

SEMINARIO DE LENGUAJES

OPCIÓN ANDROID



Almacenamiento de datos.

Esp. Fernández Sosa Juan

Almacenamiento

Existen distintas alternativas para almacenar datos en **Android**:

- Archivos
- Preferencias
- Base de datos
- Proveedores de Contenido
- Servicios a través de la Red (Internet, nube)

Archivos

- Podemos usar el **sistema de archivos** para almacenar información permanente.
- Los archivos pueden guardarse en la **memoria interna** del dispositivo o en **memoria externa** como puede ser una **tarjeta SD** (medio de almacenamiento removible).
- También se puede utilizar **archivos añadidos** a nuestra aplicación **como recursos**, pero en este caso el acceso es de **sólo lectura**.

Archivos

- También existen facilidades para guardar información estructurada en **archivos XML**
- XML es un **estándar** universalmente aceptado para la **representación de datos**, en Internet y en muchos otros entornos.
- En **Android** disponemos de las librerías **SAX** y **DOM** para manipular datos en XML.
- Hoy la tendencia está en el uso de **JSON** por ser más **liviano**, más **fácil** de trabajar en Kotlin y más **compatible** con APIs modernas (como Firebase, REST, etc.).

Preferencias

- Constituye otra alternativa para almacenar información. Es un **mecanismo liviano** que permite almacenar y recuperar datos primitivos en la forma de pares **clave/valor**.
- Este mecanismo se suele utilizar para almacenar los **parámetros de configuración** de una aplicación pero también es útil para propósitos generales.

Bases de Datos

- Las APIs de Android contienen soporte para SQLite.
- Una aplicación Android puede crear y usar base de datos SQLite de forma muy sencilla y con toda la potencia que nos da el lenguaje SQL.
- No es mucho más complejo que almacenar los datos en archivos ni requiere muchos más recursos, sin embargo es mucho más potente.

Content Providers (proveedor de contenido)

- Un proveedor de contenido expone el acceso de lectura / escritura de sus datos a otras aplicaciones.
- Implementan una sintaxis estándar para acceder a sus datos mediante **URI** (Uniform Resource Identifiers) y un mecanismo de acceso para devolver los datos similar a **SQL**.
- Una app como WhatsApp accede a los contactos del teléfono usando un Content Provider del sistema, que le permite obtener esa información de forma segura, sin acceder directamente a los archivos ni violar la privacidad del usuario.

Servicios a través de la red

- También es posible utilizar la red para almacenar y recuperar información.
- Podemos definir nuestros propios protocolos usando **sockets** o utilizar protocolos de transferencias de archivos como **FTP** o **HTTP**
- Otra alternativa es utilizar **Servicios Web**

Servicios a través de la red

- **Firestore** permite **almacenar datos** en la **nube** y mantenerlos **sincronizados** entre todos los clientes conectados en **tiempo real**.
- **Firestore Realtime Database** es una **base de datos** que **sincroniza automáticamente** los cambios hechos por cualquier usuario (aplicación o dispositivo) con todos los demás, sin necesidad de recargar o volver a consultar los datos.

Almacenamiento en el sistema de archivos



Almacenando datos en Archivos

- Android hereda el sistema de archivos de Linux.
- Cuando se instala una aplicación se crea un nuevo usuario para esa aplicación, por lo tanto los archivos guardados en el espacio de la aplicación sólo son accesibles por la aplicación, ni siquiera el usuario del dispositivo puede accederlos.

Almacenando datos en Archivos

- Existen mecanismos (hoy desaconsejados) para **dar acceso** de lectura o escritura a los archivos internos **al resto de las aplicaciones**
 - Antes de **Android N** (versión API ≤ 23), otras apps podían acceder a los archivos internos **flexibilizando los permisos** del sistema de archivos.
 - A partir de la versión 24 de la API esto ya no es posible. Para dar acceso al contenido de un archivo privado, nuestra app puede usar un **FileProvider**
 - Un **FileProvider** es un componente de Android que **permite compartir archivos internos** de tu app con otras apps de forma segura, sin exponer rutas absolutas del sistema
 - Ej. Al enviar por Whatsapp una foto tomada con otra App se utiliza un FileProvider para compartir el archivo sin violar las restricciones de Android.

