COMP-304
Fall 2018

Review of Lecture 6

- Applying **tweened animation transformations** to View objects
  - define tweening transformations as **XML resource** files or **programmatically:**
    - **Transparency** changes (Alpha)
    - **Rotations** (Rotate)
    - **Scaling** (Scale)
    - **Movement** (Translate)
  - store animation sequences as specially formatted XML files within the **/res/anim/** resource directory
  - load an animation from xml file using AnimationUtils.**loadAnimation method**
- **Shared Preferences**
  - A **lightweight data storage** mechanism called **shared preferences** for storing: **application state** simple **user information configuration options**
  - Store private primitive data in **key-value** pairs

SharedPreferences settingsActivity = **getPreferences**(MODE_PRIVATE);

SharedPreferences settings = **getSharedPreferences**("MyPrefs", 0);

Review of Lecture 6

- **SQLite** databases
  - Open-source
  - Standards-compliant
  - Lightweight
  - Single-tier
  - stored in the **/data/data/<package_name>/databases** folder on your device (or emulator).
- **Content Values** are used to **insert** new rows into tables
- **Cursor** class provides navigation methods
- **Create a helper class** which extends the **SQLiteOpenHelper** abstract class:
  - override the **constructor**, **onCreate**, and **onUpgrade**
  - To open a writable database:

SQLiteDatabase db = hoardDBOpenHelper.**getWritableDatabase()**;

  - **Use query** method to execute queries
  - Use the **moveTo<location>** methods to position the cursor at the correct row of the result **Cursor**
  - Use the type-safe **get<type>** methods to return the value stored at the current row for the specified column

Review of Lecture 6

- **Use ContentValues and insert, update, delete methods of SQLiteDatabase** object to insert, update, delete records.
- The helper class uses **a class that describes the table** (entity)
- Also, helper class implements **CRUD operations**

Files and Content Providers

Objectives:

- Use Files and Directories in Android apps
- Explain **content providers**
- Develop Android **apps that share data** and **create** their own **content providers**

Working with Files and Directories

- Android SDK provides a variety of standard Java file utility classes (such as **java.io**) for handling different types of files, such as **text files**, **binary files**, and **XML files**.
- Android application **files are stored in a standard directory hierarchy** on the Android file system.
- You can browse an application's directory structure using the View/**Tool Window/Device Explorer** Android Studio.

Working with Files and Directories

- Android application data is stored on the Android file system in the following top-level directory:

  /data/data/<package name>/

- Several default subdirectories are created for storing **databases**, **preferences**, and **files** as necessary.
- You can also create other custom directories as needed.
- File operators all begin by interacting with the application Context
- You can use all the standard java.io package utilities to work with FileStream objects.

Working with Files and Directories

Table 10.3 **Important `android.content.Context` File and Directory Management Methods**

Method	Purpose
`Context.openFileInput()`	Opens an application file for reading. These files are located in the `/files` subdirectory.
`Context.openFileOutput()`	Creates or opens an application file for writing. These files are located in the `/files` subdirectory.
`Context.deleteFile()`	Deletes an application file by name. These files must be located in the `/files` subdirectory.
`Context.fileList()`	Gets a list of all files in the `/files` subdirectory.
`Context.getFilesDir()`	Retrieves the application `/files` subdirectory object.
`Context.getCacheDir()`	Retrieves the application `/cache` subdirectory object.
`Context.getDir()`	Creates or retrieves an application subdirectory by name.

