Application Development Lab (CSE-2216)

Tutorial 2

Drag and Reorder Items in RecyclerView

Team Name: AudioPhile

Team Members:

1. Yeamin Kaiser (01)
2. Tasnim Bin Anwar (05)
3. Mohima Ahmed Joyee (42)

Introduction

RecyclerView makes it easy to efficiently display large sets of data. We supply the data and define how each item looks, and the RecyclerView library dynamically creates the elements when they're needed.

As the name implies, RecyclerView *recycles* those individual elements. When an item scrolls off the screen, RecyclerView doesn't destroy its view. Instead, RecyclerView reuses the view for new items that have scrolled on screen. This reuse vastly improves performance, improving your app's responsiveness and reducing power consumption.

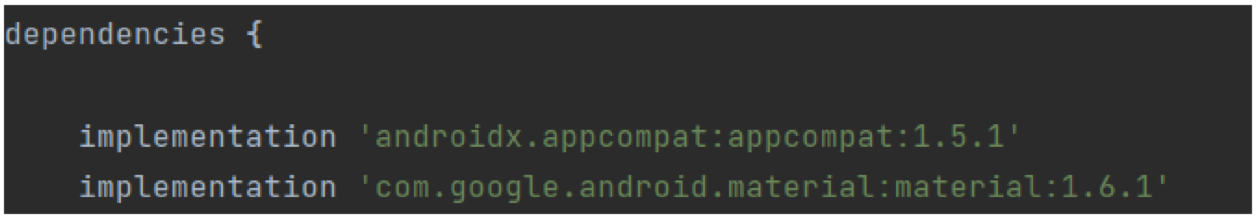
The following tutorial discusses and implements the Drag and Drop functionality over RecyclerView in an Android Application.

Creating a new project

1. Open Android Studio .
2. From the top left of the screen, select File > New > New Project.
3. Select Empty Activity and click on Next.
4. Name the project.
5. Choose the location to save the project or let it remain as it is.
6. For the language, we'll choose Java.
7. We're using the minimum SDK : API 21, Android 5.0.
8. Click on Finish to set up the project.

