Exercise 2: RecyclerView

Due: Feb. 26, 23:59 PDT

Overview

This exercise provides an example of using RecyclerView to display and manage a list.

The resulting app will display a list shown on Fig. 1. When the header of a row is clicked, the text will change. Also, if user swipe a row, the corresponding row will be removed.

Prerequisites:

- Know the process of submitting your work (exercise 0)
- Know Android project structure (exercise 1, lesson 2)
- Know how to use the Layout Editor (exercise 1, lesson 3)
- Has set up an emulator (API 29)

Procedure related to the above topics will not be provided in the instruction. Refer to corresponding exercise/lecture notes if needed.

If you set any different view id's, filenames, etc., remember to modify the corresponding part of code.

Set the project name as "Exercise2YourName"

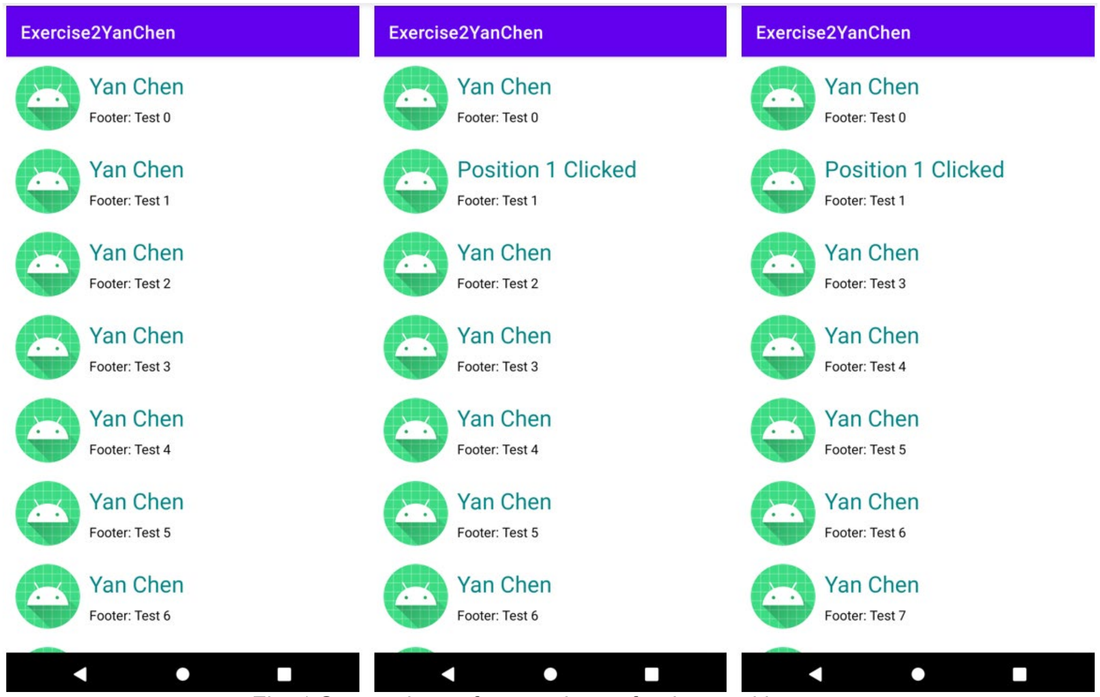

Fig. 1 Screenshots of a sample run for the resulting app:
launch page (left),
result of click the 2nd (test 1) row (middle),
and result of swipe and delete the 3rd (test 2) row (right)

Step 0. Configure build.gradle (module level)

Where to find the module level build.gradle file was shown in lesson 2 page 20.
The following steps may be optional.

0.1 Set View Binding

The example uses view binding. If you want to use classic findViewById, you can skip this step. Otherwise, add “buildFeatures {viewBinding true}” under android (highlighted in the code below 0.2).

0.2 Add RecyclerView dependency

If you are using the latest Android Studio, you don’t need to do this because it is included in the ‘com.google.android.material:material:1.3.0’ (any version), which is added in your project by default.

Otherwise, or if Android Studio complain about can’t find the package when you do the next steps, add the highlighted dependency.

```
plugins {
    id 'com.android.application'
}

android {
    buildFeatures {
        viewBinding true
    }
    ...
}

dependencies {
    implementation "androidx.recyclerview:recyclerview:1.1.0"
    implementation 'androidx.appcompat:appcompat:1.2.0'
    ...
}
```

Remember to click “sync” when you done. It should be popped out after you done any change. Or you can press the button shown in Fig. 2.