Creating and Writing to Files to the Default Application Directory

- Android applications that require only the occasional file rely upon the helpful method called **openFileOutput()**.
- Use this method to create files in the default location under the application data directory:

  /data/data/<package name>/files/

- For example, the following code snippet creates and opens a file called Filename.txt.
  - We write a single line of text to the file and then close the file:

```
FileOutputStream fos;
String strFileContents = “Some text to write to the file.”;
fos = openFileOutput(“Filename.txt”, MODE_PRIVATE);
fos.write(strFileContents.getBytes());
fos.close();
```

Creating and Writing to Files to the Default Application Directory

- We can append data to the file by opening it with the mode set to MODE_APPEND:

```
FileOutputStream fos;
String strFileContents = “More text to write to the file.”;
fos = openFileOutput(“Filename.txt”, MODE_APPEND);
fos.write(strFileContents.getBytes());
fos.close();
```

- The file we created has the following path on the Android file system:

  /data/data/<package name>/files/Filename.txt

Reading from Files in the Default Application Directory

- We have a shortcut for reading files stored in the default /files subdirectory.
- The following code snippet opens a file called Filename.txt for read operations:

```
String strFileName = “Filename.txt”;
FileInputStream fis = openFileInput(strFileName);
```

Reading Raw Files Byte-by-Byte

- You handle file I/O operations using standard Java methods.
- Subclasses of **java.io.InputStream** are used for reading bytes from different types of primitive file types.
- For example, **DataInputStream** is useful for reading one line at a time.
- Here's a simple example of how to read a text file, line by line, and store it in a StringBuffer:

```
FileInputStream fis = openFileInput(filename);
StringBuffer sBuffer = new StringBuffer();
DataInputStream dataIO = new DataInputStream(fis);
String strLine = null;
while ((strLine = dataIO.readLine()) != null) {
sBuffer.append(strLine + "\n");
}
dataIO.close();
fis.close();
```

Working with Other Directories and Files on the Android File System

- To manage your files the Android file system uses the standard **java.io.File** class methods.
- The following code gets a File object for the /files application subdirectory and retrieves a list of all filenames in that directory:

```
File pathForAppFiles = getFilesDir();
String[] fileList = pathForAppFiles.list();
```

Working with Other Directories and Files on the Android File System

- A more generic method to create a file on the file system works anywhere on the Android file system you have permission to access, not the **/files** directory:

```
File fileDir = getFilesDir();
String strNewFileName = “myFile.dat”;
String strFileContents = “Some data for our file”;
File newFile = new File(fileDir, strNewFileName);
newFile.createNewFile();
FileOutputStream fo =
new FileOutputStream(newFile.getAbsolutePath());
fo.write(strFileContents.getBytes());
fo.close();
```

Cache files

- You can create a **cache file** to cache some data to **speed up** your application's performance.
- There is also a **special application directory** for storing cache files.
  - Cache files are stored in the following location on the Android file system:

    /data/data/<package name>/**cache**/
- You should use **getCacheDir()** to open a File that represents the internal directory where your application should save temporary cache files.
- When the device is low on internal storage space, Android may delete these cache files to recover space.

