Databases + Android Studio

SEG2105 - Introduction to Software Engineering – Fall 2021

Lab 4 (2%)

Lab Goals

- Practice database management
- Connect a database with an application
- Practice creating different user interfaces with different functionality in Android Studio

Lab Objectives

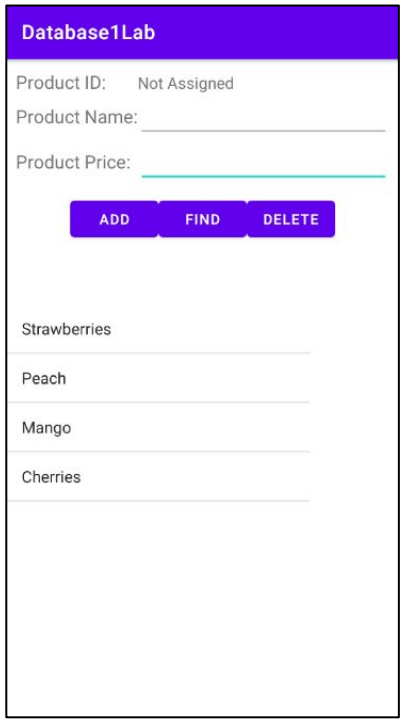
- Create a product manager app
- Use a database for the app
- Find a specific product
- Add a new product
- Delete an existing product
- View a list of all products

Database Options

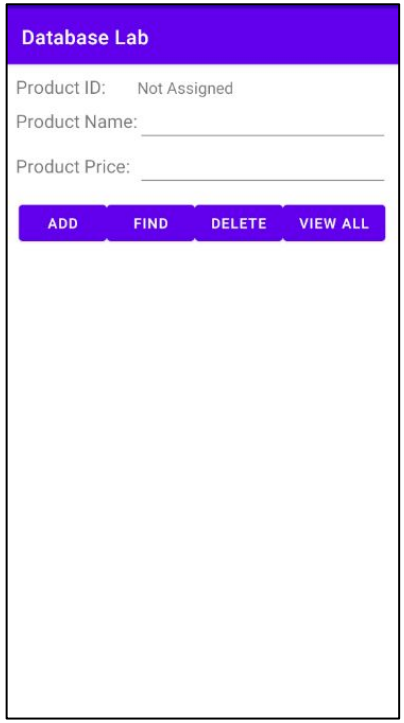
- SQLite (Tutorial 5)
  - Local embedded database
  - Good for basic development and small apps
- Firebase (Tutorial 6)
  - Realtime database, cloud-hosted, NoSQL
  - Note: Firebase is blocked in China
- Other
  - any database that you have experience with and would like to use

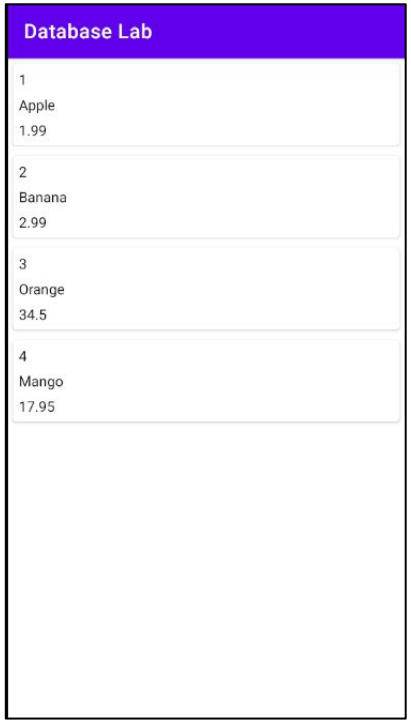
Display Options

1. Display Products on the Same Page
2. Display Products on a New Page



or





You are welcome to explore your own display! These are simply the 2 that I will be presenting in this lab.

Layout

- Full flexibility!
- Just make sure your app is able to add new products, find products, delete existing products, and display all products
- As a simple example, here is a plain layout:

- TableLayout
  - TableRow
    - TextView
    - TextView
  - TableRow
    - TextView
    - EditText
  - TableRow
    - TextView
    - EditText
- LinearLayout (horizontal)
  - Button
  - Button
  - Button

Display on Same Page

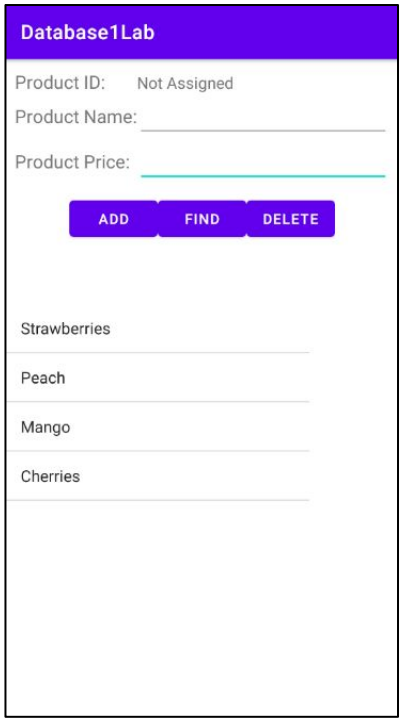
Using a ListView on the same page as the main layout to display data.

