5 – ALMACENAMIENTO INTERNO

Shared Preferences

Los Shared Preferences son archivos XML que residen en la memoria del dispositivo y me permiten almacenar información en la forma

CLAVE: VALOR

Shared Preferences

Los tipos de datos que me permite almacenar son:

- Boolean
- String
- Numéricos: float, int, long

Shared Preferences

Son parecidos a los ficheros de propiedades, pero se almacenan como un XML

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="nombre">prueba</string>
    <string name="email">modificado@email.com</string>
</map>
```

Shared Preferences

El procedimiento general de trabajo con estos ficheros será

1 Obtener referencia

2 Añadir/Consultar valores

3 Guardar los cambios

Shared Preferences

1 Obtener referencia

SharedPreferences prefs = getSharedPreferences(String nombre,MODO);

Shared Preferences

Donde, MODO, es una constante de Context y puede ser:

MODE_PRIVATE. Sólo nuestra aplicación tiene acceso a estas preferencias.

MODE_WORLD_READABLE

MODE_WORLD_WRITABLE

Shared Preferences

2 Consultar/Añadir Valores

```
String correo = prefs.getString("email", "
por_defecto@email.com");
SharedPreferences.Editor editor = prefs.edit();
editor.putString("email", "modificado@email.com");
```

Shared Preferences

3 Guardar

editor.commit();// Hago efectivos los cambios

Bases de Datos

SQL (Standard Query Lenguage) es un lenguaje soportado por todas “las marcas” de base de datos relacionales, que nos permite manipular y crear una base da datos

El Modelo E/R, me permite definir a priori qué datos y qué relaciones existen en mi app

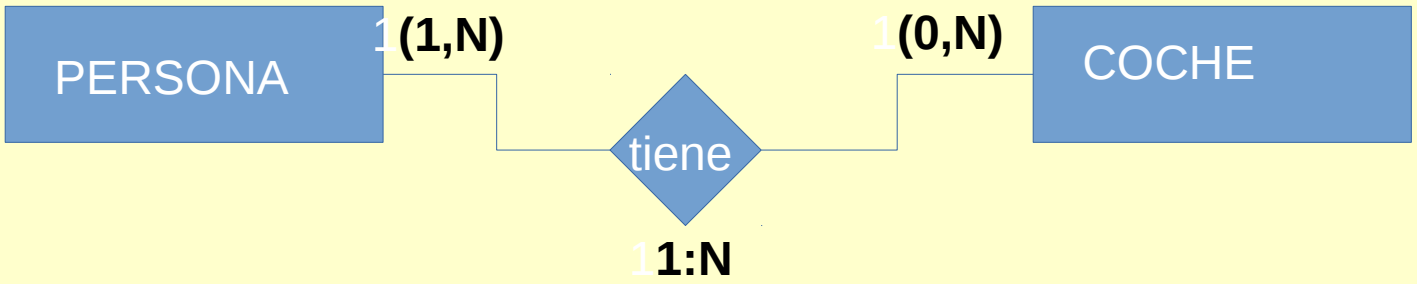
Bases de Datos

Android da soporte al uso de bases de datos relacionales mediante la clase SQLiteOpenHelper, que emplea un gestor de capacidades reducidas denominado

SQLite (www.sqlite.com)

Bases de Datos

Una base de datos relacional modela los datos como Entidades (tablas) y las relaciones entre ellos mediante y claves



Bases de Datos

TABLA personas

ID	Nombre
1	Paco
2	Jose
3	María

TABLA coche

ID	Coche	Persona
1	Ferrari	3
2	Seat	3
3	Ford	1

Nociones básicas

Cada entidad es una TABLA

Cada tabla debe tener un ID (clave primaria)

Las tablas, mantienen las relaciones (integridad referencial) mediante sus ID's.

Cuando un ID de una tabla está presente en otra, decimos que es una clave ajena (FK)

SQLiteOpenHelper

Para crear nuestra base de datos, creamos una clase que herede de esta. En ella, debemos sobreescribir los métodos:

public void onCreate(SQLiteDatabase db)

public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)

SQLiteDataBase

Es la clase que instanciamos para trabajar con la base de datos definida en la clase anterior.

SQLiteDataBase

La clase SQLiteOpenHelper, me proporciona instancias de SQLteDataBase mediante dos métodos:

getWritableDataBase() (modo L/E)

getReadableDatabase() (modo L)

SQLite Sintaxis

El lenguaje para trabjar con la base de datos se divide en:

- Lenguaje de definición (CREATE)
- Lenguaje de manipulación (INSERT)
- Lenguaje de consulta (SELECT)

Referencia SQLite

Conceptos avanzados

DISPARADORES (Triggers)

VISTAS (Views)

TRANSACCIONES (Commit/ Rollback)

Properties

Las Properties o ficheros de propiedades son semjantes, aunque tienen algunas diferencias de formato y conceptuales de uso

ruta_general=/marcabox/programas

ruta_ultimo=/marcabox/ultimo

Properties

Properties es un fichero plano y no un XML

Todos los datos son leídos como String

Su uso debe reservarse para almacenar información de configuración y no preferencias de usuario

Su emplazamiento debería ser /assets

Mem Interna VS Externa

Todas las operaciones se realizan por defecto en la memoria interna del dispositivo

Para trabajar con la memoria externa, hace falta asignar el permiso en el Manifest

android.permission.WRITE_EXTERNAL_STORAGE

Mem Interna VS Externa

Para trabajar con la memoria exeterna, podemos obtener las rutas de la clase Enviroment