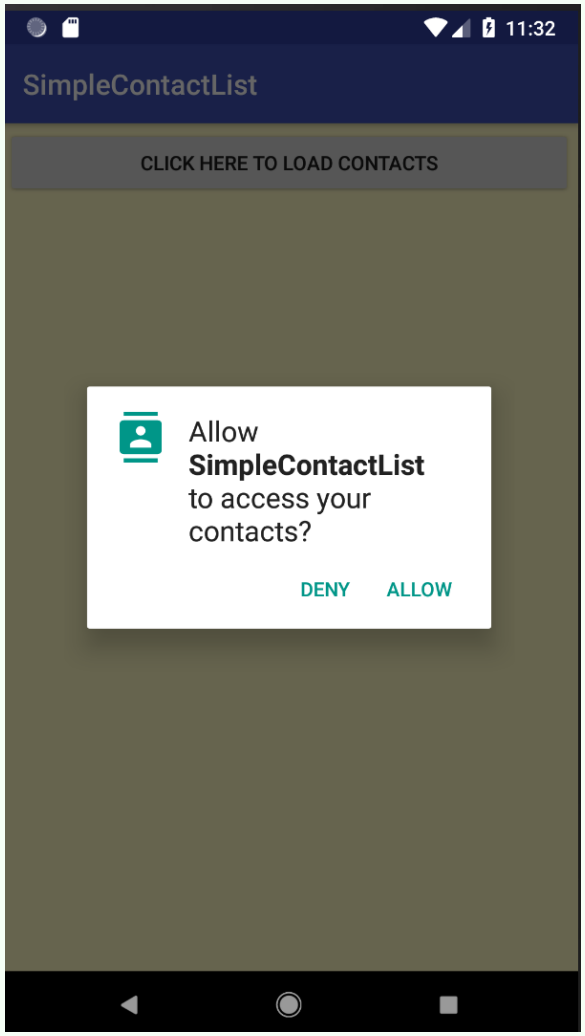
Cache files

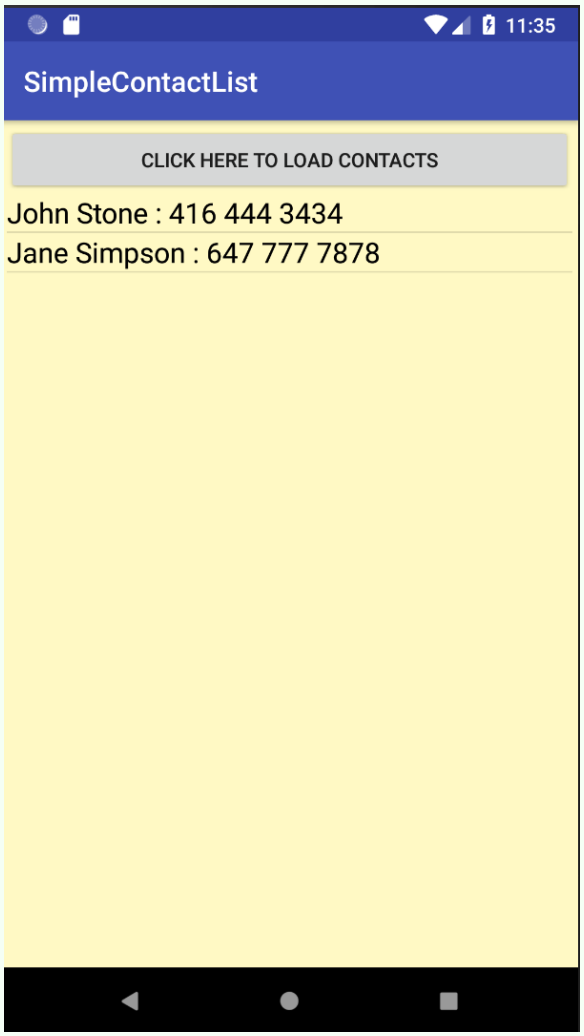
- The following code gets a File object for the /cache application subdirectory, creates a new file in that specific directory, writes some data to the file, closes the file, and then deletes it:

```
File pathCacheDir = getCacheDir();
String strCacheFileName = “myCacheFile.cache”;
String strFileContents = “Some data for our file”;
File newCacheFile = new File(pathCacheDir, strCacheFileName);
newCacheFile.createNewFile();
FileOutputStream foCache =
new FileOutputStream(newCacheFile.getAbsolutePath());
foCache.write(strFileContents.getBytes());
foCache.close();
newCacheFile.delete();
```

Content Providers

- Retrieve list of contacts (SimpleContactList example):



