SEMINARIO DE LENGUAJES OPCIÓN ANDROID

Almacenamiento de datos.

Mg. Corbalán Leonardo, Esp. Delía Lisandro

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.

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.

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

- Una de las alternativas más atractivas es utilizar FireBase (servicio de Google).
- FireBase es una base de datos en la nube en tiempo real. Cualquier cambio realizado en los datos por cualquier cliente (usuario, aplicación, dispositivo...) se sincronizará de forma inmediata (siempre que la conexión lo permita) en el resto de clientes, sin necesidad de que éstos vuelvan 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

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

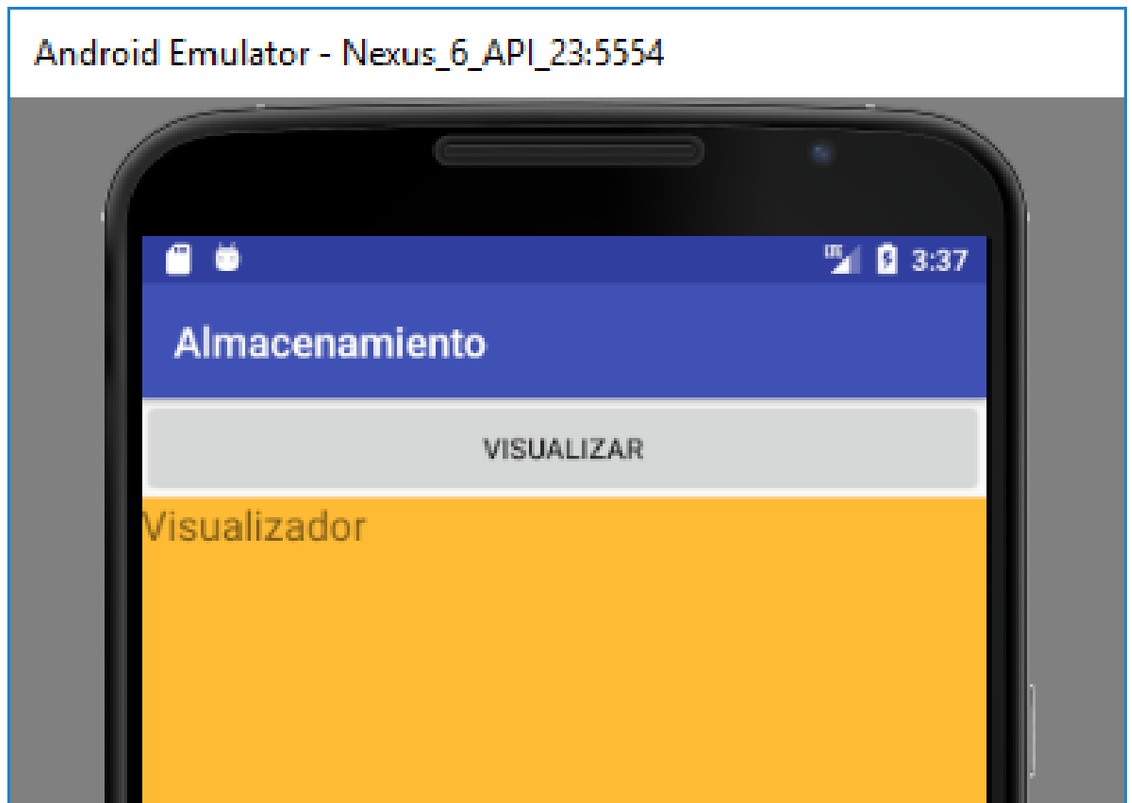
- Almacenamiento interno vs. Almacenamiento externo
  - Cuando el usuario desinstala nuestra app, se borran los archivos de la app del almacenamiento interno.
  - Sin embargo sólo se `borrarán` los archivos de la memoria `externa` si se guardaron en el directorio que se obtiene con `getExternalFileDir()`

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.
  - 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

Almacenamiento – Actividad guiada

- Cree una nueva aplicación en Android Studio denominada Almacenamiento (Minimun SDK = Api 19) 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="visualizar"
        />
    <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.java

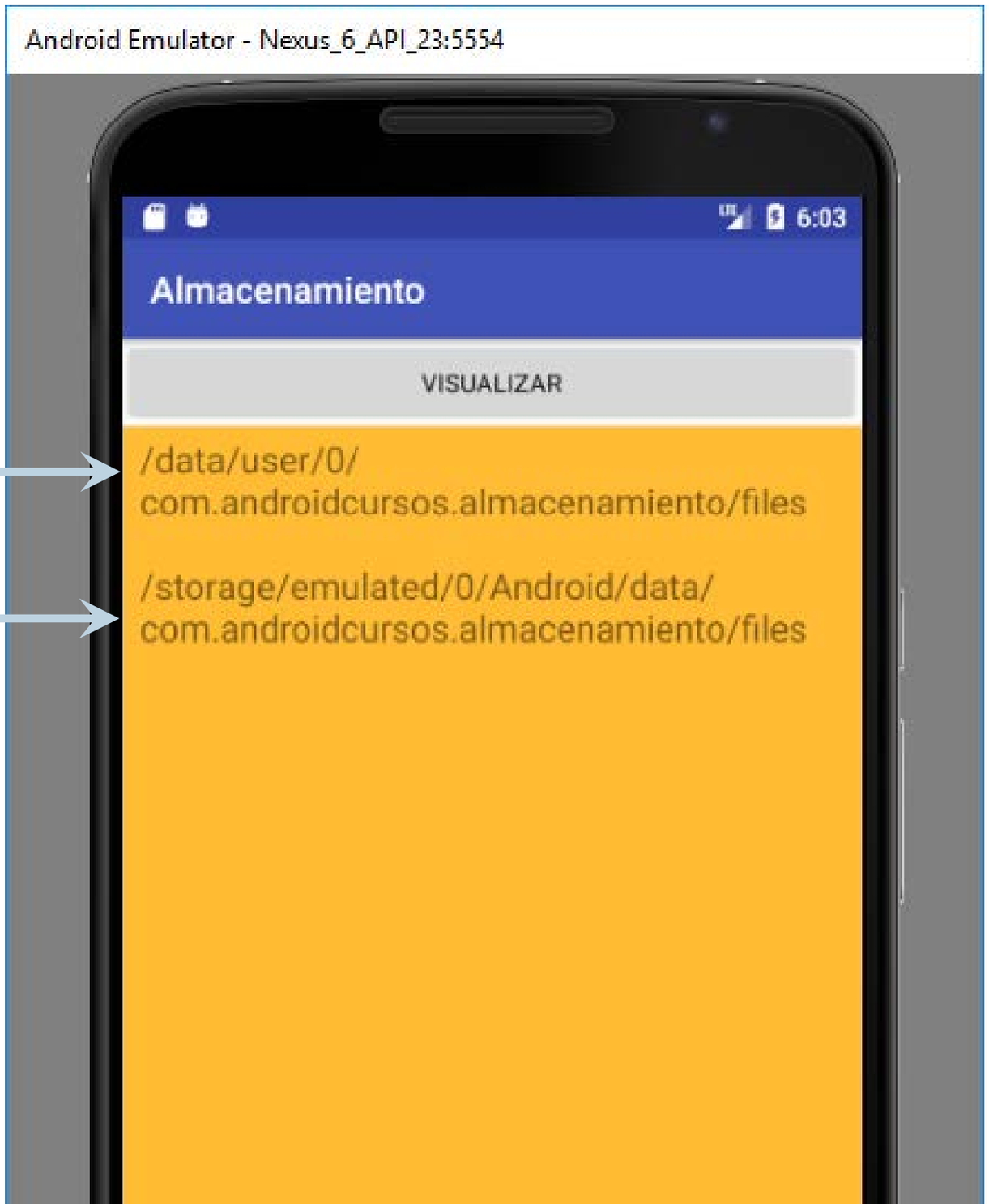
```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public void visualizar(View v)
    {
        String st= this.getFilesDir().getAbsolutePath() + "\n\n";
        st+=this.getExternalFilesDir(null).toString() + "\n\n\n";
        ((TextView)findViewById(R.id.visualizador)).setText(st);
    }
}
```