Environment.getExternalStorageDirectory()

Ficheros en JAVA

Para la interacción con ficheros, y entrada y salida de datos, Java cuenta con su propia API, agrupada en el paquete java.io

Ficheros en JAVA

Las clases más importantes son:

File
FileInputStream, FileOutputStream
FileReader, FileWriter
ObjectOutputStream, ObjectInputStream
BufferedReader, BufferedWriter

Clase File

La clase File nos permite obtener información sobre los directorios y archivos de nuestro sistema de ficheros
Nos permitirá crear y eliminar archivos dentro de nuestro sistema de ficheros
Pero no nos permitirá leer o escribir en ningún archivo o directorio

Clase File

Los métodos más importantes de la clase File son:

public File (String ruta) //constructor
boolean exists ()
boolean isDirectory()
boolean isFile()
String[] list () //obtenemos el listado del directorio
boolean createNewFile ()
boolean delete()

Stream VS Reader

- Cuando necesitemos procesar ficheros binarios (fotos, ejecutables, audio), debemos emplear clases que hereden de InputStream (FileInputStream) **Estamos leyendo bytes**
- Cuando necesitemos procesar ficheros de texto, debemos utilizar clases que hereden de Reader (FileReader), ya que al leer un byte, automáticamente se va a traducir como char, según la codificación del SO.

FileInputStream

- Es la clase que emplearemos cuando necesitemos leer archivos binarios
- Métodos destacados
  - FileInputStream (String ruta)
  - FileInputStream (File file)
  - int read (byte [] b) //lee b.length bytes
  - int read () //lee un byte
  - getFD() // obtenemos el objeto FileDescriptor

FileOutputStream

- Es la clase que emplearemos cuando necesitemos escribir archivos binarios
- Métodos destacados
  - FileOutputStream (String ruta)
  - FileOutputStream (File file)
  - int wirte (byte [] b) //escribe b.length bytes
  - int read () //escribe un byte
  - getFD() // obtenemos el objeto FileDescriptor

FileReader

- Es la clase que emplearemos cuando necesitemos leer archivos de caracteres
- Métodos destacados
  - FileReader (String ruta)
  - FileReader (File file)
  - int read () // heredada de InputStreamReader

FileWriter

- Es la clase que emplearemos cuando necesitemos escribir archivos de caracteres
- Métodos destacados
  - FileWriter (String ruta, boolean añadir)
  - FileWriter (File file, boolean añadir)
  - int write (String cadena int inicio int final) // heredada de OutputStreamWriter

Buffers

- Aunque FileReader y FileWriter nos permiten leer y escribir en un fichero, existe un método mucho más eficiente y empleado, que consiste en emplear buffer de datos, para eliminar accesos a disco
- Además, nos proporcionan una interfaz más cómoda, con unos métodos más potentes

BufferedReader

- Instanciaremos un objeto de esta clase, pasándole un FileReader

  BufferedReader br = new BufferedReader (objetoFileReader)

  Y ya, podremos usar métodos de BR, como

  - br.readLine() //que nos lee una línea de golpe (devuelve null si llegamos al final)

BufferedWriter

- Análogo a la clase anterior, Instanciaremos un objeto de esta clase, pasándole un FileWriter

  BufferedWriter bw = new BufferedWriter(objetoFileWriter)

  Y ya, podremos usar métodos de BW, como

  - bw.write(String cadena)
  - bw.newLine(); añadimos un intro

Serialización

Serializar un objeto en Java, es una capacidad que ofrece el lenguaje para poder escribir Objetos en disco y después recuperarlos cómodamente, así como poder transmitirlos por la red o de una clase a otra o almacenarlos en una base de datos.

Serializar = escribir objetos

Visualizar Fichero

Para visualizar los ficheros creados, es necesario acceder por el terminal con los permisos del propietario de la app. Para ello, hay que seguir los siguientes pasos

Visualizar Fichero

1. Entra en terminal / cmd como root
2. Ir al directorio sdk/platform-tools
3. Ejecutar ./adb shell
4. Ejecutar run-as nombrepaquete
5. Visualizar /data/data/paquete/files/ficheroscreados