File Structure

- I will provide code for how to add, find, delete, and display products using SQLite
- I will walk you through the 1 xml file:
  - activity_main.xml
- I will walk you through these 3 java files:
  - Product.java
  - MyDBHandler.java
  - MainActivity.java

activity_main.xml

- This is the main page that allows the use to enter data using EditText and also displays the data in realtime
- Contains an add, find, and delete button
- Add strong id's to the buttons, edittexts, the textview for product ID, and the listview



- TableLayout
  - TableRow
    - TextView
    - TextView
  - TableRow
    - TextView
    - EditText
  - TableRow
    - TextView
    - EditText
- LinearLayout (horizontal)
  - Button
  - Button
  - Button
- ListView

Product.java

- Create your Product.java file and declare 3 private variables
- Create your constructors

```
2
3   public class Product {
4       private int _id;
5       private String _productname;
6       private double _price;
7
8       // constructors
9       public Product() {
10      }
11      public Product(int id, String productname, double price) {
12          _id = id;
13          _productname = productname;
14          _price = price;
15      }
16      public Product(String productname, double price) {
17          _productname = productname;
18          _price = price;
19      }
```

Product.java

- Add your getters and setters to the class

```
20
21      // setters and getters
22      public void setID(int id) { _id = id; }
25      public int getID() { return _id; }
28      public void setProductName(String productname) { _productname = productname; }
31      public String getProductName() { return _productname; }
34      public void setPrice(double price) { _price = price; }
37      public double getPrice() { return _price; }
40  }
```

MyDBHandler.java

- Create your MyDBHandler.java file
- Import libraries as you go by clicking "alt+enter" over red text
- Create private variables for your database schema

```
2
3   import android.content.ContentValues;
4   import android.content.Context;
5   import android.database.Cursor;
6   import android.database.sqlite.SQLiteDatabase;
7   import android.database.sqlite.SQLiteOpenHelper;
8
9   public class MyDBHandler extends SQLiteOpenHelper {
10      //defining the schema
11      private static final int DATABASE_VERSION = 1;
12      private static final String DATABASE_NAME = "productDB.db";
13      private static final String TABLE_PRODUCTS = "products";
14      private static final String COLUMN_ID = "_id";
15      private static final String COLUMN_PRODUCTNAME = "productname";
16      private static final String COLUMN_PRICE = "price";
17
```

MyDBHandler.java

- Add a constructor

```
9   public class MyDBHandler extends SQLiteOpenHelper {
10      //defining the schema
11      private static final int DATABASE_VERSION = 1;
12      private static final String DATABASE_NAME = "productDB.db";
13      private static final String TABLE_PRODUCTS = "products";
14      private static final String COLUMN_ID = "_id";
15      private static final String COLUMN_PRODUCTNAME = "productname";
16      private static final String COLUMN_PRICE = "price";
17
18      // constructor
19      public MyDBHandler(Context context){
20          super(context, DATABASE_NAME,  factory: null, DATABASE_VERSION);
21      }
22
```

MyDBHandler.java

- The onCreate() method will create a table in our database with the proper SQL query

```
25      // create the table
26      @Override
27  @   public void onCreate(SQLiteDatabase db){
28          // CREATE TABLE TABLE_PRODUCTS (COLUMN_ID INTEGER PRIMARY KEY, COLUMN_PRODUCTNAME TEXT,
29          // COLUMN_PRICE DOUBLE)
30          String CREATE_PRODUCTS_TABLE = "CREATE TABLE " + TABLE_PRODUCTS
31                  + "(" + COLUMN_ID + " INTEGER PRIMARY KEY,"
32                  + COLUMN_PRODUCTNAME + " TEXT,"
33                  + COLUMN_PRICE + " DOUBLE" +
34                  ")";
35          db.execSQL(CREATE_PRODUCTS_TABLE);
36      }
```

MyDBHandler.java

- The onUpgrade() method will remove the table if it exists by using an SQL query

```
37
38      // deletes old tables and creates a new one
39      // change tables by incrementing the database version number
40      @Override
41      public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion){
42          db.execSQL("DROP TABLE IF EXISTS " + TABLE_PRODUCTS);
43          onCreate(db);
44      }
45
```

