la posición aproximada mediante la red: wifi y torres de telefonía.
- `android.permission.ACCESS_FINE_LOCATION`: acceso a la posición más precisa, utilizando además el GPS para obtenerla.

Y además, añade dos etiquetas meta-data con la información sobre la versión de los *Google Play Services* y la clave que hemos añadido al archivo `google_maps_api.xml`.

El *layout* de la aplicación constará de un fragmento que será manipulado desde el fichero `MapsActivity.java`. Si nos fijamos con el código de la actividad, podemos ver como acaba llamando a `setUpMap()` que añade un marcador en el mapa en la coordenada (0,0) con el título de "Marker".

Ejecute el proyecto y visualizará un mapa del mundo con un marcador en la coordenada geográfica (0,0), es decir, a la latitud y longitud 0.

Para añadir controles de *zoom* a su aplicación de forma que pueda acercarse desde el emulador de una manera cómoda necesario que cambie el método `setUpMapIfNeeded()` para que quede de la siguiente manera:

```
private void setUpMapIfNeeded() {
// Do a null check to confirm that we have not already instantiated the map.
    if (mMap == null) {
// Try to obtain the map from the SupportMapFragment.
        mMap = ((SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.map))
            .getMap();
// Check if we were successful in obtaining the map.
        if (mMap != null) {
            setUpMap();
            UiSettings settings = mMap.getUiSettings();
            settings.setZoomControlsEnabled(true);
        }
    }
}
```

Es decir, añadimos las líneas `UiSettings settings = mMap.getUiSettings();` y `settings.setZoomControlsEnabled(true);` para obtener la configuración del mapa y

modificar el valor de `ZoomControlsEnabled` para ponerlo a `true`.

Podrá encontrar más métodos para configurar su mapa mediante la variable `mmap`; por ejemplo, si desea cambiar el tipo de mapa puede acceder al método `mMap.setMapType (int type)`, que recibe como argumento el tipo de Pruebe los diferentes valores y observe los cambios que se producen en la representación.

5.3. Seguimiento de su posición

Si queremos que el mapa se actualice con nuestra posición deberemos implementar `LocationListener` y, cada vez que se produzca un cambio de éstos, actualizar el mapa.

Así, haga que la clase `MapsActivity` implemente `LocationListener` y recuerde añadir los métodos que Android Studio nos exige.

```
public class MapsActivity extends FragmentActivity implements
LocationListener {
    ...
    @Override
    public void onLocationChanged(Location location) {

    }

    @Override
    public void onStatusChanged(String provider, int status, Bundle extras) {

    }

    @Override
    public void onProviderEnabled(String provider) {

    }

    @Override
    public void onProviderDisabled(String provider) {

    }
    ...
}
```

Desde los métodos `onStatusChanged`, `onProviderEnabled` y `onProviderDisabled` podríamos informar sobre los cambios que se produzcan en el servicio GPS; ahora, sin embargo, nos centraremos en el método `onLocationChanged`, donde tendremos que:

- Obtener la posición a partir del parámetro `location`.
- Crear la cámara (área del mapa que se visualiza en un momento determinado)

con la posición y un nivel de zoom.

- Desplazar la cámara en el punto propuesto.

LatLng almacena un par de coordenadas, la latitud y la longitud.

Así, añadiremos el siguiente código a `onLocationChanged`:

```
@Override
public void onLocationChanged (Location location) {
    // Creamos un LatLng a partir de la posición
    LatLng posicion = new LatLng (location.getLatitude(),
                                location.getLongitude ());
    // Añadimos la cámara con el punto generado antes y un nivel de zoom
    CameraUpdate cámara = CameraUpdateFactory.newLatLngZoom(posicion, 19);
    // Desplazamos la cámara en el nuevo punto
    mMap.moveCamera (camara);
}
```

Una vez hemos obtenido la posición, hacemos la llamada al método `newLatLngZoom (LatLng latLng, float zoom)` de `CameraUpdateFactory` y que recibe dos parámetros: el primero determinará el punto donde se situará la cámara y el segundo especificará el nivel de *zoom*, *que deberá tomar un valor entre 2.0f y 21.0f*. Finalmente, actualizamos la cámara del mapa.

Para que el *listener* comience a controlar los cambios de posición, recuerde añadir al `onCreate` el siguiente código:

```
LocationManager gestorLoc = (LocationManager) getSystemService
    (Context.LOCATION_SERVICE);
gestorLoc.requestLocationUpdates(LocationManager.GPS_PROVIDER,1000, 1, this);
```

Si intenta enviar unas coordenadas GPS en el emulador, comprobará cómo cambia la vista. Lógicamente, cuanto más *zoom* aplicamos, más evidente será el movimiento. Para ver el efecto es recomendable cargar un fichero GPX con el DDMS (Tools /

Android / Android Device Monitor).

5.4. Añadiendo marcas en el mapa

A menudo las aplicaciones que muestran mapas añaden marcas para señalar al usuario los puntos que le sean interesantes, como los restaurantes más cercanos, las paradas de metro o las ciudades que ha visitado. En este apartado verá cómo añadir estas señales (iconos) en su mapa, y cómo colocarlos donde desee. Para ello continuará con la aplicación anterior.

Para ir añadiendo las marcas a medida que va cambiando la posición tendremos que modificar el método `onLocationChanged`. El procedimiento será el siguiente:

- Creamos un elemento del tipo `MarkerOptions`.
- Añadimos las propiedades deseadas: Título (*title*), posición (*position*), descripción (*snippet*) e icono (*icon*).
- Añadimos el marcador en el mapa.
- Si hemos de manipular el marcador posteriormente, podemos guardar el objeto *Marker* que devuelve la función `addMarker`; una posible utilidad de guardarlo es hacer la llamada a `showInfoWindow()` para hacer que se muestre la información del marcador sin tener que hacer clic sobre él.

El código del método `onLocationChanged` quedará de la siguiente manera:

```
@Override
public void onLocationChanged (Location location) {

    // Creamos un LatLng a partir de la posición
    LatLng posicion = new LatLng (location.getLatitude (),
        location.getLongitude ());

    // Añadimos la cámara con el punto generado antes y un nivel de zoom
    CameraUpdate camara = CameraUpdateFactory.newLatLngZoom (posicion, 19);

    // Desplazamos la cámara en el nuevo punto
```

```

mmap.moveCamera (camara);

// Creamos y añadimos el marcador
MarkerOptions opciones = new MarkerOptions ();
opciones.title("Título");
opciones.position(posicipn);
opciones.snippet("Descripción del marcador");
opciones.icon (BitmapDescriptorFactory.defaultMarker
(BitmapDescriptorFactory.HUE_ORANGE));

// Aquí el añadimos el mapa y nos lo guardamos en una variable (ya se ve el
icono en el mapa pero hay que hacer clic para ver la información)
Marker marca = mmap.addMarker (opciones);

// Hacemos que se muestre la información sin tener que hacer clic
marca.showInfoWindow();
}

```

Si ejecutamos la aplicación y vamos enviando diferentes posiciones utilizando el *AndroidDevice Manager*.