Almacenando datos en Archivos

- Almacenamiento interno vs. Almacenamiento externo
 - En los primeros años de **Android** la mayoría de los dispositivos proveían una **memoria interna no volátil** y la posibilidad de ampliarla con una **memoria externa SD**
 - Sin embargo, hoy en día, la **memoria externa** puede ser una **partición no extraíble** del espacio de almacenamiento permanente del dispositivo

Almacenando datos en Archivos

- Almacenamiento interno vs. Almacenamiento externo
 - El comportamiento de la **API** es el mismo, independientemente de que el almacenamiento externo sea extraíble o no
 - El almacenamiento **interno** está siempre **disponible** en todos los dispositivos, sin embargo el **externo** puede estar **ausente**

Almacenando datos en Archivos

- Almacenamiento interno vs. Almacenamiento externo
 - Sólo **nuestra aplicación** puede acceder a sus propios archivos guardados en el **almacenamiento interno**
 - Sin embargo los archivos que se guarden en el **almacenamiento externo** pueden ser accedidos por cualquier usuario.

Almacenando datos en Archivos

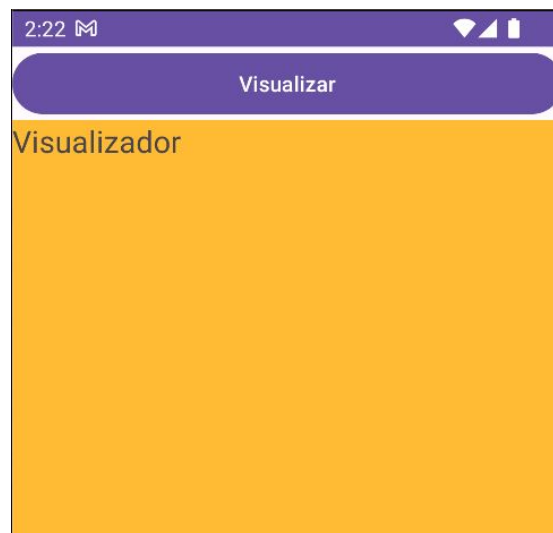
- Almacenamiento interno vs. Almacenamiento externo
 - Cuando el usuario **desinstala** nuestra app, se **borran** los archivos de la app del almacenamiento **interno**.
 - Sin embargo sólo se **borrarán** los archivos de la memoria **externa** si se guardaron en el directorio que se obtiene con **getExternalFileDir()**

Almacenando datos en Archivos

- Almacenamiento interno vs. Almacenamiento externo
 - El almacenamiento **interno** es ideal si queremos asegurar que ni el usuario del dispositivo ni otras apps puedan acceder a nuestros archivos.
 - Una app de notas guarda un borrador en un archivo interno (filesDir) que ningún usuario ni otra app puede ver.
 - El almacenamiento **externo** es ideal para los archivos que no requieren restricciones de acceso, **compartiéndolo** con otras apps y permitiendo que el usuario pueda accederlos desde una computadora
 - Una app de cámara guarda las fotos en `getExternalFilesDir(Environment.DIRECTORY_PICTURES)` para que el usuario pueda verlas con la galería.

Almacenamiento - Actividad guiada

- Cree una nueva aplicación en **Android Studio** denominada **Almacenamiento** (**Minimun SDK = Api 24 - Nougat**) con una **Empty Activity**
- Agregar al layout de la activity un **Button** y un **TextView** para visualizar información.



