|  |
| --- |
| **National Day Parade Theme Song Compilation** |

***Resources and References***

**SQLiteOpenHelper** <http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html>

**Android SQLite Database Tutorial**

<http://www.androidhive.info/2011/11/android-sqlite-database-tutorial/>

**ListView Tutorial**

<http://www.vogella.com/tutorials/AndroidListView/article.html>

**SharedPreferences**

<http://developer.android.com/guide/topics/data/data-storage.html#pref>

Table of Contents

[**National Day Parade Theme Song Compilation** 1](#_Toc138764083)

[Section A: Recap database 2](#_Toc138764084)

[Section B: Creating Activity layout for data insertion and retrieval 6](#_Toc138764085)

[Section C: The Second Activity – Update/Delete record 8](#_Toc138764086)

[Section D: Auto refresh of UI Components 13](#_Toc138764087)

[Section E: Using ListView 13](#_Toc138764088)

[Section F: Filtered Selection 14](#_Toc138764089)

[Section G: Upgrading Table with Existing Data 16](#_Toc138764090)

[**Understanding SharedPreferences and Android Activity lifecycle** 17](#_Toc138764091)

[Section H: When to Save 17](#_Toc138764092)

[Section I: Saving and Retrieving Data in the App 22](#_Toc138764093)

[**Applying SharedPreferences in Saving User Input** 25](#_Toc138764094)

[Section J 25](#_Toc138764095)

[**Handling the Problem Statement** 29](#_Toc138764096)

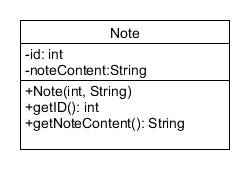
## Section A: Recap database

1. Create a new project named **Demo DatabaseCRUD** with a suitable package name and the rest of the options as default.

|  |  |
| --- | --- |
| **Project Template** | Empty View Activity |
| **Application Name** | Demo DatabaseCRUD |
| **Package Name** | sg.edu.rp.c346.id<your student ID>.demodatabasecrud |
| **Project Location** | D:\C346\Workspace\demodatabasecrud |
| **Language** | Java |
| **Minimum API Level** | API 24 |

\*Import into Github

1. The data we will be working on within the App will be as the Java Class Diagram given below. The corresponding table scheme is given on the right below.



|  |
| --- |
| Note |
| \_id INTEGER PRIMARY KEY AUTOINCREMENT  noteContent TEXT |

\

What is the SQL statement to create the table?

|  |
| --- |
| CREATE Table Note (\_id INTEGER PRIMARY KEY AUTOINCREMENT, noteContent TEXT); |
| Create a new Java Class accordingly in the project. You may notice the class implements Serializable. You would understand later that we need to pass the class object to another Activity and implementing the Serializable interface lets us pass the class object as a Serializable object to another Activity.  Note.java |
| public class Note implements Serializable {   private int id;   private String noteContent;  public Note( int id, String noteContent ) {  this.id = id;   this.noteContent = noteContent;   }   public int getId() { return id; }  public String getNoteContent() { return noteContent; }   public void setNoteContent(String noteContent) {   this.noteContent = noteContent;  }  @Override  public String toString() { return "ID:" + id + ", " + noteContent; }  } |

Because of the toString() method, even with a complex data, a simple ListView can be used with a simple ArrayAdapter. The ArrayAdapter will display only 1 line of string, which is defined above.

1. To use SQLite database in the app, you’ll need to prepare the necessary plumbing works. To do this, create the Java Class DBHeper.java which extends SQLiteOpenHelper.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | public class DBHelper extends SQLiteOpenHelper {   private static final String *DATABASE\_NAME* = "simplenotes.db";  private static final int *DATABASE\_VERSION* = 1;  private static final String *TABLE\_NOTE* = "note";  private static final String *COLUMN\_ID* = "\_id";  private static final String *COLUMN\_NOTE\_CONTENT* = "note\_content";   public DBHelper(Context context) {  super(context, *DATABASE\_NAME*, null, *DATABASE\_VERSION*);  }   @Override  public void onCreate(SQLiteDatabase db) {  String createNoteTableSql = "CREATE TABLE " + *TABLE\_NOTE* + "("  + *COLUMN\_ID* + " INTEGER PRIMARY KEY AUTOINCREMENT,"  + *COLUMN\_NOTE\_CONTENT* + " TEXT ) ";  db.execSQL(createNoteTableSql);  }   @Override  public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {  db.execSQL("DROP TABLE IF EXISTS " + *TABLE\_NOTE*);  onCreate(db);  }  } |

1. Last Lesson, you have learned how to **insert** a new record into the table. Add the following method into DBHelper.java

|  |  |
| --- | --- |
| 32  33  34  35  36  37  38  39  40 | public long insertNote(String noteContent) {  SQLiteDatabase db = this.getWritableDatabase();  ContentValues values = new ContentValues();  values.put(*COLUMN\_NOTE\_CONTENT*, noteContent);  long result = db.insert(*TABLE\_NOTE*, null, values);  db.close();  Log.*d*("SQL Insert","ID:"+ result); *//id returned, shouldn’t be -1*  return result;  } |

Line 36, will get a number, which is the record id (the primary key, ***\_id***) of the table for the record inserted. The id will be -1 if the insert failed. Thus, we can use it to check if a record is inserted successfully.

