U.T.5 PROGRAMACIÓN PARA DISPOSITIVOS MÓVILES. ALMACENAMIENTO.

GUARDAR Y RECONSTRUIR EL ESTADO DE UNA ACTIVIDAD.

La destrucción y reconstrucción de las actividades pueden ocurrir:

- Cuando Android detecta que necesita memoria y no tiene suficiente. En este caso puede eliminar actividades que estén ocultas. Si más adelante el usuario vuelve a ellas, las volverá a crear, y éstas tendrán que ser capaces de reconstruirse en el estado original, para que el usuario no note que fueron eliminadas temporalmente.
- Cuando el usuario gira el teléfono, Android también reinicia la actividad. El objetivo no es otro que darle la oportunidad de reconstruirse utilizando un layout nuevo, específico de la nueva orientación.

En el guardado y recuperación del estado de la actividad entran en juego dos métodos:

onSaveInstanceState(Bundle outInstance)

onCreate(Bundle savedInstanceState)

GUARDAR Y RECONSTRUIR EL ESTADO DE UNA ACTIVIDAD.

onSaveInstanceState(Bundle outInstance): es un método que debemos sobreescribir en la actividad, y que será invocado automáticamente por Android antes de destruir la actividad para guardar en Bundle las parejas de atributo-valor que consideremos necesarias para posteriormente reconstruir la actividad.

onCreate(Bundle savedInstanceState): es el método al que Android invoca al iniciar una actividad. El parámetro savedInstanceState será null inicialmente. Sin embargo, si se está reconstruyendo la actividad a partir de un estado anterior, el parámetro contendrá una copia del bundle que rellenamos en la llamada a onSaveInstanceState().

```
public void onSaveInstanceState(Bundle estado) {
        super.onSaveInstanceState(estado);
// Guardamos los valores que nos interesan en el Bundle estado.
        estado.putInt(EDAD, valorEdad);
        estado.putString(NOMBRE, valorNombre);
        }
onCreate(Bundle savedInstanceState){
        if(savedInstanceState!=null){
//obtenemos los datos almacenados en el Bundle savedInstanceState
        savedInstanceState.getInt(EDAD);
        savedInstanceState.getString(NOMBRE);
        }
}
```

Almacenar preferencias en una aplicación

El estado de la aplicación que se almacena con onSaveInstanceState() lo gestiona el propio sistema y únicamente sirve para reconstruir la actividad a su estado anterior si fue Android quien la destruyó. Pero si el usuario pulsa el botón volver, entonces nuestra actividad partirá de cero.

Si queremos que algo de nuestro estado perdure, necesitaremos preocuparnos nosotros mismos de guardarlo. Android nos facilita esta tarea a través de la clase SharedPreferences.

Las preferencias no son más que datos que una aplicación debe guardar para personalizar la experiencia del usuario, por ejemplo información personal, opciones de presentación, puntos conseguidos en un Juego, etc.

SharedPreferences

Cada aplicación tiene un directorio para guardar sus ficheros de preferencias. Además, es posible tener más de un fichero. Para obtener una referencia a un fichero determinado utilizaremos el método getSharedPrefences() al que pasaremos el identificador y un modo de acceso.

```
SharedPreferences preferencias =getSharedPreferences(<nombre>, <tipoAcceso>);
```

El nombre de fichero si no existe. lo crea cuando llamamos a edit y commit.

El tipo de acceso será uno de los siguientes:

- Context.MODE_PRIVATE: el fichero será accesible únicamente para la aplicación. Es el más usado.
- Context.MODE_WORLD_READABLE: el fichero podrá ser leído por el resto de aplicaciones.
- Context.MODE_WORLD_WRITABLE: el fichero podrá ser escrito por el resto de aplicaciones.

Guardar datos en un fichero de preferencias

Cada preferencia se almacenará en forma de **clave-valor**, es decir, (identificador único, valor asociado) Ejemplo: (nombre, "Ana").

La escritura de las preferencias requiere la ayuda de una clase adicional, SharedPreferences.Editor que accedemos mediante el método edit() de la clase.

```
SharedPreferences.Editor editor;
editor = preferencias.edit();
editor.putString("nombre", "Ana");
… editor.commit;
```

Una vez obtenida la referencia al editor, utilizaremos los métodos *put* correspondientes al tipo de datos, por ejemplo putString(clave, valor). tendremos también putInt(), putFloat(), putBoolean(), etc. Finalmente, llamaremos al método commit().

Recuperar datos de un fichero de preferencias

Una vez conseguida una referencia a SharedPreferences como sigue:

```
SharedPreferences preferencias;
preferencias = getPreferences("fichPreferencias", Context.MODE_PRIVATE);
```

podemos obtener los atributos que contiene con los métodos get asociados a los tipos de datos básicos, por ejemplo getString(), getInt(), getLong(),getFloat(), getBoolean(), …

Ejemplo:

```
String name = preferencias.getString("nombre", "valorPorDefecto");
```

Al método getString() le pasamos el nombre de la preferencia que queremos recuperar y un segundo parámetro con un valor por defecto para que lo retorne en el caso de que no exista en las preferencias el atributo solicitado.

FICHEROS

En Android, podremos leer y escribir ficheros localizados en:

- La propia aplicación, en forma de recurso. El directorio res/ contiene ficheros que son automáticamente gestionados por Android. Podemos guardar ficheros como recursos en las carpetas res/raw y res/assets aunque con este último quedamos fuera de los identificadores del fichero R.java.
- La tarjeta SD externa, si existe. El uso de los ficheros en el directorio res/raw o assets/ del .apk tienen el problema de que son de sólo lectura. Si queremos utilizar ficheros para guardar información tendremos que hacerlo en memoria con openFileOutput(String nombre, int tipoAcceso); que retorna un FileOutputStream
- La memoria interna del dispositivo. debemos tener en cuenta las limitaciones de espacio que tienen muchos dispositivos, por lo que no debemos usarla para ficheros de gran tamaño. Para escribir ficheros en la memoria interna, utilizamos el mismo método openFileOutput(),

Lectura de ficheros como recursos

Para guardar ficheros propios creamos una carpeta raw dentro de la carpeta de recursos res/raw/, estos ficheros no serán interpretados por Android pero se pueden abrir muy fácilmente utilizando un identificador de recurso.

Además, podemos tener varios directorios raw con cualificadores (por ejemplo raw-en/, raw-es/,...) para los distintos idiomas.

Para abrir uno de estos ficheros, accedemos en primer lugar a los recursos de la aplicación con el método getResources() y sobre éstos utilizaremos el método openRawResource(*id_del_recurso*), y nos retorna un ImputStream que puede ser leido con las clases Scanner o BufferedReader de Java

InputStream is = getResources().openRawResource(R.raw.<id>);

Escritura y Lectura de ficheros externos

```
FileOutputStream fos;
//abre el fichero de escritura, para añadir datos por el final.
fos = openFileOutput(NOMBRE_FICHERO, Context.MODE_PRIVATE | Context.MODE_APPEND);
// crea un flujo de escritura
OutputStreamWriter out = new OutputStreamWriter(fos);
out.write("datos");
out.close();
```

```
FileInputStream fis;
/abre el fichero de lectura,
fis=openFileInput(NOMBRE_FICHERO);
// crea un flujo de lectura con Scanner
Scanner sc = new Scanner(fis);
while (sc.hasNextLine()) {
        linea = sc.nextLine();
}
sc.close();
```

Android almacena por defecto los ficheros creados en una ruta determinada, que en este caso seguirá el siguiente patrón: /data/data/paquete.java/files/nombre_fichero