Actividad guiada - activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Visualizar"
        android:onClick="getStorePath"
    />
    <TextView
        android:id="@+id/visualizador"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="@android:color/holo_orange_light"
        android:textSize="20sp"
        android:text="Visualizador"
    />
</LinearLayout>
```

Almacenamiento - Actividad guiada

Cuando el usuario presiona el botón se debe visualizar en el **TextView** el path de los directorios en el almacenamiento **interno** y **externo** de la aplicación.

Utilice para ello los métodos de activity **getFilesDir()** y **getExternalFilesDir(null)** (null para el directorio raíz asociado a nuestra aplicación, si no debe especificarse un string que identifica el tipo de archivos).

Observe que estos métodos devuelven un objeto **File**. Utilice el **IDE** (info. de autocompletar) para averiguar cómo obtener a partir de ellos el **path completo**.

Actividad guiada - MainActivity.kt

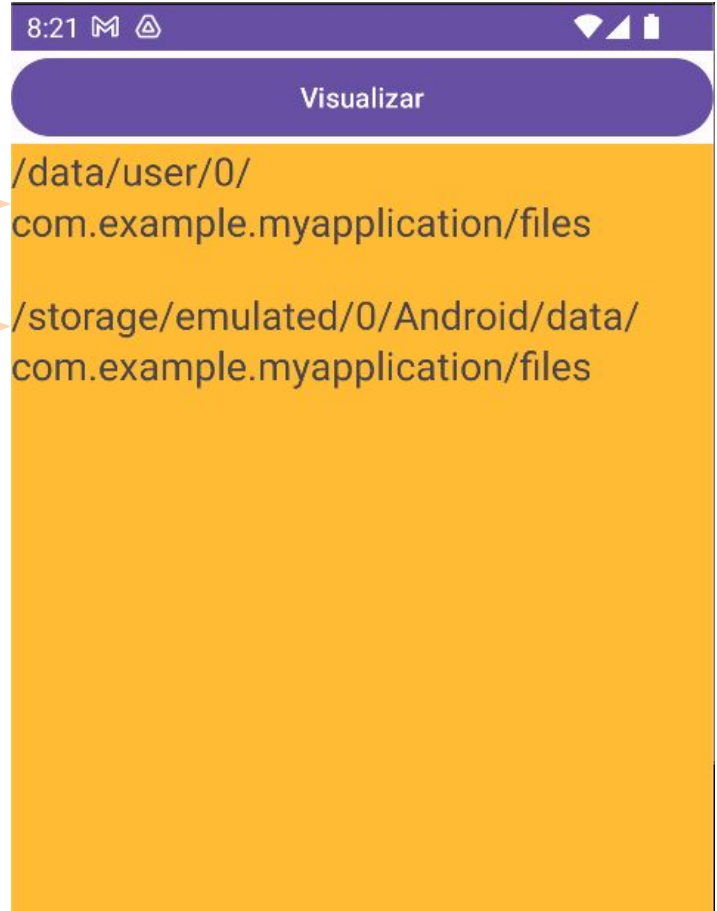
```
fun getStorePath(v: View ) {  
    var st = filesDir.absolutePath + "\n\n";  
    st+=getExternalFilesDir(null).toString() + "\n\n";  
    findViewById<TextView>(R.id.visualizador).text = st;  
}
```

Puede usarse **toString()** o **getAbsolutePath()** para obtener el string correspondiente al path completo del objeto **File** devuelto

Almacenamiento - Actividad guiada

Almacenamiento interno

Partición del espacio de almacenamiento interno montada como memoria externa



Inspección del dispositivo

Para versiones de Android Studio ≥ 3.1

Desde el panel lateral Device File Explorer, es posible analizar el file system del dispositivo, y en caso de un dispositivo emulado obtiene acceso total (permisos de root) incluido el almacenamiento interno

Name	Permissions	Date	Size
acct	drwxr-xr-x	2021-12-20 12:07	4 KB
apex	drwxr-xr-x	2023-02-10 11:13	1.1 KB
bin	lrw-r--r--	2021-12-20 12:53	11 B
cache	drwxrwx---	2021-12-20 12:07	4 KB
config	drwxr-xr-x	1969-12-31 21:00	0 B
d	lrw-r--r--	2021-12-20 12:53	17 B
data	drwxrwx--x	2023-02-10 11:13	4 KB
data_mirror	drwx-----	2023-02-10 11:13	120 B
debug_ramdisk	drwxr-xr-x	2021-12-20 12:07	4 KB
dev	drwxr-xr-x	2023-02-10 11:13	2.7 KB
etc	lrw-r--r--	2021-12-20 12:53	11 B
linkerconfig	drwx-----	2023-02-10 11:13	240 B
lost+found	drwx-----	2021-12-20 12:53	16 KB
metadata	drwxr-xr-x	2022-04-21 10:13	4 KB
mnt	drwxr-xr-x	2023-02-10 11:13	320 B
odm	drwxr-xr-x	2021-12-20 12:07	4 KB
odm_dkrm	drwxr-xr-x	2021-12-20 12:07	4 KB
postinstall	drwxr-xr-x	2021-12-20 12:07	4 KB
proc	dr-xr-xr-x	2023-02-10 11:13	0 B
product	drwxr-xr-x	2021-12-20 12:51	4 KB
sdcard	lrw-r--r--	2021-12-20 12:53	21 B
second_stage_resources	drwxr-xr-x	2021-12-20 12:07	4 KB
storage	drwx--x--x	2023-02-10 11:13	80 B
emulated	drwxrwx---	2022-04-21 14:30	4 KB
0	drwxrws---	2022-04-21 14:30	4 KB
Alarms	drwxrws---	2022-04-21 14:30	4 KB
Android	drwxrws--x	2022-04-21 14:30	4 KB
AudioBooks	drwxrws---	2022-04-21 14:30	4 KB
DCIM	drwxrws---	2022-04-21 14:30	4 KB
Documents	drwxrws---	2022-04-21 14:30	4 KB
Download	drwxrws---	2022-06-15 15:39	4 KB
Movies	drwxrws---	2022-04-21 14:30	4 KB
Music	drwxrws---	2022-04-21 14:30	4 KB
Notifications	drwxrws---	2022-04-21 14:30	4 KB
Pictures	drwxrws---	2022-04-21 14:30	4 KB
Podcasts	drwxrws---	2022-04-21 14:30	4 KB
Recordings	drwxrws---	2022-04-21 14:30	4 KB
