Mobile solutions

Kocsis Gergely

2019.01.12.

LiveData

Continue the project started at the last lesson.

Replace the `refreshScrollView` method by a more professional solution as a first step.

Use the LiveData class that is the tool of Android SDK to follow dynamic data. It is the implementation of the Observer pattern.

LiveData

Continue the project started at the last lesson.

Replace the `refreshScrollView` method by a more professional solution as a first step.

Use the LiveData class that is the tool of Android SDK to follow dynamic data. It is the implementation of the Observer pattern.

Change our DAO class so that the query method returns a LiveData typed result:

```
@Query("SELECT * FROM ShoppingList")
LiveData<List<ShoppingListItem>> getAllItems();
```

Our data became observable as a result.

LiveData

In out MainActivity-ben delete the `refreshScrollView` calls.
Set the observer in the OnCreate method.

```
shoppingListDatabase.shoppingListDAO().getAllItems().observe(this,
new Observer<List<ShoppingListItem>>() {
    @Override
    public void onChanged(List<ShoppingListItem> shoppingListItems) {
        itemsView.setText(shoppingListItems.toString());
    }
});
```

If everything went well, now it is working.

RecyclerView

Upgrade the UI as a second task.

In order to make our project more easy to be tested add a Clear button.

```
//MainActivity
public void clearDB(View view) {
    new Thread(new Runnable() {
        @Override
        public void run() {
            shoppingListDatabase.shoppingListDAO().clearDB();
        }
    }).start();
}

//ShoppingListDAO
@Query("DELETE FROM ShoppingList")
void clearDB();
```

RecyclerView

Replace the ScrollView by a RecyclerView.
Be careful, because will will get errors at the places of using the references to the elements just deleted.

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/my_recycler_view"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:layout_marginStart="16dp"
    android:layout_marginLeft="16dp"
    android:layout_marginTop="16dp"
    android:layout_marginEnd="16dp"
    android:layout_marginRight="16dp"
    android:layout_marginBottom="16dp"
    app:layout_constraintBottom_toTopOf="@+id/editText"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

RecyclerView

This RecyclerView will be the holder of the elements of our list. We need also a new XML file that describes how the layout of one item shoud look like.

Create this XML file at res→layout. The name should be item_layout.xml.

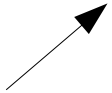
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="wrap_content">
            <TextView
                android:id="@+id/textView2"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Item name: " />

            <TextView
                android:id="@+id/textView"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:textAppearance="@style/TextAppearance.AppCompat.Large"/>
</LinearLayout>
```

RecyclerView

Add an Adapter class that puts the items to the RecyclerView

```
public class ViewAdatpter extends RecyclerView.Adapter<ViewAdatpter.MyViewHolder>
```



Inner class linking the XML file to the View

The inner class:

```
public static class MyViewHolder extends RecyclerView.ViewHolder {
    TextView tv;
    public MyViewHolder(View view){
        super(view);
        MyViewHolder.this.tv=view.findViewById(R.id.textView);
    }
}
```

RecyclerView

Add an Adapter class that puts the items to the RecyclerView

The data are stored in an instance variable. This is initialized at the time of instantiation. We use this variable in the A getItemCount method also.

```
private List<ShoppingListItem> data;

public ViewAdatpter(List<ShoppingListItem> data){
    this.data=data;
}

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

RecyclerView

Add an Adapter class that puts the items to the RecyclerView

The remaining two methods link the data to the holder class.

```
@NonNull
@Override
public MyViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    View v = LayoutInflater.from(parent.getContext()).inflate(R.layout.item_layout,
parent, false);
    return new MyViewHolder(tv);
}

@Override
public void onBindViewHolder(@NonNull MyViewHolder holder, int position) {
    holder.tv.setText(data.get(position).getItemName());
}
```

RecyclerView

Add an Adapter class that puts the items to the RecyclerView

Update the onCreate method as follows:l:

```
mRecyclerView = findViewById(R.id.my_recycler_view);
mRecyclerView.setLayoutManager(new LinearLayoutManager(MainActivity.this));

shoppingListDatabase.shoppingListDAO().getAllItems().observe(this, new
Observer<List<ShoppingListItem>>() {
    @Override
    public void onChanged(List<ShoppingListItem> shoppingListItems) {

        mRecyclerView.setAdapter(new ViewAdatpter(shoppingListItems));
    }
});
```

RecyclerView

Use CardView to pimp up your UI.

Make the UI interactive:

```
public static class MyViewHolder extends RecyclerView.ViewHolder implements
View.OnClickListener{
    TextView tv;
    final ViewAdatpter mAdapter;
    public MyViewHolder(View tv, ViewAdatpter va){
        super(tv);
        MyViewHolder.this.tv=tv.findViewById(R.id.textView);
        itemView.setOnClickListener(this);
        this.mAdapter=va;
    }

    @Override
    public void onClick(View view) {
        int mPosition = getLayoutPosition();

        ShoppingListItem element = data.get(mPosition);
        element.setItemName("done");
        data.set(mPosition, element);

        mAdapter.notifyDataSetChanged();
    }
}
```