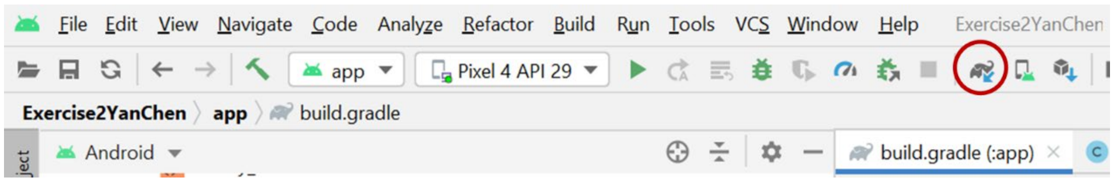

Fig. 2 Sync project button (circled)

Step 1. Set up UI files

As discussed in class, you need to have 2 UI files in total, one for the whole list, and one for each row.

1.1 Set up main_activity.xml

Use the main_activity.xml file as your UI for the whole list. Simply add a RecyclerView element, don't need to add any constraints.

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

1.2 Add and Set up a New UI File

Add a new UI file (lesson 3 page 18) for each row, name it row_layout.xml (you can use other root layout besides constraint layout). In the file:

- Add the following widgets and set the UI as shown in Fig. 3 (don't have to be the exact same).

Widget Type	id	Content
ImageView	image	@mipmap/ic_launcher
TextView	header	Your Name
TextView	footer	Will set later

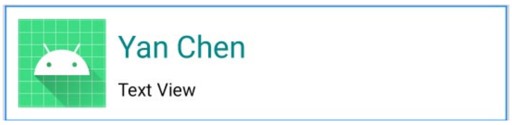

Fig. 3 UI of one row (no border - the border here just for easy reading)

- Set the height of the root layout to "wrap_content" or "?android:attr/listPreferredItemHeight".

```
<androidx.constraintlayout.widget.ConstraintLayout
    ...
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <ImageView
        android:id="@+id/image"
        ...
        app:srcCompat="@mipmap/ic_launcher" />

    <TextView
        android:id="@+id/header"
        .../>

    <TextView
        android:id="@+id/footer"
        android:text="@string/name"
        .../>
</androidx.constraintlayout.widget.ConstraintLayout>
```

In the next two steps, if you still want to use the classical findViewById, please refer to the example given in lesson 6 page 20 - 21 if needed.

Step 2. Implement a MyAdapter class

You need to implement a custom adapter to apply the same UI for each row.

Create a Java class named "MyAdapter" and do the following.

2.1 Set a Constructor

You will need to pass in an ArrayList<String> when creating a MyAdapter object. Therefore, set an ArrayList<String> named "data" as a class attribute and initialize it with the argument passed in the constructor. That is all you need for the constructor.

2.2 Implement a ViewHolder class

As explained in class, the ViewHolder is a wrapper around a View that contains the layout for each row in the list. You can set it as an inner class of MyAdapter. It is binding with the row_layout.xml.

```
public class ViewHolder extends RecyclerView.ViewHolder {
    protected final RowLayoutBinding binding;

    public ViewHolder(RowLayoutBinding binding) {
        super(binding.getRoot());
        this.binding = binding;
    }
}
```

2.3 Extend to RecyclerView.Adapter<MyAdapter.ViewHolder> and Implement Required Methods

Extend MyAdapter class to RecyclerView.Adapter<MyAdapter.ViewHolder>. Alt + Enter (Option + Enter for Mac) and choose "Implement methods" in the drop-down menu. If the shortcut doesn't work, right click the line with the warning, choose "Show Context Actions" to see the drop-down menu.

