SharedPreferences

Using the SharedPreferences class you can create named maps of key/value pairs within your application that can be shared among application components running in the same application context.

Creating new Shared Preferences

```java
SharedPreferences myPref = getSharedPreferences("myPrefName", Activity.MODE_PRIVATE);
SharedPreferences.Editor edit = myPref.edit();
edit.putInt("myInt", 50);
edit.putString("myString", "word");
edit.commit();
```

Retrieving Shared Preferences

```java
SharedPreferences myPref = getSharedPreferences("myPrefName", Activity.MODE_PRIVATE);

int x = myPref.getInt("myInt", 0);
String y = myPref.getString("myString", "");
```

This is available across the whole application, so we can put values in some activity and retrieve it from another one.

Saving And Loading Files && Including Static Files As Resources page 207 can be useful but it's not recommended to use files to save state.
SQLite Database

Explore both attached DB.java and UsingDatabase.java classes

Cursors

Queries in Android are returned as Cursor objects. Rather than extracting and returning a copy of the result values, Cursors are pointers to the result set within the underlying data. Cursors provide a managed way of controlling your position (row) in the result set of a database query. The Cursor class includes a number of navigation functions including, but not limited to, the following:

- `moveToFirst` Moves the cursor to the first row in the query result
- `moveToNext` Moves the cursor to the next row
- `moveToPrevious` Moves the cursor to the previous row
- `getCount` Returns the number of rows in the result set
- `getColumnIndexOrThrow` Returns the index for the column with the specified name (throwing an exception if no column exists with that name)
- `getColumnName` Returns the name of the specified column index
- `getColumnNames` Returns a string array of all the column names in the current Cursor
- `moveToPosition` Moves the Cursor to the specified row
- `getPosition` Returns the current Cursor position

Lab Work 1

- Suggest a scenario for using shared preferences and use it.
Content Providers

While using the database approach is the recommended way to save structured and complex data, sharing data is a challenge because the database is accessible to only the package that created it.

In Android, using a content provider is the recommended way to share data across packages.

A content provider behaves very much like a database — you can query it, edit its content, as well as add or delete its content. However, unlike a database, a content provider can use different ways to store its data. The data can be stored in a database, in files, or even over a network.

Android ships with many useful content providers, including the following:

- **Browser** — Stores data such as browser bookmarks, browser history, and so on
- **CallLog** — Stores data such as missed calls, call details, and so on
- **ContactsContract** — Stores contact details
- **MediaStore** — Stores media files such as audio, video and images
- **Settings** — Stores the device’s settings and preferences

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:

```
content://<authority>/<data_path>/<id>
```

- The **authority** specifies the name of the content provider. An example would be contacts for the built-in Contacts content provider. 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.

<table>
<thead>
<tr>
<th>QUERY STRING</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>content://media/internal/images</td>
<td>Returns a list of all the Internal Images on the device</td>
</tr>
<tr>
<td>content://media/external/images</td>
<td>Returns a list of all the images stored on the external storage (e.g., SD card) on the device</td>
</tr>
<tr>
<td>content://call_log/calls</td>
<td>Returns a list of all calls registered in the Call Log</td>
</tr>
<tr>
<td>content://browser/bookmarks</td>
<td>Returns a list of bookmarks stored in the browser</td>
</tr>
</tbody>
</table>

Example:

**Get all audio files on external storage**

```java
Cursor c = managedQuery(MediaStore.Audio.Media.EXTERNAL_CONTENT_URI, null, null, null, null);

String[] columns = c.getColumnNames();

for (int i = 0; i < columns.length; i++) {
    Log.d("Columns", columns[i]);
}

**This could be useful to explore new providers!**
```

```java
Cursor c = managedQuery(ContactsContract.Data.CONTENT_URI, null, null, null, null);

String[] columns = c.getColumnNames();

for (int i = 0; i < columns.length; i++) {
    Log.d("Columns", columns[i]);
}

Log.d("location", c.getString(c.getColumnIndex("data4")));
Log.d("name", c.getString(c.getColumnIndex("display_name")));```
Cursor `c` = managedQuery(*Uri*, `null`, `null`, `null`, `null`);

Uri examples are:

➤➤ ContactsContract.Contacts.CONTENT_URI
➤➤ ContactsContract.Data.CONTENT_URI
➤➤ Browser.BOOKMARKS_URI
➤➤ Browser.SEARCHES_URI
➤➤ CallLog.CONTENT_URI
➤➤ MediaStore.Images.Media.INTERNAL_CONTENT_URI
➤➤ MediaStore.Images.Media.EXTERNAL_CONTENT_URI
➤➤ Settings.CONTENT_URI

The ContactsContract.Data Content Provider is used to store all the contact details — such as addresses, phone numbers, and e-mail addresses — making it the best approach when searching for one of these details.