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

Almacenamiento – Actividad guiada

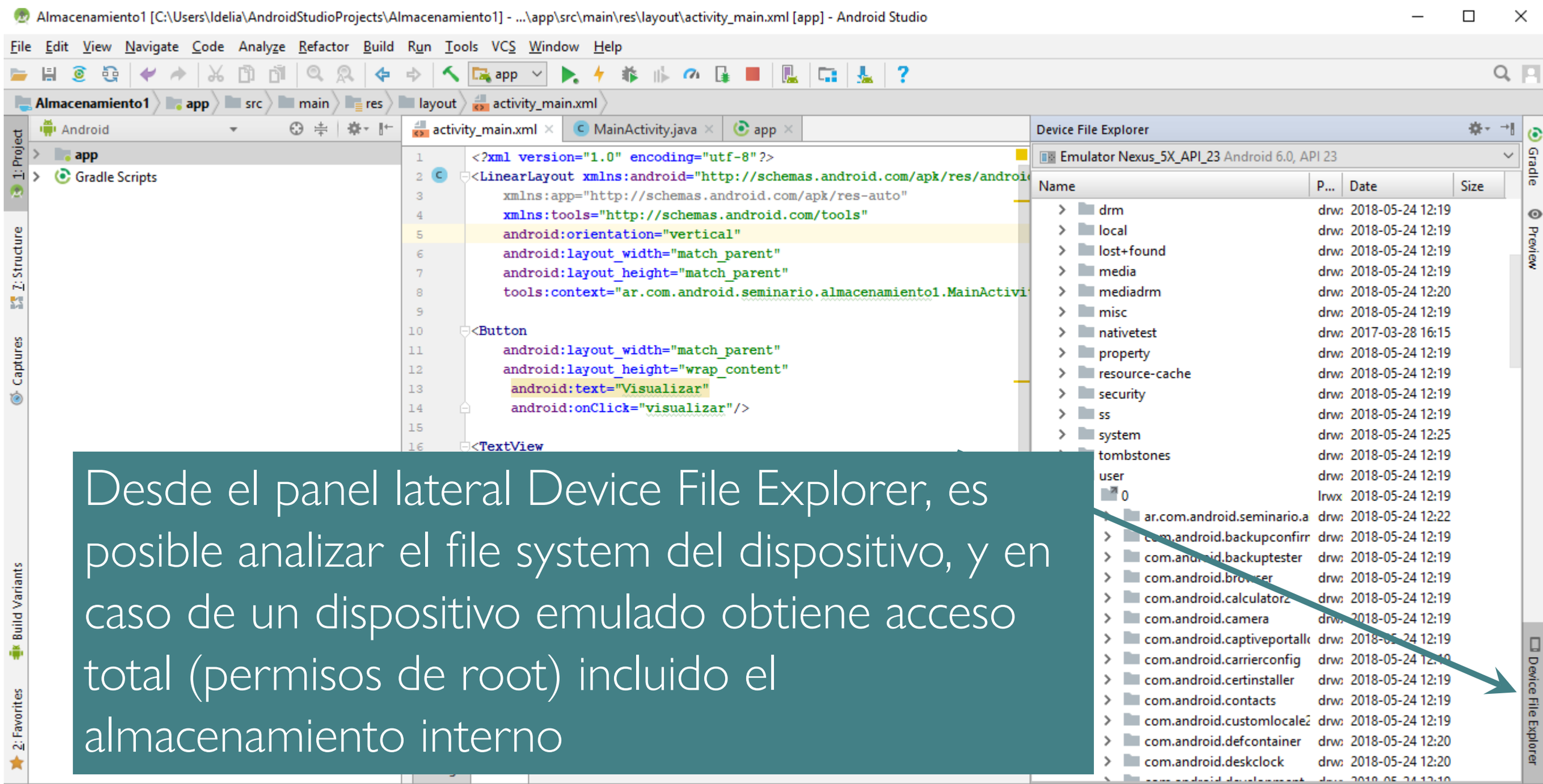
Almacenamiento interno

Partición del espacio de almacenamiento interno montada como memoria externa



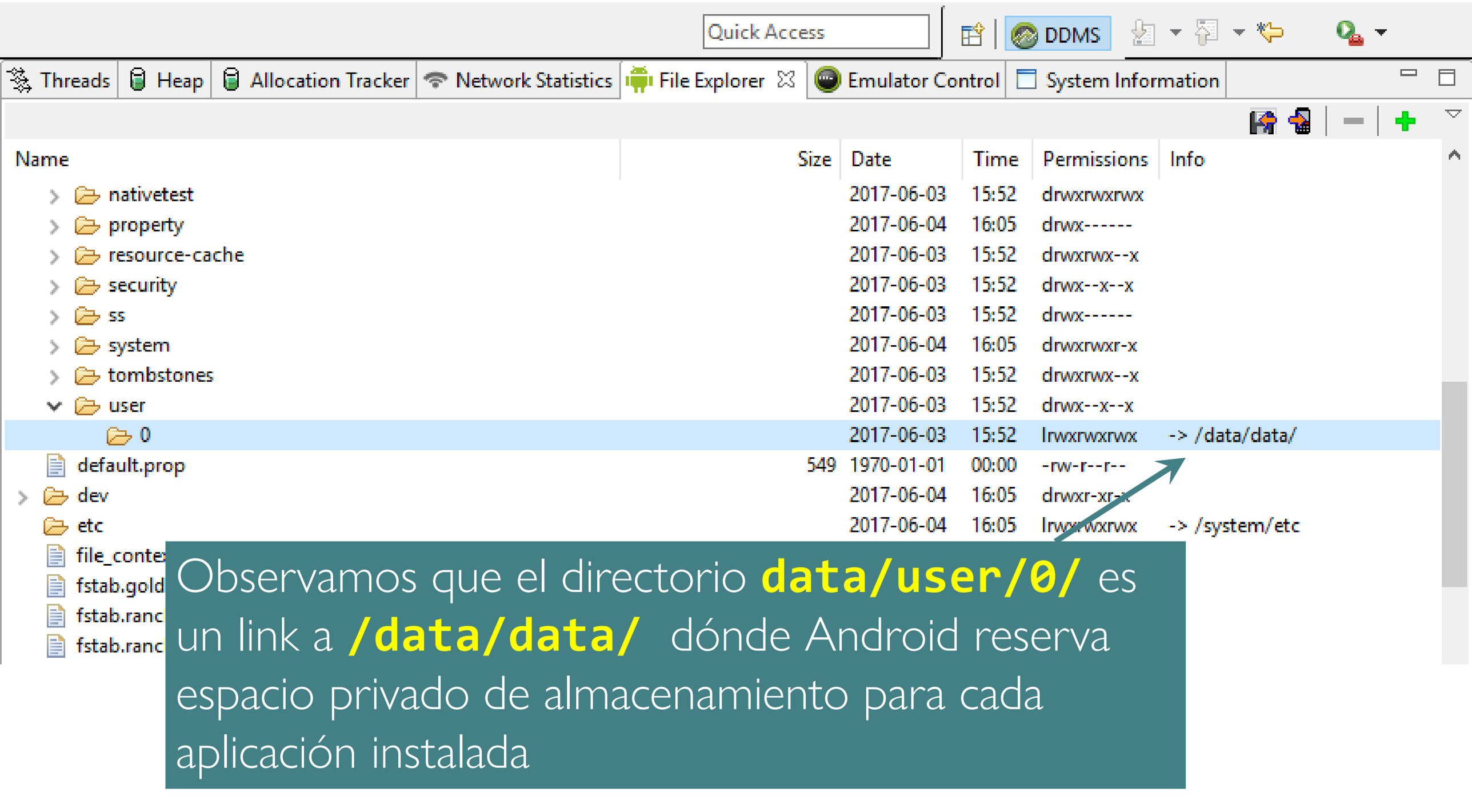
Inspección del dispositivo

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

Almacenamiento – Actividad guiada



Almacenamiento – Actividad guiada

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`

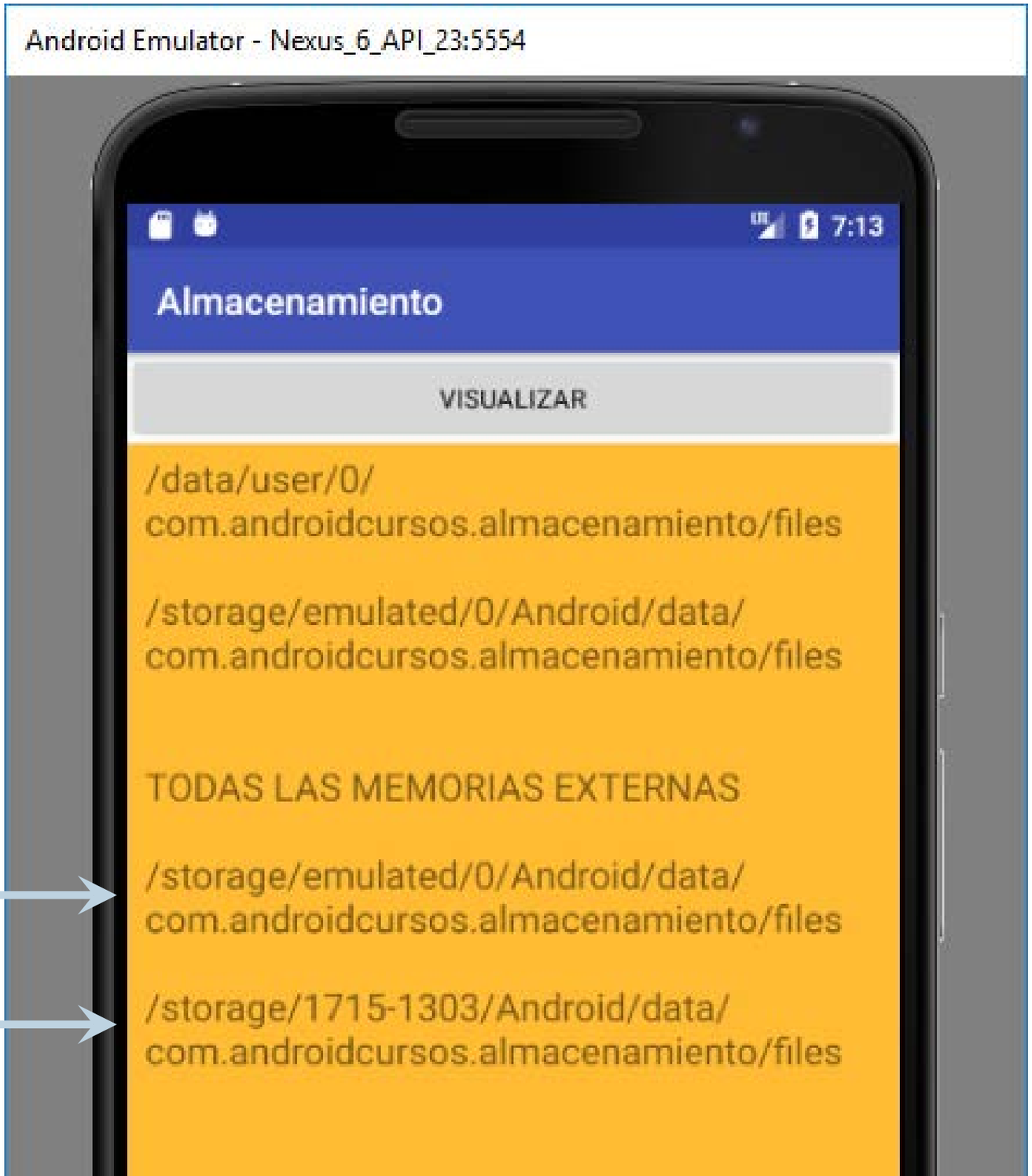
Actividad guiada – MainActivity.java

```
public void visualizar(View v)
{
    String st = this.getFilesDir().getAbsolutePath() + "\n\n";
    st += this.getExternalFilesDir(null).toString() + "\n\n\n";
    st += "TODAS LAS MEMORIAS EXTERNAS\n\n";
    File[] directoriosExternos = this.getExternalFilesDirs(null);
    for (int i = 0; i < directoriosExternos.length; i++) {
        st += directoriosExternos[i].getAbsolutePath() + "\n\n";
    }
    ((TextView) findViewById(R.id.visualizador)).setText(st);
}
```

