

Diseño y Programación de Dispositivos Móviles

EIF 204

Profesor: Ms.C Gregorio Villalobos Camacho

Correo: gregorio.villalobos.camacho@una.ac.cr

Almacenamiento Interno Android

Android: Almacenamiento Interno

- Es otra opción para guardar datos de Android
- Se utiliza un archivo de texto
- Se almacena dentro del móvil
- Permite almacenar más información
- La información solo se puede leer por el app
- Los datos están ocultos
 - Para otras aplicaciones
 - Para el usuario del dispositivo móvil

Android: Almacenamiento Interno

- Cada app tiene un espacio para almacenar (data/data/package_name/files)
 - Aquí se almacenan los archivos del app
 - Si se desinstala el app todo esto se borra/elimina
- Al iniciar el app hay que cargar el o los archivos
 - Para esto utilizamos un método llamado `fileList()`
 - Este método devuelve un arreglo de datos
 - Contiene los nombres de los archivos almacenados

Android: Almacenamiento Interno

- En el onCreate
 - Llamamos al método leerArchivo()
 - Se incluye el try-catch
 - Para capturar cualquier error de input/output

```
try {
    leerArchivo();
}catch(IOException e){
}
```

Android: Almacenamiento Interno

- Método leerArchivo()
 - Declara un arreglo string para los nombres de los archivos
 - Se carga el arreglo con el método fileList()
 - Se verifica si el archivo que buscamos existe
 - Se puede utilizar un método booleano separado
 - Si el archivo existe
 - Creamos un InputStreamReader para leer nuestro archivo
 - Creamos un BufferedReader para extraer el contenido
 - Creamos un String línea, para leer por línea del BufferedReader

Android: Almacenamiento Interno

- Método leerArchivo() (continuación)
 - Mientras el string de línea no esté vacío
 - Se carga todo en un string lista
 - Se coloca un salto de línea al final para acomodar bien
 - Se cierra el buffer
 - Se cierra el archivo
 - Se coloca el string lista dentro del texto multi-línea del app para mostrarlo al usuario

Android: Almacenamiento Interno

- Método boolean `existeArchivo()`
 - Este método recibe
 - El arreglo con el o los nombres de archivos
 - El nombre del archivo a ser buscado
 - Recorreremos el arreglo buscando el archivo
 - Si lo encuentra se devuelve verdadero
 - En caso contrario se devuelve false
 - Se debe comparar con `.equals()` el string

Android: Almacenamiento Interno

- Método salvar(View vista)
 - Se agrega View así lo puede ver el botón
 - Se crea un OutputStreamWriter
 - Se le pide OpenFileOutput con el nombre del archivo
 - Una vez abierto se escribe su contenido
 - Al finalizar se debe limpiar el buffer con flush()
 - Finalmente se cierra con .close()
- Veamos ahora todo el código fuente

Android: Almacenamiento Interno

- onCreate

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    datos = (EditText) findViewById(R.id.et_ToDoList);
    bt_Guardar = findViewById(R.id.bt_Guardar);
    String archivos[] = fileList();
    try {
        leerArchivo();
    } catch (IOException e){

    }
}
```

Android: Almacenamiento Interno

```
private void leerArchivo() throws IOException {
    String archivos [] = fileList();
    if(existeArchivo(archivos, nombre: "tareas.txt")){
        InputStreamReader archivo = new InputStreamReader((openFileInput(name: "tareas.txt")));
        BufferedReader miBuffer = new BufferedReader(archivo);
        String linea = miBuffer.readLine();
        String listaCompleta = "";
        while(linea != null){
            listaCompleta = listaCompleta + linea + "\n";
            linea = miBuffer.readLine();
        }
        miBuffer.close();
        archivo.close();
        datos.setText(listaCompleta);
    }
}
```

Android: Almacenamiento Interno

```
public boolean existeArchivo(String archivos[], String nombre){
    for(int i = 0; i < archivos.length; i++){
        if(nombre.equals(archivos[i])){
            return true;
        }
    }
    return false;
}
```

Android: Almacenamiento Interno

