

DATA MANAGEMENT

TODAY'S TOPICS

SHARED PREFERENCES

INTERNAL STORAGE

EXTERNAL STORAGE

SQLITE DATABASES

SHARED PREFERENCES

SMALL AMOUNTS OF PRIMITIVE DATA

INTERNAL STORAGE

SMALL TO MEDIUM AMOUNTS OF PRIVATE
DATA

EXTERNAL STORAGE

LARGER AMOUNTS OF NON-PRIVATE DATA

DATABASES

STORE SMALL TO LARGE AMOUNTS OF
PRIVATE, STRUCTURED DATA

SHARED PREFERENCES

A PERSISTENT MAP

HOLDS KEY-VALUE PAIRS OF SIMPLE DATA
TYPES

AUTOMATICALLY PERSISTED ACROSS
APPLICATION SESSIONS

SHARED PREFERENCES

OFTEN USED FOR LONG-TERM STORAGE OF
CUSTOMIZABLE APPLICATION DATA

ACCOUNT NAME

FAVORITE WIFI NETWORKS

USER CUSTOMIZATIONS

ACTIVITY SHARED PREFERENCES

TO GET A SharedPreferences OBJECT
ASSOCIATED WITH A GIVEN ACTIVITY

Activity.getSharedPreferences (int mode)

MODE_PRIVATE

NAMED SHARED PREFERENCES

```
Context.getSharedPreferences (  
    String name, int mode)
```

NAME – NAME OF SharedPreferences FILE

MODE – MODE_PRIVATE

WRITING SHARED PREFERENCES

CALL `SharedPreferences.edit()`

RETURNS A `SharedPreferences.Editor`
INSTANCE

WRITING SHARED PREFERENCES

ADD VALUES TO SharedPreferences USING
SharedPreferences.Editor instance

`putInt(String key, int value)`

`putString(String key, String value)`

`remove(String key)`

WRITING SHARED PREFERENCES

COMMIT EDITED VALUES WITH
`SharedPreferences.Editor.commit()`

READING SHARED PREFERENCES

USE SharedPreferences METHODS TO READ
VALUES

`getAll()`

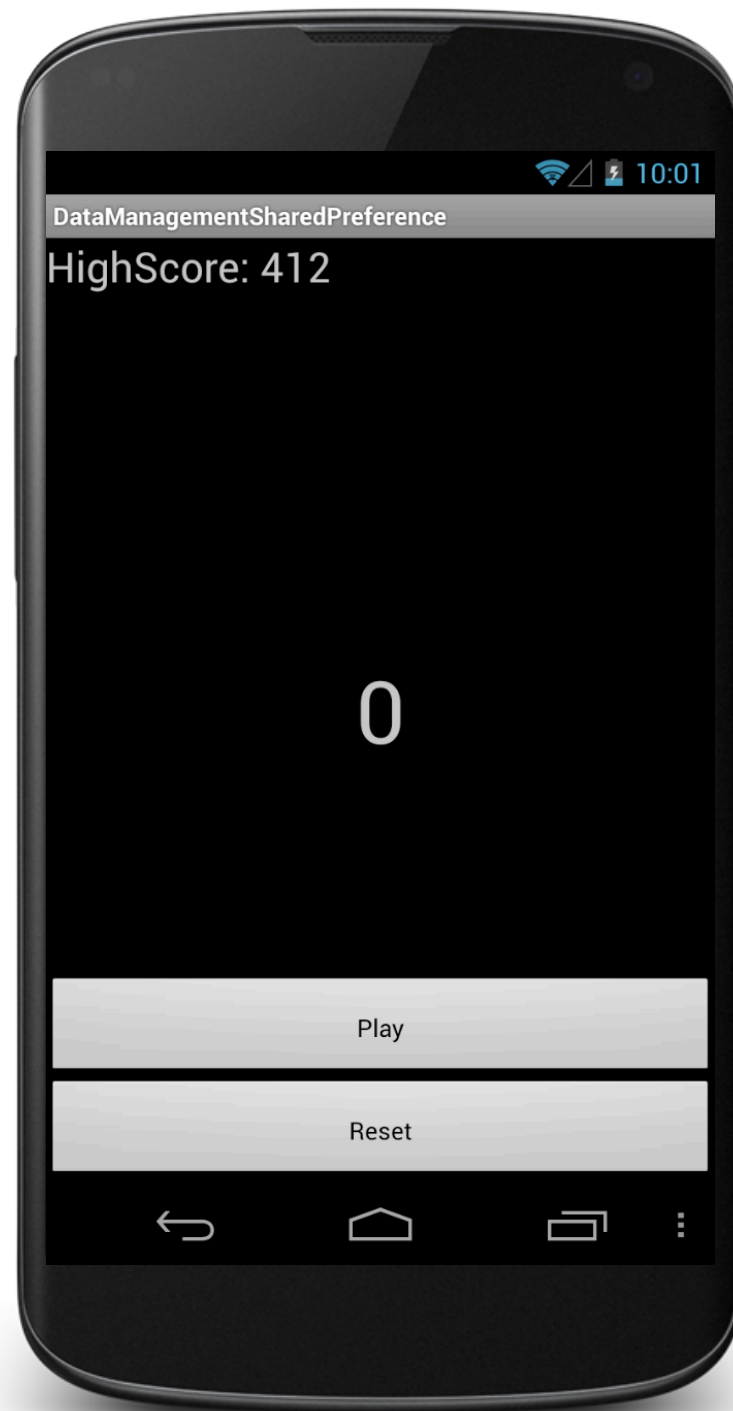
`getBoolean(String key, ...)`

`getString(String key, ...)`

DATAMANAGEMENTSHAREDREFERENCES

WHEN THE USER PRESSES THE PLAY BUTTON,
THE APPLICATION DISPLAYS A RANDOM
NUMBER

THE APPLICATION KEEPS TRACK OF THE
HIGHEST NUMBER SEEN SO FAR



DATAMANAGEMENTSHAREDREFERENCES

```
final SharedPreferences prefs = getPreferences(MODE_PRIVATE);

setContentView(R.layout.main);

// High Score
final TextView highScore = (TextView) findViewById(R.id.high_score_text);
highScore.setText(String.valueOf(prefs.getInt(HIGH_SCORE, 0)));

//Game Score
final TextView gameScore = (TextView) findViewById(R.id.game_score_text);
gameScore.setText(String.valueOf("0"));
```

DATAMANAGEMENTSHAREDREFERENCES

```
// Play Button
final Button playButton = (Button) findViewById(R.id.play_button);
playButton.setOnClickListener(new OnClickListener() {

    @Override
    public void onClick(View v) {

        Random r = new Random();
        int val = r.nextInt(1000);
        gameScore.setText(String.valueOf(val));

        // Get Stored High Score
        if (val > prefs.getInt(HIGH_SCORE, 0)) {

            // Get and edit high score
            SharedPreferences.Editor editor = prefs.edit();
            editor.putInt(HIGH_SCORE, val);
            editor.commit();

            highScore.setText(String.valueOf(val));

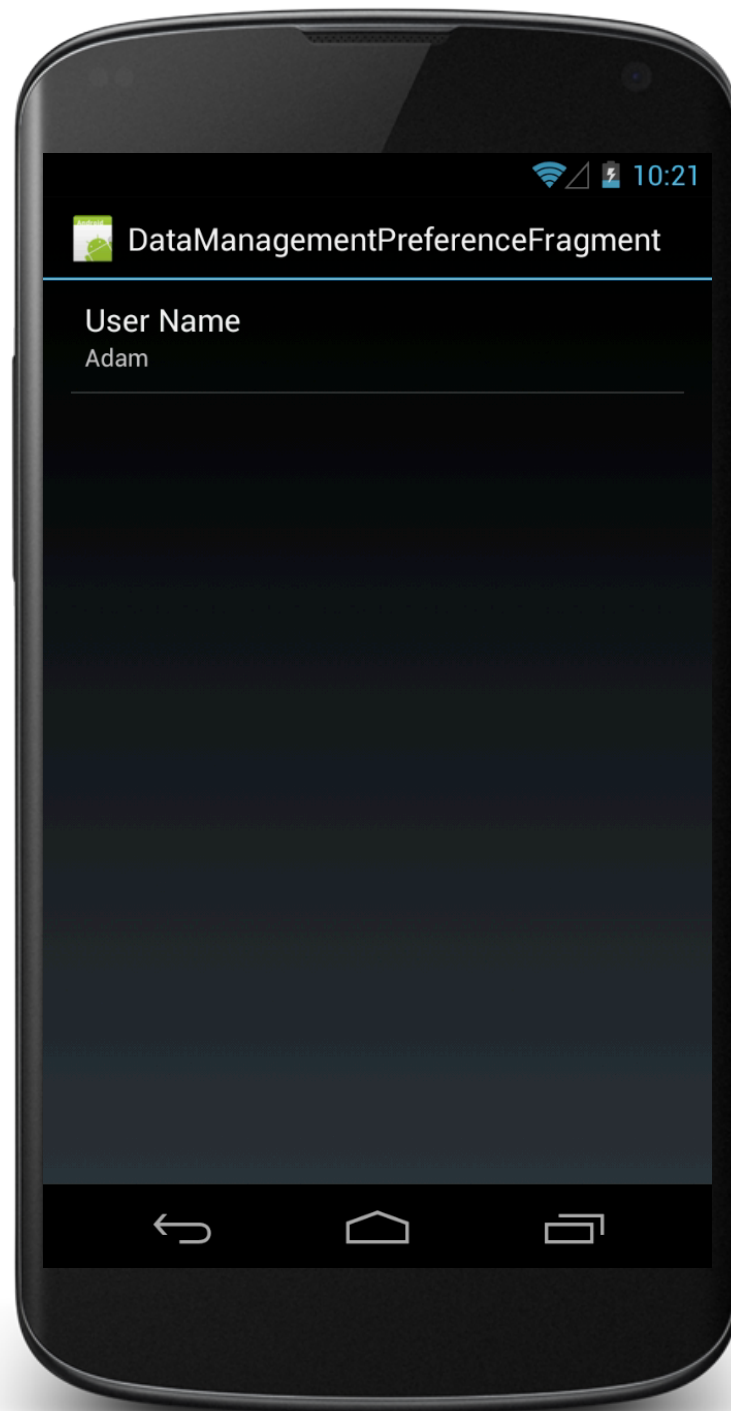
        }
    }
});
```

PREFERENCEFRAGMENT

A CLASS THAT SUPPORTS DISPLAYING &
MODIFYING USER PREFERENCES

DATAMANAGEMENT PREFERENCEFRAGMENT

THIS APPLICATION DISPLAYS A
PREFERENCEFRAGMENT, WHICH ALLOWS THE
USER TO ENTER AND CHANGE A PERSISTENT
USER NAME



DATAMANAGEMENTPREFERENCEFRAGMENT

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Load the preferences from an XML resource
    addPreferencesFromResource(R.xml.user_prefs);

    // Get the username Preference
    mUserNamePreference = (Preference) getPreferenceManager()
        .findPreference(USERNAME);

    // Attach a listener to update summary when username changes
    mListener = new OnSharedPreferenceChangeListener() {
        @Override
        public void onSharedPreferenceChanged(
            SharedPreferences sharedPreferences, String key) {
            mUserNamePreference.setSummary(sharedPreferences.getString(
                USERNAME, "None Set"));
        }
    };
};
```

DATAMANAGEMENTPREFERENCEFRAGMENT

```
// Get SharedPreferences object managed by the PreferenceManager for
// this Fragment
SharedPreferences prefs = getPreferenceManager()
    .getSharedPreferences();

// Register a listener on the SharedPreferences object
prefs.registerOnSharedPreferenceChangeListener(mListener);

// Invoke callback manually to display the current username
mListener.onSharedPreferenceChanged(prefs, USERNAME);

}
```

FILE

CLASS REPRESENTS A FILE SYSTEM ENTITY
IDENTIFIED BY A PATHNAME

FILE

STORAGE AREAS ARE CLASSIFIED AS
INTERNAL OR EXTERNAL

INTERNAL MEMORY USUALLY USED FOR
SMALLER, APPLICATION PRIVATE DATA SETS

EXTERNAL MEMORY USUALLY USED FOR
LARGER, NON-PRIVATE DATA SETS

FILE API

FILEOUTPUTSTREAM

OPENFILEOUTPUT (STRING NAME, INT MODE)

OPEN PRIVATE FILE FOR WRITING. CREATES THE
FILE IF IT DOESN'T ALREADY EXIST

FILEINPUTSTREAM

OPENFILEINPUT (STRING NAME)

OPEN PRIVATE FILE FOR READING

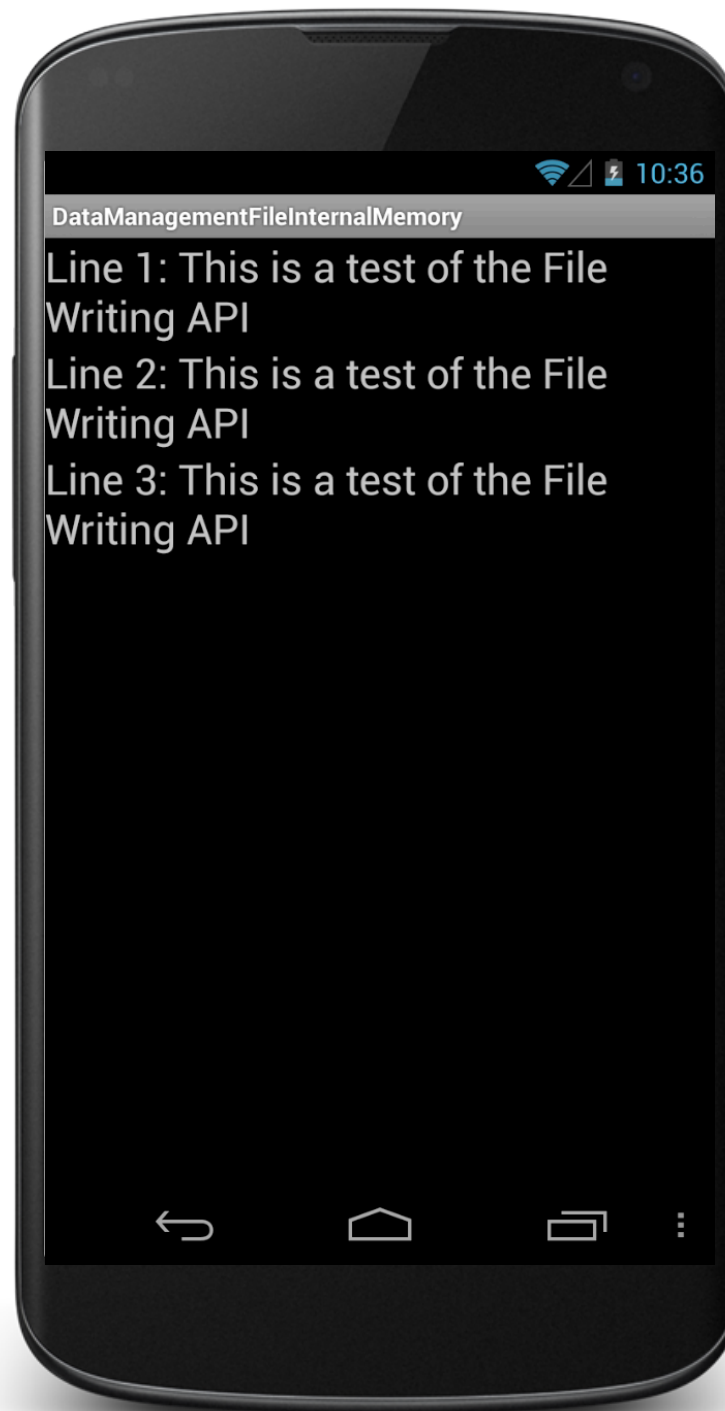
MANY OTHERS. SEE DOCUMENTATION.

DATAMANAGEMENT

FILEINTERNALMEMORY

IF A TEXT FILE DOES NOT ALREADY EXIST,
APPLICATION WRITES TEXT TO THAT TEXT FILE

APPLICATION THEN READS DATA FROM THE
TEXT FILE AND DISPLAYS IT



DATAMANAGEMENTFILEINTERNALMEMO RY

```
private void writeFile() throws FileNotFoundException {  
  
    FileOutputStream fos = openFileOutput(fileName, MODE_PRIVATE);  
  
    PrintWriter pw = new PrintWriter(new BufferedWriter(  
        new OutputStreamWriter(fos)));  
  
    pw.println("Line 1: This is a test of the File Writing API");  
    pw.println("Line 2: This is a test of the File Writing API");  
    pw.println("Line 3: This is a test of the File Writing API");  
  
    pw.close();  
  
}
```

DATAMANAGEMENTFILEINTERNALMEMO RY

```
private void readFile(LinearLayout ll) throws IOException {  
  
    FileInputStream fis = openFileInput(fileName);  
    BufferedReader br = new BufferedReader(new InputStreamReader(fis));  
  
    String line = "";  
  
    while (null != (line = br.readLine())) {  
  
        TextView tv = new TextView(this);  
        tv.setTextSize(24);  
        tv.setText(line);  
  
        ll.addView(tv);  
  
    }  
  
    br.close();  
}
```

USING EXTERNAL MEMORY FILES

REMOVABLE MEDIA MAY APPEAR/DISAPPEAR
WITHOUT WARNING

USING EXTERNAL MEMORY FILES

String Environment.

`getExternalStorageState()`

`MEDIA_MOUNTED` – PRESENT & MOUNTED
WITH READ/WRITE ACCESS

`MEDIA_MOUNTED_READ_ONLY` – PRESENT
& MOUNTED WITH READ-ONLY ACCESS

`MEDIA_REMOVED` – NOT PRESENT

ETC.

USING EXTERNAL MEMORY FILES

PERMISSION TO WRITE EXTERNAL FILES

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

DATAMANAGEMENT

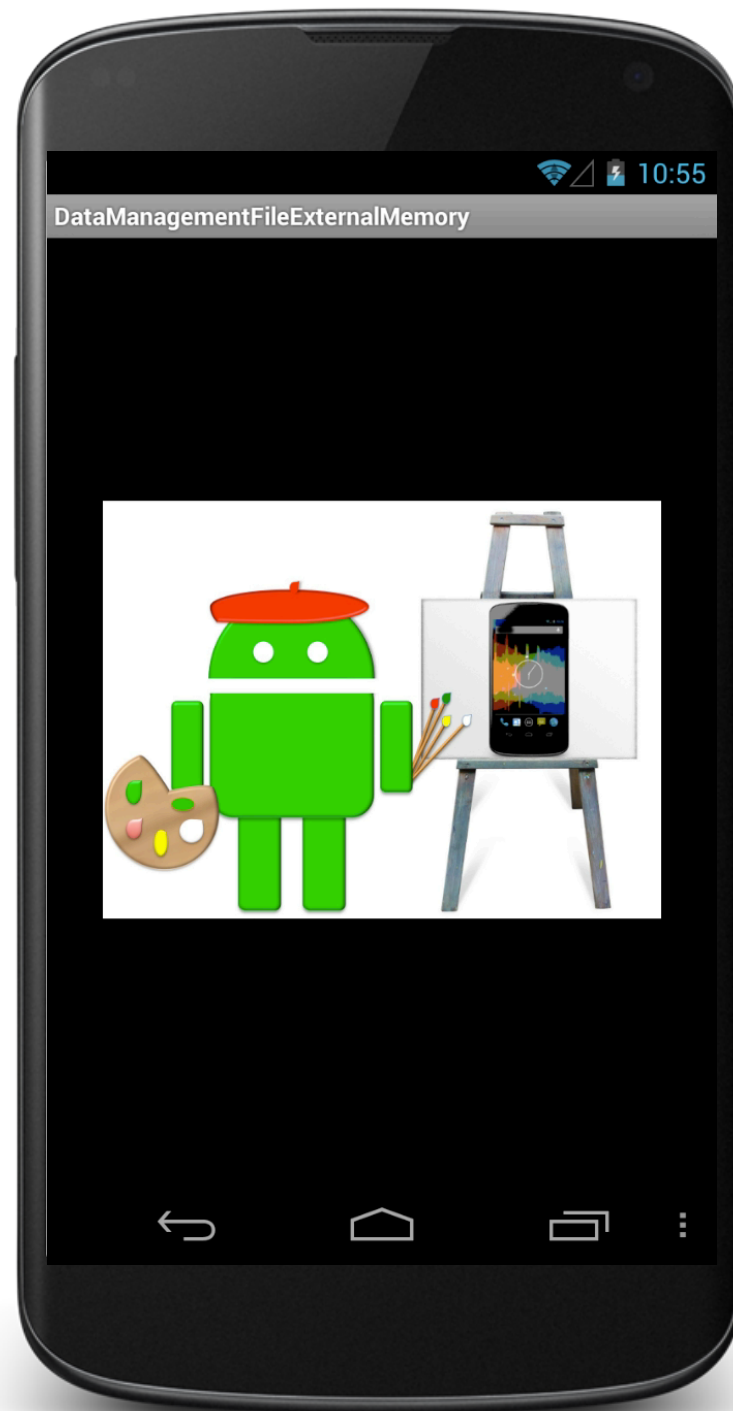
FILEEXTERNALMEMORY

APPLICATION READS AN IMAGE FILE FROM
THE RESOURCES DIRECTORY

COPIES THAT FILE TO EXTERNAL STORAGE

READS IMAGE DATA FROM THE FILE IN
EXTERNAL STORAGE

THEN DISPLAYS THE IMAGE



DATAMANAGEMENTFILEEXTERNALMEMORY

```
private void copyImageToMemory(File outFile) {  
    try {  
  
        BufferedOutputStream os = new BufferedOutputStream(  
            new FileOutputStream(outFile));  
  
        BufferedInputStream is = new BufferedInputStream(getResources()  
            .openRawResource(R.raw.painter));  
  
        copy(is, os);  
  
    } catch (FileNotFoundException e) {  
        Log.e(TAG, "FileNotFoundException");  
    }  
}
```

DATAMANAGEMENTFILEEXTERNALMEMORY

```
private void copy(InputStream is, OutputStream os) {  
    final byte[] buf = new byte[1024];  
    int numBytes;  
    try {  
        while (-1 != (numBytes = is.read(buf))) {  
            os.write(buf, 0, numBytes);  
        }  
    } catch (IOException e) {  
        e.printStackTrace();  
    } finally {  
        try {  
            is.close();  
            os.close();  
        } catch (IOException e) {  
            Log.e(TAG, "IOException");  
        }  
    }  
}
```

CACHE FILES

TEMPORARY FILES THAT MAY BE DELETED BY
THE SYSTEM WHEN STORAGE IS LOW

FILES REMOVED WHEN APPLICATION
UNINSTALLED

CACHE FILES

File Context.getCacheDir()

RETURNS ABSOLUTE PATH TO AN
APPLICATION-SPECIFIC DIRECTORY THAT CAN
BE USED FOR TEMPORARY FILES

SAVING CACHE FILES

`Context.getExternalCacheDir()`

RETURNS A FILE REPRESENTING EXTERNAL
STORAGE DIRECTORY FOR CACHE FILES

SQLITE

SQLITE PROVIDES IN-MEMORY DATABASE

DESIGNED TO OPERATE WITHIN A VERY SMALL
FOOTPRINT (<300kB)

IMPLEMENTS MOST OF SQL92

SUPPORTS ACID TRANSACTIONS

ATOMIC, CONSISTENT, ISOLATED & DURABLE

USING A DATABASE

RECOMMENDED METHOD RELIES ON A HELPER
CLASS CALLED SQLiteOpenHelper

USING A DATABASE

SUBCLASS SQLiteOpenHelper

CALL SUPER() FROM SUBCLASS

CONSTRUCTOR TO INITIALIZE UNDERLYING
DATABASE

USING A DATABASE

OVERRIDE onCreate()

OVERRIDE onUpgrade()

EXECUTE CREATE TABLE COMMANDS

USING A DATABASE

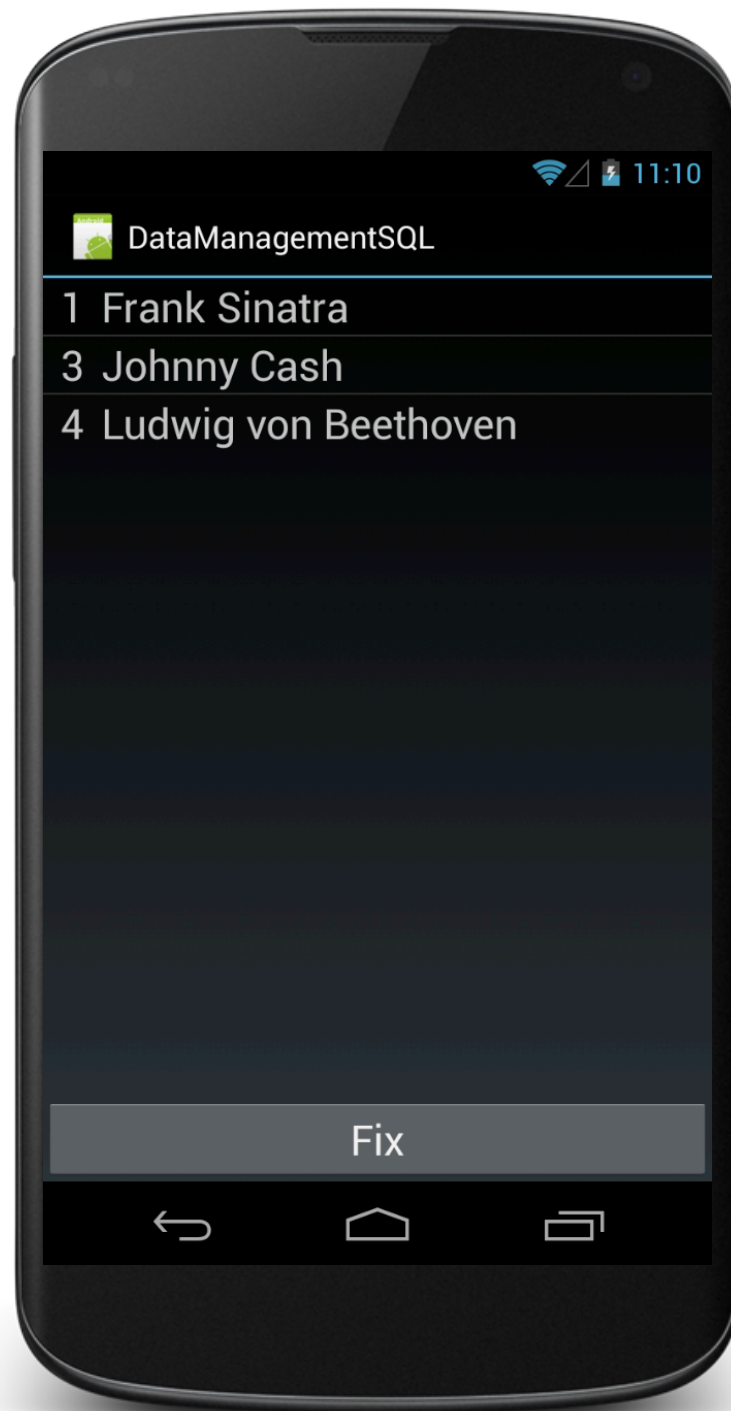
USE SQLITEOPENHELPER METHODS TO
OPEN & RETURN UNDERLYING DATABASE

EXECUTE OPERATIONS ON UNDERLYING
DATABASE

DATAMANAGEMENTSQL

APPLICATION CREATES AN SQLITE DATABASE AND INSERTS RECORDS, SOME WITH ERRORS, INTO IT

WHEN USER PRESSES THE FIX BUTTON, THE APPLICATION DELETES, UPDATES AND REDISPLAYS THE CORRECTED DATABASE RECORDS



DATAMANAGEMENTSQL

```
// Insert several artist records
private void insertArtists() {

    ContentValues values = new ContentValues();

    values.put(DatabaseOpenHelper.ARTIST_NAME, "Frank Sinatra");
    mDbHelper.getWritableDatabase().insert(DatabaseOpenHelper.TABLE_NAME, null, values);

    values.clear();

    values.put(DatabaseOpenHelper.ARTIST_NAME, "Lady Gaga");
    mDbHelper.getWritableDatabase().insert(DatabaseOpenHelper.TABLE_NAME, null, values);

    values.clear();

    values.put(DatabaseOpenHelper.ARTIST_NAME, "Jawny Cash");
    mDbHelper.getWritableDatabase().insert(DatabaseOpenHelper.TABLE_NAME, null, values);

    values.clear();

    values.put(DatabaseOpenHelper.ARTIST_NAME, "Ludwig von Beethoven");
    mDbHelper.getWritableDatabase().insert(DatabaseOpenHelper.TABLE_NAME, null, values);
}
```


DATAMANAGEMENTSQL

```
// Returns all artist records in the database
private Cursor readArtists() {
    return mDbHelper.getWritableDatabase().query(DatabaseOpenHelper.TABLE_NAME,
        DatabaseOpenHelper.columns, null, new String[] {}, null, null,
        null);
}
```

DATAMANAGEMENTSQL

```
// Modify the contents of the database
private void fix() {

    // Sorry Lady Gaga :-(
    mDbHelper.getWritableDatabase().delete(DatabaseOpenHelper.TABLE_NAME,
        DatabaseOpenHelper.ARTIST_NAME + "=?",
        new String[] { "Lady Gaga" });

    // fix the Man in Black
    ContentValues values = new ContentValues();
    values.put(DatabaseOpenHelper.ARTIST_NAME, "Johnny Cash");

    mDbHelper.getWritableDatabase().update(DatabaseOpenHelper.TABLE_NAME, values,
        DatabaseOpenHelper.ARTIST_NAME + "=?",
        new String[] { "Jawny Cash" });

}
```

DATAMANAGEMENTSQL

```
// Delete all records
private void clearAll() {

    mDbHelper.getWritableDatabase().delete(DatabaseOpenHelper.TABLE_NAME, null, null);

}

// Close database
@Override
protected void onDestroy() {

    mDbHelper.getWritableDatabase().close();
    mDbHelper.deleteDatabase();

    super.onDestroy();

}
```

EXAMINING THE DATABASE REMOTELY

DATABASES STORED IN

/data/data/<package name>/databases/

CAN EXAMINE DATABASE WITH SQLITE3

```
# adb -s emulator-5554 shell
```

```
# sqlite3 /data/data/  
course.examples.DataManagement.Data  
BaseExample/databases/artist_dbd
```

NEXT TIME

CONTENTPROVIDER