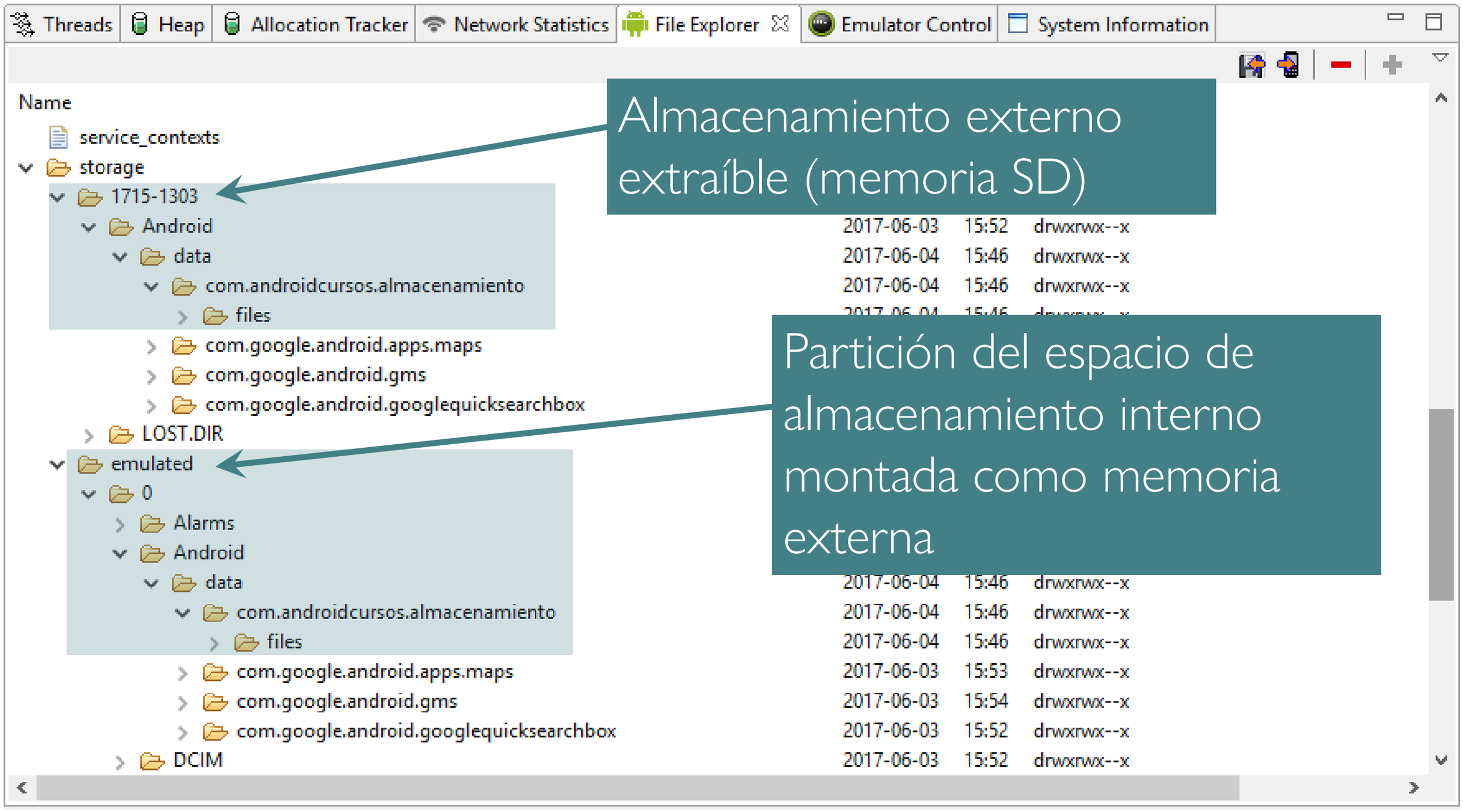
Almacenamiento – Actividad guiada

Partición del espacio de almacenamiento interno montada como memoria externa

Almacenamiento externo extraíble (memoria SD)



Almacenamiento – Actividad guiada



Archivos en la memoria interna

- Además de las clases del paquete `java.io` que pueden utilizarse, para trabajar con archivos en el espacio privado de la aplicación, se han agregado métodos muy útiles a la clase `Context` (superclase de `Activity`).
- `openFileInput()` y `openFileOutput()` se utilizan para abrir un archivo (lectura o escritura respectivamente). Se indica el nombre del archivo (sin la ruta) pues trabaja sobre el directorio privado `files` de la aplicación.

Guardando archivos en la memoria interna

- `openFileOutput()` devuelve un objeto `FileOutputStream` que es útil para escribir `bytes` en el archivo.
- Sin embargo, para escribir datos de distintos tipos utilizaremos un objeto `DataOutputStream` aprovechándonos métodos como `writeInt()`, `writeDouble()`, `writeChar()`, `writeChars()`, `writeBoolean()`, etc.

Guardando archivos en la memoria interna

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

```
private void guardar()
{
    try
    {
        FileOutputStream fos = openFileOutput("datos.bin",
                                              MODE_PRIVATE);
        DataOutputStream salida = new DataOutputStream(fos);

        salida.writeDouble(3.14);
        salida.writeChar('A');

        salida.close();

    } catch (IOException e) {}
}
```