```
public void guardar(View vista){
    try {
        OutputStreamWriter archivo = new OutputStreamWriter(openFileOutput("tareass.txt", Activity.MODE_PRIVATE));
        archivo.write(datos.getText().toString());
        archivo.flush();
        archivo.close();
        Toast.makeText(context: this, text: "Datos salvados!", Toast.LENGTH_SHORT).show();
    } catch (IOException e){

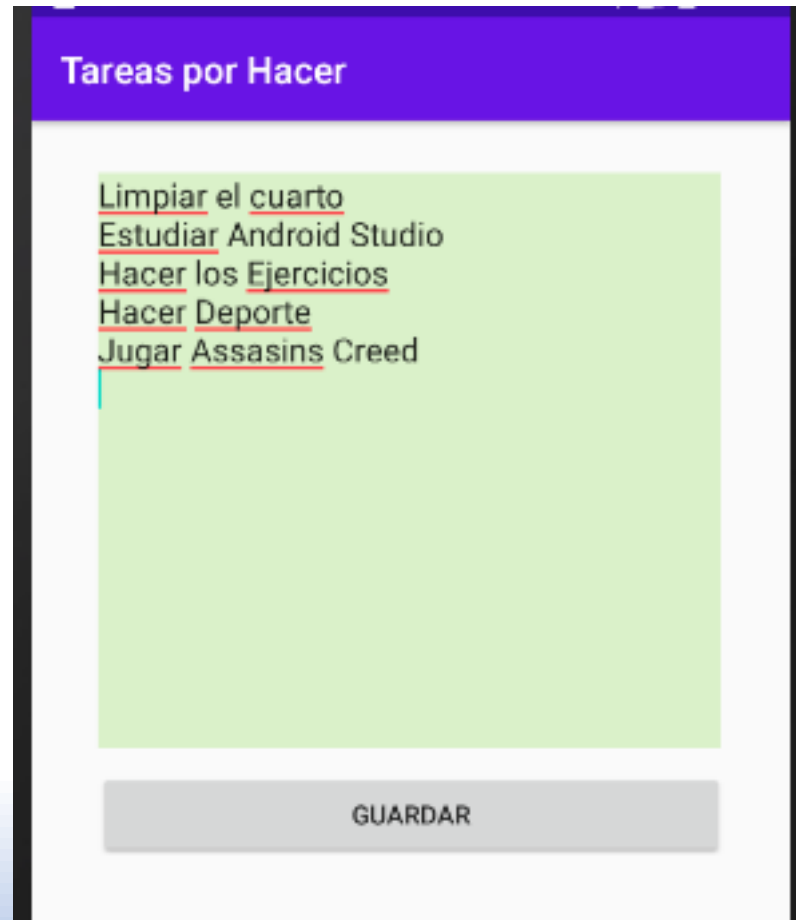
    }
    //finish();
}
```

Android: Almacenamiento Interno

- Haga el ejercicio explicado anteriormente
- Note que la pantalla tiene
 - Un Multi-Línea
 - Un botón para salvar
 - Este botón salva y cierra el app

Android: Almacenamiento Interno

- Haga el ejercicio explicado anteriormente



The screenshot shows an Android application with a purple header bar containing the text "Tareas por Hacer". Below the header is a light green rectangular area containing a list of tasks, each underlined: "Limpiar el cuarto", "Estudiar Android Studio", "Hacer los Ejercicios", "Hacer Deporte", and "Jugar Assasins Creed". At the bottom of the application is a grey button with the text "GUARDAR".

Almacenamiento Utilizando SD Card

Almacenamiento SD

- Múltiples opciones de almacenamiento
- El usuario tiene más control
 - Coloca nombre del archivo
 - Coloca contenido
- Tenemos todo el espacio disponible en el SD
- Se comporta similar al proyecto anterior
- El cambio es dónde estamos almacenando

Almacenamiento SD

- Podemos crear un proyecto igual al anterior
 - Con esto tenemos la estructura externa
 - Le vamos a agregar los cambios para usar el SD
- Ejercicio
- Construya la estructura externa
 - 1 Edit text para el nombre
 - 1 Multi-Line Text para el detalle
 - Dos botones SALVAR y CONSULTAR
- Acomode su estructura básica como siempre

Almacenamiento SD Ejercicio

- Debemos modificar el Manifest de nuevo
 - Tenemos que dar permiso para usar SD

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

- Vea que la instrucción es similar a la anterior
- Para salvar y consultar
 - Debemos usar try-catch por IOExceptions

Almacenamiento SD Ejercicio: Salvar

