Persistencia de datos

Cuando estamos utilizando nuestro dispositivo móvil, en ocasiones, deseamos que algunos datos dentro de una aplicación puedan ser guardados, aún cuando nosotros cerremos la aplicación o apaguemos el dispositivo. A esto se le conoce como *Persistencia de datos*.

La persistencia de datos puede lograrse de diferentes maneras:

1. Haciendo uso de la clase SharedPreferences.
2. Guardando archivos en la memoria interna del dispositivo.
3. Guardando archivos en una memoria externa.
4. Guardar la información en una base de datos.

Con esto surge la cuestión: ¿Cuándo utilizar cada uno de esos métodos? A continuación analizaremos cada uno de los métodos para persistencia de datos y se hará notar en qué casos es mejor usar uno que otro.

La clase SharedPreferences

Si lo que se pretende es almacenar una cantidad limitada de datos, como por ejemplo, información personal, opciones de presentación, entre otros, es preferible el uso de la clase SharedPreferences. Las preferencias compartidas o *shared preferences* se almacenarán en una forma de clave-valor, lo que quiere decir que cada una de ellas tendrá un identificador único con el cual acceder (ejemplo: "nombre") y dicho identificador tendrá un valor asociado (ejemplo: "Juan"). *¿Y dónde se guarda toda esa información?* La respuesta es sencilla; es en archivos XML.

¿Cómo se manejan estas preferencias compartidas? Gracias al lenguaje de programación utilizado, podemos hacer uso de *Colecciones*. En una aplicación, podemos manejar varias colecciones de preferencias, y estas tendrán identificador único. Si queremos obtener una referencia a una colección, tendremos que utilizar el método `getSharedPreferences()`, el cual recibe 2 argumentos: El identificador de la colección (En forma de cadena) y un modo de acceso. El modo de acceso nos sirve para indicar qué aplicaciones pueden tener acceso a la colección de preferencias. Generalmente existen 3 modos de acceso:

- `MODE_PRIVATE`. Nos indica que sólo la misma aplicación tendrá acceso a las preferencias.
- `MODE_WORLD_READABLE`. Nos indica que todas las aplicaciones pueden leer estas preferencias, sin embargo, sólo la principal podrá modificarlas.
- `MODE_WORLD_WRITABLE`. Nos indica que todas las aplicaciones pueden leer y modificar estas preferencias.

Sin embargo, a pesar de que existen diferentes modos de acceso, es muy común utilizar `MODE_PRIVATE` debido a la seguridad que le da a nuestras preferencias para poder ser modificadas. De hecho, a partir de la API 17 (Android 4.2), las otras dos opciones quedaron obsoletas.

Hagamos un ejemplo práctico para ver su funcionamiento.

El diseño de la aplicación tiene la siguiente forma:

Lo que queremos hacer es verificar si hay algún dato(en este caso una cadena) guardada en nuestra aplicación por medio de un SharedPreference. Eso lo podemos conocer presionando el botón *Verificar*. Si hay algo guardado, lo mostrará en un Toast, en caso contrario, nos mandará un mensaje indicando que no se tiene algún dato guardado.

El código Java es el siguiente:

```
package com.example.rodrigo.persistenciashaared;

import android.content.Context;
import android.content.SharedPreferences;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    EditText et1;
    Button btnIngresar;
    Button btnVerificar;


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

        et1 = (EditText) findViewById(R.id.txtDato);
        btnIngresar = (Button) findViewById(R.id.btnIngresar);
        btnVerificar = (Button) findViewById(R.id.btnVerificar);

        btnIngresar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                String dato = et1.getText().toString();
                SharedPreferences sp = getSharedPreferences("datosingresados",Context.MODE_PRIVATE);
                SharedPreferences.Editor editor = sp.edit();
                editor.putString("dato", dato);
                editor.commit();
            }
        });

        btnVerificar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                SharedPreferences sp = getSharedPreferences("datosingresados", Context.MODE_PRIVATE);
                String valorguardado = sp.getString("dato", "No guardaste nada");
                Toast.makeText(getApplicationContext(),"Dato guardado: "+valorguardado, Toast.LENGTH_LONG).show();
            }
        });
```

```
    }

}
```

Puede apreciarse en el código, que la funcionalidad de la aplicación recae en los dos botones. En el botón *Ingresar* se tienen las siguientes operaciones:

