Labsheet 6 – Friday 1ST October 2021

Databases

Software Development for Portable Devices

Information about HD TA:

AASHITA DUTTA <h20201030130@hyderabad.bits-pilani.ac.in>

Breakout room link: https://meet.google.com/erq-uxqi-egv

Information about FD TA:

ARUNEEMA DESHMUKH <f20180568@hyderabad.bits-pilani.ac.in>

Breakout room link: https://meet.google.com/pvo-osiz-cuh

In this practical, we will be creating an Android application that uses the room persistent library to create and manage SQLite databases. We are going to create a simple list application called NotesApp. The list will be stored locally using the Room library.

Room is Google's persistence library designed to make it easier to build offline apps. It tries to expose APIs that can leverage the full power of SQL while still providing an abstraction layer for managing the data as Java objects. It also works well seamlessly with Google's Architecture Components library for building robust high-quality production apps and can also be used along with the Paging Library for handling large data sets.

Room vs SQLite

Room is an ORM, Object Relational Mapping library. In other words, Room will map the database objects to Java objects. Room provides an abstraction layer over SQLite to allow fluent database access while harnessing the full power of SQLite.

Why use Room?

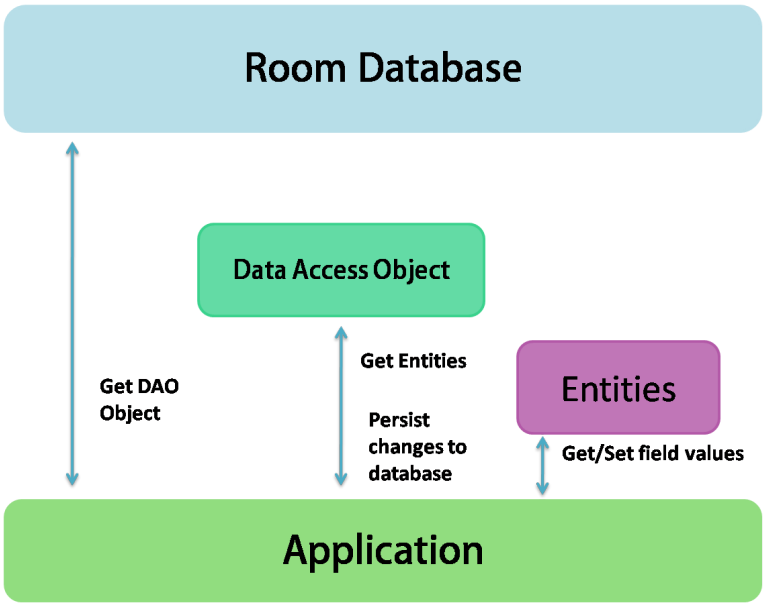
1. Compile-time verification of SQL queries. Each @Query and @Entity is checked at the compile time, which preserves your app from crash issues at runtime and not only it checks the only syntax, but also missing tables.
2. Convenience annotations that minimize repetitive and error-prone boilerplate code.
3. Easily integrated with other Architecture components (like LiveData). Room is built to work with LiveData and RxJava for data observation.
4. Streamlined database migration paths.
5. Room annotations are used:
    - To Database and entities where entities are POJO classes representing table structures.
    - To specify operation for retrieval, updation, and deletion.
    - To add constraints such as foreign keys.
    - Support for LiveData

Major problems with SQLite usage are

1. There is no compile-time verification of raw SQL queries. For example, if you write a SQL query with a wrong column name that does not exist in the real database then it will give an exception during run time and you can not capture this issue during compile time.
2. As your schema changes, you need to update the affected SQL queries manually. This process can be time-consuming and error-prone.
3. You need to use lots of boilerplate code to convert between SQL queries and Java data objects (POJO).