MyDBHandler.java

- Create an addProduct() method that adds a product to the database

```
45
46      // insert into database
47  @   public void addProduct(Product product){
48          SQLiteDatabase db = this.getWritableDatabase();
49
50          // creating an empty set of values
51          ContentValues values = new ContentValues();
52          // add values to the set
53          values.put(COLUMN_PRODUCTNAME, product.getProductName());
54          values.put(COLUMN_PRICE, product.getPrice());
55
56          // insert the set into the products table and close
57          db.insert(TABLE_PRODUCTS,  nullColumnHack: null, values);
58          db.close();
59      }
60
```

MyDBHandler.java

- Create a findProduct() method that we use when searching for a product

```
60
61      // find a product from database
62      public Product findProduct(String productname){
63          SQLiteDatabase db = this.getWritableDatabase();
64
65          // run a query to find the product with the specified product name
66          // SELECT * FROM TABLE_PRODUCTS WHERE COLUMN_PRODUCTNAME = "productname"
67          String query = "SELECT * FROM " + TABLE_PRODUCTS
68                  + " WHERE " + COLUMN_PRODUCTNAME
69                  + " = \"" + productname + "\"";
70          // passing the query
71          Cursor cursor = db.rawQuery(query,  selectionArgs: null);
72
73          Product product = new Product();
74
75          // moves cursor to the first row
76          if(cursor.moveToFirst()){
77              product.setID(Integer.parseInt(cursor.getString( i: 0)));
78              product.setProductName(cursor.getString( i: 1));
79              product.setPrice(Double.parseDouble(cursor.getString( i: 2)));
80              cursor.close();
81          }else{
82              product = null;
83          }
84          db.close();
85
86          // we return the first product in the query result with the specified product name
87          // or null if there is no product with that name
88          return product;
89      }
```

MyDBHandler.java

- Create a deleteProduct() method to delete a product

```
90
91      // delete from database
92      public boolean deleteProduct(String productname){
93          boolean result = false;
94          SQLiteDatabase db = this.getWritableDatabase();
95
96          // run a query to find the product with the specified name, then delete
97          // SELECT * FROM TABLE_PRODUCTS WHERE COLUMN_PRODUCTNAME = "productname"
98          String query = "SELECT * FROM " + TABLE_PRODUCTS
99                  + " WHERE " + COLUMN_PRODUCTNAME
100                 + " = \"" + productname + "\"";
101         // passing the query
102         Cursor cursor = db.rawQuery(query,  selectionArgs: null);
103
104         // moves cursor to the first row
105         // this deletes the first occurrence of the product with the specified name
106         if(cursor.moveToFirst()){
107             String idStr = cursor.getString( i: 0);
108             db.delete(TABLE_PRODUCTS,  whereClause: COLUMN_ID + " = " + idStr,  whereArgs: null);
109             cursor.close();
110             result = true;
111         }
112         db.close();
113
114         // if product is deleted this returns true
115         return result;
116     }
117
```

MyDBHandler.java

- The viewData() method is to read all of the data in the table using the correct SQL query

```
116
117     // read all from table
118     public Cursor viewData(){
119         SQLiteDatabase db = this.getReadableDatabase();
120         String query = "SELECT * FROM " + TABLE_PRODUCTS;
121
122         // passing the query
123         Cursor cursor = db.rawQuery(query,  selectionArgs: null);
124
125         // returns all products from table
126         return cursor;
127     }
128 }
129
```

MainActivity.java

- In the MainActivity.java file, import libraries as you go along
- Declare some variables in the class

```
2
3   import androidx.appcompat.app.AppCompatActivity;
4
5   import android.database.Cursor;
6   import android.view.View;
7   import android.widget.AdapterView;
8   import android.widget.ArrayAdapter;
9   import android.widget.EditText;
10  import android.widget.ListView;
11  import android.widget.TextView;
12  import android.os.Bundle;
13  import android.widget.Toast;
14
15  import java.util.ArrayList;
16
17  public class MainActivity extends AppCompatActivity {
18
19      TextView idView;
20      EditText productBox;
21      EditText priceBox;
22
23      ListView productlist;
24      ArrayList<String> listItem;
25      ArrayAdapter adapter;
26
```

MainActivity.java

- In the onCreate() method, use findViewById() to set new variables
- Make sure the ids correspond to the ids of your elements in your activity_main.xml layout class
- viewData() displays all existing products on the page

```
27      @Override
28      protected void onCreate(Bundle savedInstanceState) {
29          super.onCreate(savedInstanceState);
30          setContentView(R.layout.activity_main);
31
32          // set variables to the ids of .xml elements
33          idView = (TextView) findViewById(R.id.productID);
34          productBox = (EditText) findViewById(R.id.productName);
35          priceBox = (EditText) findViewById(R.id.productPrice);
36          productlist = (ListView) findViewById(R.id.productListView);
37
38          MyDBHandler dbHandler = new MyDBHandler( context: this);
39          listItem = new ArrayList<>();
40
41          // call the viewData() method to display the existing products
42          viewData();
43
44          // when a product in the list is clicked, a toast is displayed with the name of the product
45          productlist.setOnItemClickListener(new AdapterView.OnItemClickListener(){
46              @Override
47              public void onItemClick(AdapterView<?> adapterView, View view, int i, long l){
48                  String text = productlist.getItemAtPosition(i).toString();
49                  Toast.makeText( context: MainActivity.this,  text: ""+text, Toast.LENGTH_SHORT).show();
50              }
51          });
52
53      }
```

