Mobile Computing

Data Storage Options in Android

Data Storage Options in Android

- Saving key-value pairs of simple data types in a shared preferences file
  - Shared preferences file for storing small amounts of information in key-value pairs.
- Saving arbitrary files in Android's file system
  - Save a basic file, such as to store long sequences of data that are generally read in order.
- Using databases managed by SQLite
  - SQLite database to read and write structured data.

Saving Key-Value Sets

- **getSharedPreferences()** : For multiple shared preference files identified by name specifies with the first parameter. This can be called from any Context in your app.
- **getPreferences()** : For only one shared preference file for the activity. Don't need to supply a name.

getSharedPreferences()

```
Context context = getActivity();
SharedPreferences sharedPref = context.getSharedPreferences(
        getString(R.string.preference_file_key), Context.MODE_PRIVATE);
```

getPreferences()

```
SharedPreferences sharedPref = getActivity().getPreferences(Context.MODE_PRIVATE);
```

Read and Write to Shared Preferences

Write

```
SharedPreferences sharedPref = getActivity().getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sharedPref.edit();
editor.putInt(getString(R.string.saved_high_score), newHighScore);
editor.commit();
```

Read

The value can be retrieved providing the key. Optionally a default value can be returned if the key isn't present.

```
SharedPreferences sharedPref = getActivity().getPreferences(Context.MODE_PRIVATE);
int defaultValue = getResources().getInteger(R.string.saved_high_score_default);
long highScore = sharedPref.getInt(getString(R.string.saved_high_score), defaultValue);
```

Exercise

- Create an Activity which asks for user's details such as name, age, gender and school. When the app is closing the details must be saved in a shared preference. When the activity is recreated the details should retrieved from the shared preference and set in relevant fields.

Saving Files

- Internal Storage
  - Always available.
  - Files saved here are accessible by only the app which the files is saved, by default.
  - When the user uninstalls your app, the system removes all the app's files from internal storage.
- External Storages
  - Not always available
  - Files saved here may be read outside of the control of the app which created the file.
  - When the user uninstalls the app, the system removes the app's files from here only if the app save them in the directory from getExternalFilesDir().
- **Obtain Permissions for External Storage**

Permission to Write

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

Permission to Read

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

Save a File on Internal Storage

- getFilesDir()
  - Returns a File representing an internal directory for your app. i.e. Returns the absolute path to the directory on the file system where files created with openFileOutput(String, int) are stored.
- app.getCacheDir()
  - Returns a File representing an internal directory for the app's temporary cache files.
  - Be sure to delete each file once it is no longer needed and implement a reasonable size limit for the amount of memory to use at any given time.
  - If the system begins running low on storage, it may delete the cache files without warning.

```
String filename = "myfile";
String string = "Hello world!";
FileOutputStream outputStream;

try {
  outputStream = openFileOutput(filename, Context.MODE_PRIVATE);
  outputStream.write(string.getBytes());
  outputStream.close();
} catch (Exception e) {
  e.printStackTrace();
}
```

Save a File on Internal Storage

app.getCacheDir()

```
public File getTempFile(Context context, String url) {
    File file;
    try {
        String fileName = Uri.parse(url).getLastPathSegment();
        file = File.createTempFile(fileName, null, context.getCacheDir());
    catch (IOException e) {
        // Error while creating file
    }
    return file;
}
```

Check The State of External Storage

```
/* Checks if external storage is available for read and write */
public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}

/* Checks if external storage is available to at least read */
public boolean isExternalStorageReadable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state) ||
        Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
        return true;
    }
    return false;
}
```

Manipulate Directories on External Storage

```
String pathToExternalStorage = Environment.getExternalStorageDirectory().toString();
File appDirectory = new File(pathToExternalStorage + "/" + "AppName");
// have the object build the directory structure, if needed.
appDirectory.mkdirs();
```

