Ficheros

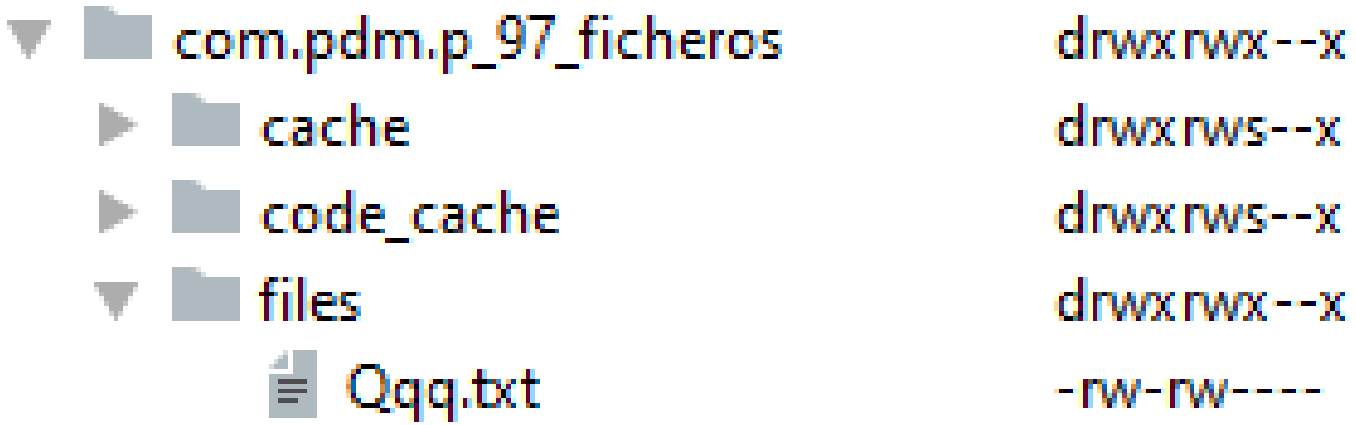
Almacenamiento de datos simples

Ficheros

- En Android también podremos manipular ficheros tradicionales de una forma muy similar a como se realiza en Java .
- Lo primero que hay que tener en cuenta es dónde queremos almacenar los ficheros y el tipo de acceso que queremos tener a ellos. Así, podremos leer y escribir ficheros localizados en:
  - La memoria interna del dispositivo.
  - La propia aplicación, en forma de recurso.
  - La tarjeta SD externa, si está preparada.

Ficheros en la memoria interna del dispositivo.

- Los ficheros se guardan en la ruta /data/data/paquetejava/files



Escritura en memoria interna: método openFileOutput()

```
EditText editText=findViewById(R.id.editText);
String nombreFichero=editText.getText().toString()+".txt";
EditText editText2=findViewById(R.id.editText2);
String texto=editText2.getText().toString();
BufferedWriter writer = null;
try {
   writer = new BufferedWriter(new OutputStreamWriter(openFileOutput(nombreFichero, Context.MODE_PRIVATE)));
   writer.write(texto);
} catch (Exception e) {
   Toast.makeText(getApplicationContext(), "Problemas", Toast.LENGTH_SHORT).show();
} finally {
   if (writer != null) {
      try {
         writer.close();
      } catch (IOException e) {
         Toast.makeText(getApplicationContext(), "Problemas", Toast.LENGTH_SHORT).show();
      }
   }
}
```

- El método openFileOutput() recibe como parámetros el nombre del fichero y el modo de acceso con el que queremos abrir el fichero. Este modo de acceso es igual que para las preferencias.
- Este método devuelve una referencia al *stream* de salida asociado al fichero, a partir del cual ya podremos utilizar los métodos de manipulación de ficheros tradicionales del lenguaje java (api java.io).

