Lab assignment 5

Due date: Tuesday, June 14

40 points

Team of 3 to 4 students

Divide the team into two subgroups with two team members per group.

The group works on the lab assignment and can discuss the code with another group. Each group demonstrates the lab to each other. The team then upload the solution from one of the subgroups.

Write the contribution to the lab assignment, the contribution percentage (maximum of 100%) on the comment box for each team member.

All the team members must agree on the percentage. If the team members have 50% and the lab assignment grade is 40/40, they get only 20 points. The instructor will review the paper summarizing and validate the percentage of each team member. If you have problems with team members, please let me know.

Problem

In this lab assignment you will build a Candy Store app using menus and SQLite.

Design

Version 0

You start an app using the Basic Activity template, Android Studio generates two layout XML files and one menu XML file: activity_main.xml, content_main.xml, and menu_main.xml. This is different from the Empty Activity template, which only generates the activity_main.xml file.

The activity_main.xml file, uses a `CoordinatorLayout` to arrange the elements. A `CoordinatorLayout` is typically used as a top-level container for an app and as a container for specific interactions with one or more child Views.

It includes three elements:

- An `AppBarLayout` , itself including a `ToolBar`. This is the action bar and this is where the menu items go.
- The View defined by content_main.xml , a `RelativeLayout` with a `TextView` inside it.
- A `FloatingActionButton` positioned at the bottom right and whose icon is a standard email icon from the Android icon library. In this app, you do not want that functionality, so you delete it.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.coordinatorlayout.widget.CoordinatorLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <com.google.android.material.appbar.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/Theme.CandyStore.AppBarOverlay">

        <androidx.appcompat.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            app:popupTheme="@style/Theme.CandyStore.PopupOverlay" />

    </com.google.android.material.appbar.AppBarLayout>

    <include layout="@layout/content_main" />

    <com.google.android.material.floatingactionbutton.FloatingActionButton
        android:id="@+id/fab"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|end"
        android:layout_margin="@dimen/fab_margin"
        app:srcCompat="@android:drawable/ic_dialog_email" />

</androidx.coordinatorlayout.widget.CoordinatorLayout>
```

Inside the `onCreate` method of the `MainActivity` class, there is existing code to handle user interaction with the floating action button. Since you do not use the floating action button for this app, you delete that code inside the `MainActivity` class.

MainActivity.java

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        FloatingActionButton fab = findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Snackbar.make(view, "Replace with your own action",
Snackbar.LENGTH_LONG)
                        .setAction("Action", null).show();
            }
        });
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }
}
```

The menu_main.xml file defines a menu with one item. Menu items are shown in the action bar, starting from the right. Hoyouver, if you run the skeleton app, no menu item shows. This is because the only menu item has the value `never` for the attribute `app:showAsAction.`

menu_main.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="com.android.example.candystore.MainActivity">
    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:title="@string/action_settings"
        app:showAsAction="never" />
</menu>
```

Updated menu_main.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="com.android.example.candystore.MainActivity">
    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:title="@string/action_settings"
        app:showAsAction="never" />
    <item android:title="@string/add" app:showAsAction="ifRoom"
android:id="@+id/action_add"/>
<item android:title="@string/add" app:showAsAction="ifRoom"
android:id="@+id/action_add" android:icon="@android:drawable/ic_menu_add"/>

//YOUR CODE


</menu>
```

MainActivity class

You deleted the code related to the floating action button inside the `onCreate` method. `onCreate`, you get a reference to the `Toolbar` defined in activity_main.xml. You call the `setSupportActionBar` , to set it as the action bar for this app.

The `onCreateOptionsMenu` method inflates menu_main.xml in order to create a menu and places that menu in the toolbar. The `onOptionsItemSelected` method is called when the user selects a menu item. Its parameter is a `MenuItem` reference to the menu item selected. You retrieve its id and use a `switch` statement to compare it to the ids of the various menu items defined in menu_main.xml and output what action is selected to Logcat.

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

    @Override
```

```java
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();
        switch ( id ) {
            case R.id.action_add:
                Log.w( "MainActivity", "Add selected" );
                return true;
        //YOUR CODE

