IES DR LLUÍS SIMARRO
ENSENYAMENT PÚBLIC
XATIVA

Programación multimedia y dispositivos móviles

# UD08. Persistencia de datos I: Preferencias.

## Desarrollo de Aplicaciones Multiplataforma

**Anna Sanchis Perales**
**Enric Giménez Ribes**

# ÍNDICE

# 1. OBJETIVOS

Los objetivos a conocer y comprender son:

- Conocer las diferentes formas de almacenar datos de forma permanente.
- Utilizar las preferencias como mecanismo de almacenamiento.
- Añadir una lista de preferencia a nuestra aplicación.
- Conocer los diferentes tipos de opciones que podemos añadir en nuestras preferencias.

# 2. PERSISTENCIA DE DATOS

Las aplicaciones descritas hasta este tema representaban la información a procesar en forma de variables. El problema de estas variables es que dejan de existir en el momento en que la aplicación es destruida. En muchas ocasiones vamos a necesitar **almacenar información de manera permanente**. Las alternativas más habituales para conservar esta información son **los ficheros, las bases de datos o servicios a través de la red**. Estas técnicas no solo permiten mantener a buen recaudo los datos de la aplicación, sino que también vamos a poder compartir estos datos con otras aplicaciones y usuarios. De forma adicional, el sistema Android pone a nuestra disposición otros mecanismos para almacenar datos, como las **Preferencias** y **ContentProvider**.

A lo largo de este tema estudiaremos cómo utilizar una de estas técnicas, que son las Preferencias.

En los siguientes temas estudiaremos el acceso a bases de datos.

## 2.1. Alternativas para guardar datos permanentemente en Android

Existen muchas alternativas para almacenar información de forma permanente en un sistema informático. A continuación mostramos una lista de las más habituales utilizadas en Android:

- **Preferencias:** Es un mecanismo liviano que permite almacenar y recuperar datos primitivos en la forma de pares clave/valor. Este mecanismo es típicamente usado para almacenar los parámetros de configuración de una aplicación.
- **Ficheros:** Puedes almacenar los ficheros en la memoria interna del dispositivo o en un medio de almacenamiento removible como una tarjeta SD. También puedes utilizar ficheros añadidos a tu aplicación como recursos.
- **XML:** Se trata de un tipo de archivo que sigue un determinado estándar. Ampliamente utilizado en Internet y en muchos otros sitios (como en el Android SDK). Disponemos de las librerías SAX y DOM para manipular estos ficheros desde Android.
- **Base de datos:** Las APIs de Android contienen soporte para SQLite y para ROOM. Tu aplicación puede crear y usar base de datos local de forma muy sencilla.
- **Proveedores de contenidos:** Un proveedor de contenidos es un componente opcional de

una aplicación que expone el acceso de lectura / escritura de sus datos a otras aplicaciones. Está sujeto a las restricciones de seguridad que quieras imponer. Los proveedores de contenidos implementan una sintaxis estándar para solicitar datos (URIs) y un mecanismo de acceso para devolver los datos (similar a SQL). Android provee algunos proveedores de contenidos para tipos de datos estándar, tales como contactos personales, ficheros multimedia, ... Se estudiará más adelante.

- **Internet:** No te olvides que también puedes usar la nube para almacenar y recuperar datos. Se estudiará más adelante.

# 3. PREFERENCIAS

Las **preferencias** (**clase SharedPreferences o Preferences**) también pueden ser utilizadas como un mecanismo sencillo para almacenar ciertos datos que tu aplicación quiera conservar de forma permanente. Es un mecanismo sencillo que te permite almacenar una serie de variables con su nombre y su valor. Puedes almacenar variables de tipo booleano, real, entero y String.

Las preferencias son almacenadas en ficheros XML dentro de la carpeta /shared_prefs en los datos de la aplicación. Las preferencias del usuario se almacenan siempre en el paquete de la aplicación, seguido de "_preferences"(por ejemplo ).

La API para el manejo de estas preferencias es muy sencilla. Toda la gestión se centraliza en la clase SharedPrefences, que representará a una colección de preferencias. Una aplicación Android puede gestionar varias colecciones de preferencias, que se diferenciarán mediante un identificador único. Para obtener una referencia a una colección determinada utilizaremos el método getSharedPrefences() al que pasaremos el identificador de la colección y un modo de acceso. El modo de acceso indicará qué aplicaciones tendrán acceso a la colección de preferencias y qué operaciones tendrán permitido realizar sobre ellas. Así, tendremos tres posibilidades principales:

- **MODE_PRIVATE.** Sólo nuestra aplicación tiene acceso a estas preferencias.
- **MODE_WORLD_READABLE.** Todas las aplicaciones pueden leer estas preferencias, pero sólo la nuestra puede modificarlas.
- **MODE_WORLD_WRITABLE.** Todas las aplicaciones pueden leerlas y modificarlas.

Las dos últimas opciones son relativamente "peligrosas" por lo que en condiciones normales no deberían usarse. De hecho, se han declarado como obsoletas en la API 17 (Android 4.2).

