Persistencia. SharedPreferences.

La escritura de ficheros puede llegar a ser tediosa y para almacenar datos simples una base de datos a veces es demasiado, para ello Android tiene los `SharePreferences`, son ficheros preparados para preferencias de usuario y que almacenan tipos de datos simples. Los datos guardados son persistentes, por lo que podremos recuperarlos, aunque cerremos la aplicación.

Crear preferencias

Para almacenar las preferencias en primer lugar debemos obtener un objeto de tipo `SharedPreferences` que nos dé acceso a un almacén concreto de preferencias. Según el almacén que queramos utilizar tendremos que usar una de las siguientes formas de obtener dicho objeto:

- `getSharedPreferences`(nombre, modo): nos permite abrir o crear un almacén de preferencias global a la aplicación proporcionando un nombre, que será lo que lo identifique, y un modo. Si existe un almacén de preferencias con dicho nombre, lo abrirá, y en caso contrario, lo creará cuando introduzcamos datos en él. Podemos indicar los permisos de acceso mediante el parámetro modo, que podrá ser de uso privado a la aplicación (`Context.MODE_PRIVATE` o 0), lo cual es el funcionamiento por defecto o también podemos dar permisos de lectura (`Context.MODE_WORLD_READABLE`) o de lectura/escritura (`Context.MODE_WORLD_WRITABLE`) desde fuera de nuestra aplicación.

```
SharedPreference sp = getSharedPreferences(NOMBRE_ALMACEN, 0);
```

- getPreferences(modo) nos da acceso a un almacén de preferencias propio de la actividad actual. En este caso no será necesario proporcionar un nombre, ya que como nombre del almacén utilizará el mismo nombre de la actividad.

```
SharedPreference sp = getSharedPreferences(0);
```

Guardar preferencias

Para guardar preferencias debemos crear, a partir del objeto SharedPreferences, un editor de la siguiente manera:

```
SharedPreferences.Editor editor = sp.edit();
```

Podemos añadir o modificar las preferencias del almacén mediante una serie de métodos `put[TIPO]` que nos proporciona el editor. Estos métodos reciben dos parámetros: una cadena con la clave y el valor de la preferencia. Según el método `put` que utilicemos podremos guardar diferentes tipos de datos, por ejemplo:

```
editor.putString[“nombre”, “Pablo”];
editor.putInt(“edad”, 25);
editor.putBoolean(“casado”, false);
editor.putFloat(“altura”, 1.78);
```

Una vez hayamos terminado de introducir datos, debemos confirmar los cambios con `commit` para que queden guardados en el almacén. Es importante que llamemos a este método ya que si no los cambios realizados se perderían.

```
editor.commit();
```

Sería buena idea llamar a este método a través de una confirmación del usuario o por ejemplo cuando la actividad pasar por `onStop()` para asegurarse de que quedan guardadas.

Leer preferencias

Una vez tengamos los datos guardados, para cargarlos un buen sitio sería el método onCreate de la actividad.

Para recuperar la información, al igual que para insertar teníamos métodos put, en este caso tenemos métodos get[TIPO], estos métodos reciben dos parámetros: el primero es la clave asociada al dato guardado en el almacén, y el segundo es el valor por defecto, en el caso de que no exista ningún dato con la clave asociada.

```
String nombre = sp.getString(“nombre”, “Anónimo”);
int edad = sp.getInt(“edad”, -1);
boolean casado = sp.getBoolean(“casado”, false);
float altura = sp.getFloat(“altura”, -1.0f);
```

Layout y PreferenceFragments.

Las pantallas 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 número de opciones, las cuales pueden ser de distintos tipos, entre los que destacan:

- CheckBoxPreference. Marca seleccionable.
- 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 apartados.