        }

    }
}
```

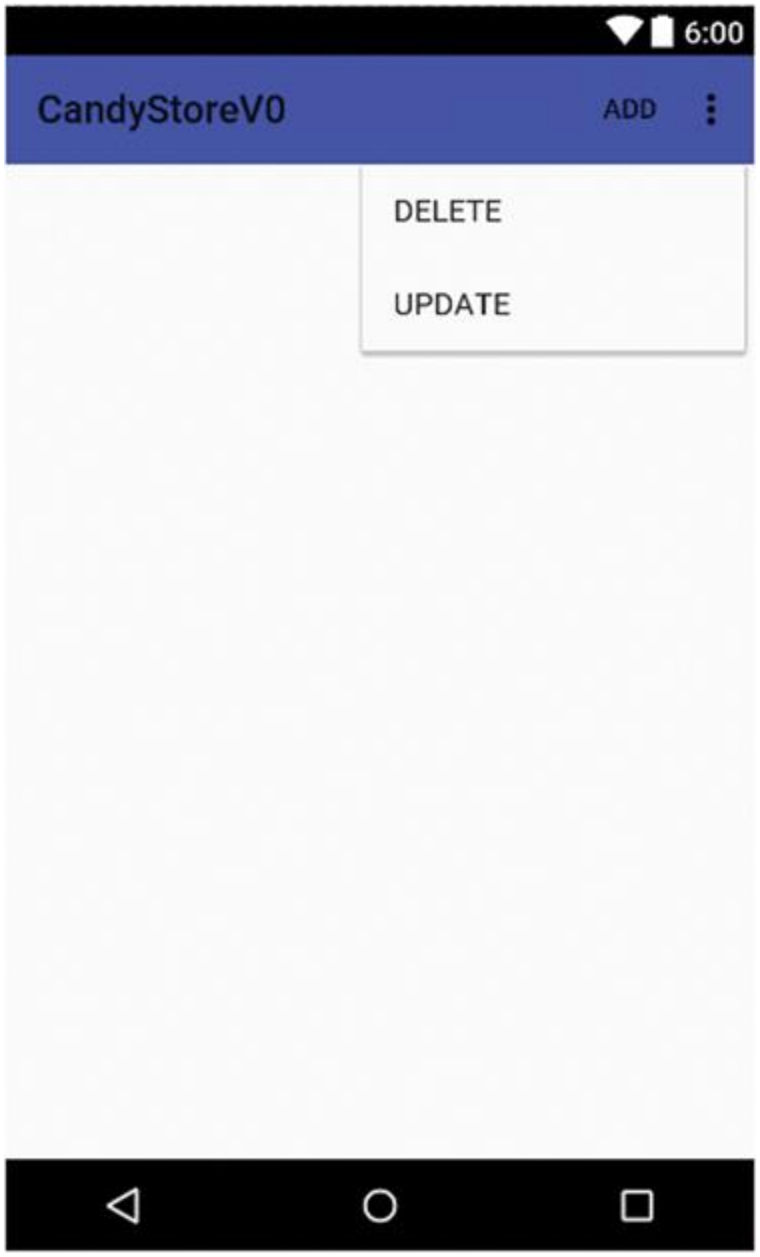
fragment_first.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".FirstFragment">


</androidx.constraintlayout.widget.ConstraintLayout>
```

FirstFragment class

```java
public class FirstFragment extends Fragment {

    @Override
    public View onCreateView(
            LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState
    ) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_first, container, false);
    }

    public void onViewCreated(@NonNull View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);


    }
}
```

Run the app inside the emulator.



Icon Items – Version 1

In Version 1, you use icons instead of `Strings` for the menu items and you provide an activity with its layout for the add menu item. To include an icon for a menu item, you use the `android:icon` XML attribute. You can use existing icons available in the Android library or create our own icons. Existing icons can be referenced using the following syntax and pattern:

```
@android:drawable/name_of_icon
```

For add, edit (update), and delete, the names of the icon resources are `ic_menu_add`, `ic_menu_edit`, and `ic_menu_delete`.

Updated menu_main.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="com.android.example.candystore.MainActivity">
    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:title="@string/action_settings"
        app:showAsAction="never" />
    <item android:title="@string/add" app:showAsAction="ifRoom"
android:id="@+id/action_add" android:icon="@android:drawable/ic_menu_add"/>
        //YOUR CODE

</menu>
```

When the user clicks on the add icon, you want to enable the user to add a candy to the database. You keep things simple and our database only contains one table storing candies: a candy has an id, a name, and a price. Ids are expected to be integers starting at 0 and being automatically incremented by 1 as you add candies. Thus, our second screen, where the user can add a candy to the database, only includes two widgets for user input: one for the name and one for the price of the candy.

activity_insert.xml.

```
<?xml version="1.0" encoding="UTF-8"?>

    <RelativeLayout android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingTop="@dimen/activity_vertical_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:layout_height="match_parent" android:layout_width="match_parent"
xmlns:android="http://schemas.android.com/apk/res/android">

    <TextView android:layout_height="wrap_content"
```

```
android:layout_width="wrap_content" android:text="@string/label_name"
android:layout_marginTop="50dp" android:id="@+id/label_name"/>

    <EditText android:layout_height="wrap_content"
android:layout_width="wrap_content" android:id="@+id/input_name"
android:orientation="horizontal" android:layout_marginLeft="50dp"
android:layout_alignBottom="@+id/label_name"
android:layout_toRightOf="@+id/label_name"/>

    <TextView android:layout_height="wrap_content"
android:layout_width="wrap_content" android:text="@string/label_price"
android:layout_marginTop="50dp" android:id="@+id/label_price"
android:layout_below="@+id/label_name"/>
        //YOUR CODE


    <Button android:layout_height="wrap_content" android:layout_width="wrap_content"
android:text="@string/button_add" android:layout_marginTop="50dp"
android:id="@+id/button_add" android:layout_below="@+id/label_price"
android:onClick="insert" android:layout_centerHorizontal="true"/>
        //YOUR CODE

</RelativeLayout>
```

Updated MainActivity class

You update the `MainActivity` class, so that when the user clicks on the Add icon, you start an `InsertActivity`. When the user clicks on the ADD icon, you create an `Intent` for an `InsertActivity` and start that activity.

```
case R.id.action_add:
    Log.w( "MainActivity", "Add selected" );
        //YOUR CODE
```

InsertActivity class

You create the `InsertActivity` class to control the second screen, defined in activity_insert.xml. You inflate the XML layout defined in the insert_activity.xml file, and you include the `insert` and `goBack` methods. The `goBack` method executes when the user clicks on the BACK button; it pops the current activity off the activity stack, returning the app to the previous activity (i.e., the first screen). Inside the `insert` method, you retrieve the user input at lines, plan to insert a new candy in the database using user input, and clear the two `EditTexts` in case the user wants to add another candy.