En cuanto a los métodos para obtener las referencias hemos comentado uno, pero existen dos:

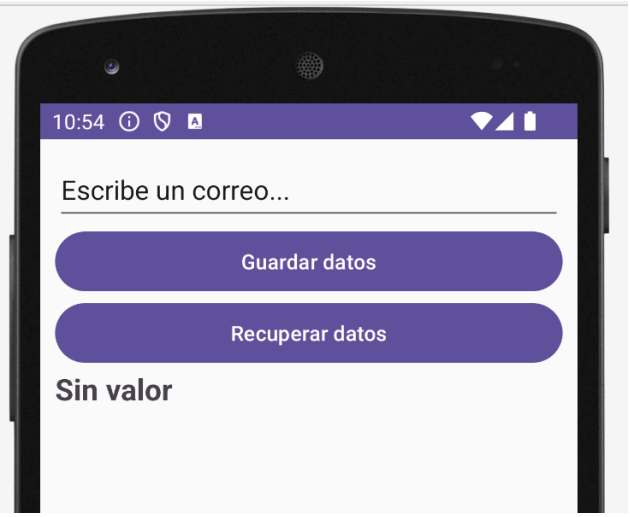
- **getSharedPreferences():** Te permite indicar de forma explícita el nombre de un fichero de preferencias. Puedes utilizarlo cuando necesites varios ficheros de preferencias o acceder al mismo fichero desde varias actividades.
- **getPreferences():** No tienes que indicar ningún nombre. Puedes utilizarlo cuando solo necesites un fichero de preferencias en la actividad donde se usa.

Una llamada a uno de estos dos métodos te devolverá un objeto de la clase SharedPreferences.

**Ejemplo Preferencias**

Vamos a hacer un pequeño ejemplo mostrando cómo escribir un valor (guardaremos un email) en las preferencias y cómo recuperarlo. Para ello crearemos un proyecto de ejemplo:

1. Creamos un proyecto denominado **Tema8App1**.
2. Vamos a diseñar un layout cuyo contenido será el siguiente:




3. Por lo que no iremos a la carpeta /res/layout y modificaremos el fichero activity_main.xml con el siguiente contenido:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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:orientation="vertical"
   android:padding="10dp"
   tools:context=".MainActivity" >

   <EditText
       android:id="@+id/edtCorreo"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:minHeight="48dp"
       android:text="Escribe un correo..."
       android:inputType="text" />
```

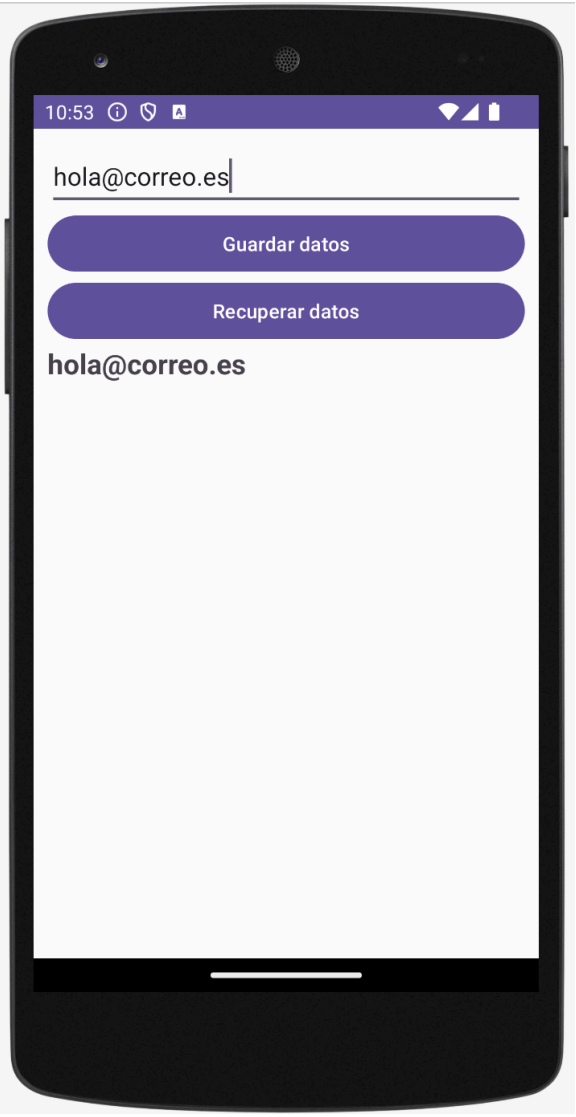
```
        <Button
            android:id="@+id/btnGuardar"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Guardar datos" />

        <Button
            android:id="@+id/btnRecuperar"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Recuperar datos" />

        <TextView
            android:id="@+id/tvValorGuardado"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Sin valor"
            android:textSize="20sp"
            android:textStyle="bold" />
    </LinearLayout>
```

4. Tras ello nos vamos a la clase principal donde añadiremos el siguiente código a la clase principal y ahora lo explicamos:

```
class MainActivity : AppCompatActivity() {

   private lateinit var binding: ActivityMainBinding


   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       binding = ActivityMainBinding.inflate(layoutInflater)
       setContentView(binding.root)

       // Obtenemos la referencia a la colección de preferencias
       val sharedPreferences = getSharedPreferences("MiPreferencia",
Context.MODE_PRIVATE)

       // Declaramos el evento onClick en el botón de guardar el correo,
       // de forma que salvamos los datos
       binding.btnGuardar.setOnClickListener {
           val editor = sharedPreferences.edit()
           editor.putString("email", binding.edtCorreo.text.toString())  // Puedes
usar putInt, putBoolean, etc., según el tipo de dato
           editor.apply()
       }

       // Declaramos el evento onClick en el botón de cargar el correo
```

```
        // de las preferencias
        binding.btnRecuperar.setOnClickListener {
            val valorRecuperado = sharedPreferences.getString("email", "Sin valor")
            binding.tvValorGuardado.text = valorRecuperado