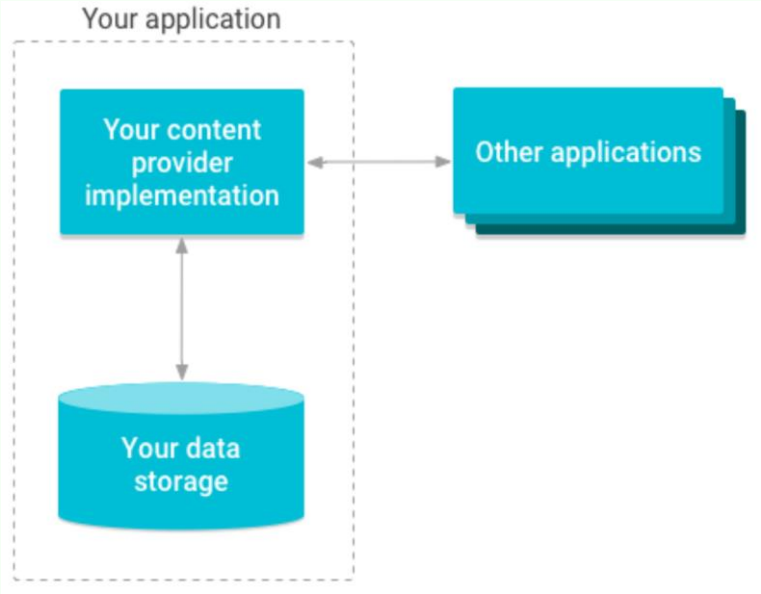


Content Providers

- A content provider **presents data to external applications** as one or more tables that are similar to the tables found in a relational database.
  - External apps **can query** it, **edit** its content, as well as **add** or **delete** content.
- A content provider can **use different ways to store its data**:
  - in a **database**
  - in **files**
  - over a **network**.

Built-in Content Providers

- **Browser** - stores data such as browser bookmarks, browser history, etc.
- **CallLog** - stores data such as missed calls, call details, etc.
- **Contacts** - stores contact details
- **MediaStore** - stores media files such as audio, video, and images
- **Settings** - stores the device's settings and preferences



Built-in Content Providers

- To query a content provider, you specify the **query string** in the form of a **URI**, with an optional specifier for a particular row.
- The format of the query URI is as follows:

  <standard_prefix>://<authority>/<data_path>/<id>
- The various parts of the URI are as follows:
  - The **standard prefix** for content providers is always **content://**.
  - The **authority** specifies the **name of the content provider**.
    - An example would be **contacts** for the built-in Contacts content provider.

Built-in Content Providers

- For third-party content providers, this could be the **fully qualified name**, such as *com.wrox.provider* or *net.learn2develop.provider*.
- The **data path** specifies the kind of data requested.
  - For example, if you are getting all the contacts from the **Contacts** content provider, then the data path would be **people**, and the URI would look like this: **content://contacts/people**.
- The **id** specifies the specific record requested.
- For example, if you are looking for contact number 2 in the Contacts content provider, the URI would look like this:

content://contacts/people/2

Built-in Content Providers

TABLE 7-1: Example Query Strings

QUERY STRING	DESCRIPTION
`content://media/internal/images`	Returns a list of all the internal images on the device
`content://media/external/images`	Returns a list of all the images stored on the external storage (e.g., SD card) on the device
`content://call_log/calls`	Returns a list of all calls registered in the Call Log
`content://browser/bookmarks`	Returns a list of bookmarks stored in the browser

Create custom Content Providers

1. Chose **how to store data** (in a SQLite database)
2. Create **a subclass** of the abstract **ContentProvider** class and override the its methods.
3. Use an **UriMatcher** object **to parse the content URI** that is passed to the content provider through a **ContentResolver**.
4. Create **a subclass of the SQLiteOpenHelper** helper class to help manage your database.
5. **Register your content provider** with Android, modify the AndroidManifest.xml file by adding the <**provider**> element.
6. Specify **permissions** that other applications must have in order to access the provider's data..

Create custom Content Providers

- Example: create a simple content provider that stores a list of books in an SQLite database.
  - The **content provider stores the books in a database table containing three fields**, as shown in Figure 7-3.