```
public void salvar(View vista){
    String nombre = et_Nombre.getText().toString();
    String detalle = et_Datos.getText().toString();
    try {
        File tarjetaSD = Environment.getExternalStorageDirectory();
        Toast.makeText(context: this, tarjetaSD.getPath(), Toast.LENGTH_LONG).show();
        File rutaArchivo = new File(tarjetaSD.getPath(), nombre);
        OutputStreamWriter escribirArchivo = new OutputStreamWriter(openFileOutput(nombre, Activity.MODE_PRIVATE));
        escribirArchivo.write(detalle);
        escribirArchivo.flush();
        escribirArchivo.close();
        Toast.makeText(context: this, text: "Almacenado correctamente!", Toast.LENGTH_SHORT).show();
        et_Nombre.setText("");
        et_Datos.setText("");
    } catch (IOException e){
        Toast.makeText(context: this, text: "No se pudo guardar", Toast.LENGTH_SHORT).show();
    }
}
```

Almacenamiento SD Ejercicio: Consultar

```
public void consultar(View vista){
    String nombre = et_Nombre.getText().toString();
    try {
        File tarjetaSD = Environment.getExternalStorageDirectory();
        File rutaArchivo = new File(tarjetaSD.getPath(), nombre);
        InputStreamReader abrirArchivo = new InputStreamReader(openFileInput(nombre));
        BufferedReader leerArchivo = new BufferedReader(abrirArchivo);
        String linea = leerArchivo.readLine();
        String contenido = "";
        while(linea != null){
            contenido = contenido + linea + "\n";
            linea = leerArchivo.readLine();
        }
        leerArchivo.close();
        abrirArchivo.close();
        et_Datos.setText(contenido);
    }catch (IOException e){
        Toast.makeText(context: this, text: "No se pudo abrir", Toast.LENGTH_SHORT).show();
    }
}
```

Almacenamiento SQL Utilizando SQLite

Almacenamiento SQL

- Android tiene SQL Lite a disposición
 - Es un gestor muy popular
 - Es pequeño
 - No necesita servidor
 - Necesita poca configuración
 - Es transaccional (SQL)
 - Es de código libre
- En Android usamos [SQLiteOpenHelper](#)

Almacenamiento SQL

- SQLiteOpenHelper
 - Tiene solo un constructor
 - No se necesita sobre-escribir
 - Métodos abstractos
 - onCreate() y onUpgrade()
 - Debemos implementarlos
 - Con ellos creamos y actualizamos nuestra base de datos

Almacenamiento SQL: Ejercicio

- Vamos a trabajar con una app como esta



The image shows a web application interface for managing data in a SQL database. It features three input fields for data entry: 'Código', 'Nombre', and 'Precio'. Below these fields are four buttons arranged in a 2x2 grid: 'INSERTAR', 'MODIFICAR', 'CONSULTAR', and 'ELIMINAR'. The interface is clean and minimalist, with a light gray background and white input fields.

Almacenamiento SQL: Ejercicio

- Creamos un proyecto nuevo
- En la carpeta Java agregamos una clase nueva
 - En la misma carpeta donde está el MainActivity.java
- Al inicio de la clase debemos importar
`import android.database.sqlite.SQLiteOpenHelper;`
- La clase hereda de `SQLiteOpenHelper`
- Le pedimos a Android que construya
 - Los métodos abstractos que hay que implementar
 - El constructor con 4 parámetros

Almacenamiento SQL: Ejercicio

- Modificamos el onCreate de la nueva clase

```
@Override
public void onCreate(SQLiteDatabase BaseDeDatos) {
    BaseDeDatos.execSQL("create table articulos(codigo int primary key, descripcion text, precio real)");
}
```

- En el MainActivity.java
 - Creamos los EditText
 - Conectamos los EditText con la parte visual

Almacenamiento SQL: Ejercicio

- En el MainActivity.java
 - Podemos crear un método para cada botón
 - Insertar()
 - Eliminar()
 - Consultar()
 - Modificar()
- Ver las siguientes filminas para su respectivo código