Example:

|  |
| --- |
| long result = db.insert(*TABLE\_NOTE*, null, values);  if (result == -1){  Log.d(“DBHelper”, “Insert failed”);  } |

1. You have also learned how to perform records retrieval from the database table. Append the following method into DBHelper.java

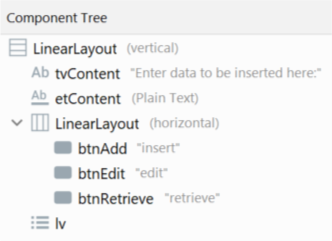
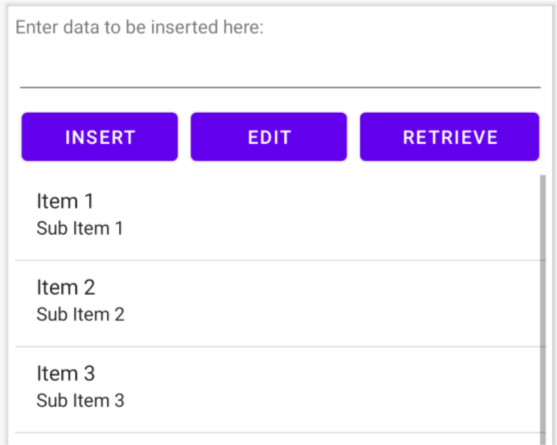
|  |  |
| --- | --- |
| 40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61 | public ArrayList<Note> getAllNotes() {  ArrayList<Note> notes = new ArrayList<Note>();  SQLiteDatabase db = this.getReadableDatabase();  String[] columns= {*COLUMN\_ID*, *COLUMN\_NOTE\_CONTENT*};  Cursor cursor = db.query(*TABLE\_NOTE*, columns, null, null,  null, null, null, null);    if (cursor.moveToFirst()) {  do {  int id = cursor.getInt(0);  String noteContent = cursor.getString(1);  Note note = new Note(id, noteContent);  notes.add(note);  } while (cursor.moveToNext());  }  cursor.close();  db.close();  return notes;  } |

This method will retrieve the Note records. Thereafter, the records are put into an ArrayList to be returned.

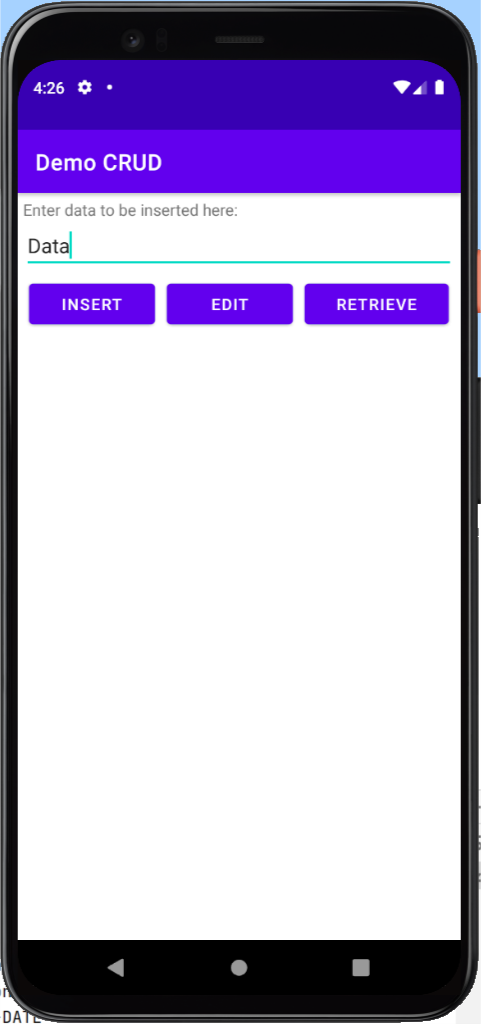
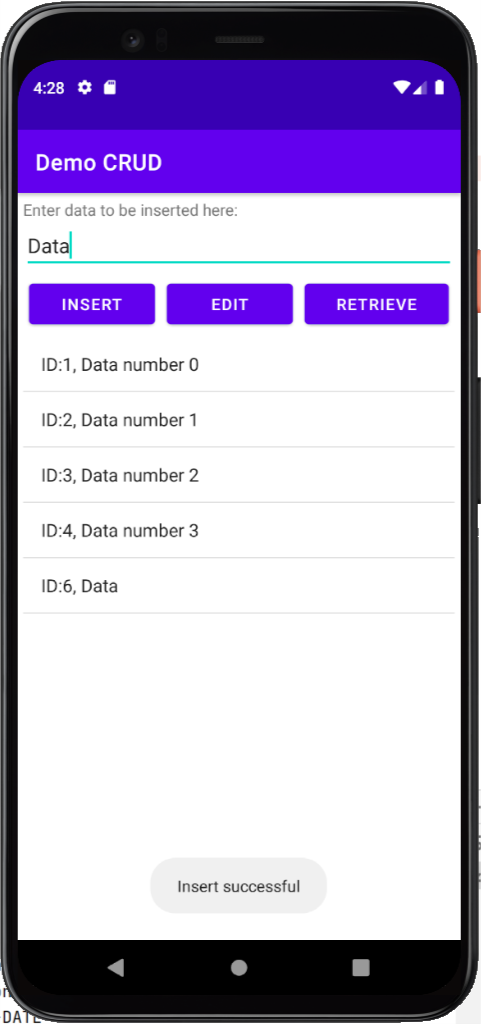
1. To prepare the database for testing purposes, we could create some dummy data during the creation of the table. Append the following code in DBHelper.java, under the method **onCreate()**