```
public class InsertActivity extends AppCompatActivity {
    public void onCreate( Bundle savedInstanceState ) {
        super.onCreate( savedInstanceState );
        //YOUR CODE
        setContentView( R.layout.activity_insert );
    }
    public void insert( View v ) {
        // Retrieve name and price
        //YOUR CODE

        // insert new candy in database

        // clear data
        nameEditText.setText( "" );
        priceEditText.setText( "" );
    }

    public void goBack( View v ) {
        //YOUR CODE

    }
}
```

You add an `activity` element in the AndroidManifest.xml file. You also allow both activities to run in vertical orientation only.

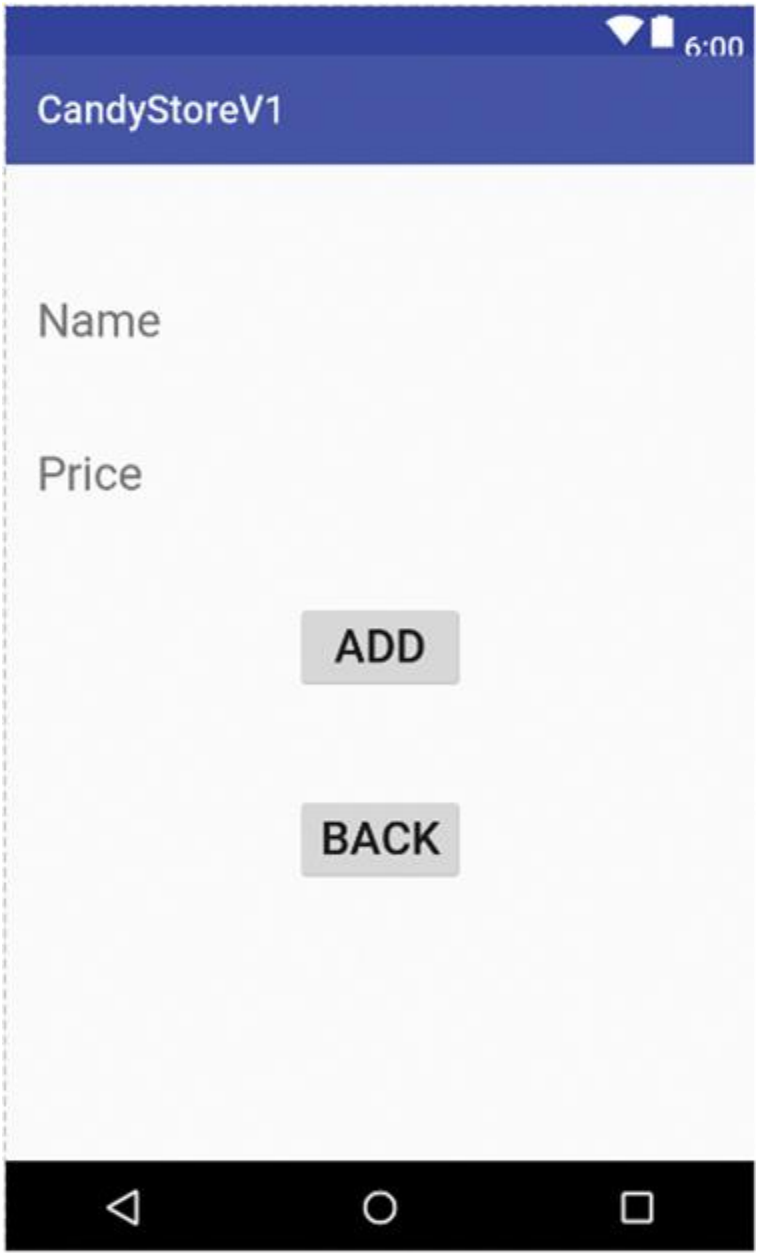
AndroidManifest.xml

```
<activity
    android:name=".MainActivity"
    android:label="@string/app_name"
    android:theme="@style/Theme.CandyStore.NoActionBar">
    android:screenOrientation="portrait"
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
        //YOUR CODE
```

Finally, in themes.xml, you specify that the text size in the various views is 24sp,

```
<style name="Theme.CandyStore"
parent="Theme.MaterialComponents.DayNight.DarkActionBar">
            //YOUR CODE
```

Run the app inside the emulator.



SQLite: Creating a Database, a Table, and Inserting Data – Version 2

In Version 2, we create a database, create a table to store candies, and insert data in that table. The android.database.sqlite package contains classes and interfaces to manage databases, execute SQL queries, process their results, etc.

Selected Classes of the `android.database.sqlite` package

Class	Description
SQLiteOpenHelper	Extend this abstract class to manage a database and its version . We must override the onCreate and onUpgrade methods.
SQLiteDatabase	Includes methods to execute SQL statements
Cursor	Encapsulates a table returned by a select SQL query.

Candy class