MainActivity.java

- The newProduct() method adds a product to the database based on the inputs from the user
- List of products is updated

activity_main.xml

```
<Button
    android:text="Add"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/add"
    android:layout_weight="1"
    android:onClick="newProduct" />
```

```
54
55      // we use onClick for the Add button in our layout to call this method
56      public void newProduct (View view) {
57          MyDBHandler dbHandler = new MyDBHandler( context: this);
58
59          // get price from the text box
60          double price = Double.parseDouble(priceBox.getText().toString());
61
62          // get product name from the text box
63          // use the constructor from Product.java
64          Product product = new Product(productBox.getText().toString(), price);
65
66          // add to database with the addProduct() method from MyDBHandler.java
67          dbHandler.addProduct(product);
68
69          // clear the text boxes
70          productBox.setText("");
71          priceBox.setText("");
72
73          // clearing the list of products
74          // calling viewData() method to display the updated list of products
75          // this means what is displayed in the ListView is always current
76          listItem.clear();
77          viewData();
78      }
```

MainActivity.java

- The lookupProduct() method finds the details of a product based on the product name a user inputs
- If there is no product with that name, "no match found" is displayed

activity_main.xml

```
<Button
    android:text="Find"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/find"
    android:layout_weight="1"
    android:onClick="lookupProduct" />
```

```
79
80      // we use onClick for the Find button in our layout to call this method
81      public void lookupProduct (View view) {
82          MyDBHandler dbHandler = new MyDBHandler( context: this);
83
84          // get product in the database using findProduct() method from MyDBHandler.java
85          Product product = dbHandler.findProduct(productBox.getText().toString());
86
87          // if found, then display the product details
88          // if not, display "No Match Found"
89          if (product != null) {
90              idView.setText(String.valueOf(product.getID()));
91              priceBox.setText(String.valueOf(product.getPrice()));
92          } else {
93              idView.setText("No Match Found");
94          }
95      }
```

MainActivity.java

- The removeProduct() method deletes a product based on the input from the use
- List of products displayed is updated

activity_main.xml

```
<Button
    android:text="Delete"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/delete"
    android:layout_weight="1"
    android:onClick="removeProduct" />
```

```
96
97      // we use onClick for the Delete button in our layout to call this method
98      public void removeProduct (View view) {
99          MyDBHandler dbHandler = new MyDBHandler( context: this);
100
101         // delete product in the database using deleteProduct() method from MyDBHandler.java
102         boolean result = dbHandler.deleteProduct(productBox.getText().toString());
103
104         // clearing the list of products
105         // calling viewData() method to display the updated list of products
106         // this means what is displayed in the ListView is always current
107         listItem.clear();
108         viewData();
109
110         // "Record Deleted" or "No Match Found"
111         if (result) {
112             idView.setText("Record Deleted");
113             productBox.setText("");
114             priceBox.setText("");
115         }
116         else
117             idView.setText("No Match Found");
118     }
```

MainActivity.java

- The viewData() method displays all of the products in the database in the listview, item by item

```
120     private void viewData(){
121         MyDBHandler dbHandler = new MyDBHandler( context: this);
122
123         // call the viewData() method in MyDBHandler that runs the query
124         Cursor cursor = dbHandler.viewData();
125
126         // if there are no products in the table a toast says there is no data to show
127         // otherwise, while there are products, keep moving to the next product
128         if(cursor.getCount() == 0){
129             Toast.makeText( context: this,  text: "Not data to show", Toast.LENGTH_SHORT).show();
130         }else{
131             while(cursor.moveToNext()){
132                 // I just want to display the product name so I only get column 1 from the table row
133                 listItem.add(cursor.getString( i: 1));
134             }
135             // create an array adapter that provides a view for each item
136             // simple_list_item_1 is a built-in xml layout from Android
137             // I decided to use this instead of creating my own .xml file for items of the ListView
138             adapter = new ArrayAdapter<>( context: this, android.R.layout.simple_list_item_1, listItem);
139
140             // attaching the adapter to the ListView
141             productlist.setAdapter(adapter);
142         }
143     }
144 }
```

Display on New Page

When the user clicks a button, a new page is opened that displays data using a RecyclerView.