Ringtones	drwxrws---	2022-04-21 14:30	4 KB
obb	drwxrwx---	2022-04-21 14:30	4 KB

Almacenamiento - Actividad guiada

Device File Explorer

Emulador nuevo API 32 Android 12L (Sv2)

Name	Permissions	Date	Size
> mediaarm	drwxrwx---	2022-04-21 14:30	4 KB
> misc	drwxrwx--t	2022-04-21 14:30	4 KB
> misc_ce	drwxrwx--t	2022-04-21 14:30	4 KB
> misc_de	drwxrwx--t	2022-04-21 14:30	4 KB
> nfc	drwxrwx---	2022-04-21 14:30	4 KB
> ota	drwxrwx--x	2022-04-21 14:30	4 KB
> ota_package	drwxrwx---	2022-04-21 14:30	4 KB
> per_boot	drwx-----	2023-02-10 11:13	4 KB
> preloads	drwxrwxr-x	2022-04-21 14:30	4 KB
> property	drwx-----	2023-02-22 12:41	4 KB
> resource-cache	drwxrwx--x	2023-02-10 11:13	8 KB
> rollback	drwx-----	2022-04-21 14:30	4 KB
> rollback-history	drwx-----	2022-04-21 14:30	4 KB
> rollback-observer	drwx-----	2022-04-21 14:30	4 KB
> server_configurable_flags	drwxrwxr-x	2022-04-21 14:30	4 KB
> ss	drwx-----	2022-04-21 14:30	4 KB
> system	drwxrwxr-x	2023-06-03 18:09	4 KB
> system_ce	drwxrwx---	2022-04-21 14:30	4 KB
> system_de	drwxrwx---	2022-04-21 14:30	4 KB
> tombstones	drwxrwx--x	2023-02-17 11:46	4 KB
> unencrypted	drwx-----	2022-04-21 14:30	4 KB
✓ user	drwx--x--x	2022-04-21 14:30	4 KB
> 0	drwxrwx--x	2023-06-03 18:09	12 KB
> user_de	drwx--x--x	2022-04-21 14:30	4 KB
> vendor	drwxrwx--x	2022-04-21 14:30	4 KB
> vendor_ce	drwxrwx--x	2022-04-21 14:30	4 KB
> vendor_de	drwxrwx--x	2022-04-21 14:30	4 KB
gsi_persistent_data	-rw-----	2023-02-10 11:13	1 B
local.prop	-rw-r--r--	2022-04-21 10:13	46 B
> data_mirror	drwx-----	2023-02-10 11:13	120 B
> debug_ramdisk	drwxr-xr-x	2021-12-20 12:07	4 KB
> dev	drwxr-xr-x	2023-02-10 11:13	2.7 KB
> etc	lrw-r--r--	2021-12-20 12:53	11 B
> linkerconfig	drwxr-xr-x	2023-02-10 11:13	240 B
> lost+found	drwx-----	2021-12-20 12:53	16 KB

Observamos que el directorio **data/user/0/** es un link a **/data/data/** dónde Android reserva espacio privado de almacenamiento para cada aplicación instalada

Almacenamiento - Actividad guiada

Device File Explorer

Emulador nu

Name

/data/data/com.example.myapplication espacio privado de nuestra aplicación

> com.android.wallpaperpicker	drwx-----	2022-04-21 14:30	4 KB
> com.app.vici	drwx-----	2023-02-24 15:12	4 KB
> com.breel.wallpapers18	drwx-----	2022-04-21 14:30	4 KB
✓ com.example.myapplication	drwx-----	2023-06-03 19:24	4 KB
> cache	drwxrws--x	2023-06-03 18:09	4 KB
> code_cache	drwxrws--x	2023-06-03 18:15	4 KB
> files	drwxrwx--x	2023-06-03 18:20	4 KB
> shared_prefs	drwxrwx--x	2023-06-03 19:24	4 KB
> com			
> com.google.android.apps.docs	drwx-----	2023-06-03 18:16	4 KB
> com.google.android.apps.enterprise	drwx-----	2022-04-21 14:30	4 KB

es el directorio devuelto por **getFileDirs()** o **filesDir**

Nota: Es bien conocido que el Android Device Monitor tiene un problema cuando el emulador corre un Android con API > 23 no pudiendo visualizar el espacio de almacenamiento interno

Almacenamiento - Actividad guiada

`getExternalFilesDirs(null)` (observar que termina con una **s** a diferencia del visto anteriormente) es un método de **Activity** (versión de API ≥ 19) que devuelve un vector de objetos **File** identificando los directorios asociados a nuestra aplicación en todas las memorias externas disponibles en el dispositivo

Modifique la aplicación para mostrar también esta información en el **TextView**

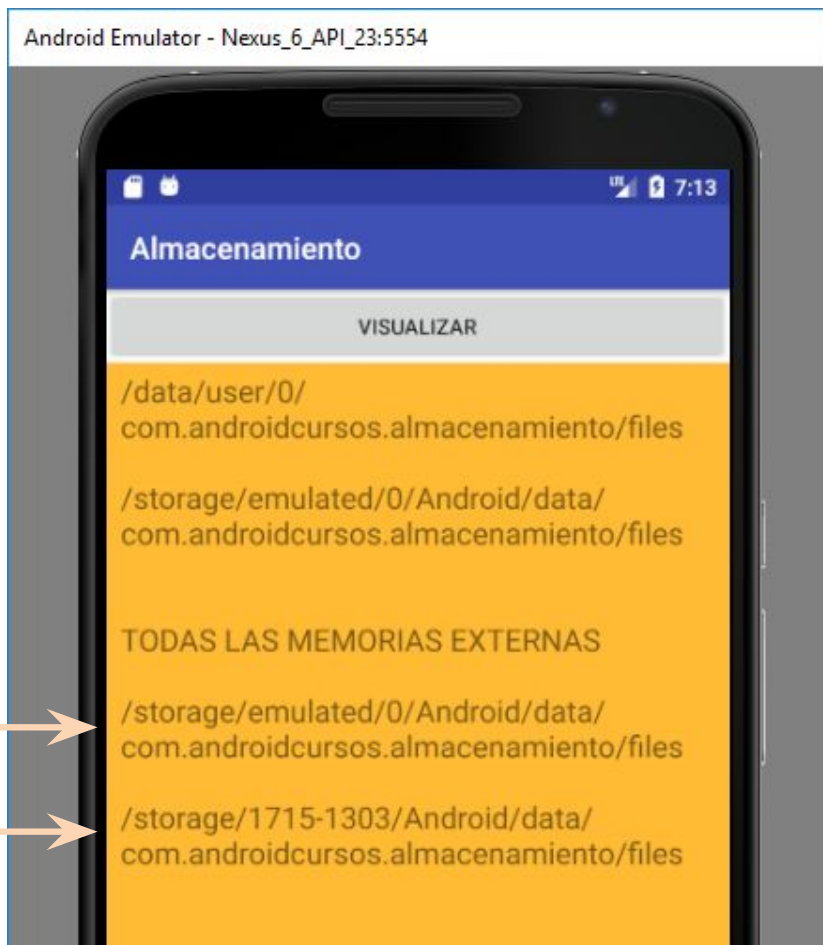
Actividad guiada - MainActivity.kt