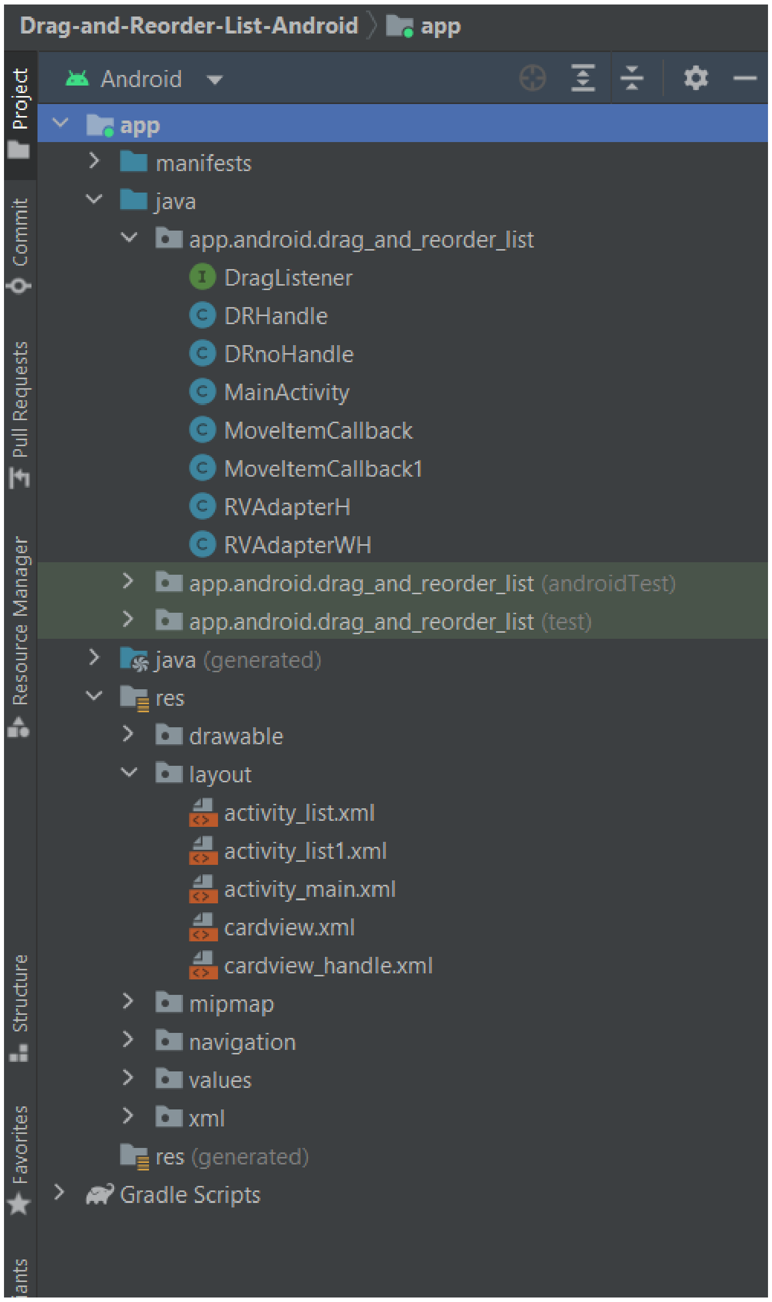
Add dependencies

We'll be using material design here to make our application look attractive.
Add the dependency from the following link:
material-components-android/getting-started.md at master



```
dependencies {

    implementation 'androidx.appcompat:appcompat:1.5.1'
    implementation 'com.google.android.material:material:1.6.1'
```

Project Structure



```
app
  manifests
  java
    app.android.drag_and_reorder_list
      DragListener
      DRHandle
      DRnoHandle
      MainActivity
      MoveItemCallback
      MoveItemCallback1
      RVAdapterH
      RVAdapterWH
    app.android.drag_and_reorder_list (androidTest)
    app.android.drag_and_reorder_list (test)
  java (generated)
  res
    drawable
    layout
      activity_list.xml
      activity_list1.xml
      activity_main.xml
      cardview.xml
      cardview_handle.xml
    mipmap
    navigation
    values
    xml
  res (generated)
Gradle Scripts
```

Code:

We will see two ways of implementing the Drag and Drop feature:

1. Without using a handle
2. Using a handle

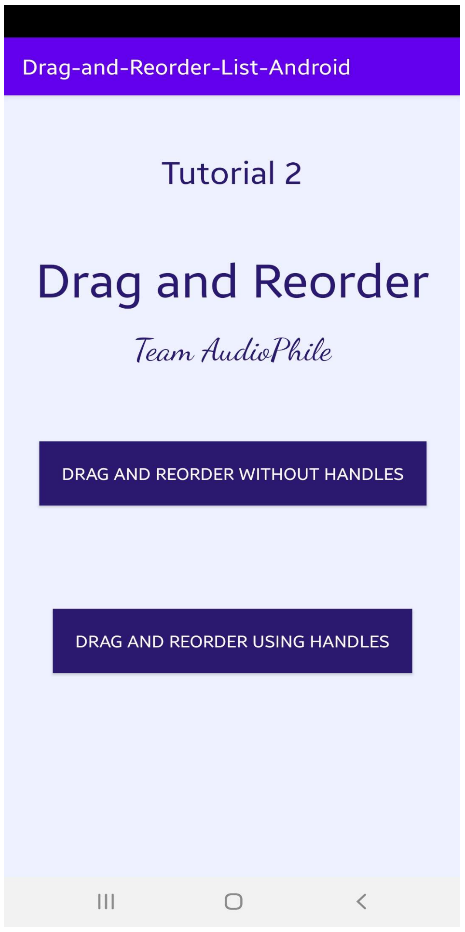
For implementing the feature using a handle, editing minor parts of the basic drag and reorder code is enough. We will see the basic implementation and the edits as we go along.