Room Basics

There are 3 major components in Room



1. **Entity** : A class annotated with the @Entity annotation is mapped to a table in the database. Every entity is persisted in its own table and every field in class represents the column name.
- tableName attribute is used to define the name of the table
- Every entity class must have at-least one Primary Key field, annotated with @PrimaryKey
- Fields in entity class can be annotated with @ColumnInfo(name = “name_of_column”) annotation to give specific column names

2. **DAO** : Data Access Object can either be an interface or an abstract class annotated with @Doa annotation, containing all the methods to define the operations to be performed on data. The methods can be annotated with
    - @Query to retrieve data from database
    - @Insert to insert data into database
    - @Delete to delete data from database
    - @Update to update data in database

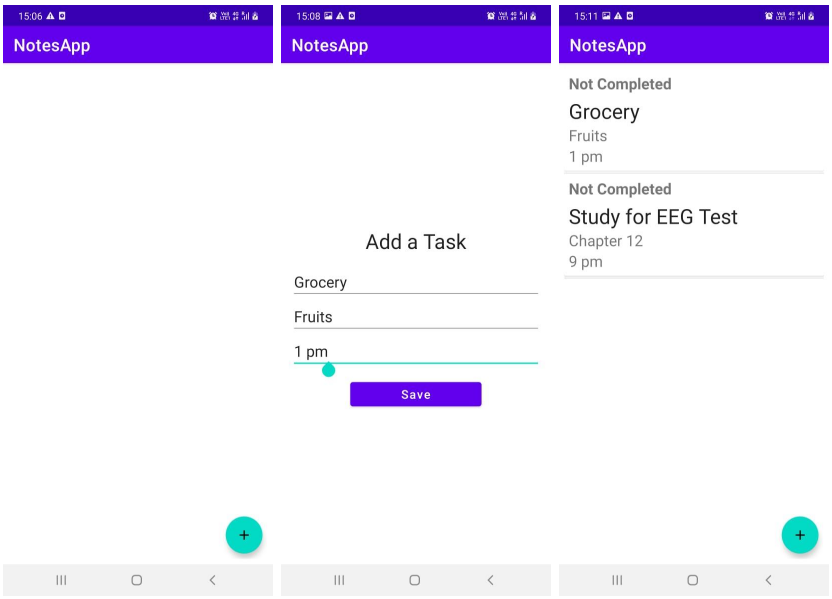
The result of SQLite queries are composed into cursor objects, DAO methods abstract the conversion of cursor to Entity objects and vice-versa.

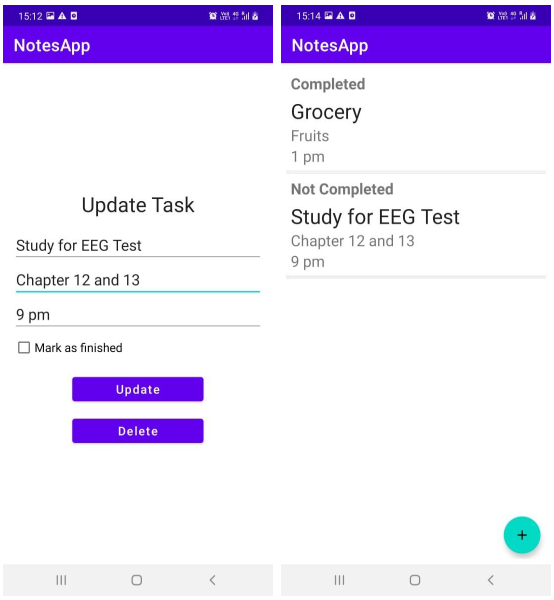
3. **Database** : Database is a container for tables. An abstract class annotated with @Database annotation is used to create a database with a given name along with the database version.
    - version = intValueForDBVersion is used to define the database version
    - entities = {EntityClassOne.class, ...} is used to define list of entities for database

We will build a Notes App that will allow the user to:

1. **Create** and **Save** tasks in database
2. **Display** a list of tasks
3. **Update** and **Delete** tasks

