Opciones de almacenamiento persistente

Opciones de almacenamiento

- En esa parte seguimos el tutorial https://developer.android.com/guide/topics/data/data-storage.html

Opciones de almacenamiento

- Almacenamiento persistente: los datos se mantienen entre distintas invocaciones de la aplicación, incluso si la aplicación es destruida por el usuario.
- Preferencias compartidas. Datos privados en parejas key – value (clave – valor).
- Almacenamiento interno. Datos privados en la memoria del dispositivo.
- Almacenamiento externo. Datos públicos en la memoria del dispositivo.

Opciones de almacenamiento

- Base de datos SQLite. Datos estructurados en una base de datos.
- Red. Datos en un servidor.

Preferencias compartidas

- Clase *SharedPreferences*.
- Dos formas de obtener un objeto *SharedPreferences*:

1. *getSharedPreferences*(*name*, *modo*). Si se necesita varios archivos de preferencias.
2. *getPreferences*(*modo*). Si solo se usa un archivo de preferencias.

- Nota: usar *modo* = 0 o *modo* = *MODE_PRIVATE* para la operación por default.

Para guardar valores

- Llamar a *edit*() para obtener un objeto *SharedPreferences.Editor*.
- Agregar valores con, por ejemplo, *putBoolean*() o *putString*().
- Llamar a *commit*() para comprometer los valores.

Para leer valores

- Usar métodos de *SharedPreferences* como *getBoolean*() o *getString*().

Ejemplo

```
public class Calc extends Activity {
    public static final String PREFS_NAME = "MyPrefsFile";

    @Override
    protected void onCreate(Bundle state){
       super.onCreate(state);
       . . .

       // Restore preferences
       SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
       boolean silent = settings.getBoolean("silentMode", false);
       setSilent(silent);
    }
```

Ejemplo

```
    @Override
    protected void onStop(){
       super.onStop();

      // We need an Editor object to make preference changes.
      // All objects are from android.context.Context
      SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
      SharedPreferences.Editor editor = settings.edit();
      editor.putBoolean("silentMode", mSilentMode);

      // Commit the edits!
      editor.commit();
    }
}
```

Almacenamiento interno

- Por default los archivos son privados.
- No pueden ser accesados por otras aplicaciones ni por el usuario.
- Al desinstalar la aplicación los archivos se borran.

Para escribir en un archivo

- Llamar a *openFileOutput*(*nombre*, *modo*). Regresa un objeto *FileOutputStream*.
- Llamar a *write*() para escribir al archivo.
- Llamar a *close*() para cerrar el archivo.
- *modo* = *MODE_PRIVATE* para la operación por default.
- *modo* = *MODE_APPEND* para agregar un archivo ya existente.

Ejemplo

```
String FILENAME = "hello_file";
String string = "hello world!";

FileOutputStream fos = openFileOutput(FILENAME, Context.MODE_PRIVATE);
fos.write(string.getBytes());
fos.close();
```

Para leer un archivo

- Llamar a *openFileInput*(). Regresa un objeto *FileInputStream*.
- Llamar a *read*() para leer bytes.
- Llamar a *close*() para cerrar el archivo.

Archivos externos

- Pueden estar en la memoria interna (no removible) o externa (removible o SD card).
- Se necesita pedir permiso en el manifiesto:

```
<manifest ...>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    ...
</manifest>
```

Revisar si la memoria está disponible

```
/* Checks if external storage is available for read and write */
public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}

/* Checks if external storage is available to at least read */
public boolean isExternalStorageReadable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state) ||
        Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
        return true;
    }
    return false;
}
```

Leer un archivo de texto

```
public String readFile(String fileName)
{
    String state = Environment.getExternalStorageState();
    Toast.makeText(this, "Estado de la SD externa " + state, Toast.LENGTH_LONG).show();
    StringBuilder builder = new StringBuilder();

    if (Environment.MEDIA_MOUNTED.equals(state) ||
            Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
        File path = new File(Environment.getExternalStorageDirectory() + "/Documents");
        File file = new File(path, fileName);
        try {
            path.mkdirs();
            BufferedReader handle = new BufferedReader(new FileReader(file));
            String linea;
            while ((linea = handle.readLine()) != null) {
                builder.append(linea).append("\n");
            }
            handle.close();
        }
        catch (Exception e) {
            Toast.makeText(this, "Error al leer en la SD externa " + e,
                    Toast.LENGTH_LONG).show();
        }
    }
    else {
        Toast.makeText(this, "No hay SD externa", Toast.LENGTH_LONG).show();
    }
    return builder.toString();
}
```