```
fun getStorePath(v: View ) {  
    var st = filesDir.absolutePath + "\n\n";  
    st+=getExternalFilesDir(null).toString() + "\n\n";  
    st+= "TODAS LAS MEMORIAS EXTERNAS \n\n";  
    val directoriosExternos = getExternalFilesDirs(null);  
    directoriosExternos.forEach {  
        st += it.absolutePath + "\n\n";  
    }  
    findViewById<TextView>(R.id.visualizador).text = st;  
}
```

Almacenamiento - Actividad guiada

Partición del espacio de almacenamiento interno montada como memoria externa

Almacenamiento externo extraíble (memoria SD)



Almacenamiento - Actividad guiada

The screenshot shows the File Explorer interface of an Android emulator. The left pane displays a tree view of the storage system. The right pane shows a list of files and folders with their metadata.

Almacenamiento externo extraíble (memoria SD)

This callout points to the **storage** partition. The tree view shows the following structure:

- service_contexts
- storage
 - 1715-1303
 - Android
 - data
 - com.androidursos.almacenamiento
 - files
 - com.google.android.apps.maps
 - com.google.android.gms
 - com.google.android.googlequicksearchbox
 - LOST.DIR

Partición del espacio de almacenamiento interno montada como memoria externa

This callout points to the **emulated** partition. The tree view shows the following structure:

- emulated
 - 0
 - Alarms
 - Android
 - data
 - com.androidursos.almacenamiento
 - files
 - com.google.android.apps.maps
 - com.google.android.gms
 - com.google.android.googlequicksearchbox
 - DCIM

The right pane displays a list of files and folders with their metadata:

Name	Date	Time	Permissions
1715-1303	2017-06-04	15:46	drwxrwx--x
Android	2017-06-04	15:46	drwxrwx--x
data	2017-06-04	15:46	drwxrwx--x
com.androidursos.almacenamiento	2017-06-04	15:46	drwxrwx--x
files	2017-06-04	15:46	drwxrwx--x
com.google.android.apps.maps	2017-06-04	15:46	drwxrwx--x
com.google.android.gms	2017-06-04	15:46	drwxrwx--x
com.google.android.googlequicksearchbox	2017-06-04	15:46	drwxrwx--x
LOST.DIR	2017-06-04	15:46	drwxrwx--x
emulated	2017-06-04	15:46	drwxrwx--x
0	2017-06-04	15:46	drwxrwx--x
Alarms	2017-06-04	15:46	drwxrwx--x
Android	2017-06-04	15:46	drwxrwx--x
data	2017-06-04	15:46	drwxrwx--x
com.androidursos.almacenamiento	2017-06-04	15:46	drwxrwx--x
files	2017-06-04	15:46	drwxrwx--x
com.google.android.apps.maps	2017-06-04	15:46	drwxrwx--x
com.google.android.gms	2017-06-04	15:46	drwxrwx--x
com.google.android.googlequicksearchbox	2017-06-04	15:46	drwxrwx--x
DCIM	2017-06-04	15:46	drwxrwx--x

Archivos en la memoria interna

- Android permite guardar archivos dentro del almacenamiento interno privado de la app.
- Este espacio es inaccesible para otras apps o el usuario.
- Una forma moderna y simple de trabajar con archivos en Kotlin es usando la clase **File**.

Guardando archivos en la memoria interna

Codificar el siguiente método e invocarlo en el onCreate de la Activity