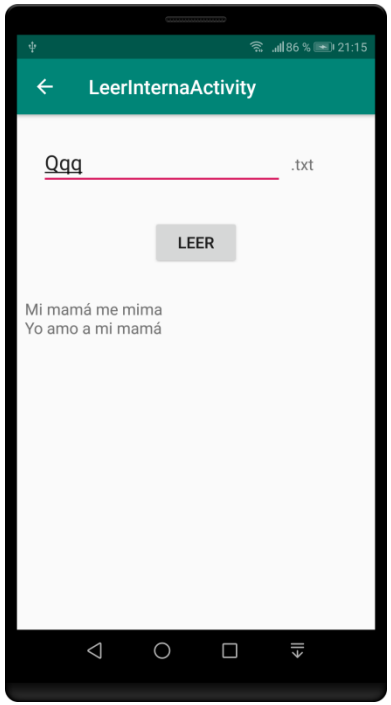
Modos de creación de ficheros para el método openFileOutput()

Modo	Descripción
MODE_APPEND	Si el fichero existe, añadir contenido al final del mismo. Si no existe, se crea
MODE_PRIVATE	Modo por defecto donde el fichero sólo puede ser accedido por la aplicación que lo crea
MODE_WORLD_READABLE	Permite a las demás aplicaciones tener acceso de lectura al fichero creado
MODE_WORLD_WRITEABLE	Permite a las demás aplicaciones tener acceso de escritura al fichero creado

Obsoletos por motivos de seguridad. Se usan otros mecanismos

Lectura de memoria interna: método openFileInput()

```
EditText editText=findViewById(R.id.editText);
String nombreFichero=editText.getText().toString()+".txt";
TextView textView=findViewById(R.id.textView2);
String eol = System.getProperty("line.separator");
BufferedReader input = null;
try {
   input = new BufferedReader(new InputStreamReader(openFileInput(nombreFichero)));
   String line;
   StringBuffer buffer = new StringBuffer();
   while ((line = input.readLine()) != null) {
      buffer.append(line).append(eol);
   }
   textView.setText(buffer);
} catch (Exception e) {
   textView.setText("No existe fichero");
} finally {
   if (input != null) {
      try {
         input.close();
      } catch (IOException e) {
         Toast.makeText(getApplicationContext(), "Problemas", Toast.LENGTH_SHORT).show();
      }
   }
}
```



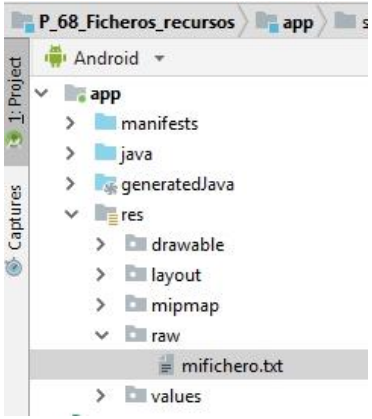
- Para leer ficheros desde la memoria interna se utiliza el método openFileInput() para abrir el fichero que devuelve una referencia al *stream* asociado al fichero a partir del cual ya podremos utilizar los métodos de manipulación de ficheros tradicionales del lenguaje java (api java.io).

Métodos de la clase Context para el manejo de ficheros

Método	Descripción
`getFilesDir()`	Obtiene la ruta absoluta del fichero en el sistema de directorios cuando el fichero interno ha sido creado
`getDir()`	Crea (o abre si ya existe) un directorio dentro de la memoria interna
`deleteFile()`	Borra un fichero guardado en la memoria interna
`fileList()`	Devuelve un array de ficheros que están guardados en la carpeta de la aplicación

Ficheros incluidos como recurso en la propia aplicación

- Aunque este método es útil en muchos casos, sólo se debe utilizar **cuando no se necesite realizar modificaciones sobre los ficheros**, ya que tienen limitado el acceso a sólo lectura.



- Para incluir un fichero como recurso de la aplicación hay que colocarlo en la carpeta /res/raw de la aplicación que no suele estar creada por defecto.
- Se utiliza la clase InputStream

Lectura de datos almacenados en ficheros incluidos como recursos de la aplicación

Importante: el nombre del fichero solo puede contener caracteres a-z y 0-9

```
try {
   InputStream fichero = getResources().openRawResource(R.raw.fichero);
   BufferedReader leer = new BufferedReader(new InputStreamReader(fichero));
   StringBuilder todo = new StringBuilder();
   String linea = leer.readLine();
   while (linea != null) {
      todo.append(linea).append("\n");
      linea = leer.readLine();
   }
   textView.setText(todo.toString());
   fichero.close();
} catch (Exception ex) {
   Toast.makeText(getApplicationContext(), "Problemas con el recurso", Toast.LENGTH_SHORT).show();
}
```