|  |  |
| --- | --- |
| 25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | @Override  public void onCreate(SQLiteDatabase db) {  String createNoteTableSql = "CREATE TABLE " + *TABLE\_NOTE* + "("  + *COLUMN\_ID* + " INTEGER PRIMARY KEY AUTOINCREMENT,"  + *COLUMN\_NOTE\_CONTENT* + " TEXT ) ";  db.execSQL(createNoteTableSql);  Log.*i*("info", "created tables");   *//Dummy records, to be inserted when the database is created*  for (int i = 0; i< 4; i++) {  ContentValues values = new ContentValues();  values.put(*COLUMN\_NOTE\_CONTENT*, "Data number " + i);  db.insert(*TABLE\_NOTE*, null, values);  }  Log.*i*("info", "dummy records inserted");  } |

## Section B: Creating Activity layout for data insertion and retrieval

1. Create the above layout for our first Activity in activity\_main.xml
2. When the **INSERT** button is clicked, the content in the EditText (**etContent**) is inserted into the database. When the **RETRIEVE** button is clicked, the records will be displayed in the ListView **lv**.

1. Modify and add the Java code below to perform the **Insert** action and **Select** action. Run it to see if it works?

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | public class MainActivity extends AppCompatActivity {  Button btnAdd, btnEdit, btnRetrieve;  TextView tvDBContent;  EditText etContent;  ArrayList<Note> al;  ListView lv;  ArrayAdapter<Note> aa;  @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.*activity\_main*);   //initialize the variables with UI here  al = new ArrayList<Note>();  aa = new ArrayAdapter<Note>(this,  android.R.layout.*simple\_list\_item\_1*, al);  lv.setAdapter(aa);  btnAdd.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  String data = etContent.getText().toString();  DBHelper dbh = new DBHelper(MainActivity.this);  long inserted\_id = dbh.insertNote(data);  if (inserted\_id != -1){  al.clear();  al.addAll(dbh.getAllNotes());  aa.notifyDataSetChanged();  Toast.makeText(MainActivity.this, "Insert successful",  Toast.LENGTH\_SHORT).show();  }  }  });  btnRetrieve.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  DBHelper dbh = new DBHelper(MainActivity.this);  al.clear();  al.addAll(dbh.getAllNotes());  aa.notifyDataSetChanged();  }  });  } } |

What do lines 30-31 do?

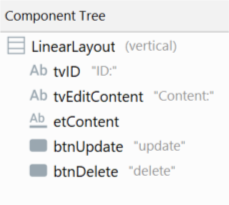
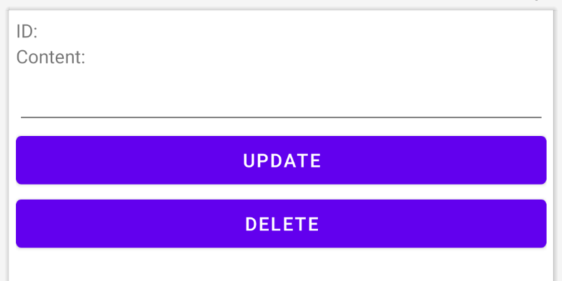
|  |
| --- |
|  |

## Section C: The Second Activity – Update/Delete record

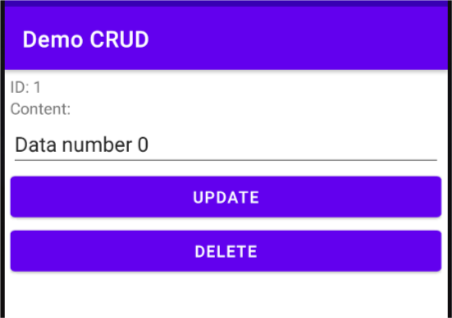
## 

1. The **EDIT** in MainActivity button will launch another Activity – EditActivity to let user perform update and delete operations on the database.