CheckBoxPreference

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 (android:key), texto a mostrar (android:title) y descripción de la opción (android:summary). Veamos un ejemplo:

```
<CheckBoxPreference
    android:key="opcion1"
    android:title="Preferencia 1"
    android:summary="Descripción de la preferencia 1" />
```

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 android:dialogTitle. Un ejemplo sería el siguiente:

```
<EditTextPreference
   android:key="opcion2"
   android:title="Preferencia 2"
   android:summary="Descripción de la preferencia 2"
   android:dialogTitle="Introduce valor" />
```

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/xml. 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 “*codigospaises.xml*“:

```
1. <?xml version="1.0" encoding="utf-8" ?>
2. <resources>
3.    <string-array name="pais">
4.       <item>España</item>
5.       <item>Francia</item>
6.       <item>Alemania</item>
7.    </string-array>
8.    <string-array name="codigopais">
9.       <item>ESP</item>
10.            <item>FRA</item>
11.            <item>ALE</item>
12.         </string-array>
13.      </resources>
```

En la preferencia utilizaremos los atributos android:entries y android:entryValues para hacer referencia a estas listas, como vemos en el ejemplo siguiente:

```
<ListPreference
    android:key="opcion3"
    android:title="Preferencia 3"
    android:summary="Descripción de la preferencia 3"
    android:dialogTitle="Indicar Pais"
    android:entries="@array/pais"
    android:entryValues="@array/codigopais" />
```

MultiSelectListPreference

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.

```
<MultiSelectListPreference
    android:key="opcion4"
    android:title="Preferencia 4"
    android:summary="Descripción de la preferencia 4"
    android:dialogTitle="Indicar Pais"
    android:entries="@array/pais"
    android:entryValues="@array/codigopais" />
```

Como ejemplo completo, veamos cómo quedaría definida una pantalla de opciones con las 3 primeras opciones comentadas (ya que probaré con Android 2.2), divididas en 2 categorías llamadas por simplicidad “Categoría 1” y “Categoría 2”. Llamaremos al fichero “opciones.xml”.

```
1. <PreferenceScreen
2.     xmlns:android="http://schemas.android.com/apk/res/android">

3.     <PreferenceCategory android:title="Categoría 1">
4.         <CheckBoxPreference
5.             android:key="opcion1"
6.             android:title="Preferencia 1"
7.             android:summary="Descripción de la preferencia 1" />

8.         <EditTextPreference
9.             android:key="opcion2"
10.                 android:title="Preferencia 2"
11.                 android:summary="Descripción de la preferencia
    2"
12.                 android:dialogTitle="Introduce valor" />
13.         </PreferenceCategory>
14.         <PreferenceCategory android:title="Categoría 2">
15.             <ListPreference
16.                 android:key="opcion3"
17.                 android:title="Preferencia 3"
18.                 android:summary="Descripción de la preferencia
    3"
19.                 android:dialogTitle="Indicar Pais"
20.                 android:entries="@array/pais"
21.                 android:entryValues="@array/codigopais" />
22.         </PreferenceCategory>
23.     </PreferenceScreen>
```

Ya tenemos definida la estructura de nuestra pantalla de opciones, pero aún nos queda un paso más para poder hacer uso de ella desde nuestra aplicación. Además de la definición XML de la lista de opciones, debemos implementar una nueva actividad, que será a la que hagamos referencia cuando queramos mostrar nuestra pantalla de opciones y la que se encargará internamente de gestionar todas las opciones, guardarlas, modificarlas, etc, a partir de nuestra definición XML.

Android nos facilita las cosas ofreciéndonos una clase de la que podemos derivar fácilmente la nuestra propia y que hace casi todo el trabajo por nosotros. Para ello, tenemos una clase de la cual debemos heredar `PreferenceFragment`. Los fragmentos proporcionan una arquitectura más flexible para tu aplicación, en comparación con el uso exclusivo de actividades, independientemente de la clase de actividad que esté desarrollando. Por lo tanto, usar `PreferenceFragment` para controlar la pantalla de tu configuración en lugar de `PreferenceActivity` siempre que sea posible.

Tu implementación de `PreferenceFragment` puede ser tan simple como la definición del método `onCreate()` para cargar un archivo de preferencias con `addPreferencesFromResource()`. Por ejemplo:

```
public static class SettingsFragment extends PreferenceFragment {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Load the preferences from an XML resource
        addPreferencesFromResource(R.xml.preferences);
    }
    ...
}
```

Luego puedes agregar este fragmento a una Activity de la misma forma que lo haría para cualquier otro Fragment. Por ejemplo:

```
public class SettingsActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Display the fragment as the main content.
        getFragmentManager().beginTransaction()
                .replace(android.R.id.content, new SettingsFragment())
                .commit();
    }
}
```

Para obtener una instancia del `SharedPreference` asociada al `fragment` usaríamos:

```
SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences
(context);
```