Se abre el archivo

Se escriben datos

Se cierra el archivo

Guardando archivos en la memoria interna

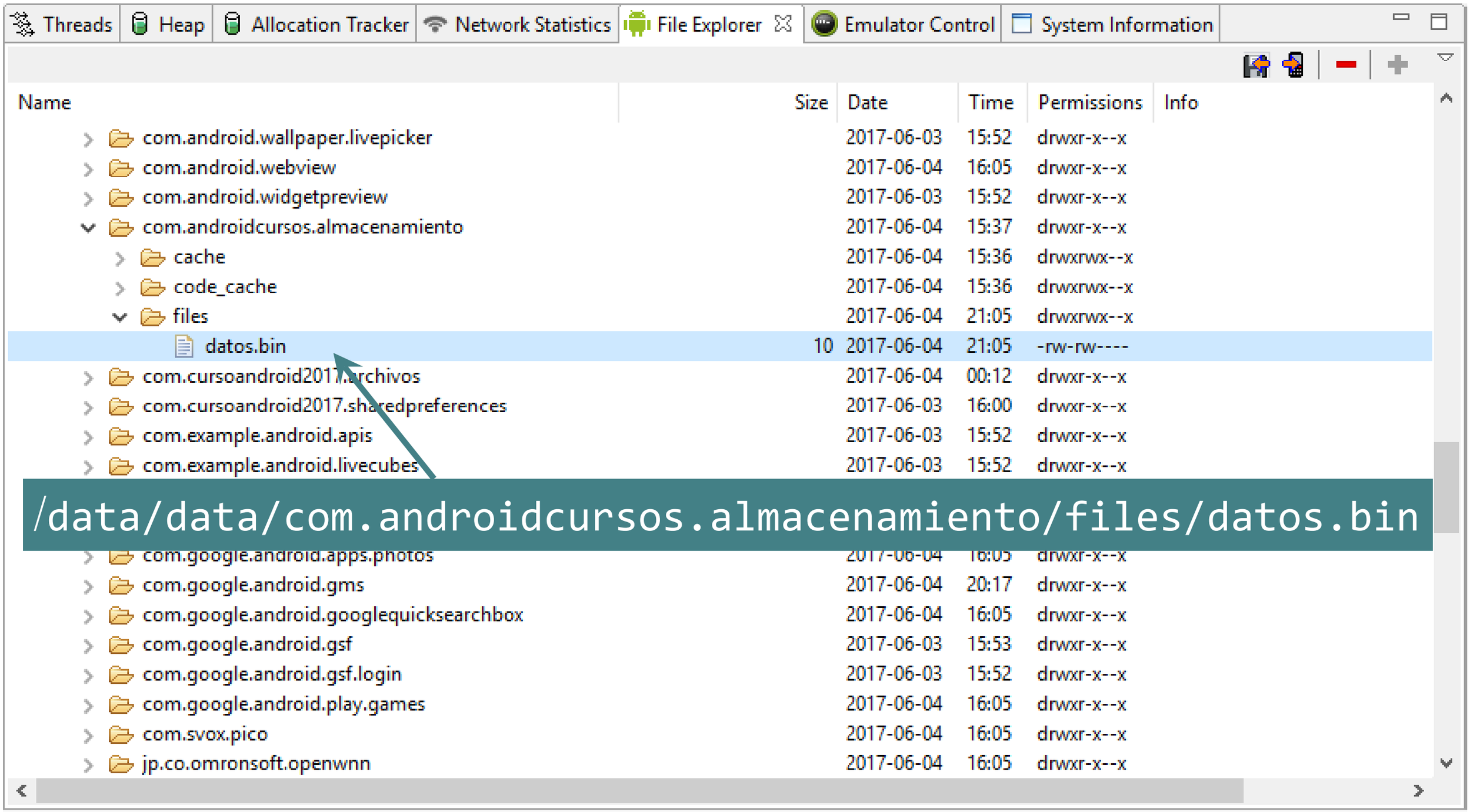
Codificar el siguiente método e invocarlo en el onCreate de la Activity

```
private void guardar()
{
    try
    {
        FileOutputStream fos = openFileOutput("datos.bin",
                                              MODE_PRIVATE);
        DataOutputStream salida = new DataOutputStream(fos);
```

Nota: Actualmente los modos válidos para abrir un **FileOutputStream** son **MODE_PRIVATE** y **MODE_APPEND**, los modos **MODE_WORLD_READABLE** y **MODE_WORLD_WRITABLE** (para permitir a otras aplicaciones acceder al archivo) han sido declarados obsoletos (*deprecated*) en la **API 17** por considerarse riesgosos

```
    }
```

Guardando archivos en la memoria interna



Leyendo archivos desde la memoria interna

- `openFileInput()` devuelve un objeto `FileInputStream` que es útil para leer `bytes` desde el archivo.
- Sin embargo, para leer datos de distintos tipos utilizaremos un objeto `DataInputStream` aprovechándonos métodos como `readInt()`, `readDouble()`, `readChar()`, `readBoolean()`, etc. (Nota: el `readLine()` está `deprecated`)

Leyendo archivos desde la memoria interna

```
private void leer()
{
    double valor=0; char caracter=' ';
    try
    {
        FileInputStream fi = openFileInput("datos.bin");
        DataInputStream entrada = new DataInputStream(fi);

        valor = entrada.readDouble();
        caracter = entrada.readChar();

        entrada.close();

    } catch (IOException e) {}
    ((TextView)findViewById(R.id.visualizador))
      .append("Contenido del archivo: " + valor + " " + caracter);
}
```

Se abre el archivo

Se leen los datos

Se cierra el archivo

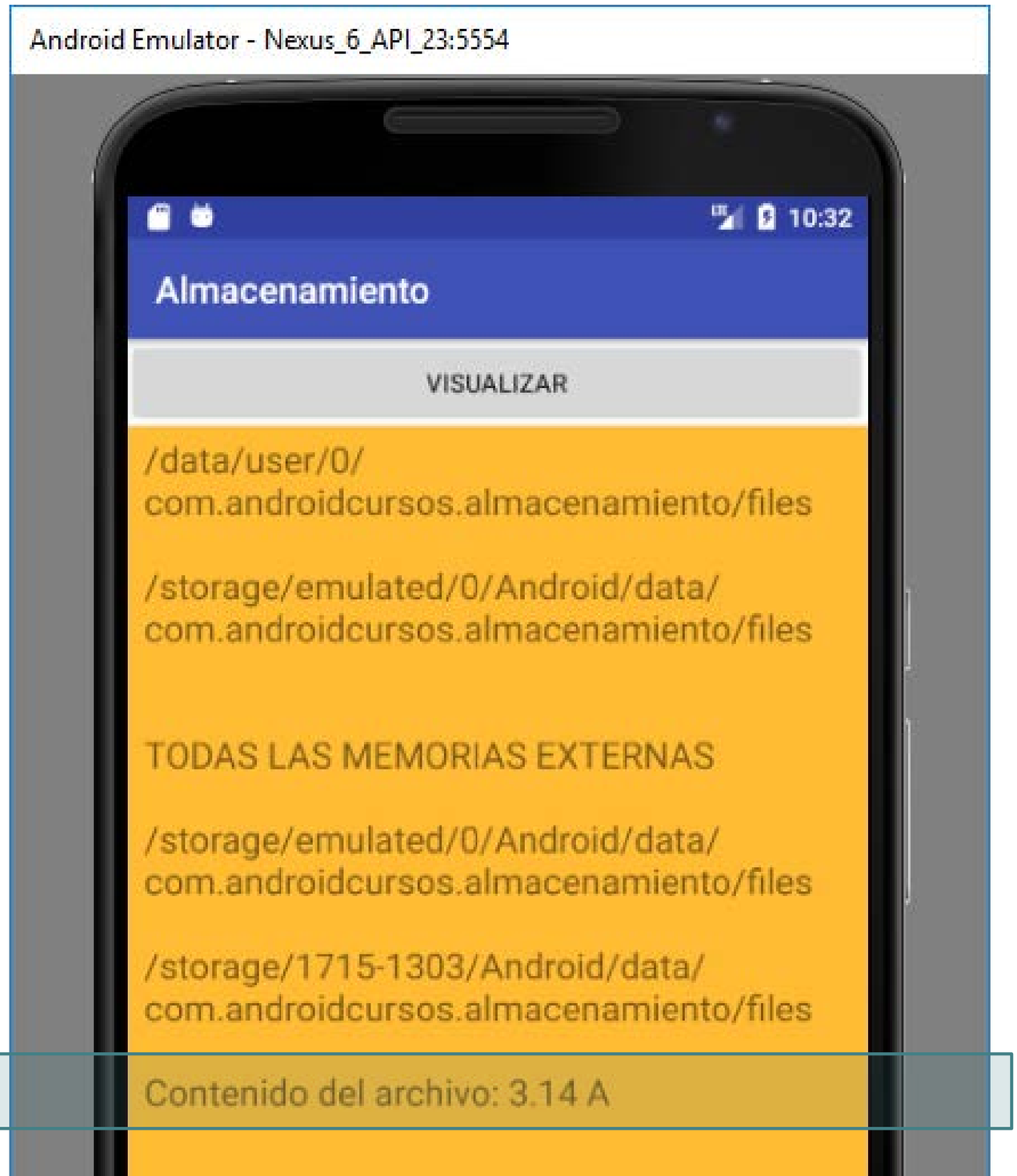
Leyendo archivos desde la memoria interna

```
public void visualizar(View v) {
    String st = this.getFilesDir().getAbsolutePath() + "\n\n";
    st += this.getExternalFilesDir(null).toString() + "\n\n\n";
    st += "TODAS LAS MEMORIAS EXTERNAS\n\n";
    File[] directoriosExternos = this.getExternalFilesDirs(null);
    for (int i = 0; i < directoriosExternos.length; i++) {
        st += directoriosExternos[i].getAbsolutePath() + "\n\n";
    }
    ((TextView) findViewById(R.id.visualizador)).setText(st);
    leer();
}
```