Create a **new** Empty Activity named EditActivity and modify the XML layout as follows:

1. Upon starting up the EditActivity, it will expect a Note object from the MainActivity.



ID is needed as the primary key to the record

Content is needed too!

I need a Note object

The ID of the Note object will be displayed in a TextView as the primary key shouldn’t be modified. The content of the object will be displayed in an EditText for modifications.

1. Below is the Java code for EditActivity.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | public class EditActivity extends AppCompatActivity {   TextView tvID;  EditText etContent;  Button btnUpdate, btnDelete;  Note data;  @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.*activity\_edit*);   //initialize the variables with UI here   Intent i = getIntent();  data = (Note) i.getSerializableExtra("data");   tvID.setText("ID: " + data.getId());  etContent.setText(data.getNoteContent());   }  } |

1. To perform the update, a method named **updateNote()** needs to be implemented in the DBHelper.java. The method will take in a Note object and update it to the database.

Add the method below to EditActivity.java to update the note when the update button is clicked.

|  |  |
| --- | --- |
| 21  22  23  24  25  26  27  28  29  30  31 | btnUpdate.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  DBHelper dbh = new DBHelper(EditActivity.this);  data.setNoteContent(etContent.getText().toString());  dbh.updateNote(data);  dbh.close();  }  }); |

Add the corresponding method to DBHelper.java to update the note.

|  |  |
| --- | --- |
| 62  63  64  65  66  67  68  69  70  71 | public int updateNote(Note data){  SQLiteDatabase db = this.getWritableDatabase();  ContentValues values = new ContentValues();  values.put(*COLUMN\_NOTE\_CONTENT*, data.getNoteContent());  String condition = *COLUMN\_ID* + "= ?";  String[] args = {String.*valueOf*(data.getId())};  int result = db.update(*TABLE\_NOTE*, values, condition, args);  db.close();  return result;  } |

Line 68 will get a number representing the number of rows affected in the table. Usually, for record updates, we expect 1 or more records to be updated. In this case, we expect only 1 record though. Thus, we can use it to check if a record is updated successfully if the affected record is 1.

int result = db.update(*TABLE\_NOTE*, values, condition, args);

if (result < 1){

Log.d(“DBHelper”, “Update failed”);

}

Example:

1. The Update SQL typically has the following syntax

Update *Table\_Name*

SET *column1 = value1, column2 = value2, ……*

WHERE *primary\_key\_column = key;*

UPDATE note

SET note\_content = “some data”  
WHERE \_id = 1;

Example:

Each **?** is substituted by a **String**

ContentValues values = new ContentValues();  
values.put(*COLUMN\_NOTE\_CONTENT*, data.getNoteContent());

String condition = *COLUMN\_ID* + "= ?";  
String[] args = { String.*valueOf*(data.getId()) };

1. To perform delete, a method named **deleteNote()** needs to be implemented in the DBHelper.java. The method will accept an ID as the primary reference to delete the record from the database.

Add the code to call the method in the onCreate() method in EditActivity.java

|  |  |
| --- | --- |
| 36  37  38  39  40  41  42  43  44 | btnDelete.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  DBHelper dbh = new DBHelper(EditActivity.this);  dbh.deleteNote(data.getId());    }  }); |

Add the corresponding method in DBHelper.java

|  |  |
| --- | --- |
| 75  76  77  78  79  80  81  82 | public int deleteNote(int id){  SQLiteDatabase db = this.getWritableDatabase();  String condition = *COLUMN\_ID* + "= ?";  String[] args = {String.*valueOf*(id)};  int result = db.delete(*TABLE\_NOTE*, condition, args);  db.close();  return result;  } |

Line 79, will get a number representing the number of rows affected in the table. Usually, we expect 1 or more records to be affected. In this case, we expect only 1 record though. Thus, we can use it to check if a record is deleted successfully if the affected record is 1.

int result = db.delete(*TABLE\_NOTE*, condition, args);

if (result < 1){

Log.d(“DBHelper”, “Delete failed”);

}

Example:

1. The Delete SQL typically has the following syntax

DELETE FROM *Table\_Name*

WHERE *primary\_key\_column = key;*

DELETE FROM note

WHERE \_id = 1;