File Structure

- I will provide code for how to add, find, delete, and display products using SQLite
- I will walk you through these 3 xml files:
  - activity_main.xml
  - activity_display_product.xml
  - product_item.xml
- I will walk you through these 4 java files:
  - Product.java
  - MyDBHandler.java
  - MainActivity.java
  - ProductAdapter.java

activity_main.xml

- This is the main page that allows the use to enter data using EditText
- Contains an add, find, delete, and view all button
- Add strong id's to the buttons, edittexts, and the textview for product ID

- TableLayout
  - TableRow
    - TextView
    - TextView
  - TableRow
    - TextView
    - EditText
  - TableRow
    - TextView
    - EditText
- LinearLayout (horizontal)
  - Button
  - Button
  - Button
  - Button

activity_display_product.xml

- This file holds the RecyclerView used to display the products
- Using RecyclerView is good for displaying a dynamic amount of data
- Clicking the "View All" button in the main activity opens this page
- You can search, then drag-and-drop the RecyclerView from the element palette onto your design
  - assign an id to the item

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/idProductDisplay"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

product_item.xml

- This file specifies the format for each item within the RecyclerView
- I use CardView (line 2) to display the data in a container
  - More about CardView: Create a Card-Based Layout

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.cardview.widget.CardView
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:app="http://schemas.android.com/apk/res-auto"
5      android:layout_width="match_parent"
6      android:layout_height="wrap_content"
7      android:layout_margin="5dp"
8      android:elevation="8dp"
9      app:cardCornerRadius="4dp">
10
11     <!-- I want each item in recyclerview in activity_display_product to have a certain format -->
12     <!-- I want to show product id, name, and price so I use a layout with 3 text views -->
13     <LinearLayout
14         android:layout_width="match_parent"
15         android:layout_height="wrap_content"
16         android:layout_margin="3dp"
17         android:orientation="vertical">
18
19         <TextView
20             android:id="@+id/idProductId"
21             android:layout_width="match_parent"
22             android:layout_height="wrap_content"
23             android:padding="3dp"
24             android:text="Product Id"
25             android:textColor="@color/black" />
26
27         <TextView
28             android:id="@+id/idProductName"
29             android:layout_width="match_parent"
30             android:layout_height="wrap_content"
```

Product.java

- Create your Product.java file and declare 3 private variables
- Create your constructors

```
2
3   public class Product {
4       private int _id;
5       private String _productname;
6       private double _price;
7
8       // constructors
9       public Product() {
10      }
11      public Product(int id, String productname, double price) {
12          _id = id;
13          _productname = productname;
14          _price = price;
15      }
16      public Product(String productname, double price) {
17          _productname = productname;
18          _price = price;
19      }
```

Product.java

- Add your getters and setters to the class

```
20
21      // setters and getters
22      public void setID(int id) { _id = id; }
25      public int getID() { return _id; }
28      public void setProductName(String productname) { _productname = productname; }
31      public String getProductName() { return _productname; }
34      public void setPrice(double price) { _price = price; }
37      public double getPrice() { return _price; }
40  }
```

MyDBHandler.java

- Create your MyDBHandler.java file
- Import libraries as you go by clicking "alt+enter" over red text
- Create private variables for your database schema

```
2
3   import android.content.ContentValues;
4   import android.content.Context;
5   import android.database.Cursor;
6   import android.database.sqlite.SQLiteDatabase;
7   import android.database.sqlite.SQLiteOpenHelper;
8
9   import java.util.ArrayList;
10
11  public class MyDBHandler extends SQLiteOpenHelper {
12      //defining the schema
13      private static final int DATABASE_VERSION = 1;
14      private static final String DATABASE_NAME = "productDB.db";
15      private static final String TABLE_PRODUCTS = "products";
16      private static final String COLUMN_ID = "_id";
17      private static final String COLUMN_PRODUCTNAME = "productname";
18      private static final String COLUMN_PRICE = "price";
19
```

MyDBHandler.java

- Add a constructor

```
9   public class MyDBHandler extends SQLiteOpenHelper {
10      //defining the schema
11      private static final int DATABASE_VERSION = 1;
12      private static final String DATABASE_NAME = "productDB.db";
13      private static final String TABLE_PRODUCTS = "products";
14      private static final String COLUMN_ID = "_id";
15      private static final String COLUMN_PRODUCTNAME = "productname";
16      private static final String COLUMN_PRICE = "price";
17
18      // constructor
19      public MyDBHandler(Context context){
20          super(context, DATABASE_NAME,  factory: null, DATABASE_VERSION);
21      }
22
```

MyDBHandler.java

- The onCreate() method will create a table in our database with the proper SQL query