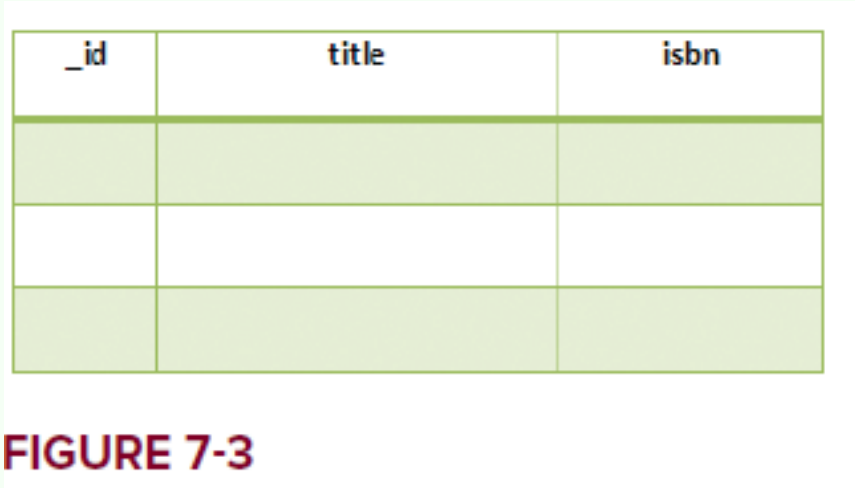


FIGURE 7-3

Create custom Content Providers

- First, create a class named *BooksProvider* that **extends the ContentProvider**
  - Override the following methods:
  - **getType()** - Returns the MIME type of the data at the given URI
  - **onCreate()** - Called when the provider is started
  - **query()** - Receives a request from a client. The result is returned as a Cursor object.
  - **insert()** - Inserts a new record into the content provider
  - **delete()** - Deletes an existing record from the content provider
  - **update()** - Updates an existing record from the content provider

ContentProviders example

- Use an **UriMatcher** object **to parse the content URI** that is passed to the content provider through a **ContentResolver**.
- For example, the following content URI represents a request for all books in the content provider:

  content://net.learn2develop.provider.Books/**books**
- The following represents a request for a particular book with _id 5:

  content://net.learn2develop.provider.Books/**books/5**

ContentProviders example

- Use the SQLiteOpenHelper helper class to help manage your database:

```
private static class DatabaseHelper extends SQLiteOpenHelper
{
        DatabaseHelper(Context context) {
                super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }
        @Override
        public void onCreate(SQLiteDatabase db)
        {
                db.execSQL(DATABASE_CREATE);
        }
        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
        {
                Log.w("Content provider database",
                "Upgrading database from version " +
                oldVersion + " to " + newVersion +
                ", which will destroy all old data");
                db.execSQL("DROP TABLE IF EXISTS titles");
                onCreate(db);
        }
}
```

Overriding ContentProvider methods

```
@Override
public String getType(Uri uri)
{
    switch (uriMatcher.match(uri))
    {
        //---get all books---
        case BOOKS:
            return “vnd.android.cursor.dir/vnd.learn2develop.books “;
        //---get a particular book---
        case BOOK_ID:
            return “vnd.android.cursor.item/vnd.learn2develop.books “;
        default:
            throw new IllegalArgumentException(“Unsupported URI: “ +
            uri);
    }
}
```

ContentProviders example

```
@Override
public Cursor query(Uri uri, String[] projection, String selection,
        String[] selectionArgs, String sortOrder)
{
        SQLiteQueryBuilder sqlBuilder = new SQLiteQueryBuilder();
        sqlBuilder.setTables(DATABASE_TABLE);
        if (uriMatcher.match(uri) == BOOK_ID)
        //---if getting a particular book---
        sqlBuilder.appendWhere(
        _ID + " = " + uri.getPathSegments().get(1));
        if (sortOrder==null || sortOrder=="")
        sortOrder = TITLE;
```

ContentProviders example

```
    Cursor c = sqlBuilder.query(
    booksDB,
    projection,
    selection,
    selectionArgs,
    null,
    null,
    sortOrder);
    //---register to watch a content URI for changes---
    c.setNotificationUri(getContext().getContentResolver(), uri);
    return c;
}
```