Example:

String condition = *COLUMN\_ID* + "= ?";  
String[] args = { String.*valueOf*(id) };

1. At the MainActivity, you need to implement the Edit button. Below is the code snippet for launching the EditActivity and to pass a Note object over.

|  |  |
| --- | --- |
| 1  35  36  37  38  39  40  41  42  43  44  45  46  47 | public class MainActivity extends AppCompatActivity {  …  @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.*activity\_main*);  …  btnEdit.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  Note target = al.get(0);  Intent i = new Intent(MainActivity.this,  EditActivity.class);  i.putExtra("data", target);  startActivity(i);  }  });  } } |

1. Run the application, **click the RETRIEVE button first** to populate the ArrayList, and proceed to experiment with the EDIT and DELETE operations. You’ll notice that after clicking the UPDATE or DELETE buttons, you are still at the EditActivity.

What code could you insert into the buttons, to return to the earlier Activity?

|  |
| --- |
|  |

1. Back to the MainActivity, you’ll notice that the TextView is filled with stale data (data not current). We need a way to automate the refresh.

How to detect that we have returned to the MainActivity from the EditActivity, so that we could trigger the record retrieval from the database?

|  |
| --- |
|  |

## Section D: Auto refresh of UI Components

1. In the MainActivity, override the method onResume() and make it perform the click on the RETRIEVE button.

|  |  |
| --- | --- |
| 1  2  48  49  50  51  52  53  54 | public class MainActivity extends AppCompatActivity {  …  @Override  protected void onResume() {  super.onResume();   btnRetrieve.performClick();  }  } |

## Section E: Using ListView

1. What are the current limitations on the Edit button that you have observed/spotted on the MainActivity?

|  |
| --- |
|  |

1. The user can select a record to be modified. You could implement OnItemClickListener() on the ListView as below.

|  |  |
| --- | --- |
| 1  2  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73 | public class MainActivity extends AppCompatActivity {  …  @Override  protected void onCreate(Bundle savedInstanceState) {  …  lv = findViewById(R.id.*lv*);  aa = new ArrayAdapter<Note>(this,  android.R.layout.*simple\_list\_item\_1*, al);  lv.setAdapter(aa);  lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {  @Override  public void onItemClick(AdapterView<?> parent, View view, int  position, long identity) {  Note data = al.get(position);  Intent i = new Intent(MainActivity.this,  EditActivity.class);  i.putExtra("data", data);  startActivity(i);  }  });  }  …  } |

## Section F: Filtered Selection

1. The Select statement can be used to filter some results instead of retrieving all records from the database. Below is an example of a method used to retrieve only records that contain a given keyword.

Add this method to DBHelper.java

|  |  |
| --- | --- |
| 85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106 | public ArrayList<Note> getAllNotes(String keyword) {  ArrayList<Note> notes = new ArrayList<Note>();   SQLiteDatabase db = this.getReadableDatabase();  String[] columns= {*COLUMN\_ID*, *COLUMN\_NOTE\_CONTENT*};  String condition = *COLUMN\_NOTE\_CONTENT* + " Like ?";  String[] args = { "%" + keyword + "%"};  Cursor cursor = db.query(*TABLE\_NOTE*, columns, condition, args,  null, null, null, null);   if (cursor.moveToFirst()) {  do {  int id = cursor.getInt(0);  String noteContent = cursor.getString(1);  Note note = new Note(id, noteContent);  notes.add(note);  } while (cursor.moveToNext());  }  cursor.close();  db.close();  return notes;  } |

More methods of SQLiteDatabase can be found here:

<http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html>

1. The Select SQL typically has the following syntax

SELECT *column1, column2*, …..  
FROM *Table\_Name*

WHERE *column = key*

GROUPBY column1

HAVING value

ORDERBY column1

LIMIT rows

SELECT \_id, note\_content

FROM note

WHERE note\_content LIKE %keyword%;

Example:

String[] columns= {*COLUMN\_ID*, *COLUMN\_NOTE\_CONTENT*};

String condition = *COLUMN\_NOTE\_CONTENT* + "Like %?%";  
String[] args = {keyword};

1. On MainActivity, we want the user to enter the text to filter in the EditText. If the user enters a text in the EditText and click on the RETRIEVE button, we would filter the result by the text. However, if the EditText is empty, we would just retrieve everything. Modify the OnClick() handler as below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | public class MainActivity extends AppCompatActivity {  ...  @Override  protected void onCreate(Bundle savedInstanceState) {  ...  btnRetrieve.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  DBHelper dbh = new DBHelper(MainActivity.this);  al.clear();  // al.addAll(dbh.getAllNotes());  String filterText = etContent.getText().toString().trim();  if(filterText.length() == 0) {  al.addAll(dbh.getAllNotes());  }  else{  al.addAll(dbh.getAllNotes(filterText));  }  aa.notifyDataSetChanged();  }  });  ...  } } |