App Screenshots: NotesApp





Task 1: Adding Dependencies

1. Select **Empty Activity** as Project Template. We already have a MainActivity created, other than this we need, AddTaskActivity and UpdateTaskActivity. Add the dependencies from below inside the app level build.gradle file (some will already be present).
2. The first dependency you added, **room-runtime**, is for the Room API containing all the classes and annotations you will need to define your database.
   The second dependency, **room-compiler,** is for the Room compiler, which will generate your database implementation based on the annotations you specify.
3. **Sync your Gradle files.**

build.gradle(:app)

```
dependencies {
    implementation 'androidx.appcompat:appcompat:1.3.1'
    implementation 'com.google.android.material:material:1.4.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.0'

    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'

    //define versions
    def room_version = "2.3.0"

    //room
    implementation "androidx.room:room-runtime:$room_version"
    annotationProcessor "androidx.room:room-compiler:$room_version"
    androidTestImplementation "androidx.room:room-testing:$room_version"
```

```
}
```

Questions:

1. Why are there multiple gradle files? Explore and find out more.

Ref:
https://gradle.org/
https://www.studytonight.com/android/introduction-to-gradle
https://developer.android.com/studio/build/dependencies

Task 2: Creating Layouts

activity_main.xml : In our main activity we will display all the added tasks, and a Floating Action Button at the bottom from where the user can add a new task.

To add the drawable resource for the FAB you can either add an icon like done in prev labs or
Right Click on drawable folder -> New -> Vector Asset -> Choose the '+' (name: ic_baseline_add_24) from Clip Art

For displaying all the added tasks we will use a Recycler View

- **RecyclerView** is the ViewGroup that contains the views corresponding to your data. It's a view itself, so you add RecyclerView into your layout the way you would add any other UI element.
- Each individual element in the list is defined by a *view holder* object. When the view holder is created, it doesn't have any data associated with it. After the view holder is created, the RecyclerView *binds* it to its data. You define the view holder by extending RecyclerView.ViewHolder.
- The RecyclerView requests those views, and binds the views to their data, by calling methods in the *adapter.* You define the adapter by extending RecyclerView.Adapter.
- The *layout manager* arranges the individual elements in your list. You can use one of the layout managers provided by the RecyclerView library, or you can define your own. Layout managers are all based on the library's LayoutManager abstract class.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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"
    android:padding="8dp"
    tools:context=".MainActivity">

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

```
    <com.google.android.material.floatingactionbutton.FloatingActionButton
        android:id="@+id/floating_button_add"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:layout_margin="8dp"
        android:src="@drawable/ic_baseline_add_24"
        app:fabSize="normal" />

</RelativeLayout>
```

activity_add_task.xml : Every task should have a name, description and time deadline

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".AddTaskActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
        android:orientation="vertical"
        android:padding="16dp">

        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="15dp"
            android:text="Add a Task"
            android:textAlignment="center"

android:textAppearance="@style/Base.TextAppearance.AppCompat.Headline" />

        <EditText
            android:id="@+id/editTextTask"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="task" />

        <EditText
            android:id="@+id/editTextDesc"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="description" />

        <EditText
            android:id="@+id/editTextFinishBy"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="time" />

        <Button
            android:id="@+id/button_save"
            android:layout_width="200dp"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
```

```
            android:layout_marginTop="15dp"
            android:text="Save"
            android:textAllCaps="false" />

    </LinearLayout>

</RelativeLayout>
```

activity_update_task.xml : Here the user will get the option to edit task details, mark the task as completed through a checkbox and delete the task

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".UpdateTaskActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
        android:orientation="vertical"
        android:padding="16dp">

        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="15dp"
            android:text="Update Task"
            android:textAlignment="center"

android:textAppearance="@style/Base.TextAppearance.AppCompat.Headline" />

        <EditText
            android:id="@+id/editTextTask"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="task" />

        <EditText
            android:id="@+id/editTextDesc"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="description" />

        <EditText
            android:id="@+id/editTextFinishBy"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="time" />

        <CheckBox
            android:id="@+id/checkBoxFinished"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Mark as finished" />

        <Button
```

```
            android:id="@+id/button_update"
            android:layout_width="200dp"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:layout_marginTop="15dp"
            android:text="Update"
            android:textAllCaps="false" />


        <Button
            android:id="@+id/button_delete"
            android:layout_width="200dp"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:layout_marginTop="15dp"
            android:text="Delete"
            android:textAllCaps="false"/>

    </LinearLayout>


</RelativeLayout>
```

recyclerview_tasks.xml : This is the layout for each RecyclerView row

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <androidx.cardview.widget.CardView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="3dp">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical"
            android:padding="7dp">