Query Free Space

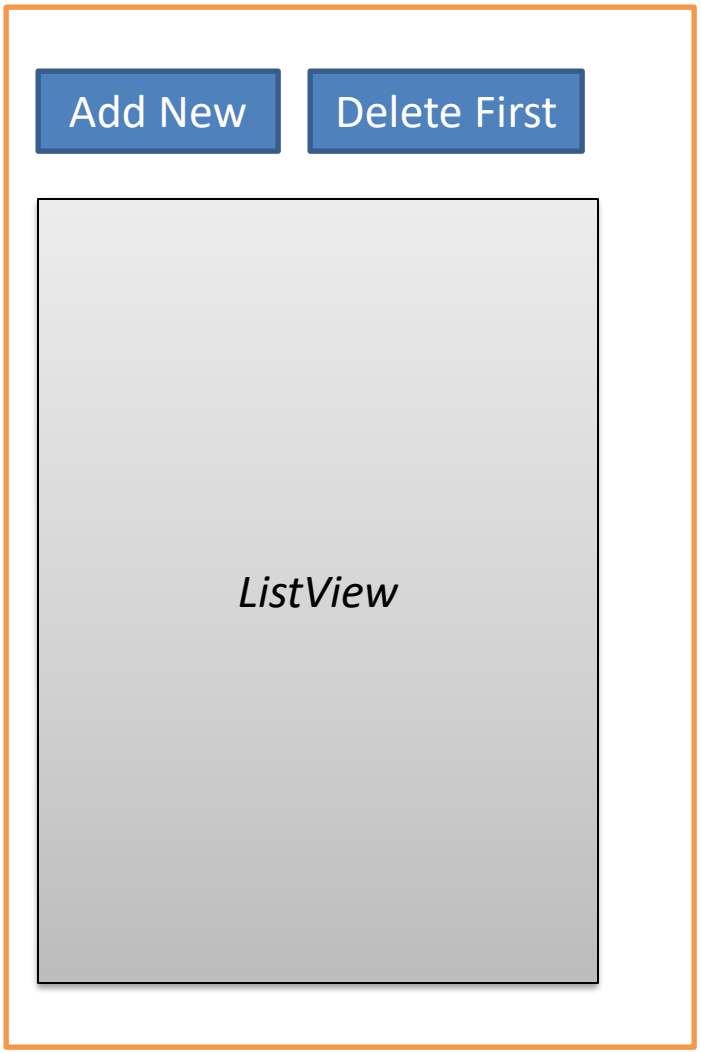
- getFreeSpace()
- getTotalSpace().

Android SQLite database

- *SQLite* is an Open Source database.
- It supports standard relational database features like SQL syntax.
- The database requires limited memory at runtime (approx. 250 KByte)
- SQLite supports three data types
  - TEXT
  - INTEGER
  - REAL
- Database is by default saved in the directory
  - DATA/data/APP_NAME/databases/FILENAME.

Example

- Create the following Layout



```
<LinearLayout
    android:id="@+id/group"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" >
    <Button
        android:id="@+id/add"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Add New"
        android:onClick="onClick"/>
    <Button
        android:id="@+id/delete"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Delete First"
        android:onClick="onClick"/>

</LinearLayout>
<ListView
    android:id="@android:id/list"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello" />
```

- Create the MySQLiteHelper class which responsible for creating the database.

```
public class MySQLiteHelper extends SQLiteOpenHelper {
  public static final String TABLE_COMMENTS = "comments";
  public static final String COLUMN_ID = "_id";
  public static final String COLUMN_COMMENT = "comment";
  private static final String DATABASE_NAME = "commments.db";
  private static final int DATABASE_VERSION = 1;
  // Database creation sql statement
  private static final String DATABASE_CREATE = "create table "
      + TABLE_COMMENTS + "(" + COLUMN_ID
      + " integer primary key autoincrement, " + COLUMN_COMMENT
      + " text not null);";
  public MySQLiteHelper(Context context) {
    super(context, DATABASE_NAME, null, DATABASE_VERSION);
  }
  @Override
  public void onCreate(SQLiteDatabase database) {
    database.execSQL(DATABASE_CREATE);
  }

...
```

```
@Override
  public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
    Log.w(MySQLiteHelper.class.getName(),
        "Upgrading database from version " + oldVersion + " to "
            + newVersion + ", which will destroy all old data");
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_COMMENTS);
    onCreate(db);
  }
}
```

