Persistencia en Android: proveedores de contenidos y SharedPreferences - Ejercicios

Índice

1. Compartir datos entre actividades con Shared Preferences

Descarga de las plantillas el proyecto *Dni*. Dicho proyecto está compuesto de dos actividades, *Formulario* y *Resumen*. El objetivo del ejercicio es conseguir que al pulsar el botón *Siguiente* en la actividad *Formulario* se muestre en la actividad *Resumen* cuáles fueron los datos introducidos.

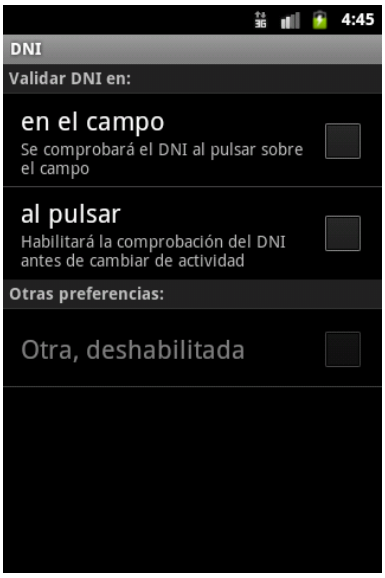
En este ejercicio vamos a hacer uso de `SharedPreferences` para pasar los datos de una actividad a la siguiente. Al pulsar en *Siguiente* en *Formulario* deberemos guardar el valor de todos los campos mediante esta plataforma. Al mostrarse la actividad *Resumen* deberemos leer estos datos, también a partir de `SharedPreferences`, para mostrarlos por pantalla.

Por último, crea un método que valide el DNI (debe tratarse de una secuencia de ocho dígitos entre 0 y 9 y una letra al final). Si se pulsa *Siguiente* y el DNI no tiene el formato correcto, no se deberá pasar a la actividad *Resumen*. En lugar de esto se mostrará un `Toast` en pantalla con un mensaje de error.

2. Actividad de preferencias

En este ejercicio seguimos trabajando con el proyecto del ejercicio anterior. Vamos a añadir un menú de opciones que permita escoger en qué momento se comprobará la validez del DNI. Para ello añade en primer lugar un menú a la actividad *Formulario* con una única opción, cuyo nombre será *Opciones*.

Al seleccionar esta opción se deberá mostrar una actividad de preferencias cuyo aspecto será el siguiente:



Preferencias de la aplicación Dni

El significado de las opciones será el siguiente:

- *En el campo*: con esta opción seleccionada, al pulsar sobre el campo de DNI en el formulario deberá mostrarse un `Toast` indicando si la sintaxis del DNI es correcta o no.
- *Al pulsar*: si se desactiva esta opción (que deberá estar activada por defecto), no se hará la comprobación del DNI al pulsar el botón *Siguiente*, por lo que siempre será posible pasar de la actividad *Formulario* a la actividad *Resumen* sea cual sea el formato del DNI introducido.
- El último check box se encontrará siempre deshabilitado. ¿Qué atributo debemos utilizar para conseguir esto?

Crea dos variables booleanas llamadas `enElCampo` y `alPulsar`. Su valor dependerá del estado de los checkboxes anteriores y se actualizará por medio del evento `onSharedPreferenceChange`. Utiliza el valor de estas variables en el código de tu actividad para que esta tenga el comportamiento indicado.

3. Proveedor de contenidos propio

Vamos a implementar otra forma de acceder a la base de datos de usuarios de la aplicación *BaseDatos* desarrollada durante los ejercicios de la sesión anterior, siguiendo esta vez el patrón de diseño `ContentProvider` de Android. Seguiremos trabajando pues con dicho proyecto.

- Creamos una nueva clase llamada `UsuariosProvider` que herede de `ContentProvider`. Esto nos obligará a sobrecargar una serie de métodos abstractos. Antes de implementar la query vamos a configurar el provider.

- Añadimos algunos campos típicos de los content provider:

```
public static final Uri CONTENT_URI =
        Uri.parse("content://es.ua.jtech.android.basedatos/usuarios");
private static final int TODAS_FILAS = 1;
private static final int UNA_FILA = 2;

private static final UriMatcher uriMatcher;

static{
        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        uriMatcher.addURI("es.ua.jtech.android.basedatos", "usuarios",
TODAS_FILAS);
        uriMatcher.addURI("es.ua.jtech.android.basedatos", "usuarios/#",
UNA_FILA);
}
```

- Vamos a acceder a la misma base de datos de usuarios utilizada en los ejercicios de la sesión anterior, pero no vamos a hacerlo a través del adaptador que tuvimos que implementar, sino que vamos a copiar de él el código que nos haga falta. Copia los campos que definen el nombre de la base de datos, de la tabla, de las columnas, la versión, así como la referencia al contexto y a la base de datos. La sentencia compilada del insert ya no va a hacer falta. Inicializa los valores necesarios en el constructor. Para inicializar la referencia a la base de datos vamos a utilizar, una vez más, MiOpenHelper. Podemos copiarlo del adaptador que ya implementamos en los ejercicios de la sesión anterior.
- Implementa de forma apropiada el `getType` para devolver un tipo MIME diferente según si se trata de una URI de una fila o de todas las filas. Para ello ayúdate del `uriMatcher`.
- Implementa el método `query`. Simplemente se trata de devolver el cursor que obtenemos al hacer una consulta a la base de datos SQLite. Algunos de los parámetros que le pasaremos los recibimos como parámetros del método del provider. Los que no tengamos irán con valor null.
- Aunque no tenemos todos los métodos del UsuariosProvider implementados, podemos probarlo. Para ello debemos registarlo en el *AndroidManifest.xml*:

```
        ...
                <provider android:name=".UsuariosProvider"
android:authorities="es.ua.jtech.android.basedatos"/>
        </application>
</manifest>
```

- En la clase `Main` se añadió codigo para insertar una serie de valores en la base de datos y mostrarlos en el campo de texto. Manteniendo este código, vamos a añadir al campo de texto el resultado obtenido con la consulta del `UsuariosProvider` para comprobar que tanto el adaptador de la base de datos como el proveedor nos devuelven el mismo resultado.

4. ¿Por qué conviene crear proveedores de contenidos? (*)

Porque es la forma estándar que establece Android de acceder a contenidos. Además, el

proveedor de contenidos nos permitirá notificar al `ContentResolver` de los cambios ocurridos. Así componentes en la pantalla podrán refrescarse de forma automática.

- Utiliza el proyecto *ProveedorContenidos* de las plantillas. Implementa la inserción en el proveedor de contenidos. Pruébala insertando algunos usuarios de ejemplo en la clase `Main`. Implementa también el `OnClickListener` del botón que inserta nuevos usuarios. El nombre del nuevo usuario irá indicado en el `EditText`.
- Comprueba que la inserción funciona y que, gracias a la siguiente línea y al estar usando un proveedor de contenidos, la lista se actualiza automáticamente cuando ocurre algún cambio, sin necesidad de pedir explícitamente la actualización al pulsar el botón.

```
cursor.setNotificationUri(cr, UsuariosProvider.CONTENT_URI);
```

Nota:

Esto no va a funcionar si se notifica que se ha producido un cambio al insertar o borrar un elemento. Para ello usamos `getContext().getContentResolver().notifyChange( UsuariosProvider.CONTENT_URI, null);`.

- Implementa el método `delete` del proveedor de contenidos. Pruébalo en la clase `Main`. Termina de implementar el `onCreateContextMenuListener` que se ejecutará cada vez que se haga una pulsación larga sobre alguna entrada de la lista. Comprueba que funciona (eliminando el usuario correspondiente, y no otro).