            <TextView
                android:id="@+id/textViewStatus"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginBottom="5dp"
                android:text="Completed"

android:textAppearance="@style/Base.TextAppearance.AppCompat.Medium"
                android:textStyle="bold" />

            <TextView
                android:id="@+id/textViewTask"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="SPDP"
```

```
android:textAppearance="@style/Base.TextAppearance.AppCompat.Headline" />

            <TextView
                android:id="@+id/textViewDesc"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="Quiz 3"

android:textAppearance="@style/Base.TextAppearance.AppCompat.Medium" />

            <TextView
                android:id="@+id/textViewFinishBy"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="5pm today"

android:textAppearance="@style/Base.TextAppearance.AppCompat.Medium" />


        </LinearLayout>

    </androidx.cardview.widget.CardView>

</RelativeLayout>
```

Questions:

1. What do you mean by app:fabSize in FloatingButton?
2. Explore more about Recycler View and Card View.
3. Why do we need an additional layout file - recyclerview_tasks.xml?
4. recyclerview_task.xml has TextView with pre-filled values, what if we remove that android:text from the TextView?

Task 3: Database

Let's dive into 3 components of Room:

1. Creating Entity

For every table that we need we need to create an entity. And for this application we need a single entity. Every class is an entity.

So create a **class named Task.java** and write the following code. We have annotated the class with @Entity, this is our table, and for the column id we have used @PrimaryKey(autoGenerate = true) this means this id will be auto increment, for other columns we used @ColumnInfo(name="columnname").

Task.java

```
package com.example.notesapp;


import androidx.room.ColumnInfo;
import androidx.room.Entity;
import androidx.room.PrimaryKey;
```

```
import java.io.Serializable;

@Entity
public class Task implements Serializable {

   @PrimaryKey(autoGenerate = true)
   private int id;

   @ColumnInfo(name = "task")
   private String task;

   @ColumnInfo(name = "desc")
   private String desc;

   @ColumnInfo(name = "finish_by")
   private String finishBy;

   @ColumnInfo(name = "finished")
   private boolean finished;


   /*
    * Getters and Setters
    * */
   public int getId() {
       return id;
   }

   public void setId(int id) {
       this.id = id;
   }

   public String getTask() {
       return task;
   }

   public void setTask(String task) {
       this.task = task;
   }

   public String getDesc() {
       return desc;
   }

   public void setDesc(String desc) {
       this.desc = desc;
   }

   public String getFinishBy() {
       return finishBy;
   }

   public void setFinishBy(String finishBy) {
       this.finishBy = finishBy;
   }

   public boolean isFinished() {
       return finished;
   }

   public void setFinished(boolean finished) {
```

```
        this.finished = finished;
    }
}
```

Questions:

1. Why do we have to implement a Serializable interface?
2. Why do we use the get and set functions?
3. Try running this app without serializable implementation (later).

2. Creating Dao

The first step to interacting with your database tables is to create a Dao (Data Access Object). A DAO is an interface that contains functions for each database operation you want to perform. So create an **interface named TaskDao.java** and write the following code. We defined all the methods needed for the Create, Read, Update and Delete operation.

The @Query annotation expects a string containing a SQL command as input.
SELECT * FROM task asks Room to pull all columns for all rows in the database table.

TaskDao.java

```
package com.example.notesapp;

import androidx.room.Dao;
import androidx.room.Delete;
import androidx.room.Insert;
import androidx.room.Query;
import androidx.room.Update;

import java.util.List;

@Dao
public interface TaskDao {

    @Query("SELECT * FROM task")
    List<Task> getAll();

    @Insert
    void insert(Task task);

    @Delete
    void delete(Task task);

    @Update
    void update(Task task);