```
25      // create the table
26      @Override
27  @   public void onCreate(SQLiteDatabase db){
28          // CREATE TABLE TABLE_PRODUCTS (COLUMN_ID INTEGER PRIMARY KEY, COLUMN_PRODUCTNAME TEXT,
29          // COLUMN_PRICE DOUBLE)
30          String CREATE_PRODUCTS_TABLE = "CREATE TABLE " + TABLE_PRODUCTS
31                  + "(" + COLUMN_ID + " INTEGER PRIMARY KEY,"
32                  + COLUMN_PRODUCTNAME + " TEXT,"
33                  + COLUMN_PRICE + " DOUBLE" +
34                  ")";
35          db.execSQL(CREATE_PRODUCTS_TABLE);
36      }
```

MyDBHandler.java

- The onUpgrade() method will remove the table if it exists by using an SQL query

```
37
38      // deletes old tables and creates a new one
39      // change tables by incrementing the database version number
40      @Override
41      public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion){
42          db.execSQL("DROP TABLE IF EXISTS " + TABLE_PRODUCTS);
43          onCreate(db);
44      }
45
```

MyDBHandler.java

- Create an addProduct() method that adds a product to the database

```
45
46      // insert into database
47  @   public void addProduct(Product product){
48          SQLiteDatabase db = this.getWritableDatabase();
49
50          // creating an empty set of values
51          ContentValues values = new ContentValues();
52          // add values to the set
53          values.put(COLUMN_PRODUCTNAME, product.getProductName());
54          values.put(COLUMN_PRICE, product.getPrice());
55
56          // insert the set into the products table and close
57          db.insert(TABLE_PRODUCTS,  nullColumnHack: null, values);
58          db.close();
59      }
60
```

MyDBHandler.java

- Create a findProduct() method that we use when searching for a product

```
60
61      // find a product from database
62      public Product findProduct(String productname){
63          SQLiteDatabase db = this.getWritableDatabase();
64
65          // run a query to find the product with the specified product name
66          // SELECT * FROM TABLE_PRODUCTS WHERE COLUMN_PRODUCTNAME = "productname"
67          String query = "SELECT * FROM " + TABLE_PRODUCTS
68                  + " WHERE " + COLUMN_PRODUCTNAME
69                  + " = \"" + productname + "\"";
70          // passing the query
71          Cursor cursor = db.rawQuery(query,  selectionArgs: null);
72
73          Product product = new Product();
74
75          // moves cursor to the first row
76          if(cursor.moveToFirst()){
77              product.setID(Integer.parseInt(cursor.getString( i: 0)));
78              product.setProductName(cursor.getString( i: 1));
79              product.setPrice(Double.parseDouble(cursor.getString( i: 2)));
80              cursor.close();
81          }else{
82              product = null;
83          }
84          db.close();
85
86          // we return the first product in the query result with the specified product name
87          // or null if there is no product with that name
88          return product;
89      }
```

MyDBHandler.java

- Create a deleteProduct() method to delete a product

```
90
91      // delete from database
92      public boolean deleteProduct(String productname){
93          boolean result = false;
94          SQLiteDatabase db = this.getWritableDatabase();
95
96          // run a query to find the product with the specified name, then delete
97          // SELECT * FROM TABLE_PRODUCTS WHERE COLUMN_PRODUCTNAME = "productname"
98          String query = "SELECT * FROM " + TABLE_PRODUCTS
99                  + " WHERE " + COLUMN_PRODUCTNAME
100                 + " = \"" + productname + "\"";
101         // passing the query
102         Cursor cursor = db.rawQuery(query,  selectionArgs: null);
103
104         // moves cursor to the first row
105         // this deletes the first occurrence of the product with the specified name
106         if(cursor.moveToFirst()){
107             String idStr = cursor.getString( i: 0);
108             db.delete(TABLE_PRODUCTS,  whereClause: COLUMN_ID + " = " + idStr,  whereArgs: null);
109             cursor.close();
110             result = true;
111         }
112         db.close();
113
114         // if product is deleted this returns true
115         return result;
116     }
117
```

MyDBHandler.java

- Create a readProducts() method to get the id, name, and price of each product and create an ArrayList with that data

```
117
118         // read all from table
119         public ArrayList<Product> readProducts() {
120             SQLiteDatabase db = this.getReadableDatabase();
121
122             // passing the query
123             Cursor cursorProducts = db.rawQuery( sql: "SELECT * FROM " + TABLE_PRODUCTS,  selectionArgs: null);
124
125             // create arraylist for our products
126             ArrayList<Product> productArrayList = new ArrayList<>();
127
128             // while there are products in our table, keep moving to the next product
129             // we add the product id, name, and price for each new element in the arraylist
130             // column 0 is product id, column 1 is product name, column 2 is product price in our table
131             if (cursorProducts.moveToFirst()) {
132                 do {
133                     productArrayList.add(new Product(cursorProducts.getInt( i: 0),
134                             cursorProducts.getString( i: 1),
135                             cursorProducts.getDouble( i: 2)));
136                 } while (cursorProducts.moveToNext());
137             }
138             cursorProducts.close();
139             return productArrayList;
140         }
141     }
```

MainActivity.java

- In the MainActivity.java file, import libraries as you go along
- Declare 4 variables