        }

    }
}
```

5. A continuación, obtenemos una referencia a una colección de preferencias llamada por ejemplo “MiPreferencia” con modo privado.
6. Una vez hemos obtenido una referencia a nuestra colección de preferencias, ya podemos obtener, insertar o modificar preferencias utilizando los métodos get o put correspondientes al tipo de dato de cada preferencia.
7. Para ello asignamos al evento onClick() del botón btnGuardar la actualización o inserción de nuevas preferencias , aunque la actualización o inserción no la haremos directamente sobre el objeto SharedPreferences, sino sobre su objeto de edición SharedPreferences.Editor. A este último objeto accedemos mediante el método edit() de la clase SharedPreferences.
8. Una vez obtenida la referencia al editor, utilizaremos los métodos put correspondientes al tipo de datos de cada preferencia para actualizar/insertar su valor, por ejemplo putString(clave, valor), para actualizar una preferencia de tipo String. Tendremos disponibles métodos put para todos los tipos de datos básicos: putInt(), putFloat(), putBoolean()... Finalmente, una vez actualizados/insertados todos los datos necesarios llamaremos al método apply() para confirmar los cambios.
9. Para obtener el valor que hemos almacenado asignamos al evento onClick() del botón btnRecuperar, de forma que haremos uso del método getString() donde le pasamos el nombre de la preferencia que queremos recuperar y un segundo parámetro con un valor por defecto. Este valor por defecto será devuelto por el método getString() si la preferencia solicitada no existe en la colección. Además del método getString(), existen por supuesto métodos análogos para el resto de tipos de datos básicos, por ejemplo getInt(), getLong(), getFloat(), getBoolean()…
10. Si ejecutamos el proyecto podemos obtener el siguiente resultado:




## Almacenamiento en un fichero XML

¿Pero donde se almacenan estas preferencias compartidas? Las preferencias se almacenan en un fichero XML. Estos ficheros XML se almacenan en una ruta que sigue el siguiente patrón:

```
/data/data/paquete.java/shared_prefs/nombre_coleccion.xml
```

Así, por ejemplo, en nuestro caso encontraríamos nuestro fichero de preferencias en la ruta:

```
/data/data/com.example.tema8app1/shared_prefs/MiPreferencia.xml
```

Si abrimos el Device File Explorer situado en **View | Tool Windows | Device File Explorer.**

Y navegamos hasta la carpeta de la cual hemos comentado con anterioridad, accedemos al fichero y lo descargamos en nuestro ordenador observaremos el contenido, que es el siguiente:

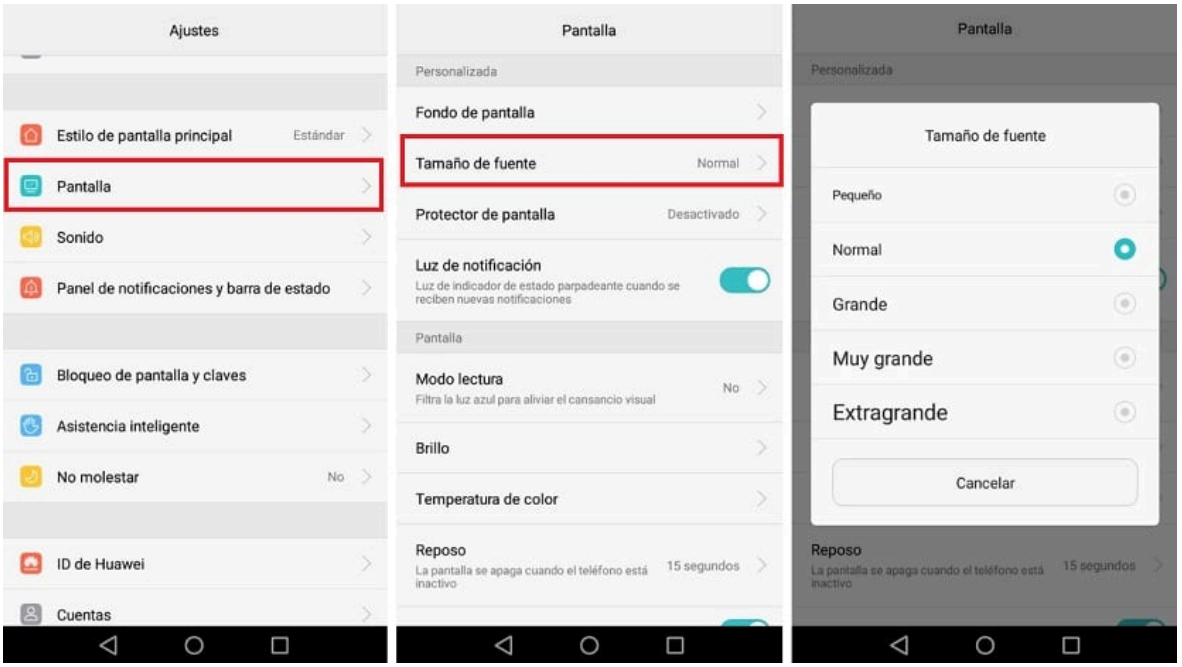
```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
      <string name="email">hola@correo.es</string>
</map>
```

En este XML podemos observar cómo se han almacenado las preferencias de ejemplo que insertamos anteriormente, con sus claves y valores correspondientes.

## 3.1. Añadiendo preferencias de usuario

**Android nos facilita la configuración de nuestro programa, al permitir añadir una lista de preferencias** que el usuario podrá modificar, de forma que se definirá mediante XML un conjunto de opciones para una aplicación y crearemos nosotros las pantallas necesarias para permitir al usuario modificarlas a su antojo. Las preferencias también pueden ser utilizadas para que tu aplicación almacene de forma permanente información (más adelante aprenderemos a utilizar esta opción).

Si nos fijamos en cualquier pantalla de preferencias estándar de Android veremos que todas comparten una interfaz común.




Si atendemos por ejemplo a la segunda imagen vemos cómo las diferentes opciones se organizan dentro de la **pantalla de opciones** en varias **categorías** (“Personalizada” y “Pantalla“). Dentro de cada categoría pueden aparecer **tipos de opciones**, como por ejemplo de tipo lista de selección (“Tamaño de la fuente”) o de tipo checkbox (“Luz de notificación”). He resaltado las palabras “pantalla de opciones”, “categorías”, y “tipos de opciones” porque serán estos los tres elementos principales con los que vamos a definir el conjunto de opciones o preferencias.

Como hemos indicado, nuestra pantalla de opciones la vamos a definir mediante un XML, de forma similar a como definimos cualquier layout, aunque en este caso deberemos colocarlo en la carpeta /res/xml. El contenedor principal de nuestra pantalla de preferencias será el elemento

<PreferenceScreen>. Este elemento representará a la pantalla de opciones en sí, dentro de la cual incluiremos el resto de elementos. Dentro de éste podremos incluir nuestra lista de opciones organizadas por categorías, que se representarán mediante el elemento <PreferenceCategory> al que daremos un texto descriptivo utilizando su atributo android:title. Dentro de cada categoría podremos añadir cualquier tipo de opciones, las cuales pueden ser de distintos tipos, entre los que destacan:

- **SwitchPreferenceCompat.** Dos opciones seleccionables.
- **EditTextPreference.** Cadena simple de texto.
- **ListPreference.** Lista de valores seleccionables (exclusiva).
- **MultiSelectListPreference.** Lista de valores seleccionables (múltiple).

Cada uno de estos tipos de preferencia requiere la definición de diferentes atributos, que iremos viendo en los siguientes ejemplos.

### 3.1.1. SwitchPreferenceCompat

Representa un tipo de opción que sólo puede tomar dos valores distintos: activada o desactivada. Es el equivalente a un control de tipo checkbox. En este caso tan sólo tendremos que especificar los atributos: nombre interno de la opción (app:key), texto a mostrar (app:title), descripción de la opción (app:summary), y si queremos una descripción diferente cuando esté activada la opción (app:summaryOn) o desactivada (app:summaryOff).

1. Creamos un proyecto denominado **Tema8App2**.
2. Para que nos cree la estructura completa vamos a crear una nueva actividad de tipo Settings Activity y luego estudiamos todas las cosas que nos ha creado para realizar nuestro ejemplo.
3. Lo primero que nos añade a Gradle es la dependencia de androidx:

```
implementation ("androidx.preference:preference:1.2.0")
```

4. Crea una nueva carpeta denominada /res/xml donde almacenará el layout de los settings con el nombre root_preferences.xml.
5. Borramos todo el código del layout root_preferences.xml para crearlo desde 0.
6. Vamos a suponer que tenemos una opción donde podemos habilitar/deshabilitar la opción

de escuchar la música de nuestra aplicación mientras la ejecutamos. Por ello, el código del fichero XML será el siguiente:

```
<PreferenceScreen xmlns:app="http://schemas.android.com/apk/res-auto">

       <SwitchPreferenceCompat
           app:key="reproducirMusica"
           app:summaryOff="No escuchar la música mientras se ejecuta la aplicación"
           app:summaryOn="Escuchar la música mientras se ejecuta la aplicación"
           app:title="Música" />

</PreferenceScreen>
```

Aunque todavía nos quedan las demás opciones por ver vamos a implementar todo lo necesario para hacerla funcionar en este momento.

7. Por ello, además de la definición XML de la lista de opciones, nos ha implementado una nueva actividad (SettingsActivity), que será a la que hacemos referencia cuando queramos mostrar nuestra pantalla de opciones y la que se encargará internamente de gestionar todas las opciones, guardarlas, modificarlas... a partir de nuestra definición XML.

   La pantalla de preferencias hace uso de Fragments. Para ello, define la clase kotlin del fragment (SettingsFragment) dentro de propia SettingsActivity, que deberá extender de PreferenceFragmentCompat.

   Para mostrar el fragment creado como contenido principal de la actividad utilizamos el FragmentManager para sustituir el contenido de la pantalla (R.id.settings) a un objeto de SettingsFragment. Esta manera de cargar el fragment va a ser de manera dinámica.