ContentProviders example

```
@Override
public Uri insert(Uri uri, ContentValues values)
{
      //---add a new book---
      long rowID = booksDB.insert( DATABASE_TABLE, "", values);
      //---if added successfully---
      if (rowID>0)
      {
      Uri  _uri = ContentUris.withAppendedId(CONTENT_URI, rowID);
      getContext().getContentResolver().notifyChange(_uri, null);
      return _uri;
     }
      throw new SQLException("Failed to insert row into " + uri);
}
```

ContentProviders example

```
@Override
public int delete(Uri arg0, String arg1, String[] arg2)
{
        // arg0 = uri
        // arg1 = selection
        // arg2 = selectionArgs
        int count=0;
        switch (uriMatcher.match(arg0)){
                case BOOKS:
                count = booksDB.delete(
                DATABASE_TABLE,
                arg1,
                arg2);
                break;
```

ContentProviders example

```
    case BOOK_ID:
        String id = arg0.getPathSegments().get(1);
        count = booksDB.delete(
        DATABASE_TABLE,
        _ID + " = " + id +
        (!TextUtils.isEmpty(arg1) ? " AND (" +
        arg1 + ')' : ""),
        arg2);
        break;
    default: throw new IllegalArgumentException("Unknown URI " + arg0);
  }
    getContext().getContentResolver().notifyChange(arg0, null);
    return count;
}
```

ContentProviders example

```
@Override
public int update(Uri uri, ContentValues values, String selection,
        String[] selectionArgs) {
        int count = 0;
        switch (uriMatcher.match(uri)){
            case BOOKS:
              count = booksDB.update(
               DATABASE_TABLE,
               values,
               selection,
               selectionArgs);
        break;
```

ContentProviders example

```
        case BOOK_ID:
            count = booksDB.update(
            DATABASE_TABLE,
            values,
            _ID + " = " + uri.getPathSegments().get(1) +
            (!TextUtils.isEmpty(selection) ? " AND (" +
            selection + ')' : ""),
            selectionArgs);
            break;
        default: throw new IllegalArgumentException("Unknown URI " + uri);
}
        getContext().getContentResolver().notifyChange(uri, null);
        return count;
}
```

ContentProviders example

- Finally, to **register your content provider** with Android, modify the AndroidManifest.xml file by adding the <**provider**> element:

```
<provider android:name=”BooksProvider”
android:authorities=”net.learn2develop.provider.Books”>
</provider>
```

Using the Content Provider - insert

- Test your new content provider from within your Android application:
- To **add a book** to the content provider, you create a new ContentValues object and then populate it with the various information about a book:

```
//---add a book---
ContentValues values = new ContentValues();
values.put(BooksProvider.TITLE, ((EditText)
findViewById(R.id.txtTitle)).getText().toString());
values.put(BooksProvider.ISBN, ((EditText)
findViewById(R.id.txtISBN)).getText().toString());
Uri uri = getContentResolver().insert(
BooksProvider.CONTENT_URI, values);
```

Using the Content Provider - insert

- To access this content provider from another package, you need to specify the field name directly, like this:

```
ContentValues values = new ContentValues();
values.put("title", ((EditText)
findViewById(R.id.txtTitle)).getText().toString());
values.put("isbn", ((EditText)
findViewById(R.id.txtISBN)).getText().toString());
Uri uri = getContentResolver().insert(
Uri.parse(
"content://net.learn2develop.provider.Books/books"),
values);
```

Using the Content Provider - insert

- Note that for external packages, you need to refer to the content URI using the fully qualified content URI:

```
Uri uri = getContentResolver().insert(
Uri.parse(
"content://net.learn2develop.provider.Books/books"),
values);
```

Using the Content Provider - retrieve

```
//---retrieve the titles---
Uri allTitles =
    Uri.parse("content://net.learn2develop.provider.Books/books");
Cursor c;
if (android.os.Build.VERSION.SDK_INT <11) {
      //---before Honeycomb---
      c = managedQuery(allTitles, null, null, null,"title desc");
}
else
{
      //---Honeycomb and later---
      CursorLoader cursorLoader = new CursorLoader(
      this, allTitles, null, null, null,"title desc");
      c = cursorLoader.loadInBackground();
}
```