```
2
3   import androidx.appcompat.app.AppCompatActivity;
4   import androidx.recyclerview.widget.LinearLayoutManager;
5   import androidx.recyclerview.widget.RecyclerView;
6
7   import android.os.Bundle;
8   import android.view.View;
9   import android.widget.Button;
10  import android.widget.EditText;
11  import android.widget.TextView;
12
13  import java.util.ArrayList;
14
15  public class MainActivity extends AppCompatActivity {
16
17      // creating variables for our different elements
18      TextView idView;
19      EditText productBox;
20      EditText priceBox;
21      Button viewProductsBtn;
```

MainActivity.java

- In the onCreate() method, use findViewById() to set new variables
- Make sure the ids correspond to the ids of your elements in your activity_main.xml layout

```
22
23      @Override
24      protected void onCreate(Bundle savedInstanceState) {
25          super.onCreate(savedInstanceState);
26          setContentView(R.layout.activity_main);
27
28          // set variables to the ids of .xml elements
29          idView = (TextView) findViewById(R.id.productID);
30          productBox = (EditText) findViewById(R.id.productName);
31          priceBox = (EditText) findViewById(R.id.productPrice);
32          viewProductsBtn = (Button) findViewById(R.id.idViewAllbtn);
33
34      }
35
```

MainActivity.java

- Create a newProduct() method that adds a new product when the user clicks the "Add" button
- Include onClick for the .xml button element

activity_main.xml

```
<Button
    android:text="Add"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/add"
    android:layout_weight="1"
    android:onClick="newProduct" />
```

```
35
36      // we use onClick for the Add button in our layout to call this method
37      public void newProduct (View view) {
38          MyDBHandler dbHandler = new MyDBHandler( context: this);
39
40          // get price from the text box
41          double price = Double.parseDouble(priceBox.getText().toString());
42
43          // get product name from the text box
44          // use the constructor from Product.java
45          Product product = new Product(productBox.getText().toString(), price);
46
47          // add to database with the addProduct() method from MyDBHandler.java
48          dbHandler.addProduct(product);
49
50          // clear the text boxes
51          productBox.setText("");
52          priceBox.setText("");
53      }
```

MainActivity.java

- Create a lookupProduct() method that finds a product when the user enters a name and clicks the "Find" button
- Include onClick for the .xml button element

activity_main.xml

```
<Button
    android:text="Find"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/find"
    android:layout_weight="1"
    android:onClick="lookupProduct" />
```

```
54
55      // we use onClick for the Find button in our layout to call this method
56      public void lookupProduct (View view) {
57          MyDBHandler dbHandler = new MyDBHandler( context: this);
58
59          // get product in the database using findProduct() method from MyDBHandler.java
60          Product product = dbHandler.findProduct(productBox.getText().toString());
61
62          // if found, then display the product details
63          // if not, display "No Match Found"
64          if (product != null) {
65              idView.setText(String.valueOf(product.getID()));
66              priceBox.setText(String.valueOf(product.getPrice()));
67          } else {
68              idView.setText("No Match Found");
69          }
70      }
```

MainActivity.java

- Create a removeProduct() method deletes a product when the user enters a product name and clicks "Delete"
- Include onClick for the .xml button element

activity_main.xml

```
<Button
    android:text="Delete"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/delete"
    android:layout_weight="1"
    android:onClick="removeProduct" />
```

```
71
72      // we use onClick for the Delete button in our layout to call this method
73      public void removeProduct (View view) {
74          MyDBHandler dbHandler = new MyDBHandler( context: this);
75
76          // delete product in the database using deleteProduct() method from MyDBHandler.java
77          boolean result = dbHandler.deleteProduct(productBox.getText().toString());
78
79          // "Record Deleted" or "No Match Found"
80          if (result) {
81              idView.setText("Record Deleted");
82              productBox.setText("");
83              priceBox.setText("");
84          }
85          else
86              idView.setText("No Match Found");
87      }
```

MainActivity.java

- Create a viewProducts() method shows all products when the user clicks "View All"
- Include onClick for the .xml button element

activity_main.xml

```
<Button
    android:id="@+id/idViewAllbtn"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:onClick="viewProducts"
    android:text="View All" />
```