Try it out by typing 0 in the EditText and click the RETRIEVE button, what do you see?

|  |
| --- |
|  |

## Section G: Upgrading Table with Existing Data

1. You had seen that we drop a table when we need to upgrade the database as the code on line 6 below shows. This is not a good way to upgrade your table for your users as it means all the data your user had accumulated would be gone.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | public class DBHelper extends SQLiteOpenHelper {  ...  @Override  public void onUpgrade(SQLiteDatabase db, int oldVersion,  int newVersion) {  db.execSQL("DROP TABLE IF EXISTS " + *TABLE\_NOTE*);  onCreate(db);  }  } |

1. To add a column called **module\_name** to an existing table, modify **onUpgrade()** to execute an SQL statement as below. To make the app calls onUpgrade(), increment the variable DATABASE\_VERSION from 1 to 2.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | public class DBHelper extends SQLiteOpenHelper {  ...  private static final int *DATABASE\_VERSION* = 2;  ...  @Override  public void onUpgrade(SQLiteDatabase db, int oldVersion,  int newVersion) {  // ~~db.execSQL("DROP TABLE IF EXISTS " + TABLE\_NOTE);~~  db.execSQL("ALTER TABLE " + TABLE\_NOTE + " ADD COLUMN module\_name TEXT ");  //~~onCreate(db); // Delete as table already created~~  }  ...  } |

# **Understanding SharedPreferences and Android Activity lifecycle**

Let us look at how to use SharedPreferences to support data persistency of a simple data structure in the following exercise. You will be creating a simple app that stores and displays a greeting message.

## Section H: When to Save

In the Android Studio, create a new project with the following configuration requirements:

|  |  |
| --- | --- |
| **Project Template** | Empty View Activity |
| **Application Name** | Demo Simple Save |
| **Package Name** | sg.edu.rp.c346.id<your student ID>.demosimplesave |
| **Project Location** | D:\C346\Workspace\DemoSimpleSave |
| **Language** | Java |
| **Minimum API Level** | API 24 |

**Design the UI Layout**

Overwrite the generated layout file with a linear vertical layout as shown below.

|  |  |
| --- | --- |
| A screenshot of a computer  Description automatically generated with medium confidence | A screenshot of a computer  Description automatically generated with low confidence |

**Implement the App without Data Persistency**

|  |
| --- |
| A picture containing text, diagram, screenshot, plan  Description automatically generated |

1. In P05, we learned a set of six callback methods in the Android Activity lifecycle. Now let’s generate one of them, “onResume()”, in the MainActivity. Since the method is overridden, you can auto-generate it using Android Studio in either of the two ways below:
2. Select a white space in the class MainActivity.

Click on the Android Studio menu, “Code 🡪 Override Methods…”

A screenshot of a computer

Description automatically generated

1. Right-click on a white space in the class MainActivity.

Select “Generate…” and then “Override Methods…”

|  |  |
| --- | --- |
| A screenshot of a computer  Description automatically generated | A screenshot of a computer  Description automatically generated |

1. In the pop-up window, search and select “onResume()” as shown below. Click **OK**.

|  |
| --- |
| A screenshot of a computer  Description automatically generated with medium confidence |

1. The “onResume()” method should be auto-generated already. Now, add the following code (in red) into the method to display a Toast message when it is called.

|  |
| --- |
| A picture containing text, screenshot, font  Description automatically generated |

1. Run the app and observe your Emulator.

|  |  |
| --- | --- |
| ? | What message is displayed?  Referring to the Activity lifecycle above, explain why this message is displayed. |
|  | |

1. Click the “Overview” button to open the “Recent Apps” window.

|  |
| --- |
| A picture containing screenshot, design  Description automatically generated |

1. Mirroring how an app is closed in the real phone, swipe the app or click the cross button on the top right corner to terminate it.

|  |
| --- |
| C:\Users\denise_quek\Desktop\Screenshot_1593399751.png |

1. Run the app again by launching it from the Emulator menu and make some observations.

|  |
| --- |
| C:\Users\denise_quek\Desktop\Screenshot_1560503173.png |

|  |  |
| --- | --- |
| ? | What message is displayed this time?  Referring to the Activity lifecycle above, explain why this message is displayed. |
|  | |

## Section I: Saving and Retrieving Data in the App

Now let us use SharedPreferences to save the message in this app.

1. Referring to how we generated the “onResume()” method in Step 1, generate the “onPause()” method in the MainActivity.

|  |
| --- |
| A screenshot of a computer program  Description automatically generated with medium confidence |

1. To save the persistent data **when the app is no longer (fully) visible**, enter the code below into the “onPause()” method.