```
class SettingsActivity : AppCompatActivity() {

   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       setContentView(R.layout.settings_activity)
       if (savedInstanceState == null) {
           supportFragmentManager
               .beginTransaction()
               .replace(R.id.settings, SettingsFragment())
               .commit()
```

```
        }
        supportActionBar?.setDisplayHomeAsUpEnabled(true)
    }

    class SettingsFragment : PreferenceFragmentCompat() {
        override fun onCreatePreferences(savedInstanceState: Bundle?, rootKey: String?)
{
            setPreferencesFromResource(R.xml.root_preferences, rootKey)
        }
    }
}
```

La clase SettingsFragment lo que va hacer es cargar el layout creado dentro de la carpeta /xml/root_preferences.xml creado anteriormente.

8. Ahora vamos al layout principal y añadimos un botón que nos permitirá lanzar el menú de preferencias. El código de /res/layout/activity_main.xml será el siguiente:

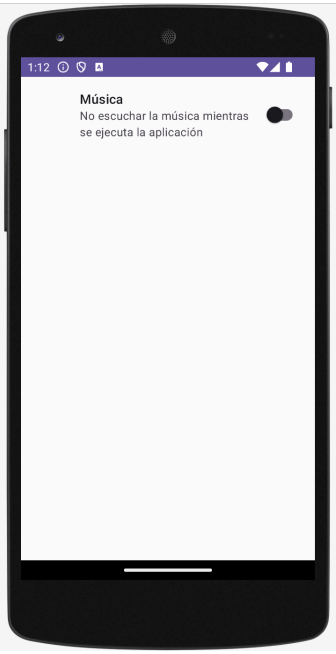
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/btnPreferencias"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Preferencias"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

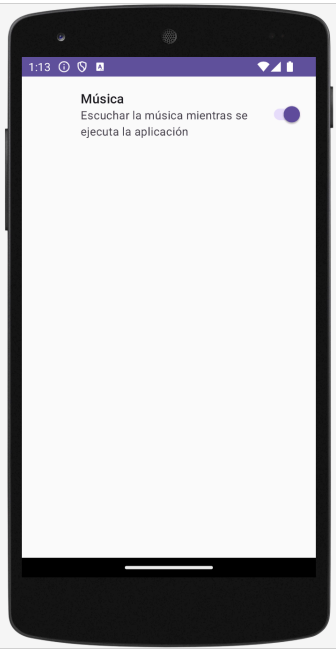
9. En la clase principal MainActivity.kt el código será el siguiente:

```
class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
```

```
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        binding.btnPreferencias.setOnClickListener {
            val intent = Intent(this, SettingsActivity::class.java)
            startActivity(intent)
        }
    }
}
```

10. Tras ello solo nos queda ejecutar la aplicación y comprobar su funcionamiento:



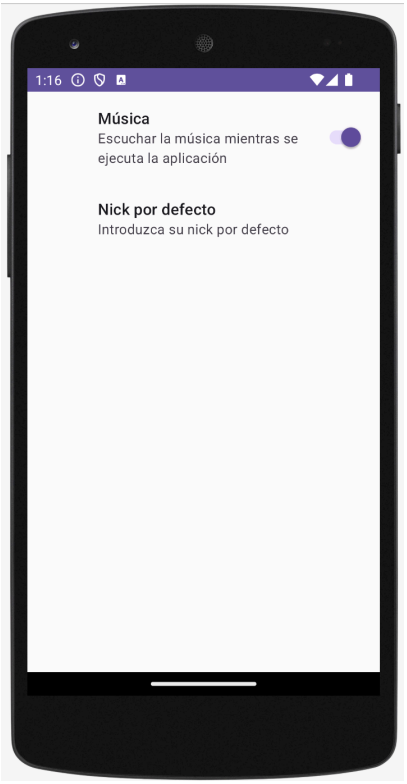

### 3.1.2. EditTextPreference

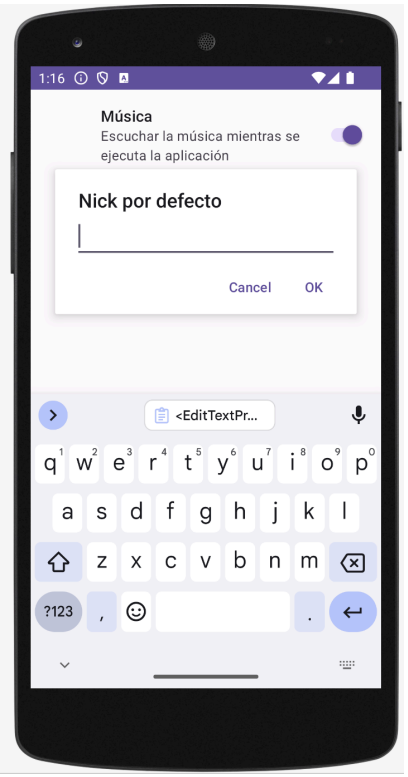
Representa un tipo de opción que puede contener como valor una cadena de texto. Al pulsar sobre una opción de este tipo se mostrará un cuadro de diálogo sencillo que solicitará al usuario el texto a almacenar. Para este tipo, además de los tres atributos comunes a todas las opciones (key, title y summary) también tendremos que indicar el texto a mostrar en el cuadro de diálogo, mediante el atributo app:dialogTitle. Otra opción interesante es sustituir app:summary="Texto descriptivo" por app:useSimpleSummaryProvider="true" que nos rellenara el summary con el texto introducido.

1. Abrimos el proyecto **Tema8App2**.
2. Nos vamos al fichero /res/layout/root_preferences.xml y añadimos:

```
<EditTextPreference
    app:key="nick"
    app:title="Nick por defecto"
    app:summary="Introduzca su nick por defecto"
    app:dialogTitle="Nick por defecto" />
```

3. Si ejecutamos la aplicación obtenemos los siguiente resultados:




### 3.1.3. ListPreference

Representa un tipo de opción que puede tomar como valor un elemento, y sólo uno, seleccionado por el usuario entre una lista de valores predefinida. Al pulsar sobre una opción de este tipo se mostrará la lista de valores posibles y el usuario podrá seleccionar uno de ellos. Y en este caso seguimos añadiendo atributos. Además de los cuatro ya comentados (key, title, summary y dialogTitle) tendremos que añadir dos más, uno de ellos indicando la lista de valores a visualizar en la lista y el otro indicando los valores internos que utilizaremos para cada uno de los valores de la lista anterior (Ejemplo: al usuario podemos mostrar una lista con los valores “Español” y “Francés”, pero internamente almacenarlos como “ESP” y “FRA”).

Estas listas de valores las definiremos también como ficheros XML dentro de la carpeta /res/values. Definiremos para ello los recursos de tipos <string-array> necesarios, en este caso dos, uno para la

lista de valores visibles y otro para la lista de valores internos, cada uno de ellos con su ID único correspondiente. Veamos cómo quedarían dos listas de ejemplo, en un fichero llamado "arrays.xml", por ello modificamos este fichero en la carpeta /res/values.

1. Abrimos el proyecto **Tema8App2**.
2. Modificamos el fichero arrays.xml dentro de /res/values con el siguiente contenido:

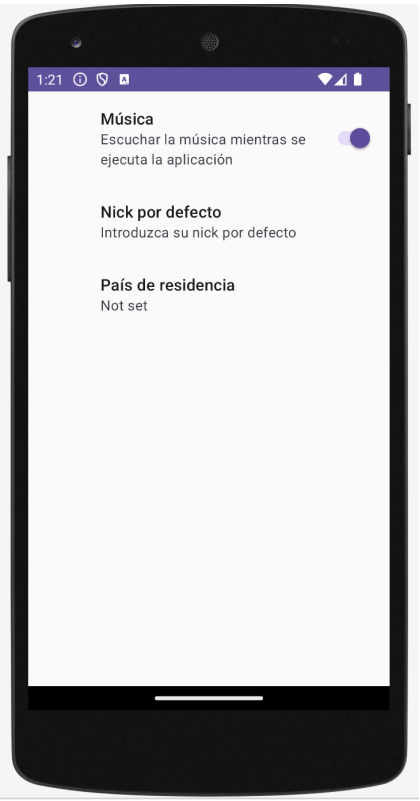
```
<?xml version="1.0" encoding="utf-8" ?>
<resources>
      <string-array name="pais">
         <item>España</item>
         <item>Francia</item>
         <item>Alemania</item>
      </string-array>
      <string-array name="codigopais">
         <item>ESP</item>
         <item>FRA</item>
         <item>ALE</item>
      </string-array>
</resources>
```

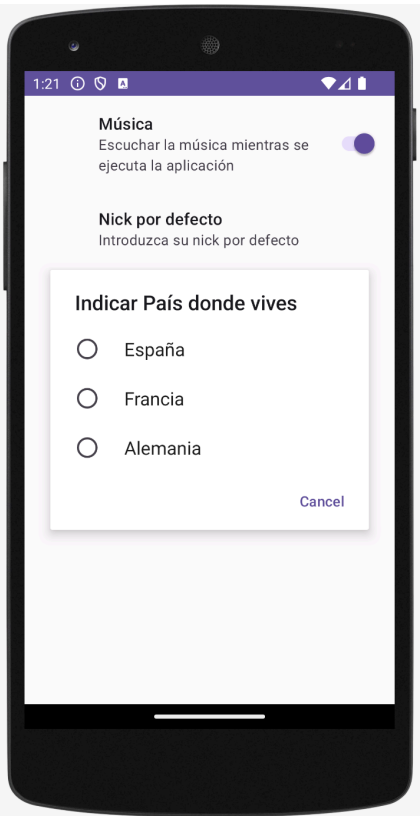
3. Nos vamos al fichero /res/layout/root_preferences.xml y lo modificamos añadiendo el siguiente código (en la preferencia utilizaremos los atributos android:entries y android:entryValues para hacer referencia a estas listas):

```
<PreferenceScreen xmlns:app="http://schemas.android.com/apk/res-auto">

      <SwitchPreferenceCompat
          app:key="reproducirMusica"
          app:summaryOff="No escuchar la música mientras se ejecuta la aplicación"
          app:summaryOn="Escuchar la música mientras se ejecuta la aplicación"
          app:title="Música" />

      <EditTextPreference
          app:key="nick"
          app:title="Nick por defecto"
          app:summary="Introduzca su nick por defecto"
          app:dialogTitle="Nick por defecto" />

      <ListPreference
          app:key="pais"
          app:title="País de residencia"
          app:useSimpleSummaryProvider="true"
          app:dialogTitle="Indicar País donde vives"
          app:entries="@array/pais"
```