}
```

Reference: https://developer.android.com/training/data-storage/room/accessing-data

Questions:

1. @Delete above will delete only one object, what if we want to remove multiple objects like an array of objects at one go?
2. Why is TaskDao.java not a class but an Interface?

3. Creating Database

Create one more **class and name it AppDatabase**, and write the following code inside the class. We will define all the entities and the database version.

The @Database annotation tells Room that this class represents a database in your app. The annotation itself requires two parameters. The first parameter is a list of entity classes, which tells Room which entity classes to use when creating and managing tables for this database. The second parameter is the version of the database.

When you first create a database, the version should be 1. As you develop your app in the future, you may add new entities and new properties to existing entities. When this happens, you will need to modify your entities list and increment your database version.

To tell the database class to generate an instance of the DAO we add an abstract function that has a return type as TaskDao.

AppDatabase.java

```
package com.example.notesapp;

import androidx.room.Database;
import androidx.room.RoomDatabase;

@Database(entities = {Task.class}, version = 1)
public abstract class AppDatabase extends RoomDatabase {
   public abstract TaskDao taskDao();
}
```

Database Client

Creating AppDatabase's object is expensive so we will create a single instance of it.
So we are applying Singleton Pattern to create and use already instantiated single instance for every database access.
Create a **class named DatabaseClient** and write the following code.

DatabaseClient.java

```
package com.example.notesapp;

import android.content.Context;
import androidx.room.Room;

public class DatabaseClient {

   private Context mCtx;
   private static DatabaseClient mInstance;

   //our app database object
```

```
    private AppDatabase appDatabase;

    private DatabaseClient(Context mCtx) {
        this.mCtx = mCtx;

        //creating the app database with Room database builder
        //MyToDos is the name of the database
        appDatabase = Room.databaseBuilder(mCtx, AppDatabase.class,
"MyToDos").build();
    }

    public static synchronized DatabaseClient getInstance(Context mCtx) {
        if (mInstance == null) {
            mInstance = new DatabaseClient(mCtx);
        }
        return mInstance;
    }

    public AppDatabase getAppDatabase() {
        return appDatabase;
    }
}
```

Questions:

1. Why do we need to pass a Context object to the getInstance method?
2. Why do you need the synchronized keyword?

Task 4: Activities

Adding tasks to database

We will create an AsyncTask to perform our operation because if we try to perform the database operation in the main thread it will crash our application. Reading from the database does not happen immediately. Because access can take so long, Room disallows all database operations on the main thread. If you try to violate this rule, Room will throw the IllegalStateException.

For saving the task we just created the object and called the insert method that we created in our TaskDao interface.

AddTaskActivity.java

```
package com.example.notesapp;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;
```

```
public class AddTaskActivity extends AppCompatActivity {

   private EditText editTextTask, editTextDesc, editTextFinishBy;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_add_task);

       editTextTask = findViewById(R.id.editTextTask);
       editTextDesc = findViewById(R.id.editTextDesc);
       editTextFinishBy = findViewById(R.id.editTextFinishBy);

       findViewById(R.id.button_save).setOnClickListener(new View.OnClickListener()
{
           @Override
           public void onClick(View view) {
               saveTask();
           }
       });
   }

   private void saveTask() {
       final String sTask = editTextTask.getText().toString().trim();
       final String sDesc = editTextDesc.getText().toString().trim();
       final String sFinishBy = editTextFinishBy.getText().toString().trim();

       if (sTask.isEmpty()) {
           editTextTask.setError("Task required");
           editTextTask.requestFocus();
           return;
       }

       if (sDesc.isEmpty()) {
           editTextDesc.setError("Desc required");
           editTextDesc.requestFocus();
           return;
       }

       if (sFinishBy.isEmpty()) {
           editTextFinishBy.setError("Finish by required");
           editTextFinishBy.requestFocus();
           return;
       }

       class SaveTask extends AsyncTask<Void, Void, Void> {

           @Override
           protected Void doInBackground(Void... voids) {

               //creating a task
               Task task = new Task();
               task.setTask(sTask);
               task.setDesc(sDesc);
               task.setFinishBy(sFinishBy);
               task.setFinished(false);

               //adding to database
               DatabaseClient.getInstance(getApplicationContext()).getAppDatabase()
                       .taskDao()
                       .insert(task);
               return null;
```