```
fun guardarUsuario() {  
    val contenido = "Juan Pérez\njuan@example.com\n2"  
    val archivo = File(filesDir, "usuario.txt")  
    try {  
        archivo.writeText(contenido)  
    } catch (e: IOException) {  
        e.printStackTrace()  
    }  
}
```

Se crea una referencia
al archivo usuario.txt

Se escriben los datos. Si el
archivo no existe lo crea, y si
no sobrescribe lo que tenía

Guardando archivos en la memoria interna

Codificar el siguiente método e invocarlo en el onCreate de la Activity

```
fun guardarUsuario() {  
    val contenido = "Juan Pérez\njuan@example.com\n2"  
    val archivo = File(filesDir, "usuario.txt")  
    try {  
        archivo.writeText(contenido)  
    } catch (e: IOException) {  
        e.printStackTrace()  
    }  
    archivo.appendText("nueva línea\n")  
}
```

agrega texto al final sin borrar lo que ya tenía el archivo.

Guardando archivos en la memoria interna

The screenshot displays the Android Studio interface. On the left, the 'Project' view shows the file structure of 'com.example.myapplication'. The 'files' directory is expanded, showing 'usuario.txt'. A red arrow points from 'usuario.txt' to a red box at the bottom containing the file path. On the right, the 'Text Editor' view shows the content of 'usuario.txt'.

Line	Content
1	Juan Pérez
2	juan@example.com
3	2

/data/data/com.example.myapplication/files/usuario.txt

Leyendo archivos desde la memoria interna