Click OK on the pop-up window. There are 3 methods to implement:

- onCreateViewHolder
  This method creates a new ViewHolder for a row when needed. It will only initialize the ViewHolder, but no data has filled in yet.
  The parent parameter is the whole list. The system, or more precisely, the Recycler View will take care of the viewType.
  This method returns a new ViewHolder for a row.

```
public MyAdapter.ViewHolder onCreateViewHolder(ViewGroup parent,
                                               int viewType) {
    // Inflate (= create) a new view
    LayoutInflater inflater = LayoutInflater.from(parent.getContext());
    RowLayoutBinding row = RowLayoutBinding.inflate(inflater);
    return new ViewHolder(row);
}
```

- onBindViewHolder
  The method displays the view holder's layout using the corresponding data at the specified position. This means, when it is called, the ViewHolder is bound to specific data.
  Therefore, you need to set data and add listeners if needed under this method.
  Note that the parameter position is final, which means it won't change if the list changed later (i.e., it's the initial position of each row). As mentioned before, the resulting app allows the user to remove a row. Therefore, we need to call getAdapterPosition() to get the current position of a row.

```
public void onBindViewHolder(ViewHolder holder, final int position) {
    // get text from your data at this position
    final String current = data.get(position);
    // Set the footer to display corresponding text
    holder.binding.footer.setText("Footer: " + current);
    // Set onClickListener for header
    holder.binding.header.setOnClickListener(v -> {
        String text =
            "Position " + holder.getAdapterPosition() + " Clicked";
        holder.binding.header.setText(text);
    });
}
```

- getItemCount()
  This method simply returns the size of the data you passed in.

```
public int getItemCount() {
    return data.size();
}
```

All these methods will be called and managed by Recycler View, you don't need to worry about when and how they will be called.

Step 3. Implement MainActivity.java

All sub steps should be in the onCreate method. The code for initializing the binding is omitted. Refer to lesson 3 page 12 if needed.

3.1 Initialize Data

Let's set the data as an ArrayList of strings from "Test 0" to "Test 99":

```
ArrayList<String> input = new ArrayList<>();
for (int i = 0; i < 100; i++) {
    input.add("Test " + i);
}
```

3.2 Set up RecyclerView

Assume you've done initializing the binding. If you did a findViewById, simply remove "binding" before "recycler". See the comments for explanations of the code.

```
// Ensure each row has the same size
binding.recycler.setHasFixedSize(true);
// Set it as a list (linear layout)
RecyclerView.LayoutManager layoutManager = new LinearLayoutManager(this);
binding.recycler.setLayoutManager(layoutManager);
// Define and attach the recycler view to an adapter
RecyclerView.Adapter<MyAdapter.ViewHolder> mAdapter = new MyAdapter(input);
binding.recycler.setAdapter(mAdapter);
```

3.3 Set onSwiped

ItemTouchHelper is a utility class to add swipe to dismiss support to RecyclerView.

Let’s just do a SimpleCallback since we only want the onSwiped feature. onMove method is also required to implement, since we don’t need it, simply return false.

```
// Set onSwiped
ItemTouchHelper.SimpleCallback simpleItemTouchCallback =
        new ItemTouchHelper.SimpleCallback(0,
                ItemTouchHelper.LEFT | ItemTouchHelper.RIGHT) {
            @Override
            public boolean onMove(@NonNull RecyclerView recyclerView,
                                  @NonNull RecyclerView.ViewHolder viewHolder,
                                  @NonNull RecyclerView.ViewHolder target) {
                return false;
            }

            @Override
            public void onSwiped(RecyclerView.ViewHolder viewHolder,
                                 int swipeDir) {
                int position = viewHolder.getAdapterPosition();
                input.remove(position);
                mAdapter.notifyItemRemoved(position);
            }
        };
// Attach to the RecyclerView
ItemTouchHelper itemTouch = new ItemTouchHelper(simpleItemTouchCallback);
itemTouch.attachToRecyclerView(binding.recycler);
```

Step 4. Test your app

Run the app to see if the list displays correctly.

Before any swipe, when you click the header of one row, the position number should match the test number in footer.

After you swipe and delete any rows, the position number will be different from the test number in footer when the header is clicked. This is because the text in the footer won’t change, but after deleting row(s), the actual position of each remaining row will change.

Submission

- Push your project to a Bitbucket repository (name it “exercise2”) by the due date.
- Invite and share your Bitbucket repository the grader (edmond.lin@sjsu.edu) and the instructor (yan.chen01@sjsu.edu).
- Submit repository links, etc. by answering all the questions in the “Exercise 2 - RecyclerView” quiz on Canvas.
- Only your last submission before deadline will be graded based on the following criteria:
  2 pts if meets all requirements (the header of each row should be your name initially);
  1 pt if app failed/missing any requirement.