```
            }

            @Override
            protected void onPostExecute(Void aVoid) {
                super.onPostExecute(aVoid);
                finish();
                startActivity(new Intent(getApplicationContext(),
MainActivity.class));
                Toast.makeText(getApplicationContext(), "Saved",
Toast.LENGTH_LONG).show();
            }
        }

        SaveTask st = new SaveTask();
        st.execute();
    }

}
```

Questions:

1. What is the purpose of requestFocus()?
2. AsyncTask<Void, Void, Void> what does **<Void, Void, Void>** represent?
3. doInBackground method takes arguments as **Void… voids**. What does these 3 dots mean?
4. What is the role of **doInBackground** and **onPostExecute** function under AsyncTask?
5. What is **Toast**? Can you try a snackbar? How are the two different?

Displaying tasks

TasksAdapter.java :

We have designed how each element in the list is going to look in **recyclerview_tasks.xml**. Based on this design,we extend the ViewHolder class. Our version of ViewHolder provides all the functionality for the list items. The view holder is a wrapper around a View, and that view is managed by RecyclerView.

Then we need to create the adapter which will actually populate the data into the RecyclerView. The adapter's role is to convert an object at a position into a list row item to be inserted.

Every adapter has three primary methods:

1. onCreateViewHolder to inflate the item layout and create the holder,
2. onBindViewHolder to set the view attributes based on the data and
3. getItemCount to determine the number of items. We need to implement all three to finish the adapter:

For more:
https://developer.android.com/guide/topics/ui/layout/recyclerview#implement-adapter
https://guides.codepath.com/android/using-the-recyclerview

```
package com.example.notesapp;
```

```
import android.content.Context;
import android.content.Intent;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

import androidx.recyclerview.widget.RecyclerView;

import java.util.List;

public class TasksAdapter extends RecyclerView.Adapter<TasksAdapter.TasksViewHolder>
{

   private Context mCtx;
   private List<Task> taskList;

   public TasksAdapter(Context mCtx, List<Task> taskList) {
       this.mCtx = mCtx;
       this.taskList = taskList;
   }

   @Override
   public TasksViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
       View view = LayoutInflater.from(mCtx).inflate(R.layout.recyclerview_tasks,
parent, false);
       return new TasksViewHolder(view);
   }

   @Override
   public void onBindViewHolder(TasksViewHolder holder, int position) {
       Task t = taskList.get(position);
       holder.textViewTask.setText(t.getTask());
       holder.textViewDesc.setText(t.getDesc());
       holder.textViewFinishBy.setText(t.getFinishBy());

       if (t.isFinished())
           holder.textViewStatus.setText("Completed");
       else
           holder.textViewStatus.setText("Not Completed");
   }

   @Override
   public int getItemCount() {
       return taskList.size();
   }

   class TasksViewHolder extends RecyclerView.ViewHolder implements
View.OnClickListener {

       TextView textViewStatus, textViewTask, textViewDesc, textViewFinishBy;

       public TasksViewHolder(View itemView) {
           super(itemView);

           textViewStatus = itemView.findViewById(R.id.textViewStatus);
           textViewTask = itemView.findViewById(R.id.textViewTask);
           textViewDesc = itemView.findViewById(R.id.textViewDesc);
           textViewFinishBy = itemView.findViewById(R.id.textViewFinishBy);


           itemView.setOnClickListener(this);
```

```
        }

        @Override
        public void onClick(View view) {
            Task task = taskList.get(getAdapterPosition());

            Intent intent = new Intent(mCtx, UpdateTaskActivity.class);
            intent.putExtra("task", task);

            mCtx.startActivity(intent);
        }
    }
}
```

Questions:

1. Why does putExtra has two arguments? What's the difference between the two?

MainActivity.java : We will call the getAll() method that we created inside our TaskDao to get all the stored tasks from the database.
Then we are displaying the read tasks to a RecyclerView.