```
fun leerUsuario() {  
    val archivo = File(filesDir, "usuario.txt")  
    val lineas = archivo.readlines()  
  
    val nombre = lineas[0]  
    val email = lineas[1]  
    val nivel = lineas[2].toInt()  
  
    findViewById<TextView>(R.id.visualizador).text =  
        "Nombre: $nombre\nEmail: $email\nNivel: $nivel"  
}
```

Carga las líneas del archivo como una lista. El archivo se abre, se lee y se cierra automáticamente

Leyendo archivos desde la memoria interna

```
fun getStorePath(v: View ){  
    var st = filesDir.absolutePath + "\n\n";  
    st+=getExternalFilesDir(null).toString() + "\n\n";  
    st+= "TODAS LAS MEMORIAS EXTERNAS \n\n";  
    val directoriosExternos = getExternalFilesDirs(null);  
    directoriosExternos.forEach {  
        st += it.absolutePath + "\n\n";  
    }  
    findViewById<TextView>(R.id.visualizador).text = st;  
    leerUsuario();  
}
```

Agregar esta invocación en el método **getStorePath**

Archivos en la memoria interna

Este es el contenido guardado en el archivo usuario.txt y luego recuperado



Escribiendo en memoria externa

- A diferencia de la memoria interna, la memoria externa puede no estar presente en el dispositivo.
- Con el método estático `getExternalStorageStatus()` de la clase `Environment`, es posible consultar el estado de la memoria externa. Devuelve un string que nos indicará el estado de la misma.
- Alguno de los valores devueltos más importantes se muestran en la siguiente diapositiva.

Escribiendo en memoria externa

- **MEDIA_MOUNTED**: Memoria externa disponible para leer y escribir en ella.
- **MEDIA_MOUNTED_READ_ONLY**: disponible sólo para lectura.
- Otra serie de valores que indicarán que existe algún problema y que por lo tanto no podemos ni leer ni escribir en la memoria externa (**MEDIA_UNMOUNTED**, **MEDIA_REMOVED**, ... etc.).

Escribiendo en memoria externa

Por ejemplo podríamos chequear la memoria externa de esta manera

```
...  
var sdDisponible = false;  
var sdAccesoEscritura = false;  
val estado = Environment.getExternalStorageState();  
if (estado.equals(Environment.MEDIA_MOUNTED)) {  
    sdDisponible = true;  
    sdAccesoEscritura = true;  
} else if (estado.equals(Environment.MEDIA_MOUNTED_READ_ONLY)) {  
    sdDisponible = true;  
}  
...
```

Escribiendo en memoria externa

```
fun guardarEnMemoriaExterna() {  
    val carpeta = getExternalFilesDir(null)  
    val archivo = File(carpeta, "usuario_ext.txt")  
  
    val contenido = "Juan Pérez\njuan@example.com\n2"  
  
    try {  
        archivo.writeText(contenido)  
    } catch (e: IOException) {  
        e.printStackTrace()  
    }  
}
```

Escribe el contenido en la memoria externa

Espacio de la aplicación en la memoria externa

Ejemplo: Escribiendo en memoria externa

Escribiendo en memoria externa