activity_main.xml

We used constraint layout to create an introductory page that shows the name of the tutorial, the name of our team and buttons to view the two ways the feature is implemented.

```
<?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"
   android:background="#EEF1FF"
   tools:context=".MainActivity">

   <TextView
       android:id="@+id/tv0"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:textSize="30sp"
       android:text="Tutorial 2"
       app:layout_constraintVertical_chainStyle="packed"
       android:textColor="#2b1972"
       app:layout_constraintBottom_toTopOf="@+id/tv1"
       app:layout_constraintEnd_toEndOf="parent"
       app:layout_constraintStart_toStartOf="parent"
       app:layout_constraintTop_toTopOf="parent"
       />

   <TextView
       android:id="@+id/tv1"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:fontFamily="sans-serif-condensed"
       android:text="Drag and Reorder"
       android:textColor="#2b1972"
       android:textSize="45sp"
       app:layout_constraintBottom_toTopOf="@+id/tv2"
       app:layout_constraintEnd_toEndOf="parent"
       app:layout_constraintStart_toStartOf="parent"
       app:layout_constraintTop_toTopOf="parent"
       app:layout_constraintVertical_chainStyle="packed" />

   <TextView
```

```
        android:id="@+id/tv2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:fontFamily="cursive"
        android:padding="20dp"
        android:text="Team AudioPhile"
        android:textColor="#2b1972"
        android:textSize="30sp"
        app:layout_constraintBottom_toTopOf="@+id/btn1"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/tv1" />

    <androidx.appcompat.widget.AppCompatButton
        android:id="@+id/btn1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="50dp"
        android:background="#2b1972"
        android:padding="20dp"
        android:text="Drag and Reorder without Handles"
        android:textColor="@color/white"
        android:textSize="16sp"
        app:layout_constraintBottom_toTopOf="@+id/btn2"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.492"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/tv2" />

    <androidx.appcompat.widget.AppCompatButton
        android:id="@+id/btn2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="50dp"
        android:background="#2b1972"
        android:padding="20dp"
        android:text="Drag and Reorder using handles"
        android:textColor="@color/white"
        android:textSize="16sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.494"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/btn1" />

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


Drag-and-Reorder-List-Android
Tutorial 2
Drag and Reorder
Team AudioPhile
DRAG AND REORDER WITHOUT HANDLES
DRAG AND REORDER USING HANDLES

MainActivity.java

This is the launcher activity and it defines the functionality of the buttons used in activity_main.xml. The goDRnoHandle() takes the user to the page where they can drag and reorder items pressing anywhere on the card, whereas the goDRHandle() takes the user to the page where they can drag and reorder items by only pressing on the specific handle defined.

```
public class MainActivity extends AppCompatActivity {
   Button button1, button2;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);
       getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
WindowManager.LayoutParams.FLAG_FULLSCREEN);
```

```
        goDRnoHandle();
        goDRHandle();
    }

    private void goDRnoHandle() {
        button1 = findViewById(R.id.btn1);
        button1.setOnClickListener(view -> {
            Intent i = new Intent(MainActivity.this, DRnoHandle.class);
            startActivity(i);
        });
    }

    private void goDRHandle() {
        button2 = findViewById(R.id.btn2);
        button2.setOnClickListener(view -> {
            Intent i = new Intent(MainActivity.this, DRHandle.class);
            startActivity(i);
        });
    }
}
```

cardview.xml

The card created for RecyclerView:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:id="@+id/cardView"
    android:layout_width="match_parent"
    android:layout_height="64dp"
    android:layout_margin="8dp"
    card_view:cardBackgroundColor="#FFD1D1"
    card_view:cardCornerRadius="10dp"
    card_view:cardElevation="2dp">

    <RelativeLayout
        android:id="@+id/relativeLayout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:paddingBottom="8dp"
        android:paddingLeft="8dp"
        android:paddingRight="8dp">

        <TextView
            android:id="@+id/title"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerVertical="true"

 android:textAppearance="@style/TextAppearance.Compat.Notification.Title" />

    </RelativeLayout>
 </androidx.cardview.widget.CardView>
```

This is the base implementation. An image can be added for showing the placement of the handle using the following:

```
<ImageView
   android:id="@+id/pan"
   android:padding="8dp"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_alignParentEnd="true"
   android:layout_centerVertical="true"
   android:src="@drawable/ic_pan" />
```

activity_list.xml

This is used to create the list of items for the RecyclerView:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   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=".DRnoHandle">

   <androidx.recyclerview.widget.RecyclerView
       android:id="@+id/recyclerView"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:orientation="vertical"
       app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager" />

</LinearLayout>
```

DRnoHandle.java

This is the main java class that is used to create the drag and reorder feature.