```
            app:entryValues="@array/codigopais" />

</PreferenceScreen>
```

4. Si ejecutamos la aplicación obtenemos el siguiente resultado:



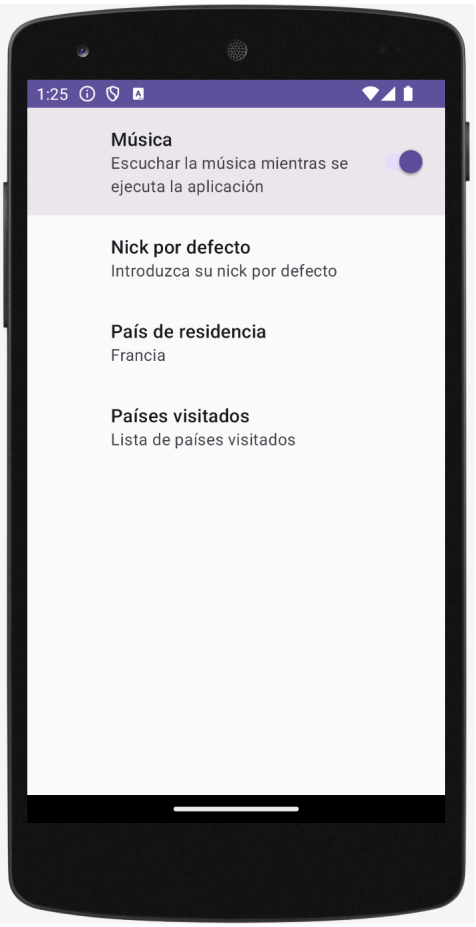
### 3.1.4. MultiSelectListPreference

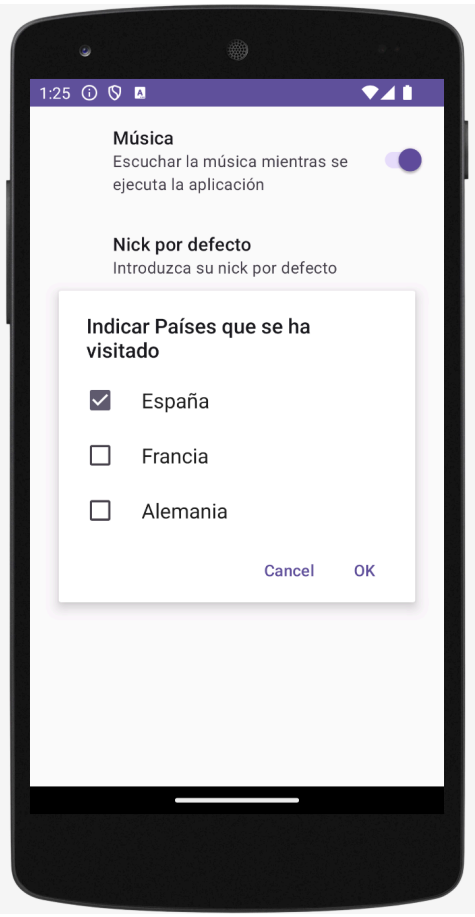
A partir de Android 3.0.x / Honeycomb las opciones de este tipo son muy similares a las ListPreference, con la diferencia de que el usuario puede seleccionar varias de las opciones de la lista de posibles valores. Los atributos a asignar son por tanto los mismos que para el tipo anterior. Veamos un ejemplo:

1. Abrimos el proyecto **Tema8App2**.
2. Nos vamos al fichero /res/layout/root_preferences.xml y lo modificamos:

```
<PreferenceScreen xmlns:app="http://schemas.android.com/apk/res-auto">

        <SwitchPreferenceCompat
            app:key="reproducirMusica"
            app:summaryOff="No escuchar la música mientras se ejecuta la aplicación"
            app:summaryOn="Escuchar la música mientras se ejecuta la aplicación"
            app:title="Música" />

        <EditTextPreference
            app:key="nick"
            app:title="Nick por defecto"
            app:summary="Introduzca su nick por defecto"
            app:dialogTitle="Nick por defecto" />

        <ListPreference
            app:key="pais"
            app:title="País de residencia"
            app:useSimpleSummaryProvider="true"
            app:dialogTitle="Indicar País donde vives"
            app:entries="@array/pais"
            app:entryValues="@array/codigopais" />

        <MultiSelectListPreference
            app:key="paisesVisitados"
            app:title="Países visitados"
            app:summary="Lista de países visitados"
            app:dialogTitle="Indicar Países que se ha visitado"
            app:entries="@array/pais"
            app:entryValues="@array/codigopais" />

</PreferenceScreen>
```

3. Si ejecutamos la aplicación obtenemos el siguiente resultado:




### 3.1.5. Definir categorías

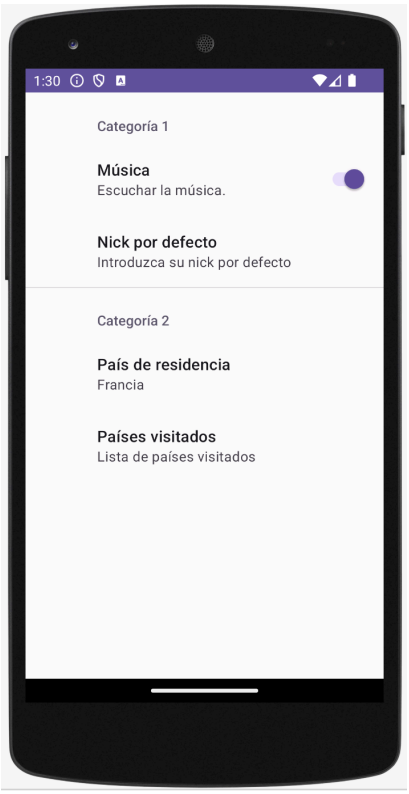
Además de esto, podemos definir categorías dentro de la pantalla, de forma que las 2 primeras opciones comentadas pertenecerán a la categoría 1, mientras que las 2 últimas opciones pertenecen a la categoría 2. Esto lo modificaremos en el fichero opciones.xml, por lo que:

1. Abrimos el proyecto **Tema8App2**.
2. Abrimos el fichero /res/xml/root_preferences.xml y lo modificamos:

```
<PreferenceScreen xmlns:app="http://schemas.android.com/apk/res-auto">

       <PreferenceCategory app:title="Categoría 1">
              <SwitchPreferenceCompat
                  app:key="reproducirMusica"
                  app:summaryOff="No escuchar la música."
                  app:summaryOn="Escuchar la música."
                  app:title="Música" />
```

```
            <EditTextPreference
                app:key="nick"
                app:title="Nick por defecto"
                app:summary="Introduzca su nick por defecto"
                app:dialogTitle="Nick por defecto" />
        </PreferenceCategory>
        <PreferenceCategory app:title="Categoría 2">
            <ListPreference
                app:key="pais"
                app:title="País de residencia"
                app:useSimpleSummaryProvider="true"
                app:dialogTitle="Indicar País donde vives"
                app:entries="@array/pais"
                app:entryValues="@array/codigopais" />

            <MultiSelectListPreference
                app:key="paisesVisitados"
                app:title="Países visitados"
                app:summary="Lista de países visitados"
                app:dialogTitle="Indicar Países que se ha visitado"
                app:entries="@array/pais"
                app:entryValues="@array/codigopais" />
        </PreferenceCategory>
</PreferenceScreen>
```

3. Si ejecutamos la aplicación obtenemos el siguiente resultado:



### 3.1.6. Accediendo a los valores de las preferencias

Una vez establecidos los valores de las preferencias podemos salir de la ventana de opciones simplemente pulsando el botón Atrás del dispositivo o del emulador. Nuestra actividad SettingsActivity se habrá ocupado por nosotros de guardar correctamente los valores de las opciones haciendo uso de la API de preferencias compartidas (Shared Preferences). Y para comprobarlo vamos a añadir otro botón (btnObtenerOpciones) a la aplicación de ejemplo que recupere el valor actual de las 3 preferencias y los escriba en el log de la aplicación.

La forma de acceder a las preferencias compartidas de la aplicación ya la vimos en el punto anterior. Obtenemos la lista de preferencias mediante el método getDefaultSharedPreferences() y posteriormente utilizamos los distintos métodos get() para recuperar el valor de cada opción dependiendo de su tipo.

Con todo esto hacemos los siguiente:

1. Abrimos el proyecto **Tema8App2**.
2. Nos vamos al layout principal que se sitúa en la carpeta /res/layout y se denomina activity_main.xml, y lo modificamos añadiendo un botón más para poder mostrar por el log el valor de las preferencias asignadas. El código del layout será:

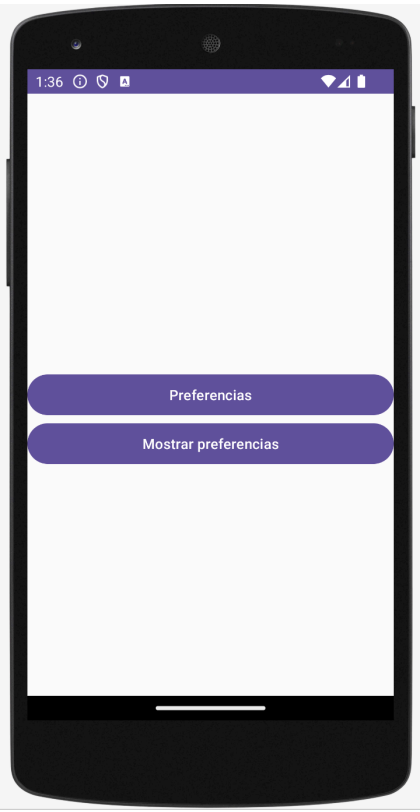
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:padding="10dp"
   tools:context=".MainActivity">

   <Button
       android:id="@+id/btnPreferencias"
       android:layout_width="0dp"
       android:layout_height="wrap_content"
       android:text="Preferencias"
       app:layout_constraintEnd_toEndOf="parent"
       app:layout_constraintHorizontal_bias="0.5"
       app:layout_constraintStart_toStartOf="parent"
       app:layout_constraintTop_toTopOf="parent" />
```

```
    <Button
        android:id="@+id/btnMostrarPreferencias"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Mostrar preferencias"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/btnPreferencias" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

3. Nos vamos a la clase principal MainActivity.kt y asignamos al nuevo botón, mediante el método onClick() que muestre en el log la información de las preferencias almacenadas. El código será el siguiente:

```
class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        binding.btnPreferencias.setOnClickListener {
            val intent = Intent(this, SettingsActivity::class.java)
            startActivity(intent)
        }

        binding.btnMostrarPreferencias.setOnClickListener {
            val pref: SharedPreferences =
                PreferenceManager.getDefaultSharedPreferences(this@MainActivity)

            Log.i("Tema8App2", "Reproducir música: "
                    + pref.getBoolean("reproducirMusica", false));
            Log.i("Tema8App2", "Nick: " + pref.getString("nick", ""));
            Log.i("Tema8App2", "Pais: " + pref.getString("pais", ""));


        }
    }
}
```

4. Tras ello, si ejecutamos la aplicación, tras guardar los valores de preferencias, si las mostramos, vemos en la ventana de Logcat lo siguiente:




```
Tema8App2              com.example.tema8app2          I  Reproducir música: true
Tema8App2              com.example.tema8app2          I  Nick: Anna
Tema8App2              com.example.tema8app2          I  Pais: FRA
```