```
public class Candy {
    private int id;
    private String name;
    private double price;

    public Candy( int newId, String newName, double newPrice ) {
        setId( newId );
        setName( newName );
        setPrice( newPrice );
    }

    public void setId( int newId ) {
        id = newId;
    }

    public void setName( String newName ) {
        name = newName;
    }

    public void setPrice( double newPrice ) {
        if( newPrice >= 0.0 )
            price = newPrice;
    }

    public int getId( ) {
        return id;
    }
    public String getName( ) {
        return name;
    }

    public double getPrice( ) {
        return price;
    }

    public String toString( ) {
        return id + "; " + name + "; " + price;
    }
}
```

As part of the Model, we include a class containing methods to execute various basic SQL statements. When executing an insert, update, or delete statement, we can use the `execSQL` method of the `SQLiteDatabase` class. When executing a select statement, we can use the `rawQuery` method of the `SQLiteDatabase` class to execute it, and the methods of the `Cursor` class to process the results.

Selected Methods of the `SQLiteDatabase` class

Method	Description
void execSQL(String sql)	Executes sql, a SQL query that does not return data. Can be used for create, insert, update, delete, but not for select queries.
Cursor rawQuery(String sql, String [ ] selectionArgs)	Executes sql and returns a Cursor; selectionArgs can be provided to match ?s in the where clause of the query.

Selected Methods of the `Cursor` class

Method	Description
boolean moveToNext( )	Move this Cursor to the next row when processing results.
DataType getDataType(int column)	Returns the value for the current row at column index column. DataType can be a basic data type, String or Blob.

Selected Methods of the `SQLiteOpenHelper` class

Method	Description
SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int newVersion)	Constructor: creates a SQLiteOpenHelper object. Name is the name of the database. Factory can be used to create Cursor objects, use null for default.
abstract void onCreate(SQLiteDatabase db)	Called when the database is created for the first time. We must implement that method.
abstract void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)	Called when the database needs to be upgraded. We must implement that method.
SQLiteDatabase getWritableDatabase( )	Creates and/or opens a database that we will use for reading and writing. Triggers a call to onCreate the first time it is called. Returns a SQLiteDatabase reference, which we can use to perform SQL operations.

DatabaseManager class

```
public class DatabaseManager extends SQLiteOpenHelper {
    private static final String DATABASE_NAME = "candyDB";
    private static final int DATABASE_VERSION = 1;
    private static final String TABLE_CANDY = "candy";
    private static final String ID = "id";
    private static final String NAME = "name";
    private static final String PRICE = "price";

    public DatabaseManager( Context context ) {
        super( context, DATABASE_NAME, null, DATABASE_VERSION );
    }

    public void onCreate( SQLiteDatabase db ) {
        // build sql create statement
        String sqlCreate = "create table " + TABLE_CANDY + "( " + ID;
        sqlCreate += " integer primary key autoincrement, " + NAME;
        sqlCreate += " text, " + PRICE + " real )" ;

        db.execSQL( sqlCreate );
    }

    public void onUpgrade( SQLiteDatabase db,
                           int oldVersion, int newVersion ) {
        // Drop old table if it exists
        db.execSQL( "drop table if exists " + TABLE_CANDY );
        // Re-create tables
        onCreate( db );
    }

    public void insert( Candy candy ) {
        SQLiteDatabase db = this._____________( );
        String sqlInsert = "insert into " + TABLE_CANDY;
        sqlInsert += " values( null, '" + ________________;
        sqlInsert += "', '" + ______________ + "' )";

        db.execSQL( sqlInsert );
        db.close( );
    }

    public void deleteById( int id ) {
        SQLiteDatabase db = this._______________( );
        String sqlDelete = "delete from " + TABLE_CANDY;
        sqlDelete += " where " + ID + " = " + _____________;

        db.execSQL( sqlDelete );
        db.close( );
    }

    public void updateById( int id, String name, double price ) {
        SQLiteDatabase db = this.getWritableDatabase();

        String sqlUpdate = "update " + TABLE_CANDY;
        sqlUpdate += " set " + NAME + " = '" + _____ + "', ";
        sqlUpdate += PRICE + " = '" + _____ + "'";
        sqlUpdate += " where " + ID + " = " + _____;

        db.execSQL( sqlUpdate );
        db.close( );
    }

    public ArrayList<Candy> selectAll( ) {
        String sqlQuery = "select * from " + _____________;

        SQLiteDatabase db = this._________________( );
        Cursor cursor = db._________________( sqlQuery, null );

        ArrayList<Candy> candies = new ArrayList<Candy>( );
        while( cursor.____________ ) {
            Candy currentCandy
                    = new Candy( Integer.parseInt( cursor.getString( _____) ),
                    _________________, cursor.getDouble( ____ ) );
```

```
            candies.add( currentCandy );
        }
        db.close( );
        return candies;
    }

    public Candy selectById( int id ) {
        String sqlQuery = "select * from " + __________;
        sqlQuery += " where " + ID + " = " + ___________;

        SQLiteDatabase db = this.getWritableDatabase( );
        Cursor cursor = db.____________( sqlQuery, null );

        Candy candy = null;
        if( cursor.______________________ )
            candy = new Candy( Integer.parseInt( cursor.getString( _____) ),
                    _________________, cursor.getDouble( ____ ) );

        return candy;
    }
}
```

The Model is ready, we can use it in the Controller, the `InsertActivity` class, in order to add a candy in our database.

We provide feedback to the user showing a `Toast`. A `Toast` is a temporary pop-up that can be used to provide visual feedback on an operation. It automatically disappears after a short time. The `Toast` class is in the `android.widget` package.

Selected Methods and Constants of the `Toast` class

Method	Description
static Toast makeText(Context context, CharSequence text, int duration)	Creates a Toast within context with content text and a duration specified by duration.
void show( )	Shows this Toast.
Constant	**Value**
LENGTH_SHORT	0; use this constant for a Toast of approximately 3 seconds.

Method	Description
LENGTH_LONG	1; use this constant for a Toast of approximately 5 seconds.

InsertActivity class