```
public void insertar(View vista){
    AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(context: this, name: "administracion", factory: null, version: 1);
    SQLiteDatabase baseDatos = admin.getWritableDatabase();
    String codigo = et_codigo.getText().toString();
    String descripcion = et_descripcion.getText().toString();
    String precio = et_precio.getText().toString();
    if(!codigo.isEmpty() && !descripcion.isEmpty() && !precio.isEmpty()){
        ContentValues registro = new ContentValues();
        registro.put("codigo", codigo);
        registro.put("descripcion", descripcion);
        registro.put("precio", precio);
        baseDatos.insert(table: "articulos", nullColumnHack: null, registro);
        baseDatos.close();
        et_codigo.setText("");
        et_precio.setText("");
        et_descripcion.setText("");
        Toast.makeText(context: this, text: "Inserción Correcta", Toast.LENGTH_SHORT).show();
    }else{
        Toast.makeText(context: this, text: "Debe llenar todos los datos", Toast.LENGTH_SHORT).show();
    }
}
```

Consultas Con SQL

Almacenamiento SQL: Ejercicio

```
public void consultar(View vista){
    AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(context: this, name: "administracion", factory: null, version: 1);
    SQLiteDatabase baseDatos = admin.getWritableDatabase();
    String codigo = et_codigo.getText().toString();
    if(!codigo.isEmpty()){
        Cursor fila = baseDatos.rawQuery(sql: "select descripcion, precio from articulos where codigo = "
            + codigo, selectionArgs: null);
        if(fila.moveToFirst()){
            et_descripcion.setText(fila.getString(columnIndex: 0));
            et_precio.setText(fila.getString(columnIndex: 1));
        }else{
            Toast.makeText(context: this, text: "Datos no encontrados!", Toast.LENGTH_SHORT).show();
        }
    }else{
        Toast.makeText(context: this, text: "Debe introducir el código para buscar", Toast.LENGTH_SHORT).show();
    }
    baseDatos.close();
}
```

Eliminar Con SQL


```
public void eliminar(View vista){
    AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(context: this, name: "administracion", factory: null, version: 1),
    SQLiteDatabase baseDatos = admin.getWritableDatabase();
    String codigo = et_codigo.getText().toString();
    if(!codigo.isEmpty()){
        int cantidad = baseDatos.delete(table: "articulos", whereClause: "codigo=" + codigo, whereArgs: null);
        if(cantidad == 1){
            Toast.makeText(context: this, text: "Datos borrados exitosamente", Toast.LENGTH_SHORT).show();
        }else{
            Toast.makeText(context: this, text: "Articulo no encontrado!", Toast.LENGTH_SHORT).show();}
        baseDatos.close();
        et_codigo.setText("");
        et_precio.setText("");
        et_descripcion.setText("");
    }else{
        Toast.makeText(context: this, text: "Debe indicar un código para eliminarlo", Toast.LENGTH_SHORT).show();
    }
}
```

Modificar Con SQL

```
public void modificar(View vista){
    AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(context: this, name: "administracion", factory: null, version: 1);
    SQLiteDatabase baseDatos = admin.getWritableDatabase();
    String codigo = et_codigo.getText().toString();
    String descripcion = et_descripcion.getText().toString();
    String precio = et_precio.getText().toString();
    if(!codigo.isEmpty() && !descripcion.isEmpty() && !precio.isEmpty()){
        ContentValues registro = new ContentValues();
        registro.put("codigo", codigo);
        registro.put("descripcion", descripcion);
        registro.put("precio", precio);
        int cantidad = baseDatos.update(table: "articulos", registro, whereClause: "codigo="+ codigo, whereArgs: null);
        baseDatos.close();
        if(cantidad == 1){
            Toast.makeText(context: this, text: "Datos modificados exitosamente", Toast.LENGTH_SHORT).show();
        }else{
            Toast.makeText(context: this, text: "Artículo no encontrado!", Toast.LENGTH_SHORT).show();
        }
    }else{
        Toast.makeText(context: this, text: "Debe llenar todos los campos", Toast.LENGTH_LONG).show();
    } }
}
```