```java
package app.android.drag_and_reorder_list;

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

import android.os.Bundle;
import java.util.ArrayList;


public class DRnoHandle extends AppCompatActivity {

   RecyclerView recyclerView;
   RVAdapterWH Adapter;
   ArrayList<String> stringArrayList = new ArrayList<>();

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

       recyclerView = findViewById(R.id.recyclerView);

       populateRecyclerView();
   }

   private void populateRecyclerView() {
       stringArrayList.add("Red");
       stringArrayList.add("Orange");
       stringArrayList.add("Yellow");
       stringArrayList.add("Blue");
       stringArrayList.add("Green");
       stringArrayList.add("Indigo");
       stringArrayList.add("Violet");
       stringArrayList.add("Pink");
       stringArrayList.add("Black");
       stringArrayList.add("White");

       Adapter = new RVAdapterWH(stringArrayList);

       ItemTouchHelper.Callback callback =
               new MoveItemCallback1(Adapter);
       ItemTouchHelper touchHelper = new ItemTouchHelper(callback);
       touchHelper.attachToRecyclerView(recyclerView);

       recyclerView.setAdapter(Adapter);
   }
}
```

In this, we've populated a RVAdapterWH.java class (recyclerview adapter without handle) with an ArrayList of Strings. We've attached an instance of the MoveItemCallback.java class on the RecyclerView to start drag and drop. Let's look at each of these files.

MoveItemCallback1.java

```
public class MoveItemCallback1 extends ItemTouchHelper.Callback {

   private final ItemTouchHelperContract Adapter;

   public MoveItemCallback1(ItemTouchHelperContract adapter) {
       Adapter = adapter;
   }

   @Override
   public boolean isLongPressDragEnabled() {
       return true;
   }

   @Override
   public boolean isItemViewSwipeEnabled() {
       return false;
   }



   @Override
   public void onSwiped(@NonNull RecyclerView.ViewHolder viewHolder, int i) {

   }

   @Override
   public int getMovementFlags(@NonNull RecyclerView recyclerView, @NonNull
RecyclerView.ViewHolder viewHolder) {
       int dragFlags = ItemTouchHelper.UP | ItemTouchHelper.DOWN;
       return makeMovementFlags(dragFlags, 0);
   }

   @Override
   public boolean onMove(@NonNull RecyclerView recyclerView,
RecyclerView.ViewHolder viewHolder,
                         RecyclerView.ViewHolder target) {
       Adapter.onRowMoved(viewHolder.getAdapterPosition(),
target.getAdapterPosition());
       return true;
   }
```

```
    @Override
    public void onSelectedChanged(RecyclerView.ViewHolder viewHolder,
                                  int actionState) {


        if (actionState != ItemTouchHelper.ACTION_STATE_IDLE) {
            if (viewHolder instanceof RVAdapterWH.MyViewHolder) {
                RVAdapterWH.MyViewHolder myViewHolder=
                        (RVAdapterWH.MyViewHolder) viewHolder;
                Adapter.onRowSelected(myViewHolder);
            }

        }

        super.onSelectedChanged(viewHolder, actionState);
    }
    @Override
    public void clearView(@NonNull RecyclerView recyclerView,
                          @NonNull RecyclerView.ViewHolder viewHolder) {
        super.clearView(recyclerView, viewHolder);

        if (viewHolder instanceof RVAdapterWH.MyViewHolder) {
            RVAdapterWH.MyViewHolder myViewHolder=
                    (RVAdapterWH.MyViewHolder) viewHolder;
            Adapter.onRowClear(myViewHolder);
        }
    }