Ficheros en la memoria externa

- La memoria interna suele ser relativamente limitada y no es aconsejable almacenar en ella ficheros de gran tamaño.
- La alternativa natural es utilizar para ello la memoria externa del dispositivo, constituida normalmente por una tarjeta de memoria microSD.
- A diferencia de la memoria interna, **la tarjeta de memoria no tiene por qué estar presente en el dispositivo, e incluso estándolo puede no estar reconocida por el sistema**.
- Por tanto, el primer paso recomendado a la hora de trabajar con ficheros en memoria externa es **asegurarnos de que dicha memoria está presente y disponible para leer y/o escribir en ella.**

Permisos

- Para tener acceso de escritura a la memoria externa tendremos que especificar en el fichero AndroidManifest.xml que nuestra aplicación necesita permiso de escritura en dicha memoria.
- Desde Android 4.1 también es necesario el permiso para lectura.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.pdm.p_66_ficheros_2">
   <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
   <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>

...
```

Permisos peligrosos

- Tener acceso de escritura o de lectura a la memoria externa son permisos considerados peligrosos, por tanto habrá que tratarlos desde código.
- Ejemplo de tratamiento de acceso de escritura a la SD:

```
if (ContextCompat.checkSelfPermission(getApplicationContext(), Manifest.permission.WRITE_EXTERNAL_STORAGE) == PackageManager.PERMISSION_GRANTED) {
        seguir();
} else {
        pedirPermiso();
}
```

Carpeta de almacenamiento

- **Ficheros públicos** (no se borran al desinstalar la aplicación):
  - En la raíz de la SD: NO RECOMENDABLE ya que contamina el espacio de nombres raíz del usuario.

    File nombre_fichero = new File(Environment.getExternalStorageDirectory(), "prueba_sd.txt");
  - En la carpeta prederminada por el tipo de fichero:

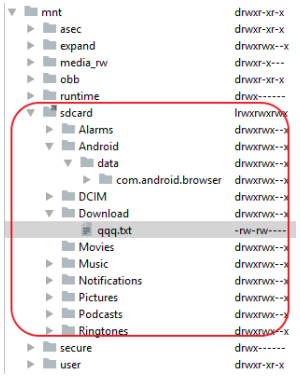
    File nombre_fichero = new File(Environment.getExternalStoragePublicDirectory(**Environment.DIRECTORY_PICTURES**), "prueba_sd.jpg");
- En **una carpeta propia para nuestra aplicación**, lo que además tendrá la ventaja de que al desinstalar la aplicación también se liberará este espacio.
  - En la carpeta raíz de la aplicación:

    File nombre_fichero = new File(getExternalFilesDir(null), "DemoFile.jpg");
  - En la carpeta de la aplicación prederminada por el tipo de fichero:

    File nombre_fichero = new File(getExternalFilesDir(Environment.DIRECTORY_PICTURES), "DemoPicture.jpg");

Carpetas predeterminadas

- Ayudan al sistema Android a saber qué tipo de contenidos hay en cada carpeta, de forma que puedan clasificarse correctamente, por ejemplo en la galería multimedia (no siempre existen!).



Fields		
public static String	DIRECTORY_ALARMS	Standard directory in which to place any audio files that should be in the list of alarms that the user can select (not as regular music).
public static String	DIRECTORY_DCIM	The traditional location for pictures and videos when mounting the device as a camera.
public static String	DIRECTORY_DOCUMENTS	Standard directory in which to place documents that have been created by the user.
public static String	DIRECTORY_DOWNLOADS	Standard directory in which to place files that have been downloaded by the user.
public static String	DIRECTORY_MOVIES	Standard directory in which to place movies that are available to the user.
public static String	DIRECTORY_MUSIC	Standard directory in which to place any audio files that should be in the regular list of music for the user.
public static String	DIRECTORY_NOTIFICATIONS	Standard directory in which to place any audio files that should be in the list of notifications that the user can select (not as regular music).
public static String	DIRECTORY_PICTURES	Standard directory in which to place pictures that are available to the user.
public static String	DIRECTORY_PODCASTS	Standard directory in which to place any audio files that should be in the list of podcasts that the user can select (not as regular music).
public static String	DIRECTORY_RINGTONES	Standard directory in which to place any audio files that should be in the list of ringtones that the user can select (not as regular music).

Requires API Level "19" or higher. To reveal, change the target API level above the left navigation.

Variables de entorno y métodos

- La clase Environment proporciona acceso a los valores de las variables de entorno.

Constants		
String	MEDIA_BAD_REMOVAL	Storage state if the media was removed before it was unmounted.
String	MEDIA_CHECKING	Storage state if the media is present and being disk-checked.
String	MEDIA_MOUNTED	Storage state if the media is present and mounted at its mount point with read/write access.
String	MEDIA_MOUNTED_READ_ONLY	Storage state if the media is present and mounted at its mount point with read-only access.
String	MEDIA_NOFS	Storage state if the media is present but is blank or is using an unsupported filesystem.
String	MEDIA_REMOVED	Storage state if the media is not present.
String	MEDIA_SHARED	Storage state if the media is present not mounted, and shared via USB mass storage.
String	MEDIA_UNKNOWN	Unknown storage state, such as when a path isn't backed by known storage media.
String	MEDIA_UNMOUNTABLE	Storage state if the media is present but cannot be mounted.
String	MEDIA_UNMOUNTED	Storage state if the media is present but not mounted.

Public Methods	
static File	getDataDirectory () Return the user data directory.
static File	getDownloadCacheDirectory () Return the download/cache content directory.
static File	getExternalStorageDirectory () Return the primary external storage directory.
static File	getExternalStoragePublicDirectory (String type) Get a top-level public external storage directory for placing files of a particular type.
static String	getExternalStorageState () Returns the current state of the primary "external" storage device.
static File	getRootDirectory () Gets the Android root directory.
static String	getStorageState (File path) Returns the current state of the storage device that provides the given path.
static boolean	isExternalStorageEmulated () Returns whether the device has an external storage device which is emulated.
static boolean	isExternalStorageRemovable () Returns whether the primary "external" storage device is removable.

Ejemplo de escritura en memoria externa