```
package com.example.notesapp;

import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;

import android.content.Intent;
import android.os.AsyncTask;
import android.os.Bundle;

import android.view.View;

import com.google.android.material.floatingactionbutton.FloatingActionButton;
import java.util.List;

public class MainActivity extends AppCompatActivity {

    private FloatingActionButton buttonAddTask;
    private RecyclerView recyclerView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        recyclerView = findViewById(R.id.recyclerview_tasks);
        recyclerView.setLayoutManager(new LinearLayoutManager(this));

        buttonAddTask = findViewById(R.id.floating_button_add);
        buttonAddTask.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new Intent(MainActivity.this, AddTaskActivity.class);
                startActivity(intent);
            }
```

```
        });


        getTasks();

    }


    private void getTasks() {
        class GetTasks extends AsyncTask<Void, Void, List<Task>> {

            @Override
            protected List<Task> doInBackground(Void... voids) {
                List<Task> taskList = DatabaseClient
                        .getInstance(getApplicationContext())
                        .getAppDatabase()
                        .taskDao()
                        .getAll();
                return taskList;
            }

            @Override
            protected void onPostExecute(List<Task> tasks) {
                super.onPostExecute(tasks);
                TasksAdapter adapter = new TasksAdapter(MainActivity.this, tasks);
                recyclerView.setAdapter(adapter);
            }
        }

        GetTasks gt = new GetTasks();
        gt.execute();
    }

}
```

Update/Delete tasks

UpdateTaskActivity.java

```
package com.example.notesapp;

import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;

import android.content.DialogInterface;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.View;
import android.widget.CheckBox;
import android.widget.EditText;
import android.widget.Toast;

public class UpdateTaskActivity extends AppCompatActivity {

    private EditText editTextTask, editTextDesc, editTextFinishBy;
    private CheckBox checkBoxFinished;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_update_task);


        editTextTask = findViewById(R.id.editTextTask);
        editTextDesc = findViewById(R.id.editTextDesc);
        editTextFinishBy = findViewById(R.id.editTextFinishBy);

        checkBoxFinished = findViewById(R.id.checkBoxFinished);


        final Task task = (Task) getIntent().getSerializableExtra("task");

        loadTask(task);

        findViewById(R.id.button_update).setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Toast.makeText(getApplicationContext(), "Clicked",
Toast.LENGTH_LONG).show();
                updateTask(task);
            }
        });

        findViewById(R.id.button_delete).setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View view) {

                AlertDialog.Builder builder = new
AlertDialog.Builder(UpdateTaskActivity.this);
                builder.setTitle("Are you sure?");
                builder.setPositiveButton("Yes", new
DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialogInterface, int i) {
                        deleteTask(task);
                    }
                });
                builder.setNegativeButton("No", new DialogInterface.OnClickListener()
{
                    @Override
                    public void onClick(DialogInterface dialogInterface, int i) {

                    }
                });

                AlertDialog ad = builder.create();
                ad.show();
            }
        });
    }

    private void loadTask(Task task) {
        editTextTask.setText(task.getTask());
        editTextDesc.setText(task.getDesc());
        editTextFinishBy.setText(task.getFinishBy());
        checkBoxFinished.setChecked(task.isFinished());
```

```
    }

    private void updateTask(final Task task) {
        final String sTask = editTextTask.getText().toString().trim();
        final String sDesc = editTextDesc.getText().toString().trim();
        final String sFinishBy = editTextFinishBy.getText().toString().trim();

        if (sTask.isEmpty()) {
            editTextTask.setError("Task required");
            editTextTask.requestFocus();
            return;
        }

        if (sDesc.isEmpty()) {
            editTextDesc.setError("Desc required");
            editTextDesc.requestFocus();
            return;
        }

        if (sFinishBy.isEmpty()) {
            editTextFinishBy.setError("Finish by required");
            editTextFinishBy.requestFocus();
            return;
        }

        class UpdateTask extends AsyncTask<Void, Void, Void> {

            @Override
            protected Void doInBackground(Void... voids) {
                task.setTask(sTask);
                task.setDesc(sDesc);
                task.setFinishBy(sFinishBy);
                task.setFinished(checkBoxFinished.isChecked());
                DatabaseClient.getInstance(getApplicationContext()).getAppDatabase()
                        .taskDao()
                        .update(task);
                return null;
            }