Using the Content Provider - retrieve

```
if (c.moveToFirst()) {
do
{
    Toast.makeText(this,
    c.getString(c.getColumnIndex(
    BooksProvider._ID)) + “, “ +
    c.getString(c.getColumnIndex(
    BooksProvider.TITLE)) + “, “ +
    c.getString(c.getColumnIndex(
    BooksProvider.ISBN)),
    Toast.LENGTH_SHORT).show();
} while (c.moveToNext());
}
```

Using the Content Provider - update

- If you want to update a book's detail, call the **update()** method with the content URI, indicating the book's ID:

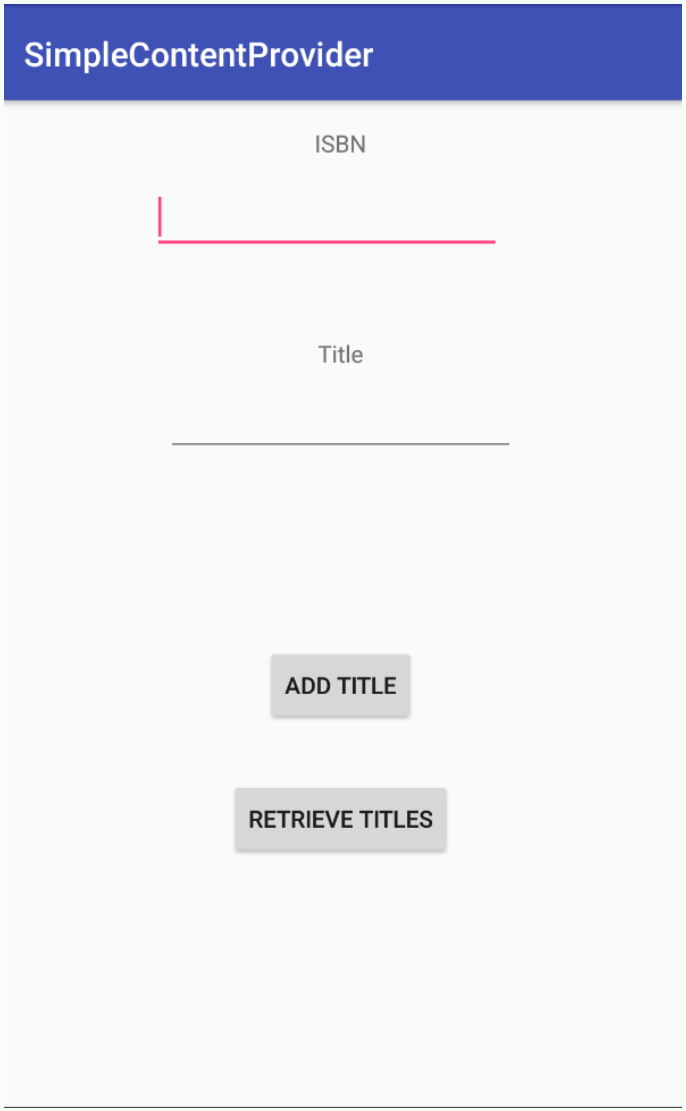
```
ContentValues editedValues = new ContentValues();
editedValues.put(BooksProvider.TITLE, "Android Tips and Tricks");
getContentResolver().update(
Uri.parse(
"content://net.learn2develop.provider.Books/books/2"),
editedValues,
null,
null);
```

Using the Content Provider - delete

- To delete a book, use the delete() method with the content URI, indicating the book's ID:

```
//---delete a title---
getContentResolver().delete(
Uri.parse("content://net.learn2develop.provider.Books/books/2"),
null, null);
```

Content Provider Example



References

- Textbook
- Android Documentation: https://developer.android.com/guide/topics/providers/content-providers.html