```

>  second_stage_resources drwxr-xr-x 2009-01-01 00:00 27 B
v  storage                drwx--x--- 2025-05-20 00:00 80 B
  v  emulated             dr-xr-x--- 2025-05-20 00:00 4 KB
    v  0                  drwxrws--- 2025-05-20 00:00 4 KB
      >  Alarms           drwxrws--- 2025-05-20 00:00 4 KB

```

Archivo creado en el espacio de la aplicación en la memoria externa. Se eliminará automáticamente al desinstalar la aplicación

```

>  com.android            drwxrws--- 2025-05-25 23:00 4 KB
v  com.example            drwxrws--- 2025-05-25 23:00 4 KB
  v  files                drwxrws--- 2025-05-26 00:00 4 KB
    usuario_ext.txt      - 2025-05-26 00:00 30 B
>  com.google             drwxrws--- 2025-05-20 00:00 4 KB
>  com.google             drwxrws--- 2025-05-20 00:00 4 KB

```



Preferencias

Preferencias

- La clase **SharedPreferences** nos permite almacenar y recuperar **datos primitivos** en la forma **clave/valor**
- Las preferencias son almacenadas en **archivos xml** dentro de la carpeta **shared_prefs** en los datos privados de la aplicación.
- Las preferencias de una aplicación **se eliminan** cuando se **desinstala la aplicación**

Preferencias

- `getSharedPreferences()` permite indicar el nombre del archivo de preferencias.
- `getPreferences()` Utiliza un nombre de archivo por defecto para la actividad.
- En ambos casos debe indicarse el tipo de permiso. Se aconseja utilizar sólo `MODE_PRIVATE`

Preferencias

```
fun guardarPreferencias() {  
    val preferencias: SharedPreferences =  
getPreferences(MODE_PRIVATE);  
    val editor = preferencias.edit();  
    editor.putString("articulo", "Paleta");  
    editor.putInt("cantidad", 3);  
    editor.commit();  
}
```

Ejemplo: Escribiendo en el archivo de preferencias por defecto de la activity

Preferencias

Device File Explorer

Emulador nuevo API 32 Android 12L (Sv2)

Name	Permissions	Date	Size
> com.android.theme.light.notosensordrwx-----	drwx-----	2022-04-21 14:30	4 KB
> com.android.traceurdrwx-----	drwx-----	2022-04-21 14:30	4 KB
> com.android.traceur.auto_generated_rdrwx-----	drwx-----	2022-04-21 14:30	4 KB
/data/data/com.androidcursos.almacenamiento espacio privado de nuestra aplicación			
> com.app.vici	drwx-----	2023-02-24 15:12	4 KB
> com.breel.wallpapers18	drwx-----	2022-04-21 14:30	4 KB
▼ com.example.myapplication	drwx-----	2023-06-03 19:24	4 KB
> cache	drwxrws--x	2023-06-03 18:09	4 KB
> code_cache	drwxrws--x	2023-06-03 18:15	4 KB
> files	drwxrws--x	2023-06-03 18:20	4 KB
▼ shared_prefs	drwxrws--x	2023-06-03 19:24	4 KB
MainActivity.xml	-rw-rw----	2023-06-03 19:24	152 B
> com.google.android.apps.customizatio	drwx-----	2022-04-21 14:30	4 KB

activity_main.xml x MainActivity.xml x MainActivity.kt x build.gradle (:app) x

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
1
2 <map>
3   <string name="articulo">Paleta</string>
4   <int name="cantidad" value="7" />
5
```

Preferencias

```
fun leerPreferencias() {  
    val preferencias: SharedPreferences =  
        getPreferences(MODE_PRIVATE);  
    val art = preferencias.getString("articulo", "valor por  
defecto");  
    val cant = preferencias.getInt("cantidad", 0);  
    findViewById<TextView>(R.id.visualizador).append("Contenido del  
Share Preferences: " + art + " " + cant);  
}
```

Ejemplo: Leyendo desde el
archivo de preferencias por
defecto de la activity