```
private void seguir() {
   boolean sdDisponible = false;
   boolean sdAccesoEscritura = false;
   String estado = Environment.getExternalStorageState();
   if (estado.equals(Environment.MEDIA_MOUNTED)) {
      sdDisponible = true;
      sdAccesoEscritura = true;
   } else if (estado.equals(Environment.MEDIA_MOUNTED_READ_ONLY)) {
      sdDisponible = true;
      sdAccesoEscritura = false;
   }
   if (!sdDisponible)
      Toast.makeText(this, "No hay tarjeta montada", Toast.LENGTH_LONG).show();
   else if (!sdAccesoEscritura)
      Toast.makeText(this, "Tarjeta de solo lectura", Toast.LENGTH_LONG).show();
   else {
      nombreFichero.setVisibility(View.VISIBLE);
      texto.setVisibility(View.VISIBLE);
      button.setVisibility(View.VISIBLE);
      button.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View view) {
            if (nombreFichero.getText().toString().length() == 0) {
               nombreFichero.requestFocus();
               Toast.makeText(getApplicationContext(), "No te olvides nombre nombreFichero", Toast.LENGTH_SHORT).show();
            } else {
               if (texto.getText().toString().trim().length() == 0) {
                  texto.requestFocus();
                  Toast.makeText(getApplicationContext(), "No te olvides de rellenar el texto", Toast.LENGTH_SHORT).show();
               } else {
                  File f = new File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS), nombreFichero.getText().toString()+".txt");
                  try {
                     OutputStreamWriter fichero = new OutputStreamWriter(new FileOutputStream(f, true));
                     fichero.append(texto.getText().toString());
                     fichero.close();
                     Toast.makeText(getApplicationContext(), "Datos guardados correctamente", Toast.LENGTH_LONG).show();
                  } catch (Exception e) {
                     Toast.makeText(getApplicationContext(), "Error", Toast.LENGTH_LONG).show();
                  }
               }
            }
         }
      });
   }
}
```

el método getExternalStorageStatus(), de la clase Environment devuelve el estado de la memoria

el valor devuelto MEDIA_MOUNTED, indica que la memoria externa está disponible y suponemos que podemos tanto leer como escribir en ella..

el valor devuelto MEDIA_MOUNTED_READ_ONLY, indica que la memoria externa está disponible pero sólo para leer.

getExternalStoragePublicDirectory(): método de la clase Environment que devuelve un objeto File con la ruta de la carpeta especificada.

Se construye otro objeto con el nombre elegido para el fichero, creando un nuevo objeto File que combine ambos elementos (ruta y nombre).

Prácticas propuestas

- Realiza la hoja de ejercicios Ejercicios_22_Ficheros