```
public class InsertActivity extends AppCompatActivity {
    private _______________dbManager;
    public void onCreate( Bundle savedInstanceState ) {
        super.onCreate( savedInstanceState );
        dbManager = new ___________________( this );
        setContentView( R.layout.activity_insert );
    }

    public void insert( View v ) {
        // Retrieve name and price
        EditText nameEditText = ______________________;
        EditText priceEditText = _____________________;
        String name = nameEditText.getText( ).toString( );
        String priceString = priceEditText.getText( ).toString( );

        // insert new candy in database
        try {
            double price = Double.parseDouble( priceString );
            Candy candy = new Candy( 0, name, price );
            dbManager.____________( candy );
            Toast.makeText( this, "Candy added", Toast.LENGTH_SHORT )._________;
        } catch( NumberFormatException nfe ) {
            Toast.makeText( this, "Price error", Toast.LENGTH_LONG ).___________;
        }
        ArrayList<Candy> candies = dbManager.__________;
        for( Candy candy : __________________ )
            Log.w("MainActivity", "candy = " + candy._________________( ) );
        // clear data
        nameEditText.setText( "" );
        priceEditText.setText( "" );
    }

    public void goBack( View v ) {
        _____________________
    }
}
```

When you run the app, enter some data and click on the ADD icon, the `Toast` message appears. If we want to check that a new row is added to the `candy` table, we can call the `selectAll` method of the `DatabaseManager` class and loop through the resulting `ArrayList` of `Candy` objects. We can write these statements at the end of the `insert` method and check the output in Logcat:

ArrayList<Candy> candies = dbManager.selectAll( );

```
for( Candy candy : candies )

  Log.w( ”MainActivity”, ”candy = ” + candy.toString( ) );
```

Deleting Data: Candy Store App, Version 3

In Version 3, you enable the user to delete a candy from the database. To implement this functionality, you need to do the following:

- Create a delete activity.
- Modify `MainActivity` so that when the user clicks on the DELETE icon, the user goes to the delete activity.
- Add an activity element in AndroidManifest.xml for the delete activity.

Selected Methods of the `RadioGroup` class

Method	Description
void setOnCheckedChangeListener(RadioGroup. OnCheckedChangeListener listener)	Registers listener on this RadioGroup. When the selected radio button changes in this group, the onCheckedChange of method of the RadioGroup. OnCheckedChangeListener interface is called.

MainActivity class

```
    case R.id.action_delete:
    //YOUR CODE

    Log.w( "MainActivity", "Delete selected" );
    return true;
```

AndroidManifest.xml

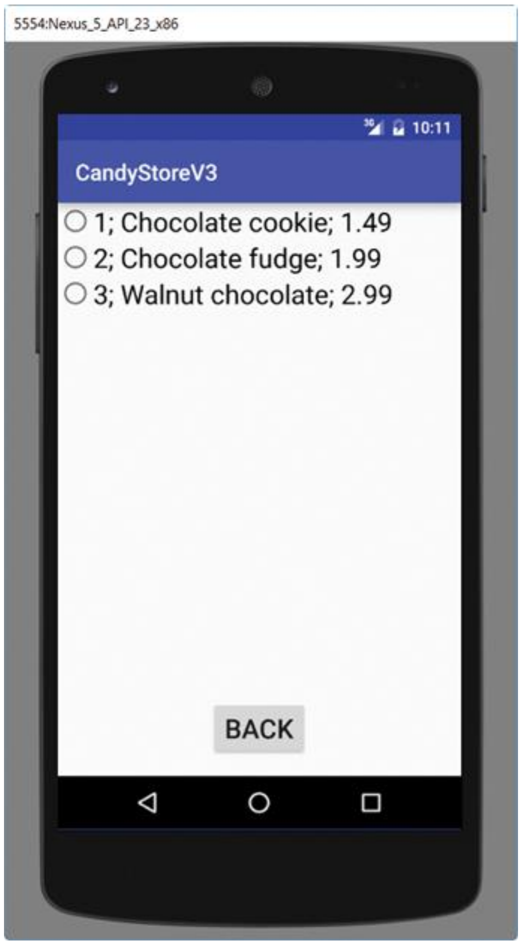
```
//YOUR CODE
```

DeleteActivity class

```
public class DeleteActivity extends AppCompatActivity {
    private DatabaseManager dbManager;

    public void onCreate( Bundle savedInstanceState ) {
        super.onCreate( savedInstanceState );
        dbManager = new DatabaseManager( this );
        updateView( );
    }

    // Build a View dynamically with all the candies
    public void updateView( ) {
        ArrayList<Candy> candies = dbManager.____________
        RelativeLayout layout = new RelativeLayout( this );
        ScrollView scrollView = new ScrollView( this );
        RadioGroup group = new ____________( this );
        for ( Candy candy : ______________ ) {
            RadioButton rb = new RadioButton( this );
            rb.setId( _____________) );
            rb.setText(________________ );
            group.__________( _____ );
        }
        // set up event handling
        RadioButtonHandler rbh = new RadioButtonHandler( );
        group.________________________(rbh);

        // create a back button
        Button backButton = new Button( this );
        backButton.setText( R.string.button_back );

        backButton.setOnClickListener( new View.OnClickListener( ) {
            public void onClick(View v) {
                ____________________________________
            }
        });

        scrollView._____________(_________);
        layout.____________( scrollView );

        // add back button at bottom
        RelativeLayout.LayoutParams params
                = new RelativeLayout.LayoutParams(
                RelativeLayout.LayoutParams.WRAP_CONTENT,
                RelativeLayout.LayoutParams.WRAP_CONTENT );
        params.addRule( RelativeLayout.ALIGN_PARENT_BOTTOM );
        params.addRule( RelativeLayout.CENTER_HORIZONTAL );
        params.setMargins( 0, 0, 0, 50 );
        layout.____________( _____________, params );

        setContentView( ___________ );
    }
```

```
  private class RadioButtonHandler
            implements RadioGroup.______________________ {
        public void onCheckedChanged( _____ group, int checkedId ) {
            // delete candy from database
            dbManager.______________( checkedId );
            Toast.makeText( DeleteActivity.this, "Candy deleted",
                    Toast.LENGTH_SHORT ).show( );

            // update screen
            updateView( );
        }
    }
}
```

The figure below shows the delete screen of the app after the user selects the DELETE icon. All candies are displayed as radio buttons. Clicking on one deletes it and refreshes the screen. Note that the database in Version 3 is different from the database in Version 2. Thus, if we run Version 3 without inserting candies, the `candy` table is empty and no candy will show when we click on the DELETE icon.



1; Chocolate cookie; 1.49
2; Chocolate fudge; 1.99
3; Walnut chocolate; 2.99

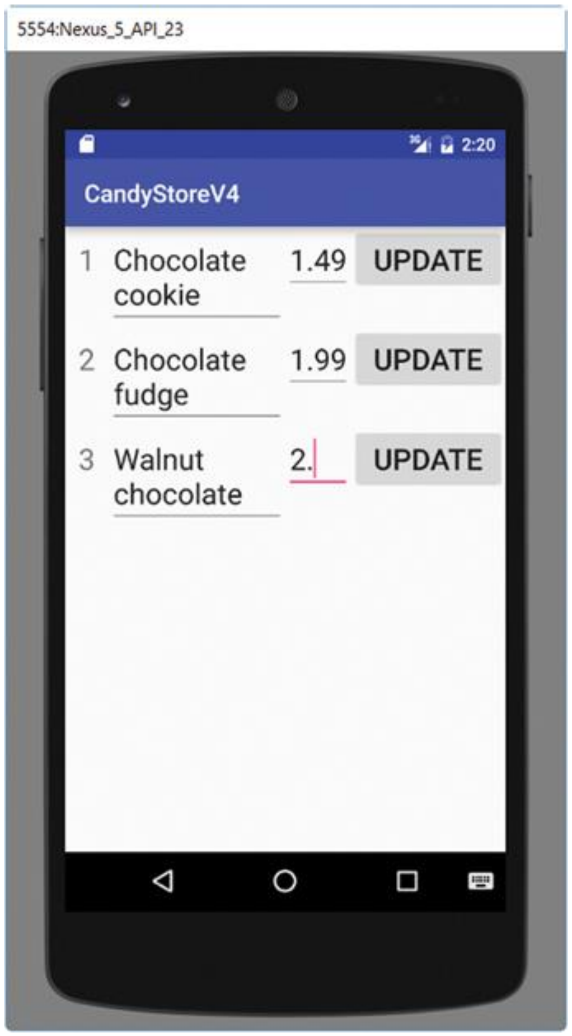
Updating Data: Candy Store App, Version 4

In Version 4, we enable the user to update the name and price of a candy from the database. To implement this functionality, we need to do the following:

- Create an update activity.
- Modify `MainActivity` so that when the user clicks on the UPDATE icon, the user goes to the update activity.
- Add an activity element in AndroidManifest.xml for the update activity.

We distribute the width of the screen among the four components across it as follows:

- 10% for the id,
- 40% for the name,
- 15% for the price, and
- 35% for the button.



The update screen of the Candy Store app, version 4

MainActivty class Update

AndroidManifest.xml Update

UpdateActivity class

```
public class UpdateActivity extends AppCompatActivity {
    ___________________ dbManager;

    public void onCreate( Bundle savedInstanceState ) {
        super.onCreate( savedInstanceState );
        dbManager = new _________________( this );

        ____________________
    }

    // Build a View dynamically with all the candies
    public void updateView( ) {
        ArrayList<Candy> candies = dbManager.selectAll( );
        if( candies.size( ) > 0 ) {
            // create ScrollView and GridLayout
            ScrollView scrollView = new ___________( this );
            GridLayout grid = new GridLayout( this );
            grid.setRowCount( ______________ );
            grid.setColumnCount( 4 );

            // create arrays of components
            TextView[] ids = new TextView[___________________];
            EditText[][] namesAndPrices = new EditText[candies.size( )][2];
            Button[] buttons = new Button[candies.size( )];
            ButtonHandler bh = new ButtonHandler( );

            // retrieve width of screen
            Point size = new Point( );
            getWindowManager( ).getDefaultDisplay( ).getSize( size );
            int width = size.x;

            int i = 0;

            for ( Candy candy : candies ) {
                // create the TextView for the candy's id
                ids[i] = new TextView( this );
                ids[i].setGravity( Gravity.CENTER );
                ids[i].setText( "" + candy.getId( ) );

                // create the two EditTexts for the candy's name and price
                namesAndPrices[i][0] = new EditText( this );
                namesAndPrices[i][1] = new EditText( this );
                namesAndPrices[i][0].setText( ____________ );
                namesAndPrices[i][1].setText( "" +_____________________ );
                namesAndPrices[i][1]
                        .setInputType( InputType.TYPE_CLASS_NUMBER );
                namesAndPrices[i][0].setId( 10 * ________________ );
                namesAndPrices[i][1].setId( 10 * candy.getId( ) + 1 );

                // create the button
                buttons[i] = new Button( this );
                buttons[i].setText( __________ );
                buttons[i].setId(________________ );

