Homework 09 – Local Database CRD

Due: Day 36 – May 4th, 2022 (beginning of class)

Objective:

The goal of this assignment is to gain experience with the CRD (i.e., Create, Read, and Delete) operations of the local SQLiteDatabase that Android uses on the user's device.

Instructions:

For this assignment, assume that you have been hired by a company to work on its vehicle repair logging app (Repair Shop) that allows a user to keep track of the vehicles they own as well as the repairs for each vehicle. On Moodle, I have provided a starting Android project that I want you to use for implementing the required features described below. **Important**: The Views in the layout files have already been given an appropriate ID name for this homework. Therefore, do not add/modify/remove any of the Views in the layout files that I provided you. You will need to setup a View Binding in your Java controller files.

Your app needs to work as follows (i.e., the features/behavior listed below will be explicitly graded for this assignment).

- **Activity # 1 – Main Activity**
  - On the main page, you will want to add the necessary code to open/transition to each "screen"/Activity based upon the button that the user selects:
    - The "Add New Vehicle" button should open the AddVehicleActivity
    - The "Add New Repair" button should open the AddRepairActivity
    - The "Search Repairs" button should open the SearchRepairsActivity
- **Activity #2 – Add Vehicle Activity**
  - On this activity, the user will enter information regarding a vehicle that they own including:
    - Year (e.g., **2009**)
    - Make and Model (e.g., **Toyota Camry**)
    - Purchase Price (e.g., **13500.83**)
    - Is New (e.g., **True** or **False**)
  - When the user presses the "Add Vehicle" button, a new record should be added into a **Vehicle** table in the user's local SQLite database.
    - Important: Make sure that you are choosing an appropriate datatype for each column in your Vehicle table (i.e., do not make everything TEXT – it will lose significant points). Also, make sure that you add an id column as the Vehicle table's primary key and name it **vid** to distinguish its name from another table that you will create later. Use DB Browser for SQLite to create your Vehicle table for the CREATE TABLE SQL statement that you will need for your DBHelper class. I will be **heavily** grading whether you are using good programming practices such as static variables instead of hardcoded text/values – please be sure to use the same practices that were discussed in class when writing the methods of your DBHelper class. You should also create a Vehicle.java class to efficiently pass data to your DBHelper – do not pass information as a large number of parameters/arguments. Your app should display a Toast to confirm when the record is successfully added to the SQLite database's Vehicle table. If so, you should download the SQLite database using the Device File Explorer and open the database in DB Browser for SQLite to confirm that data was indeed correctly added.
    - Important: The onCreate( ) method of your DBHelper class only gets called once when you add the very first record to the database. Therefore, if you make a mistake in your database's design and need to force your DBHelper's onCreate( ) method to be called again, you will need to run

the AVD, open up the Device File Explorer, navigate to data >> data >> com.depauw.repairshop >> databases, and delete both of the .db files in this folder.

- After you have confirmed that pressing the "Add Vehicle" button is correctly adding a new record into the Vehicle table, call the **finish**( ) method so that the Activity closes once the vehicle has been added

- **Activity #3 – Add Repair Activity**
  - On this activity, the user will enter information regarding a new repair for one of their vehicles.
  - When the activity is created, the Spinner should be populated with a list of all vehicles in the Vehicle table (i.e., if the user added 4 vehicles to their SQLite database, then the Spinner should display all 4 items). Each vehicle should be displayed in the Spinner in the following format: **[year] [makeModel]**
    - Example:
      - 2009 Toyota Camry
      - 2020 Honda Odyssey
      - 1967 Chevrolet Camaro
      - 1983 Ford Mustang
    - Important #1: The DBHelper method that you write must return a list of Vehicle objects using the same practices that were discussed in class.
    - Important #2: You should implement this feature first before proceeding to the remaining steps
  - The company wants a DatePicker to appear when the user taps the **edittext_repair_date.** The DatePicker should display with the current date initially selected, however, the user should be able to select any date for their repair. When the user confirms the repair's date in the DatePicker, the date that they selected should be displayed in the **edittext_repair_date** in the ISO8601 format: **yyyy-mm-dd**
    - Example: If the user selected April 7th, 2022 – it should be displayed as **2022-04-07**
  - Finally, the user will type in their repair's cost (example: *247.74*) and description (example: *Replaced brake pads and added new water pump*), and press the Add Repair button.
  - When the user presses the "Add Repair" button, a new record should be added into a **Repair** table in the user's local SQLite database.
    - Important: Make sure that you are choosing an appropriate datatype for each column in your Repair table (i.e., do not make everything TEXT). Also, make sure that you (1) add an id column as the Repair table's primary key named **rid** to distinguish its name from the Vehicle table and (2) a **foreign key** column to associate the repair with the vehicle id that they have selected in the Spinner. Use DB Browser for SQLite to create your Repair table and the CREATE TABLE SQL statement that you will need for your DBHelper class.
    - Important: The onCreate( ) method of your DBHelper class only gets called once when you add the very first record to the database. Therefore, since you have likely added records to test your Vehicle table, you will need to force your DBHelper's onCreate( ) method to be called again in order to create your Repair table. To do this, you need to run the AVD, open up the Device File Explorer, navigate to data >> data >> com.depauw.repairshop >> databases, and delete both of the .db files in this folder. When you re-run your app and add your very first record again, this will cause the onCreate( ) method to be called to create both your Vehicle table and Repair table
    - Important: You should also create a Repair.java class to efficiently pass data to your DBHelper – do not pass information as a large number of parameters/arguments. Your app should display a Toast to confirm when the record is successfully added to the SQLite database's Repair table. If so, you should download the SQLite database using the Device File Explorer and open the database in DB Browser for SQLite to confirm that data was indeed correctly added.
  - After you have confirmed that pressing the "Add Repair" button is correctly adding a new record into the Repair table, call the **finish**( ) method so that the Activity closes once the repair record has been added.