Agregar esta invocación en el método visualizar

Archivos en la memoria interna

Este es el contenido guardado en el archivo datos.bin y luego recuperado



Cómo se guarda y recupera un string usando DataOutputStream y DataInputStream

```
private void guardar() {
    try {
        FileOutputStream fos = openFileOutput("datos.bin",
                                              MODE_PRIVATE);
        DataOutputStream salida = new DataOutputStream(fos);

        salida.writeDouble(3.14);
        salida.writeChar('A');

        salida.writeChars("Hola mundo!\n");

        salida.close();

    } catch (IOException e) {
    }
}
```

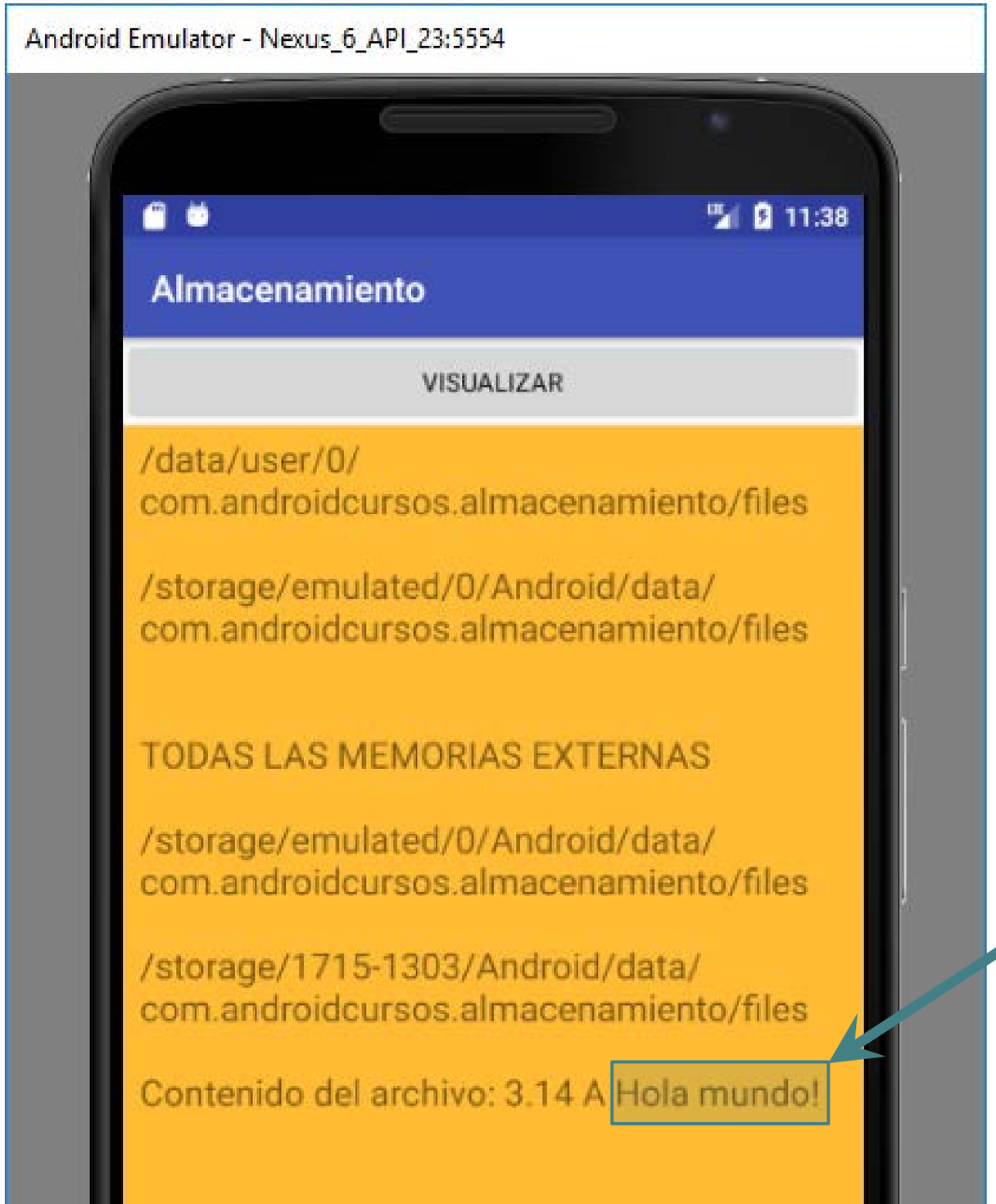
Usar el método **writeChars** del **DataOutputStream**