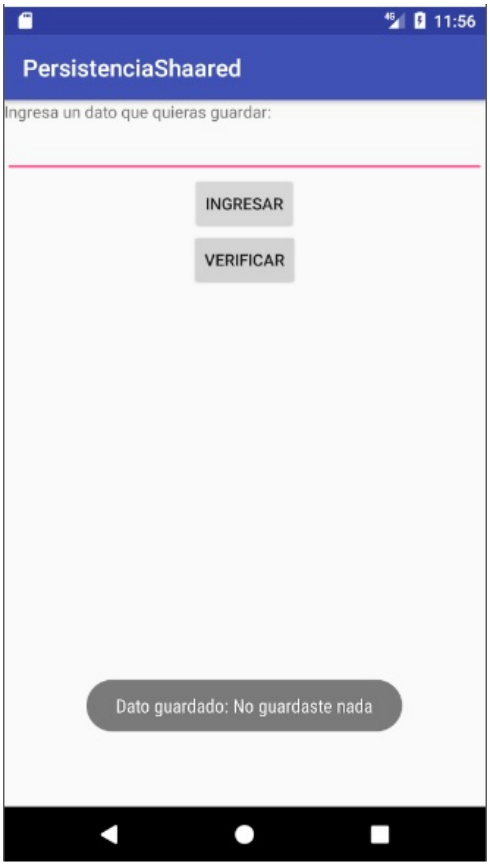
1. Guardamos la cadena ingresada en el EditText en una variable de tipo String llamada *dato*.
2. Se crea una referencia a un objeto de la clase SharedPreferences y ejecutamos el método `getSharedPreferences()`, que recibe como argumento *"datosingresados"* (nombre de la colección) y será en modo privado.
3. Mencionamos que queremos editar nuestra colección, por lo que llamamos a `SharedPreferences.Editor` y creamos un objeto. Hacemos uso del método `edit()` del objeto SharedPreferences.
4. Ingresamos la cadena que queremos guardar con el método `putString()`, que recibe la llave de un objeto dentro de nuestra colección, junto con el valor asociado. Nota: existen métodos para ingresar cualquier tipo de dato, como puede ser `putInt()`, `putDouble()`, etc.
5. Guardamos los cambios con el método `commit()` del editor.

Para el botón *Verificar* se observan las siguientes acciones:

1. Accedemos a la colección llamada *datos ingresados* con el método `getSharedPreferences()`.
2. Verificamos si existe algún valor guardado en la colección con el método `getString()`. Observamos que en el primer argumento contiene la cadena "dato", esta es la posible clave de algún valor guardado, si existe, nos devolverá el valor y se guarda en la variable *valor guardado*, en caso de que no exista, nos devolverá la cadena ingresada como segundo parametro (En este caso se ingresó "No guardaste nada").
3. Lanzamos un Toast para que nos verifique si existe o no algún dato guardado.

Probando la aplicación

Si al iniciar la aplicación presionamos el botón *Verificar*, nos arrojará un mensaje indicando que no se ha guardado nada, así como se muestra en la siguiente imagen:



Si ahora, ingresamos una cadena en el EditText, presionamos el botón *Ingresar*, y seguidamente volvemos a presionar el botón *Verificar*, ahora el Toast nos devolverá un mensaje con la cadena guardada en nuestros *SharedPreferences*. Véase la

siguiente imagen:

Incluso al cerrar la aplicación y volver a abrirla, si ahora volvemos a presionar el botón de *Verificar*, podremos ver que nuestro dato seguirá guardado.

Almacenamiento en la memoria interna del dispositivo

Existirán ocasiones en que, al momento de estar utilizando una aplicación, queramos guardar información en forma de un archivo para poder volver a utilizarlo en un futuro. Tal es el caso de los documentos de texto, en donde nosotros podemos escribir y posteriormente abrir ese archivo de nuevo para seguir editando. Los archivos que se crean deben almacenarse en alguna memoria, y en este caso, veremos cómo utilizar la memoria interna de nuesto dispositivo Android. Debemos de tener en cuenta las limitaciones que tiene el dispositivo en cuanto a su memoria, por lo que en la memoria interna es recomendable no guardar archivos de gran tamaño. Si lo que deseamos es escribir datos en un archivo, Android nos ofrece el método `openFileOutput()`, el cual recibirá como parámetros el nombre del archivo en el cual queremos escribir y el modo de acceso con el que queremos acceder al fichero. Para los métodos de acceso tenemos:

- `MODE_PRIVATE` : Para tener acceso privado, es decir, únicamente desde nuestra aplicación (Crea el archivo o lo sobreescribe si ya existe).
- `MODE_APPEND` : Para poder añadir datos a un archivo existente.
- `MODE_WORLD_READABLE` : Para que otras aplicaciones puedan leer el archivo.
- `MODE_WORLD_WRITABLE` : Para que otras aplicaciones puedan escribir sobre el archivo. Los últimos dos han sido declarados como obsoletos en la API 17.

El método anterior nos devolverá una referencia al flujo de salida asociado al archivo, por lo que ya podremos utilizar los métodos para manipulación de archivos como se hace en el lenguaje Java.

Para que esto quede más claro, hagamos un ejemplo.

El diseño de la aplicación tendrá la siguiente forma: