Laboratorio 3.- To Do list

Contenido:

Objetivos: Realizar una aplicación completa con los conocimientos adquiridos hasta ahora, usando varias actividades e ***intent***s para navegar entre ellas. Practicar el uso del método "***registerForActivityResult***", para crear actividades que retornan datos, y el uso de los ***ListView***s para mostrar datos en forma de listado. Ser capaces de personalizar el tema de la aplicación.

1 Toast

Un toast es un mensaje de tipo pop-up que se muestra durante un tiempo determinado. De este tipo de mensajes se puede personalizar el texto, la posición, la duración y el aspecto.

```
Toast.makeText(getApplicationContext(),"Mensaje",Toast.LENGTH_LONG).show();
```

El toast sólo se muestra al ejecutar el método `show()`.

```
int tiempo= Toast.LENGTH_SHORT;
Toast aviso = Toast.makeText(this, "Mensaje", tiempo);
aviso.show();
```

Si queremos interaccionar con el mensaje podemos usar un Snackbar que permite acciones .

```
Snackbar.make(contextView, R.string.text_label,
Snackbar.LENGTH_LONG).setAction(R.string.action_text) {

// Responds to click on the action

    }.show();
```

2 Aplicación a realizar: Lista de tareas

En esta práctica, la aplicación que vamos a crear será una app multiidioma (castellano e inglés) para tener un listado de tareas pendientes que el usuario vaya introduciendo. La aplicación se mostrará en el idioma en el que esté configurado el dispositivo o en inglés por defecto. Con ello, vamos a practicar el uso de *intents* para navegar entre actividades. Concretamente, haremos uso de los métodos para recoger desde la actividad principal los datos que nos proporciona la segunda actividad (o actividad llamada). Para ello, tenemos que utilizar `startActivityIntent.launch` para lanzar la segunda actividad y `registerForActivityResult` para recuperar los datos en la actividad principal. Dentro de este segundo método, sobrescribiremos el método `onActivityResult` para recoger los datos que correspondan.

La actividad principal simplemente mostrará una lista de tareas que el usuario puede ir introduciendo en la segunda actividad. Dicha lista de tareas se mostrará en pantalla mediante un *ListView* simple, y habrá que actualizarlo cada vez que recibamos una nueva tarea desde la segunda actividad.

Además, nos fijaremos en qué pasa si rotamos la pantalla una vez hayamos introducido elementos en el *ListView*, y veremos qué soluciones hay para tratar este tema.

2.1 Prototipo de la interfaz

Hay que crear una aplicación que nos sirva de lista de tareas pendientes (***ToDo*** list). Para ello, crearemos una actividad principal que tiene un botón en la parte superior con el texto "ADD A TASK" (o "AÑADIR UNA TAREA"), y debajo del botón un *ListView* con las tareas añadidas por el usuario. Inicialmente, el *ListView* estará vacío y tendrá el aspecto de la Fig. 1.

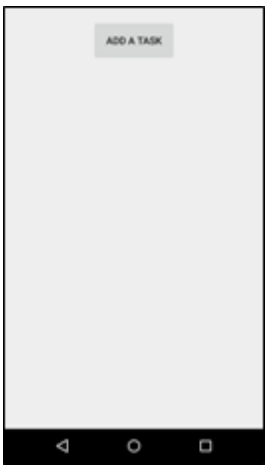

Fig. 1.- Pantalla principal

Cuando clicamos sobre el botón, mediante un ***intent*** explícito (se indica cuál es la actividad que se va a abrir) se nos abrirá una segunda actividad que usaremos para añadir una tarea. En esta actividad, tendremos un *TextView* con el texto "Enter new task name" (o "Introduce una nueva tarea"), y un *EditText*, donde el usuario puede escribir dicha descripción (ver Fig. 2).

Fig. 2.- Interfaz añadir tarea

Cuando el usuario haga clic sobre el botón "DONE" (o "LISTO"), la segunda actividad finalizará, y creará un ***intent*** con los datos pertinentes y lo enviará a la actividad principal. La actividad principal tiene que recoger los datos enviados por la segunda actividad, y añadirlos al *ListView*. Para eso, actualizaremos el contenido del array (para este tipo de operaciones, conviene usar un ***ArrayList*** en vez de un array, ya que tiene métodos para añadir –`add()`- y eliminar –`remove()`- elementos), y llamaremos al método `adapter.notifyDataSetChanged()` para que se actualice el *ListView* (Fig. 3).

Fig. 3.- Navegación entre interfaces

Además, usaremos el método `setOnItemLongClickListener()` del objeto *ListView* para añadir el método `onItemLongClick` a cada ítem del *ListView* y conseguir que se elimine del listado al hacer una pulsación larga.

```
final ArrayList<String> arraydedatos=new ArrayList<String>();
arraydedatos.add("alumno1");
arraydedatos.add("alumno2");
arraydedatos.add("alumno3");
```

```
lalista.setOnItemLongClickListener(new AdapterView.OnItemLongClickListener() {
    @Override
    public boolean onItemLongClick(AdapterView<?> adapterView, View view, int i, long l) {
        arraydedatos.remove(i);
        eladaptador.notifyDataSetChanged();
        return true;
    }
});
```

Se devuelve true si todo ha ido bien, para que no se ejecute ningún otro listener. Si se devuelve false se ejecutarían el resto de listeners definidos (por ejemplo, OnCLickListener)

Haced que tras eliminar un elemento se muestre un toast avisando de que se ha eliminado.

Cuando el ejercicio esté acabado, mirad qué pasa si se gira el dispositivo. ¿Qué se os ocurre que se puede hacer en estos casos para solucionar este tema?

3 Estilos y temas

Un **estilo** es una colección de propiedades que especifican el aspecto y el formato de una vista (de un elemento de tipo View). A través de las propiedades se puede indicar el tamaño, el color, etc. La idea es similar a las hojas de estilo (CSS) en HTML.

Los estilos se definen en formato XML en el fichero styles.xml que se encuentra en la carpeta res/values de la aplicación.

El nodo raíz del fichero XML debe ser el elemento `<resources>` y cada estilo (se pueden definir varios en el mismo fichero) debe definirse dentro de un elemento `<style>`. Las propiedades de cada estilo se definen mediante el elemento `<item>`.

```
<style name="FuenteDAS">
    <item name="android:fontFamily">sans-serif-smallcaps</item>
    <item name="android:textColor">#f00350</item>
</style>
```

Para aplicar un estilo concreto a un elemento, se indica en el XML del layout mediante el atributo ***style***.

```
<TextView
    android:id="@+id/texto"
    style="@style/FuenteDAS"
    … />
```

Los estilos pueden organizarse mediante herencia usando el atributo ***parent***. De este modo, un estilo heredará todas las propiedades del estilo de su estilo "padre".

Mantiene el color, pero cambia el tipo de fuente →

```
<style name="FuenteDASParaEditText" parent="FuenteDAS">
    <item name="android:fontFamily">casual</item>
</style>
```

Un **tema** es un estilo que, en vez de aplicarse a un elemento de la interfaz, se aplica a toda una actividad o a toda la aplicación. Mediante los temas se pueden personalizar aspectos como el fondo de las actividades, la barra que se muestra en la parte superior de las actividades denominada barra de acción (Action Bar), etc.

Como a día de hoy las actividades que se utilizan extienden la clase *AppCompatActivity*, para poder utilizar un estilo a modo de tema, debe heredar obligatoriamente de un tema correspondiente a AppCompat.

```
<style name="TemaDAS" parent="Theme.AppCompat">
    <item name="colorPrimary">#a2f079</item>
    <item name="colorPrimaryDark">#f4f4f4</item>
    <item name="colorAccent">#eba154</item>
</style>
```

Se pueden ver los temas disponibles de AppCompat desde Android Studio. Al hacer click con la tecla Ctrl pulsada sobre `"Theme.AppCompat.Light"` (o el tema `parent` elegido) en el fichero *styles.xml*, se abre el listado de temas dentro del fichero *values.xml*.

Para aplicar un tema a la aplicación completa o a una actividad concreta, se realiza a través del manifiesto de la aplicación.

Aplicar tema a la aplicación completa

```
<application
    …
    android:theme="@style/TemaDAS">
…
</application>
```

Aplicar tema a una actividad

```
<activity android:name=".MainActivity"
    android:theme="@style/TemaDAS">
    …
</activity>
```

> 💡 Se pueden definir diferentes estilos y temas para distintos tipos de dispositivos o para diferentes versiones del sistema operativo creando el fichero styles.xml en la carpeta *res/values* correspondiente mediante el uso de sufijos (values-v11, values-w820dp, etc.)

> 💡 No existe un listado actualizado de todos los atributos que se pueden utilizar a la hora de definir un estilo, pero se pueden encontrar buscándolos en el fichero XML que define los estilos: https://android.googlesource.com/platform/frameworks/base/+/refs/heads/master/core/res/res/values/styles.xml

> 💡 Google recomienda que todas las aplicaciones sigan una serie de reglas de estilo de modo que tengan un aspecto y un comportamiento "similar" que permita que los usuarios se acostumbren y no tengan que estar aprendiendo cómo funciona cada aplicación. Dichas reglas de estilo se llaman Material Design y se pueden consultar en https://material.io/.

- Cread estilos y temas y aplicadlos a la aplicación para personalizar los textos, los botones y todo lo que se os ocurra.

A partir de la versión 10 (API 29), Android soporta el tema oscuro a nivel de sistema. Este tema se aplica a la interfaz de usuario del sistema y a las aplicaciones que se ejecutan en él. Para que una aplicación sea compatible con el tema oscuro, el tema de la app debe heredar de un tema DayNight:

```
<style name="AppTheme" parent="Theme.AppCompat.DayNight">
```

Más información sobre el tema oscuro en: https://developer.android.com/guide/topics/ui/look-and-feel/darktheme