*Note: The writing of the value is in a (key, value) pair. The key is the name of the data structure used to “point” to the value that we wish to save in the local storage.*

|  |
| --- |
| A picture containing text, screenshot, font, number  Description automatically generated |

|  |  |
| --- | --- |
| ? | Referring to the Activity lifecycle above, explain why we should write the data-saving code in the “onPause()” method. |
|  | |

1. To retrieve the saved data **when the Activity is resumed (fully visible)**, modify the code in the “onResume()” method as follows.

*Note: The reading of the value is using the method getString(key, default value). The key is the name of the preference to retrieve, which points to a value in the SharedPreferences. If the value is not found (e.g., when the program is first installed), it will return the default value, “No greetings!” in this case.*

|  |
| --- |
| A screenshot of a computer code  Description automatically generated with low confidence |

1. Re-run the app on your Emulator and observe.

|  |  |
| --- | --- |
| ? | What message is displayed? Explain why. |
|  | |

1. Terminate the app and launch it again by following Step 5~7.

|  |  |
| --- | --- |
| ? | What message do you see this time? Explain why. |
|  | |

|  |  |
| --- | --- |
|  | **Learning Checkpoint** |
| **To recap what we have learned so far,**  In the above exercise, we created an Android app to save the hard-coded data and retrieve it for display using Toast. We tapped on the callback methods in the Android Activity lifecycle for data persistency, using “onPause()” method to save the data when the app dismisses to the background, and using “onResume()” method to retrieve and display the saved data, when the app becomes visible again. | |

# **Applying SharedPreferences in Saving User Input**

## 

## Section J

Let us use SharedPreferences to support the persistence of data input from the user. In the next exercise, you will be creating a simple app that stores information on GPA scores.

In Android Studio, create a new project with the following configuration requirements:

|  |  |
| --- | --- |
| **Project Template** | Empty View Activity |
| **Application Name** | Demo My Profile |
| **Package Name** | sg.edu.rp.c346.id<your student ID>.demomyprofile |
| **Project Location** | D:\C346\Workspace\DemoMyProfile |
| **Language** | Java |
| **Minimum API Level** | API 24 |

**Design the UI Layout**

Create a layout file with the following content as per the component tree and the device screen in Android Studio.

|  |  |
| --- | --- |
| A screenshot of a computer  Description automatically generated with medium confidence | A screenshot of a computer  Description automatically generated with medium confidence |

**Saving and Retrieving Data in the App**

1. Declare the following variables in the “MainActivity.java”.

|  |
| --- |
|  |

1. Bind them to the respective UI elements using “findViewById()”.
2. Generate the callback methods “onPause()” and “onResume()” in the MainActivity as we did earlier on.
3. Let us save the Username into SharedPreferences first.

Implement the “onPause()” method according to the hints in the comments below, to save the Username when the app is no longer (fully) visible.

*Note: You may refer to the code in the “Simple Save” project.*

|  |
| --- |
| A screenshot of a computer code  Description automatically generated with low confidence |

1. Implement the “onResume()” method according to the hints in the comments below, to retrieve the Username value when the Activity is resumed (fully visible).

*Note: You may refer to the code in the “Simple Save” project.*

|  |
| --- |
| A picture containing text, screenshot, font, algebra  Description automatically generated |

1. Run and test your app.

The username should be saved into SharedPreferences, whereas the GPA score is not being persistently stored yet.

1. Now let us save the GPA score into SharedPreferences.

Usually, the GPA score is decimal so we should consider declaring it as a float value. With different data types being used, the SharedPreferences method to save/retrieve the key-value pair will be different. Check the following links and complete the tables below.

<https://developer.android.com/reference/android/content/SharedPreferences.Editor.html>

|  |  |
| --- | --- |
| **Type of data to save** | **SharedPreferences method** |
| String | putString(String key, String value) |
| int | putInt(String key, int value) |
| float | putFloat(String key, float value) |
| boolean | putBoolean(String key, boolean value) |

<https://developer.android.com/reference/android/content/SharedPreferences>

|  |  |
| --- | --- |
| **Type of data to retrieve** | **SharedPreferences method** |
| String | [getString](https://developer.android.com/reference/android/content/SharedPreferences.html#getString(java.lang.String,%20java.lang.String))([String](https://developer.android.com/reference/java/lang/String.html) key, [String](https://developer.android.com/reference/java/lang/String.html) defValue) |
| int | [getInt](https://developer.android.com/reference/android/content/SharedPreferences.html#getInt(java.lang.String,%20int))([String](https://developer.android.com/reference/java/lang/String.html) key, int defValue) |
| float | [getFloat](https://developer.android.com/reference/android/content/SharedPreferences.html#getFloat(java.lang.String,%20float))([String](https://developer.android.com/reference/java/lang/String.html) key, float defValue) |
| boolean | [getBoolean](https://developer.android.com/reference/android/content/SharedPreferences.html#getBoolean(java.lang.String,%20boolean))([String](https://developer.android.com/reference/java/lang/String.html) key, boolean defValue) |