- Create the Comment class.
  - This is the model class and contains the data which will save in the database

```
public class Comment {
        private long id;
        private String comment;

        public long getId() {
            return id;
        }
        public void setId(long id) {
            this.id = id;
        }
        public String getComment() {
            return comment;
        }
        public void setComment(String comment) {
            this.comment = comment;
        }

        // Will be used by the ArrayAdapter in the ListView
        @Override
        public String toString() {
            return comment;
        }
}
```

- Create the CommentsDataSource class. This class is the data access object (DAO) which maintains the database connection and supports adding new comments and fetching all comments

```
public class CommentsDataSource {
  // Database fields
  private SQLiteDatabase database;
  private MySQLiteHelper dbHelper;
  private String[] allColumns = { MySQLiteHelper.COLUMN_ID,
      MySQLiteHelper.COLUMN_COMMENT };

  public CommentsDataSource(Context context) {
    dbHelper = new MySQLiteHelper(context);
  }

  public void open() throws SQLException {
    database = dbHelper.getWritableDatabase();
  }

  public void close() {
    dbHelper.close();
  }
```

```
 public Comment createComment(String comment) {
    ContentValues values = new ContentValues();

    values.put(MySQLiteHelper.COLUMN_COMMENT, comment);

    long insertId = database.insert(MySQLiteHelper.TABLE_COMMENTS, null,
        values);

    Cursor cursor = database.query(MySQLiteHelper.TABLE_COMMENTS,
        allColumns, MySQLiteHelper.COLUMN_ID + " = " + insertId, null,
        null, null, null);

    cursor.moveToFirst();
    Comment newComment = cursorToComment(cursor);
    cursor.close();
    return newComment;
  }

  public void deleteComment(Comment comment) {
    long id = comment.getId();
    System.out.println("Comment deleted with id: " + id);
    database.delete(MySQLiteHelper.TABLE_COMMENTS,
MySQLiteHelper.COLUMN_ID
        + " = " + id, null);
  }
```

```
public List<Comment> getAllComments() {
    List<Comment> comments = new ArrayList<Comment>();
    Cursor cursor = database.query(MySQLiteHelper.TABLE_COMMENTS,
        allColumns, null, null, null, null, null);
    cursor.moveToFirst();
    while (!cursor.isAfterLast()) {
      Comment comment = cursorToComment(cursor);
      comments.add(comment);
      cursor.moveToNext();
    }
    // make sure to close the cursor
    cursor.close();
    return comments;
  }

  private Comment cursorToComment(Cursor cursor) {
    Comment comment = new Comment();
    comment.setId(cursor.getLong(0));
    comment.setComment(cursor.getString(1));
    return comment;
  }
}
```

- The MainActivity

```
@Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    datasource = new CommentsDataSource(this);
    datasource.open();
    List<Comment> values = datasource.getAllComments();
    // use the SimpleCursorAdapter to show the
    // elements in a ListView
    ArrayAdapter<Comment> adapter = new ArrayAdapter<Comment>(this,
        android.R.layout.simple_list_item_1, values);
    setListAdapter(adapter);
  }
```

- The MainActivity

```
// Will be called via the onClick attribute
  // of the buttons in main.xml
  public void onClick(View view) {
    @SuppressWarnings("unchecked")
    ArrayAdapter<Comment> adapter = (ArrayAdapter<Comment>)
getListAdapter();
    Comment comment = null;
    switch (view.getId()) {
    case R.id.add:
      String[] comments = new String[] { "Cool", "Very nice", "Hate it"
};
      int nextInt = new Random().nextInt(3);
      // save the new comment to the database
      comment = datasource.createComment(comments[nextInt]);
      adapter.add(comment);
      break;
    case R.id.delete:
      if (getListAdapter().getCount() > 0) {
        comment = (Comment) getListAdapter().getItem(0);
        datasource.deleteComment(comment);
        adapter.remove(comment);
      }
      break;
    }
    adapter.notifyDataSetChanged();
```

- The MainActivity

```
@Override
  protected void onResume() {
    datasource.open();
    super.onResume();
  }
  @Override
  protected void onPause() {
    datasource.close();
    super.onPause();
  }
```

Exercise

- Create a notepad application.
- The application should store the title, date and the body of a note.
- The titles of stored notes should be shown in a list.
- When the list item is clicked, the particular note with the title and the date/time should be shown in a new activity.