    public interface ItemTouchHelperContract {

        void onRowMoved(int fromPosition, int toPosition);
        void onRowSelected(RVAdapterWH.MyViewHolder myViewHolder);
        void onRowClear(RVAdapterWH.MyViewHolder myViewHolder);
    }
}
```

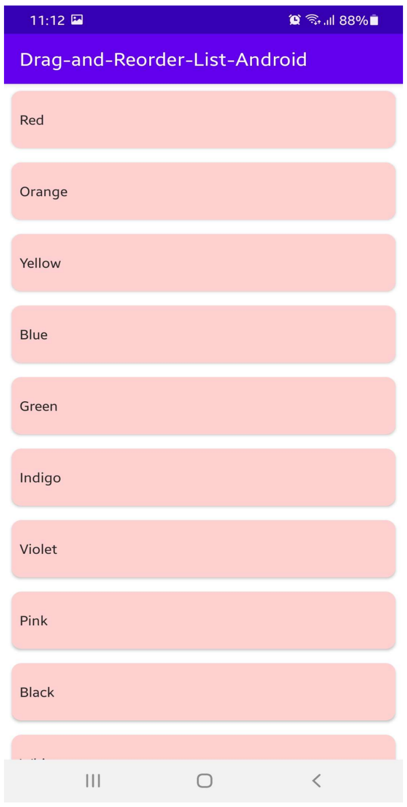
Here, we've defined an interface ItemTouchHelperContract. Each of its methods get called from the implemented methods of the ItemTouchHelper.Callback interface.

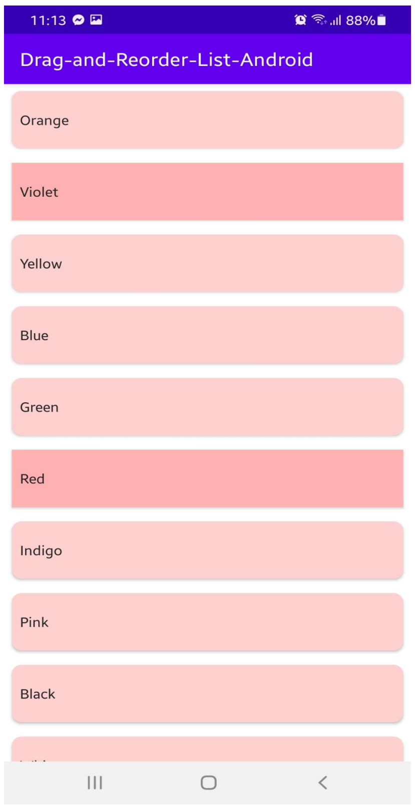
RVAdapterWH.java

```
public class RVAdapterWH extends
RecyclerView.Adapter<RVAdapterWH.MyViewHolder> implements
MoveItemCallback1.ItemTouchHelperContract {

    private ArrayList<String> data;
    public class MyViewHolder extends RecyclerView.ViewHolder {

        private TextView mTitle;
        View row;

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

            row = itemView;
            mTitle = itemView.findViewById(R.id.title);
        }
    }

    public RVAdapterWH(ArrayList<String> data) {
        this.data = data;
    }

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

    @Override
    public void onBindViewHolder(final MyViewHolder holder, int position) {
        holder.mTitle.setText(data.get(position));
    }


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


    @Override
    public void onRowMoved(int fromPosition, int toPosition) {
        if (fromPosition < toPosition) {
            for (int i = fromPosition; i < toPosition; i++) {
                Collections.swap(data, i, i + 1);
            }
        } else {
            for (int i = fromPosition; i > toPosition; i--) {
                Collections.swap(data, i, i - 1);
            }
        }
        notifyItemMoved(fromPosition, toPosition);
    }

    @Override
    public void onRowSelected(MyViewHolder myViewHolder) {
        myViewHolder.row.setBackgroundColor(Color.rgb(255,148,148));
    }

    @Override
    public void onRowClear(MyViewHolder myViewHolder) {
        myViewHolder.row.setBackgroundColor(Color.rgb(255,178,178));
    }
}
```

onRowMoved() defined in the Contract interface earlier gets called when the drag and drop is done. We swap the positions of the two rows present in the ArrayList and call notifyItemMoved to refresh the adapter.

This ends the tutorial for the general implementation of the drag and reorder feature where we can move items by pressing anywhere in the RecyclerView rows. Now we will see how to do the same by pressing only on a particular side of the RecyclerView rows.





Drag and Reorder using a handle:

In order to use a specific handle view, we need to go to MoveItemCallback.java and change the value of isLongPressDragEnabled() to false to disable the default drag and drop.

```
public boolean isLongPressDragEnabled() {
   return false;
}
```

We create an interface named DragListener.java and implement it in the DRHandle and pass it to the Adapter.