Cómo se guarda y recupera un string usando DataOutputStream y DataInputStream

```
private void leer() {
    double valor = 0;
    char caracter = ' '; String st = "";
    try {
        FileInputStream fi = openFileInput("datos.bin");
        DataInputStream entrada = new DataInputStream(fi);
        valor = entrada.readDouble();
        caracter = entrada.readChar();

        char c;
        while ((c = entrada.readChar()) != '\n') st += c;

        entrada.close();
    } catch (IOException e) {
    }
    ((TextView) findViewById(R.id.visualizador))
            .append("Contenido del archivo: " + valor + " "
                    + caracter + " " + st);
}
```

Se lee la línea carácter a carácter

Archivos en la memoria interna



En caso de un archivo de texto

- Para **escribir** en un archivo de texto puede utilizarse un `DataOutputStream` con su método `writeChars()`
- Para **leer** un archivo de texto, en lugar de utilizar un `DataInputStream`, es más sencillo hacer uso de un `BufferedReader` que cuenta con el método `readLine()`

En caso de un archivo de texto

```
private void guardarArchivoTexto() {
    try {
        FileOutputStream fos = openFileOutput("datos.txt",
                MODE_PRIVATE);
        DataOutputStream salida = new DataOutputStream(fos);
        salida.writeChars("C U R S O \n");
        salida.writeChars("D E \n");
        salida.writeChars("A N D R O I D \n");

        salida.close();

    } catch (IOException e) {
    }
}
```

Ejemplo: Escribiendo un archivo de texto

En caso de un archivo de texto

Ejemplo: Leyendo un archivo de texto

```
private void leerArchivoDeTexto() {
    String st = "";
    String linea = "";
    try {
        FileInputStream fi = openFileInput("datos.txt");
        BufferedReader entrada =
                new BufferedReader(new InputStreamReader(fi));

        while ((linea = entrada.readLine()) != null) {
            st += linea + '\n';
        }

        entrada.close();
    } catch (IOException e) {
    }
    ((TextView) findViewById(R.id.visualizador))
            .append("Contenido del archivo de texto:\n" + st);
}
```

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

. . .

```
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;
}
```

. . .

Escribiendo en memoria externa

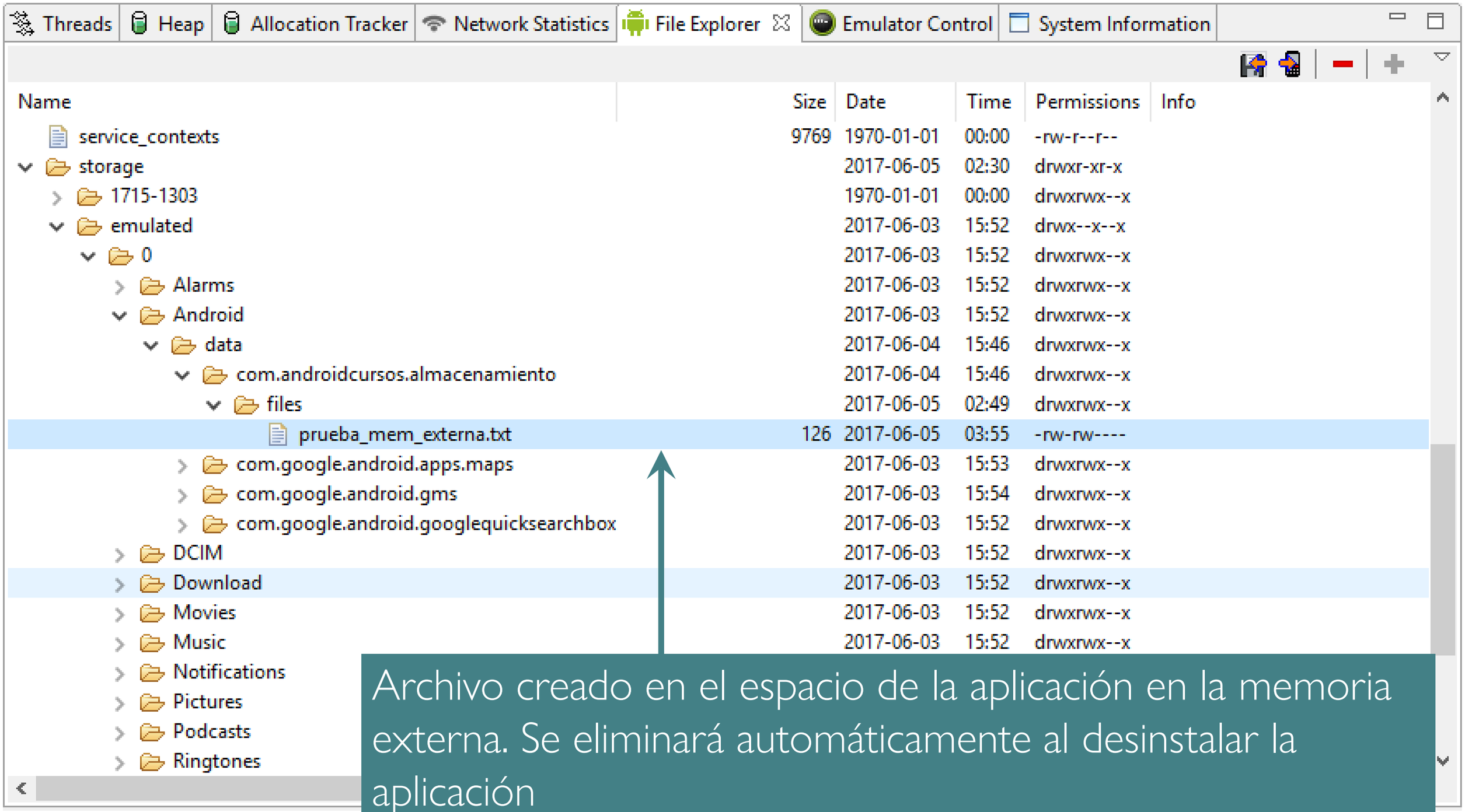
```
private void guardarArchivoTextoEnMemoriaExterna() {
  try {

    String nombreArchi = getExternalFilesDir(null)
            .getAbsolutePath() + "/prueba_mem_externa.txt";

    FileOutputStream f = new FileOutputStream(nombreArchi);
    DataOutputStream salida = new DataOutputStream(f);

    salida.writeChars("Escribiendo en memoria externa\n");
    salida.writeChars("en el espacio de la aplicación \n");

    salida.close();

  } catch (IOException e) {
  }
}
```

Espacio de la aplicación en la memoria externa

Se crea un **DataOutputStream** en base a un **FileOutputStream**

Ejemplo: Escribiendo en memoria externa

Escribiendo en memoria externa



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

Escribiendo en memoria externa

Nota: A partir de Android 4.4 (API 19) ya no es necesario contar con permisos para acceder al espacio de la aplicación en la memoria externa. Sin embargo, si la app debe correr en versiones anteriores o se necesita acceso a un directorio en la memoria externa fuera del área de la app, deben definirse los permisos correspondientes en el `manifest`.