- **Activity #4 – Search Repairs Activity**
  - On this activity, the user will type a **search phrase** (i.e., a single word or words) that they would like to search for in their repair records. When they press the "Find Repairs" button, the local SQLite database should be queried and only repair records whose description contains the search phrase should be displayed in the ListView.
    - Example #1: If the user types the word **oil** as their search phrase, then only repair records whose description contains the word **oil** should appear in the ListView
    - Example #2: If the user types the word **brake pad** as their search phrase, then only repair records whose description contains the phrase **brake pad** should appear in the ListView
    - Example #3: If the user does not type a word as their search phrase (i.e., an empty search phrase), then a list of all the repair records should display
  - When the user presses the "Find Repairs" button, the results should be displayed in the ListView using the provided **listview_results_row.xml** layout file. Open this layout file and notice that each row will display both Vehicle and Repair information. To accommodate this design, you **must** do the following:
    - Create a Java class named **RepairWithVehicle** that has two member variables: (1) a Repair object variable and (2) a Vehicle object variable. In object-oriented programming, you learned about a relationship called composition where an object is "composed" (i.e., made up of) other objects. Read this brief article introducing the concept of composition and notice how in their example, a CoffeeMachine object "*has a*" Grinder object and BrewingUnit object as its two member variables. In the same way, your **RepairWithVehicle** object now is made up of a (1) Repair object that will contain repair data and (2) a Vehicle object that will contain the associated vehicle data – therefore – each RepairWithVehicle object can represent 1 row of data when you join together the Repair table with the Vehicle table.
    - Use Android Studio to generate a constructor for your RepairWithVehicle class that takes a (1) Repair object and (2) Vehicle object as parameters (i.e., inputs) – do not create a constructor that takes a large number of parameters/arguments. Also, use Android Studio to generate the two "getters" for your RepairWithVehicle class: (1) a getRepair( ) method and (2) a getVehicle( ) method.
    - In your DBHelper class, write a method named **getRepairsWithVehicle**( ) that takes the search phrase as a parameter and returns a List of RepairWithVehicle objects. Write the appropriate SQL query to return records containing repairs with their associated vehicle information whose description contains the search phrase. You must use the appropriate single INNER JOIN query to obtain your results – failure to do so will result in a significant loss of points. For each row of your results table, you must:
      - Extract the repair column data and store these values to a temporary Repair object
      - Extract the vehicle column data and store these values to a temporary Vehicle object
      - Construct a RepairWithVehicle object by passing its constructor the temporary Repair and Vehicle objects that you created – your RepairWithVehicle object has now been created using the object-oriented principle that we call composition!
      - Add the RepairWithVehicle object to your List that you will eventually return
    - The getRepairsWithVehicle( ) method should be called when the user presses the "Find Repairs" button in order to populate the rows of the ListView. Use the provided **listview_results_row.xml** layout file to display each result's (1) vehicle year, (2) make and model, (3) repair date, (4) repair cost, and (5) repair description (*an example row is shown below*):

- In order to test whether your "Find Repairs" button is working correctly, you should first add several vehicles and repairs using the previous activities. Then, test the following scenarios:
    - Scenario #1: If I press the "Find Repairs" button without typing a search phrase, then the ListView should display all of the repairs
    - Scenario #2: If I press the "Find Repairs" button and type in a phrase (e.g., **oil**), then the ListView should display only repairs whose description contains the word **oil** in it
    - Scenario #3: If I press the "Find Repairs" button and type in a phrase (e.g., **brake**) that none of the repairs contain in their description, then the ListView should display no results

- Finally, the company would like one last feature. When the user **long click**s on a row in the ListView, the selected repair record should be deleted in the local SQLite database's Repair table. After the record has been deleted from the database, the Listview should be automatically refreshed by calling its **invalidateViews**( ) method and the deleted record should no longer be visible in the results after doing so.

Submission:

When you are finished, you must **zip** your Android Studio project. On Moodle, you should upload your zip file to the Homework 09 assignment box