```
public interface DragListener {
   void requestDrag(RecyclerView.ViewHolder viewHolder);
}
```

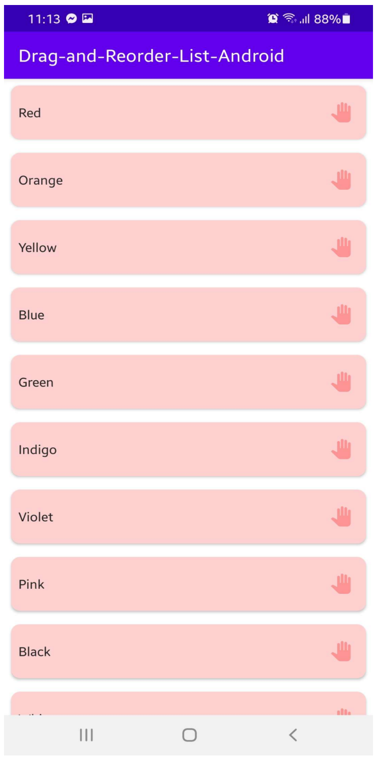
The implementation of the interface in DRHandle and the passing to Adapter is given below:

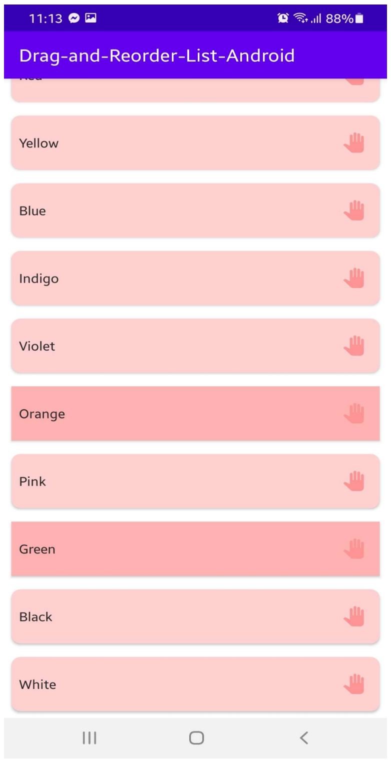
```
public void requestDrag(RecyclerView.ViewHolder viewHolder) {
   touchHelper.startDrag(viewHolder);
}

Adapter = new RVAdapterH(stringArrayList,this);
```

Next we do the following in the RVAdapterH.java

```
public void onBindViewHolder(final MyViewHolder holder, int position) {
   holder.mTitle.setText(data.get(position));

   holder.imageView.setOnTouchListener((v, event) -> {
       if (event.getAction() ==
               MotionEvent.ACTION_DOWN) {
           mAdapter.requestDrag(holder);
       }
       return false;
   });
}
```

The rest of the code is exactly the same as described for Drag and Drop without using a handle. Refer to the beginning of the tutorial to see the rest of the code snippets.





Possible Errors:

1. **Error:** Text not showing after running the app
   Solution: Mention the color of the text. Otherwise by default it will be shown in white. As a result the text can not be seen when the app is run on the phone or emulator.

2. **Error:** Button/Image/TextView not constrained when using constraint layout
   Solution: Add all the constraints properly to each element in the xml file.

3. **Error:** Activity not mentioned AndroidManifest.xml
   Solution: Check whether all the activities created are mentioned in the manifest file and also check the intent filter.

4. **Error:** Two icons created on app screen after installing app
   Solution: Check AndroiManifest.xml to see whether two activities have launcher enabled in the intent filter. Make sure only one activity is defined as the launcher activity.

5. **Error:** Not implementing all the necessary methods in the ItemTouchHelper.Callback Interface
   Solution: Make sure the following methods are implemented to ensure the feature is properly functioning –
   - ➢ **isLongPressDragEnabled** - Return true here to enable long press on the RecyclerView rows for drag and drop.
   - ➢ **isItemViewSwipeEnabled** - Enables or disables swipes. Here we are disabling it.
   - ➢ **getMovementFlags** - Pass the flags for the directions of drag and swipe. Since swipe is disabled, we pass 0 for it.
   - ➢ **onMove** - Set the code for the drag and drop.
   - ➢ **onSwipe** - Implement the code for swiping. We kept this empty in the current tutorial.
   - ➢ **onSelectedChanged** - Based on the current state of the RecyclerView and whether it's pressed or swiped, this method gets triggered. Here we can customize the RecyclerView row. For example, changing the background color.
   - ➢ **clearView** - This method gets triggered when the user interaction stops with the RecyclerView row.

Github Repository: https://github.com/Ahmed-Joyee/Drag-and-Reorder-List-Android

References:

Youtube Videos:

1. https://www.youtube.com/watch?v=TXAbYWZhpBQ&ab_channel=CodeWithMazn
2. https://www.youtube.com/watch?v=g6ySj807iTY&ab_channel=LearningWorldz

Blogs and websites:

1. https://androidapps-development-blogs.medium.com/drag-and-drop-reorder-in-recyclerview-android-2a3093d16ba2
2. https://www.digitalocean.com/community/tutorials/android-recyclerview-drag-and-drop