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

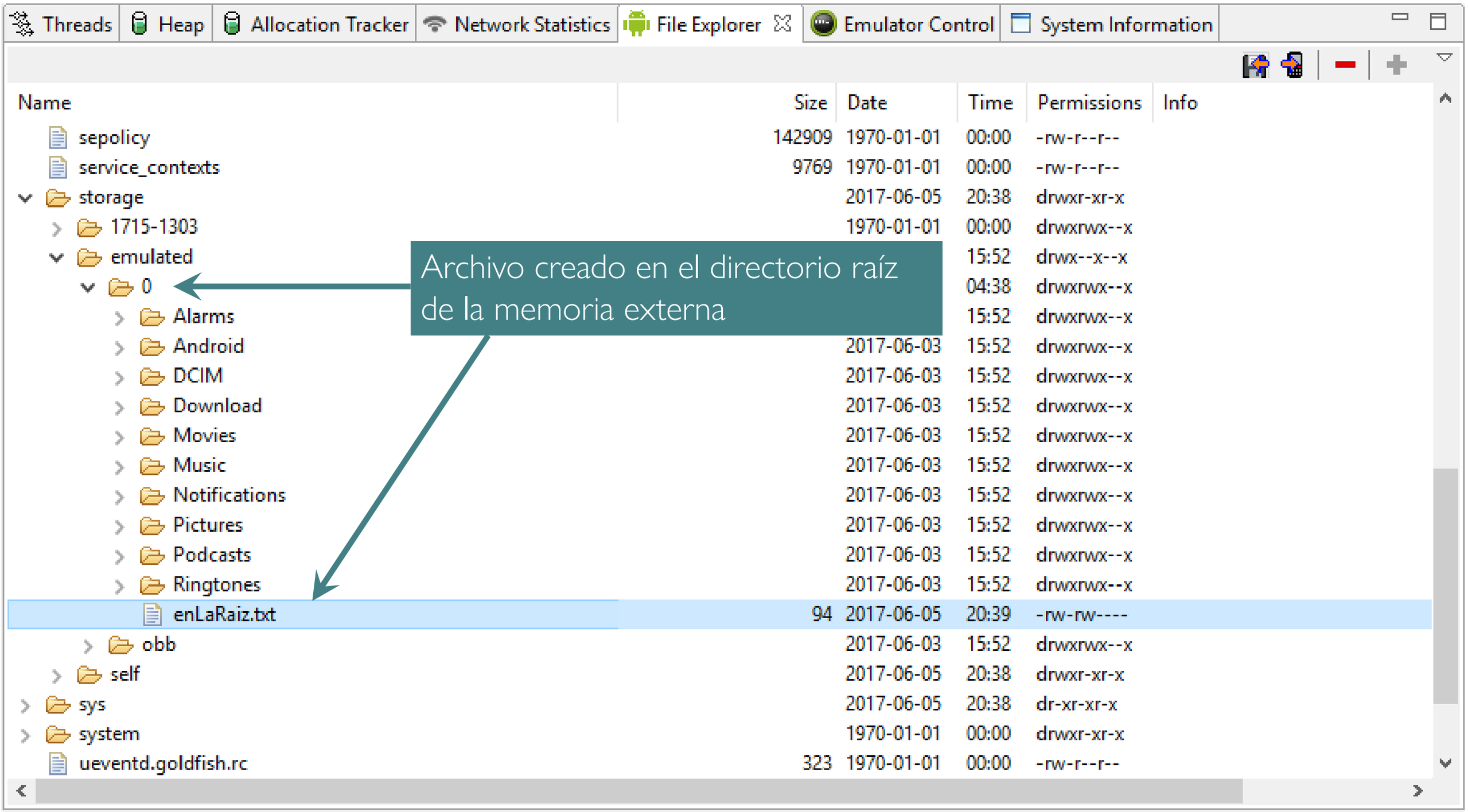
Escribiendo en memoria externa

```
private void guardarEnRaizDeMemoriaExterna() {
  try {
    String nombreArchi = Environment.getExternalStorageDirectory()
                + "/enLaRaiz.txt";

    FileOutputStream f = new FileOutputStream(nombreArchi);
    DataOutputStream salida = new DataOutputStream(f);
    salida.writeChars("Escribiendo en la raíz \n");
    salida.writeChars("de la memoria externa \n");
    salida.close();

  } catch (IOException e) {
  }
}
```

Directorio raíz de la memoria externa

Ejemplo: Escribiendo en la raíz de la memoria externa

Escribiendo en memoria externa



Leyendo desde memoria externa

- Para ejercitar en casa:
  - En base al material expuesto sobre cómo escribir un archivo en memoria externa y como leer desde uno en memoria interna, implementar análogamente un método para leer el contenido de un archivo desde la memoria externa

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. Hoy se aconseja utilizar sólo `MODE_PRIVATE` (`MODE_WORLD_READABLE` y `MODE_WORLD_WRITEABLE` están *deprecated*)

Preferencias

```
private void guardarPreferencias() {
    SharedPreferences preferencias = getPreferences(MODE_PRIVATE);
    SharedPreferences.Editor 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

Threads | Heap | Allocation Tracker | Network Statistics | File Explorer | Emulator Control | System Information

Name	Size	Date	Time	Permissions	Info
com.android.server.telecom		2017-06-03	15:52	drwxr-x--x	
com.android.settings		2017-06-03	16:04	drwxr-x--x	
com.android.statementservice		2017-06-03	15:52	drwxr-x--x	
com.android.systemui		2017-06-03	15:52	drwxr-x--x	
com.android.vending					
com.android.vpndialogs					
com.android.wallpaper.livepicker					
com.android.webview					
com.android.widgetpreview					
com.androidcursos.almacenamiento					
cache					
code_cache					
files					
shared_prefs		2017-06-05	21:38	drwxrwx--x	
MainActivity.xml	152	2017-06-05	21:38	-rw-rw----	
com.cursoandroid2017.archivos		2017-06-04	00:12	drwxr-x--x	
com.cursoandroid2017.sharedpreferences		2017-06-03	16:00	drwxr-x--x	

`/data/data/com.androidcursos.almacenamiento`
espacio privado de nuestra aplicación

Archivo de preferencias de MainActivity

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <int name="cantidad" value="3" />
    <string name="articulo">paleta</string>
</map>
```

Preferencias

```
private void leerPreferencias() {
  SharedPreferences preferencias = getPreferences(MODE_PRIVATE);
  String art = preferencias.getString("articulo", "valor por
                                                defecto");
  int cant = preferencias.getInt("cantidad", 0);
}
```

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