            @Override
            protected void onPostExecute(Void aVoid) {
                super.onPostExecute(aVoid);
                Toast.makeText(getApplicationContext(), "Updated",
Toast.LENGTH_LONG).show();
                finish();
                startActivity(new Intent(UpdateTaskActivity.this,
MainActivity.class));
            }
        }

        UpdateTask ut = new UpdateTask();
        ut.execute();
    }


    private void deleteTask(final Task task) {
        class DeleteTask extends AsyncTask<Void, Void, Void> {

            @Override
            protected Void doInBackground(Void... voids) {
                DatabaseClient.getInstance(getApplicationContext()).getAppDatabase()
                        .taskDao()
```

```
                        .delete(task);
                return null;
            }

            @Override
            protected void onPostExecute(Void aVoid) {
                super.onPostExecute(aVoid);
                Toast.makeText(getApplicationContext(), "Deleted",
Toast.LENGTH_LONG).show();
                finish();
                startActivity(new Intent(UpdateTaskActivity.this,
MainActivity.class));
            }
        }

        DeleteTask dt = new DeleteTask();
        dt.execute();

    }

}
```

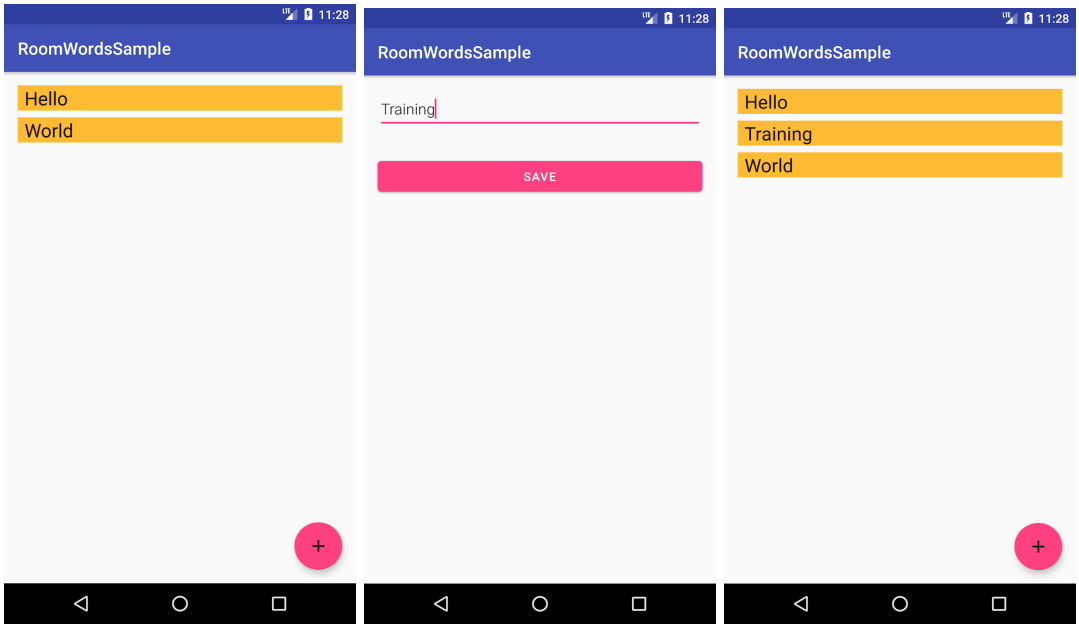
Questions:

In the code, final Task task = (Task) getIntent().getSerializableExtra("task");

1. This line in code has used the final access specifier, why?
2. Why is getSerializableExtra used?
3. What is an Alert Dialog (Dialogs)?

Homework Assignment:

Implement the following app where the list of words will be saved in the room database and when the user taps the + button, a new word can be added to the database and displayed in lexicographic order. Find below the preview of app:


11:28
RoomWordsSample
Hello
World
11:28
RoomWordsSample
Training
SAVE
11:28
RoomWordsSample
Hello
Training
World