                // set up event handling
                buttons[i].setOnClickListener( bh );

                // add the elements to grid
                grid.addView( ids[i], width / 10,
                        ViewGroup.LayoutParams.WRAP_CONTENT );
                grid.addView( namesAndPrices[i][0], ( int ) ( width * ___ ),
                        ViewGroup.LayoutParams.WRAP_CONTENT );
                grid.addView( namesAndPrices[i][1], ( int ) ( width * _____ ),
                        ViewGroup.LayoutParams.WRAP_CONTENT );
                grid.addView( buttons[i], ( int ) ( width * _______ ),
                        ViewGroup.LayoutParams.WRAP_CONTENT );

                i++;
            }
            scrollView._______( _______ );
            setContentView( ________ );
        }
    }

    private class ButtonHandler implements View.OnClickListener {
        public void onClick( View v ) {
            // retrieve name and price of the candy
            int candyId = v.getId( );
            EditText nameET = ( EditText ) findViewById( ________________ );
            EditText priceET = ( EditText ) findViewById( 10 * candyId + 1 );
            String name _______________________
            String priceString = _______________________________

            // update candy in database
            try {
                double price = Double.parseDouble( priceString );
                dbManager.______________( candyId, name, price );
                Toast.makeText( UpdateActivity.this, "Candy updated",
                        Toast.LENGTH_SHORT )._____________________;

                // update screen
                updateView( );
            } catch( NumberFormatException nfe ) {
                Toast.makeText( UpdateActivity.this,
                        "Price error", Toast.LENGTH_LONG ). _____________;
            }
        }
    }
}
```

Run your app. Clicking on an UPDATE button updates the corresponding candy and refreshes the screen. The user is updating the price of the walnut chocolate candy.

Running the Cash Register – Version 5

In Version 5, we enable the user to run the app as a cash register using the first screen. We provide a grid of buttons, one button per candy. The store employee can use the buttons to compute the total amount of money due by a customer who buys candies. Each time the user clicks on a button, we add the price of the corresponding candy to the total for that customer and show the total in a Toast.

When the user clicks on a button, we need to access the price of the candy associated with that button. An easy way to solve that problem is to create a new class, CandyButton, which extends the Button class and has a Candy instance variable.

```
public class CandyButton extends Button {
    private Candy candy;

    public CandyButton(Context context, Candy newCandy ) {
        super( context );
        candy = newCandy;
    }

    public double getPrice( ) {
        return candy.getPrice( );
    }
}
```

more than one customer. Thus, we need to have a way to reset the running total to `0` whenever we are done with the current customer and are ready for the next one. An easy way to do that is to provide an additional item in the menu and reset the total to `0` when the user clicks on that item.

We add a `String` named `reset` with value `RESET` in the strings.xml file. We also add our own icon, stored in the ic_reset.png file, which we place in the drawable directory. The file is attached to the lab assignment.

menu_main.xml

```
<item android:title="@string/reset" app:showAsAction="ifRoom"
android:icon="@drawable/ic_reset" android:id="@+id/action_reset"/>
<item android:title="@string/add" app:showAsAction="ifRoom"
android:id="@+id/action_add" android:icon="@android:drawable/ic_menu_add"/>
```

We need to place our buttons inside a ScrollView because we do not know how many there are each time we run the app . An easy way to do this is to replace the RelativeLayout inside content_main.xml with a ScrollView element. We give it an id so that we can retrieve it inside the `MainActivity` class. We also eliminate the padding inside the `ScrollView`.

content_main.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ScrollView
    android:id="___________________"
    tools:showIn="@layout/activity_main"
    tools:context="com.android.example.candystore.MainActivity"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android">
</ScrollView>
```

The MainActivity classhas four instance variables: A DatabaseManager, dbManager, so that we can query the database to retrieve the candies; total, a double, to keep track of the running total for the current customer; scrollView, a reference to the ScrollView defined in content_main.xml; and buttonWidth, the width of each button. You instantiate dbManager, initialize total to 0.0, instantiate scrollView, and calculate buttonWidth. You want to size the buttons so that their width is half the width of the screen. Thus, you assign half the width of the screen to the variable buttonWidth. When the user has finished processing the current customer and clicks on the reset icon , we reset the running total to 0.0 so that the app is ready to compute the total for the next customer.

The updateView method creates the GUI and sets up event handling. We call updateView inside onCreate when the app starts and also inside onResume when the user comes back from a secondary activity such as add, delete, or update. Indeed, the contents of the candy table may have changed when the user comes back from a secondary activity so it is necessary to update the first screen by calling updateView. The onResume method is automatically called when the user comes back to this activity.

The updateView method has similarities with the updateView method in the UpdateActivity class. It is possible that the user added, deleted, or updated one or more candies before coming back to this View. Thus, we first remove all the buttons inside scrollView before rebuilding scrollView. We create a grid of buttons dynamically, one button per candy. We include two buttons per row . We size the number of rows to guarantee to have enough room for all the buttons . As in the delete and update activities, we put the grid inside a ScrollView so that we have automatic scrolling if needed.