Modify the “onPause()” and “onResume()” using the correct SharedPreferences methods:

* To save the GPA score when the app is no longer (fully) visible
* To retrieve the GPA score when the Activity is resumed (fully visible)

*Note: You need to use Float.parseFloat() to convert the GPA score input by the user to the “float” data type before saving it into the SharedPreferences object.*

1. Did you manage to write similar code in “onPause()” and “onResume()” as shown in the box below? It is still missing the code to save and retrieve the GPA. You would need to add the code similar to how it is done for the name but do note GPA is a float.

|  |
| --- |
| @Override **protected void** onPause() {  **super**.onPause();   *// Step 1a: Get the user input from the EditText and store it in a variable* String strName = **etName**.getText().toString();  **float** gpa= Float.*parseFloat*(**etGPA**.getText().toString());   *// Step 1b: Obtain an instance of the SharedPreferences* SharedPreferences prefs = *getPreferences*(**MODE\_PRIVATE**);   *// Step 1c: Obtain an instance of the SharedPreference Editor for update later* SharedPreferences.Editor prefEdit = prefs.edit();   *// Step 1d: Add the key-value pair* prefEdit.putString(**"name"**, strName);    *// How to save GPA? Complete code here*  *// Step 1e: Call commit() to save the changes into SharedPreferences* prefEdit.commit(); }  @Override **protected void** onResume() {  **super**.onResume();   *// Step 2a: Obtain an instance of the SharedPreferences* SharedPreferences prefs = *getPreferences*(**MODE\_PRIVATE**);   *// Step 2b: Retrieve the saved data from the SharedPreferences object* String strName = prefs.getString(**"name"**, **"John"**);  *// How to retrieve GPA? Complete code here*     *// Step 2c: Update the UI element with the value* **etName**.setText(strName);  *// How to set back GPA? Complete code here* } |

1. Besides getPreferences(), you can call getSharedPreferences() to create a new shared preference file or access an existing one as well.

|  |  |
| --- | --- |
| ? | Research and summarise the different scenario where getPreferences() or getSharedPreferences() could be used.  <https://developer.android.com/training/data-storage/shared-preferences> |
|  | |

|  |  |
| --- | --- |
|  | **Learning Checkpoint** |
| **To recap what we have learned so far,**  In the above exercise, you created an Android app to save and retrieve the user input data. To save or retrieve different types of data, we need to use different SharedPreferences methods. | |

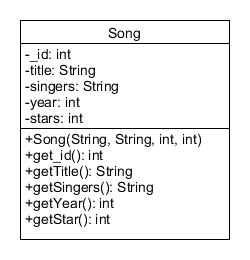
# **Handling the Problem Statement**

Create an app that lets the user input and save the songs they have come across.

*Note: You should have completed steps 1 to 5 below in the last lesson. Continue with step 6.*

1. Create a Song class as below.

|  |
| --- |
| song |
| \_id INTEGER PRIMARY KEY AUTOINCREMENT  title TEXT  singers TEXT  year INTEGER  stars INTEGER |



1. Create a Java class (you can name it DBHelper.java) which extends SQLiteOpenHelper for the database operations. Implement the following methods:
   1. Constructor
   2. onCreate -> SQL to create the table above
   3. onUpgrade
2. Implement the Create and Retrieve operations in the DBHelper.
3. Proceed to create the first Activity (MainActivity).

For the activity\_main.xml

* EditText for input like song title, singers, and year
* RadioButtons for ratings

For MainActivity.java

* Let the user insert a song record into the database
* Let the user launch a new Activity to display records

A screenshot of a music application

Description automatically generated with low confidence

1. Create the second Activity which contains a simple ListView, to retrieve and display all the songs in the database table.
2. For the second Activity

* When the user clicks one of the items in the ListView, launch a third Activity and pass the clicked item to it
* When the “SHOW ALL SONGS WITH 5 STARS” button is clicked, show all songs with 5 stars only
* Enhancement: implement a Spinner for filtering the songs by the year

A screenshot of a music video

Description automatically generated with low confidence

1. Implement the third Activity for the update/delete operations.

For the layout

* EditText for editing the song details
* Update and Delete Button
* You may need to add setters in the Song class

A screenshot of a phone

Description automatically generated with medium confidence

1. When the third Activity is closed, refresh the song list in the second Activity.