Escribir un archivo de texto

```
public boolean writeFile(String fileName, String texto)
{
    boolean error = false;
    String state = Environment.getExternalStorageState();
    tvState.setText("State: " + state);
    Toast.makeText(this, "Estado de la SD externa " + state, Toast.LENGTH_LONG).show();

    if (Environment.MEDIA_MOUNTED.equals(state) ||
            Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
        File path = new File(Environment.getExternalStorageDirectory() + "/Documents");
        File file = new File(path, fileName);
        try {
            path.mkdirs();
            BufferedWriter handle = new BufferedWriter(new FileWriter(file));
            handle.write(texto);
            handle.newLine();
            handle.close();
        }
        catch (Exception e) {
            error = true;
            Toast.makeText(this, "Error al escribir en la SD externa " + e,
                    Toast.LENGTH_LONG).show();
        }
    }
    else {
        error = true;
        Toast.makeText(this, "No hay SD externa", Toast.LENGTH_LONG).show();
    }
    return error;
}
```

Bases de datos

- Clase *SQLiteDatabase*.
- Permite ejecutar comandos SQL:
  - *execSQL*(String sql).
  - *rawQuery*(String sql, String[] selection) devuelve un objeto de tipo *Cursor*.
- Define métodos para no usar SQL:
  - insert(...)
  - update(...).
  - query(...).

Abrir o crear la base de datos

- Se hace al crear la actividad.

```
private SQLiteDatabase handle;

@Override
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    ...
    handle = openOrCreateDatabase("estudiantes", Context.MODE_PRIVATE, null);
    handle.execSQL("create table if not exists " +
      "estudiantes(expediente VARCHAR, nombre VARCHAR, calificacion VARCHAR);");
}
```

Insertar registros

```
handle.execSQL("insert into estudiantes values('" + exp + "','" +
    nom + "','" + cal + "');");
```

Listar todos los registros

```
Cursor cursor = handle.rawQuery("select * from estudiantes;", null);

if (cursor.getCount() == 0) {
    Log.d("tag", "Error: no hay registros");
    return;
}

StringBuilder builder = new StringBuilder();
while (cursor.moveToNext()) {
    builder.append("Expediente: ").append(cursor.getString(0)).append("\n");
    builder.append("Nombre: ").append(cursor.getString(1)).append("\n");
    builder.append("Calificación: ").append(cursor.getString(2)).append("\n");
}

return builder.toString();
```

Listar un registro

```
Cursor cursor = estudiantes.rawQuery("select * from estudiantes where expediente = '"
    + exp + "';", null);
StringBuilder builder = new StringBuilder();
if (cursor.moveToFirst()) {
    builder.append(cursor.getString(0)).append(" ");
    builder.append(cursor.getString(1)).append(" ");
    builder.append(cursor.getString(2)).append("\n");
}
else {
    Log.d("tag", "Error: el expediente " + exp + " no existe");
}
return builder.toString();
```

Borrar un registro

```
Cursor cursor = estudiantes.rawQuery("select * from estudiantes where expediente = '"
    + exp + "';", null);
if (cursor.moveToFirst()) {
    estudiantes.execSQL("delete from estudiantes where expediente = '" + exp + "';");
}
else {
    Log.d("tag", "Error: el expediente " + exp + " no existe");
}
```

Actualizar un registro

```
Cursor cursor = estudiantes.rawQuery("select * from estudiantes where expediente = '"
    + exp + "';", null);
if (cursor.moveToFirst()) {
    estudiantes.execSQL("update estudiantes " +
            " set nombre = '" + nom + "'," +
            "calificacion = '" + cal +
            "' where expediente = '" + exp + "';");
    limpiaCampos();
}
else {
    Log.d("tag", "Error: el expediente " + exp + " no existe");
}
```

SQLiteOpenHelper

- Clase para manejar la creación, apertura y actualización de la base de datos.
- Callbacks:
  - onCreate(…)
  - onUpdate(…)
  - onOpen(…)
- Típicamente aquí se pone el código para actualizar e interrogar la base de datos.

Aplicación con base de datos

- Clase SQLiteOpenHelper.
- Clases para representar las entidades y relaciones de la base de datos.
- Diálogos para interrogar y actualizar la base de datos.