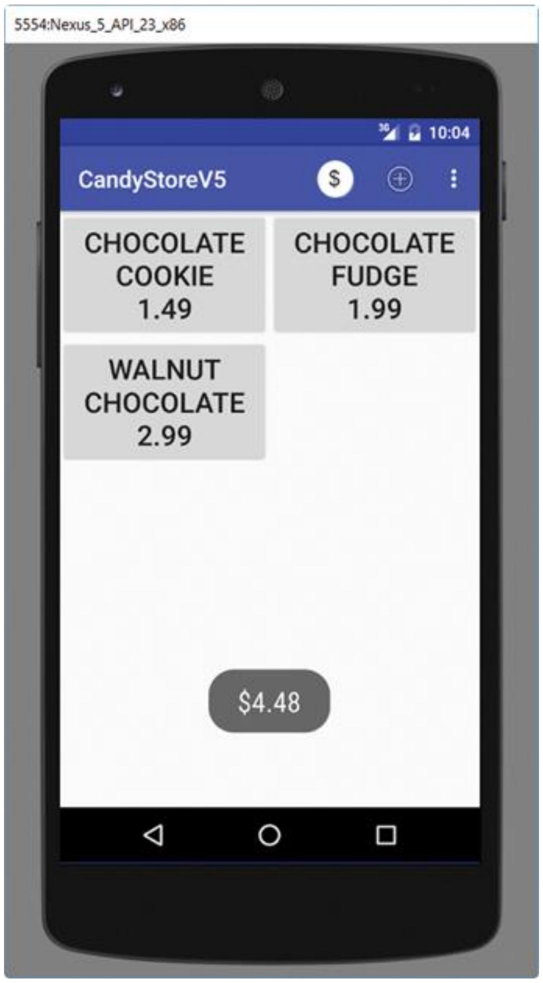
Inside each button, we put the name and the price of the candy, each on one line. Long candy names may require two lines of their own. We could set the font size of each button dynamically to make each candy name fit on one line. We set up event handling at lines.

The onClick method of the ButtonHandler class is called when the user clicks on a button. We update total and show a Toast whose value is formatted as currency. The benefit of using the CandyButton class is that we cast the View parameter v, which represents the button clicked, to a CandyButton and call getPrice to retrieve the price of the corresponding candy.

MainActivity class

```
public class MainActivity extends AppCompatActivity {
    private DatabaseManager dbManager;
    private double total;
    private ScrollView scrollView;
    private int buttonWidth;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        //YOUR CODE - Create DatabaseManager object, scrollview, initialize total to
        //0,stet the buttonWidth to half the width of the screen. Call updateView
        //which you create it later.
        //YOUR CODE
    }
    protected void onResume( ) {
        super.onResume( );
        //YOUR CODE
        ______________________
    }
    public void updateView( ) {
        ArrayList<Candy> candies = _______________________;
        if( candies.size( ) > 0 ) {
            // remove subviews inside scrollView if necessary
            scrollView.removeAllViewsInLayout( );

            // set up the grid layout
            GridLayout grid = new GridLayout( this );
            grid.___________( ( candies.size( ) + 1 ) / 2 );
            grid.____________( 2 );

            // create array of buttons, 2 per row
            CandyButton [] buttons = new CandyButton[______________];
            ButtonHandler bh = ________________________________;

            // fill the grid
            int i = 0;
            for ( Candy candy : candies ) {
                // create the button
                buttons[i] = new CandyButton( this, __________________ );
                buttons[i].setText( _______________
                        + "\n" + _____________ );

                // set up event handling
                buttons[i]._______________________;

                // add the button to grid
                grid.addView( ______________, ____________________,
                        GridLayout.LayoutParams.WRAP_CONTENT );
                i++;
            }
            scrollView.addView( ______________ );
        }
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();
        switch ( id ) {
            case R.id.action_add:
                Log.w( "MainActivity", "Add selected" );
                Intent insertIntent = new Intent( this, InsertActivity.class );
                this.startActivity( insertIntent );
                return true;
            case R.id.action_delete:
                Intent deleteIntent = new Intent( this, DeleteActivity.class );
                this.startActivity( deleteIntent );
                Log.w( "MainActivity", "Delete selected" );
                return true;
            case R.id.action_update:
                Intent updateIntent = new Intent( this, UpdateActivity.class );
                this.startActivity( updateIntent );
                Log.w( "MainActivity", "Update selected" );
                return true;
            //YOUR CODE

            default:
                return super.onOptionsItemSelected( item );
        }

    }
    private class ButtonHandler implements ____________________ {
        public void onClick( View v ) {
            // retrieve price of the candy and add it to total
            total += ( ( ____________ ) _________ ).getPrice( );
            String pay =
                    NumberFormat.getCurrencyInstance( ).format( _______ );
            Toast.makeText( MainActivity.this, pay,
                    Toast.LENGTH_LONG ).show( );
        }
    }
}
```

The figure below shows all the candies and a current running total of $4.48 after the user selected chocolate cookie and walnut chocolate. Note the three vertical dots showing on the right of the menu because there is not enough space for all the icons. Touching the three dots opens a submenu showing the missing items.



Grading:

1. Submit the project in zip file (extension zip)
2. A pdf document that contains the content of the following files: Candy class, Database Manager class, InsertActivity clas, MainActivity class, AndroidManifest.xml, DeleteActivity class, UpdateActivity class, CandyButton class, menu_main.xml, and content_main.xml.
3. A zoom link or YouTube that demonstrate the user of the CandyStore app (less than 5 minutes). Write the link on the comment section in the dropbox.
   Your video should demonstrate the following features of the app:
    a. Add a candy
    b. Update a candy
    c. Delete a candy

d. Running the cash register (3 selected candies)

Note:
App is running correctly and met all the requirements is 40 points, but
Incomplete item 1(-40 points)
Incomplete item 2 (-10 points)
Incomplete item 3 (-20 points)