```
88
89      // we use onClick for the View All button in our layout to call this method
90      public void viewProducts(View view) {
91          // move from one activity page to the activity_display_product page
92          setContentView(R.layout.activity_display_product);
93
94          // initializing variables
95          ArrayList<Product> productArrayList = new ArrayList<>();
96          MyDBHandler dbHandler = new MyDBHandler( context: this);
97
98          // getting the arraylist of products from MyDBHandler class
99          productArrayList = dbHandler.readProducts();
100
101         // here we pass the ArrayList to our adapter class
102         ProductAdapter productAdapter = new ProductAdapter(productArrayList,  context: this);
103
104         // my recyclerview is idProductDisplay in the activity_display_product.xml file
105         RecyclerView productsRV = findViewById(R.id.idProductDisplay);
106
107         // layout manager positions items within our recyclerview
108         // using a vertical recyclerview (other option is horizontal)
109         LinearLayoutManager linearLayoutManager = new LinearLayoutManager( context: this, RecyclerView.VERTICAL,  reverseLayout: false);
110         productsRV.setLayoutManager(linearLayoutManager);
111
112         // attaching the adapter to the recyclerview
113         productsRV.setAdapter(productAdapter);
114     }
115 }
```

ProductAdapter.java

- In the ProductAdapter.java file, import libraries as you go along
- ProductAdapter extends RecyclerView.Adapter
- Declare 2 variables

```
2
3   import android.content.Context;
4   import android.view.LayoutInflater;
5   import android.view.View;
6   import android.view.ViewGroup;
7   import android.widget.TextView;
8
9   import androidx.annotation.NonNull;
10  import androidx.recyclerview.widget.RecyclerView;
11
12  import java.util.ArrayList;
13
14  // adapter is used to get data from the table and then populate the recyclerview
15  // think of it as the "middle man" that connects the table with the layout view
16  public class ProductAdapter extends RecyclerView.Adapter<ProductAdapter.ViewHolder> {
17
18      // creating variables
19      private ArrayList<Product> productArrayList;
20      private Context context;
```

ProductAdapter.java

- Add a constructor

```
18      // creating variables
19      private ArrayList<Product> productArrayList;
20      private Context context;
21
22      // constructor
23      public ProductAdapter(ArrayList<Product> productModalArrayList, Context context) {
24          this.productArrayList = productModalArrayList;
25          this.context = context;
26      }
```

ProductAdapter.java

- This method creates a new RecyclerView.ViewHolder that describes an item view

```
27
28      @NonNull
29      @Override
30      // this is called when the recyclerview needs to represent an item
31      public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
32          // inflating our layout file for our recycler view items
33          // layout inflater is used to create a new product item for our layout
34          View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.product_item, parent,  attachToRoot: false);
35          return new ViewHolder(view);
36      }
37
```

ProductAdapter.java

- This method fetches the data and uses the data to fill the view holder's layout
  - for example it sets the text for holder.productPrice

```
38      @Override
39      // called by RecyclerView to display the data at the specified position
40      // it updates the contents of the recycler view item to reflect the specific product
41      public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
42          Product product = productArrayList.get(position);
43
44          // we get the product name using our getter from Product.java
45          // then we set the text in the corresponding TextView element in our layout
46          // process repeated for product price and id
47          holder.productName.setText(product.getProductName());
48          // we display data as text using setText() but price is a double and id is an int
49          // so we use valueOf() to represent the values as a string
50          holder.productPrice.setText(String.valueOf(product.getPrice()));
51          holder.productId.setText(String.valueOf(product.getID()));
52      }
```

ProductAdapter.java

- This method returns the length of the ArrayList that holds all of the products

```
54      @Override
55      public int getItemCount() {
56          // return the size of the ArrayList
57          return productArrayList.size();
58      }
```

ProductAdapter.java

- This inner class contains the layout for the individual item

```
59
60      // an inner class called ViewHolder provides the layout for an item
61      public class ViewHolder extends RecyclerView.ViewHolder {
62
63          // creating variables for the TextViews
64          private TextView productName, productPrice, productId;
65
66          public ViewHolder(@NonNull View itemView) {
67              super(itemView);
68              // initialize the TextViews
69              // use findViewById to find the view in our layout with the specified id
70              productName = itemView.findViewById(R.id.idProductName);
71              productPrice = itemView.findViewById(R.id.idProductPrice);
72              productId = itemView.findViewById(R.id.idProductId);
73          }
74      }
75  }
```

Option #2 Done!

TODO:

- You now know how to add, find, and delete entries using SQLite
- Now explore how to display a list of all the products in the database on your own
  - You can use method #1 or #2 that I presented, or come up with your own way
  - Display database content either on the same page or create a second page
  - Tutorial 6 gives a Firebase example
- Customize the application as much as you'd like (both in terms of design and feature implementation)! I provided the bare-bones.
- Lots of resources online to guide you, and you can email me for help to guide you in the right direction
- This will be useful code to reuse in your term project

Expectations

- All group members must participate in completing the lab.
- Only one member needs to submit the application via Brightspace.
- Your application should not crash when performing the expected objectives.
- Mark breakdown:
  - Full marks are awarded if all lab objectives are met
  - Part marks are awarded if some lab objectives are met
  - No marks are awarded if you do not submit the lab or submit past the deadline
- Each member must submit the Group Evaluation doc