RecyclerView

Dr. Abdelkarim Erradi

CSE@QU

Outline

1. [Implementing a RecyclerView](#)

2. [Interacting with a RecyclerView](#)

Implementing a RecyclerView

What is a RecyclerView?

- RecyclerView is scrollable container for displaying a list of views

- A flexible container for efficiently displaying, and interacting with large sets of data

  o As user scrolls, views that leave the screen are recycled and reused to display new items

  o Efficient as it reuses a limited number of views

RecyclerView

Swipe Up
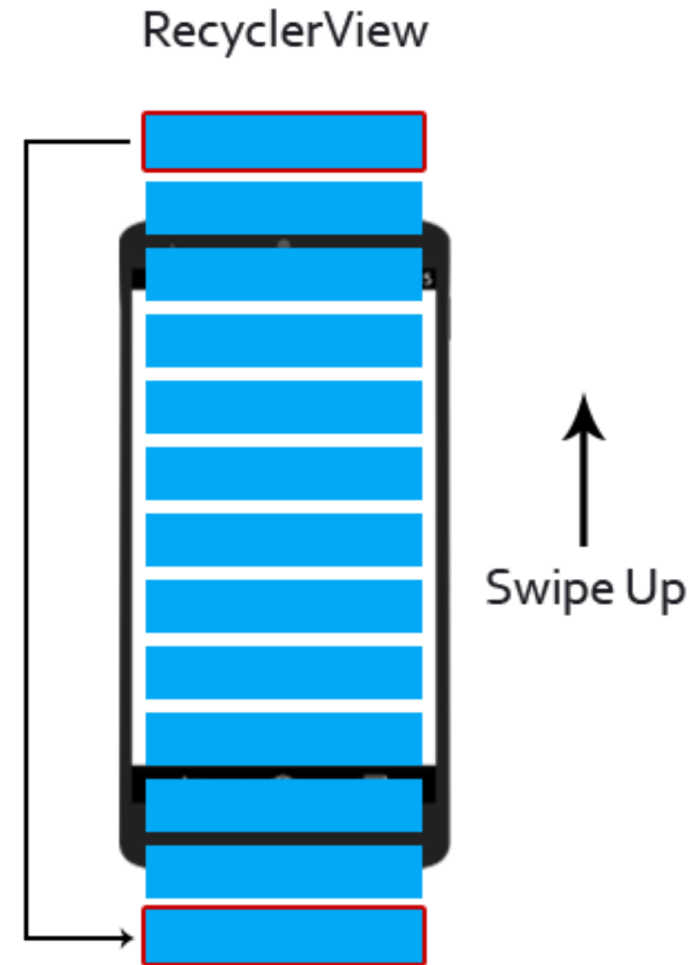
RecyclerView Key Elements
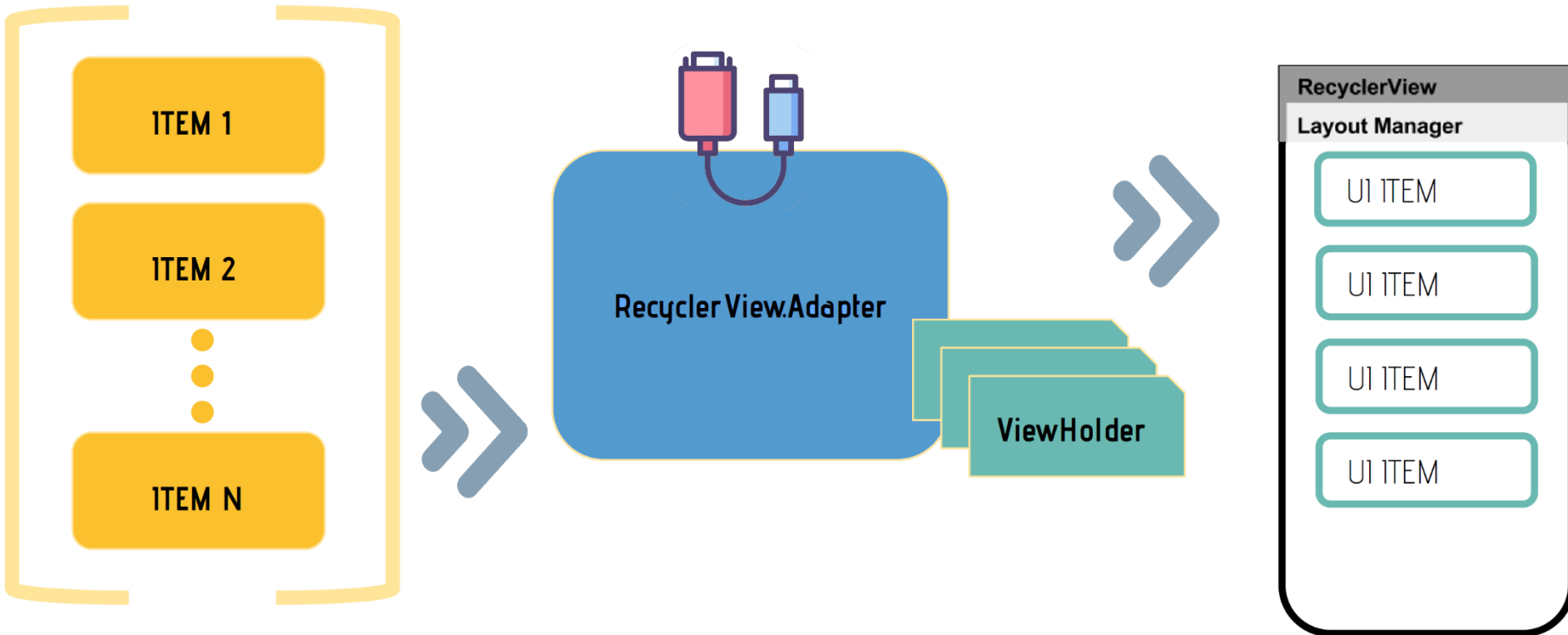
- **Adapter**: connects the *RecyclerView* to the data source
- **ViewHolder:** displays the data in the list item view

What is RecyclerView Adapter?

- In general an adapters helps **incompatible** interfaces work together

- RecyclerView connects the RecyclerView to the data source (typically a list of objects) and handles the view logic (such as filter, sort)

- It extends RecyclerView.Adapter and implements:

RecyclerView Adapter

1. Return the number of items
getItemCount()

2. Create new view instances
onCreateViewHolder()

3. Populate view items with data
onBindViewHolder()

What is a ViewHolder?

- To provide a new view, the RecyclerView either creates a new `ViewHolder` object to inflate the layout or it recycles an existing one

- The `ViewHolder` holds the view created from the list item XML Layout

- Used by the adapter to **prepare a View** with data for one list item

  - It assigns / updates the data in the view item. When a view is re-used, the previous data is overwritten

- Can have clickable elements

- Is placed in the RecyclerView by the layout manager

- Extends RecyclerView.ViewHolder

What is a layout manager?

- It **positions** the RecyclerView's items and **tells it when to recycle items** that have transitioned off-screen (i.e., no longer visible to the user)

- Built-in layout managers

  - `LinearLayoutManager:` Lays items out vertically or horizontally

  - `GridLayoutManager:` Lays items out in a grid

 - Others or your own! if you want extra customization

Implementation Steps Summary

1. Add **RecyclerView** to the layout

2. Create **item XML layout** to specify how individual items should be displayed

3. Implement `RecyclerView Adapter` that extends **RecyclerView.Adapter**

   o Connect the RecyclerView to the data source and handle the view logic

4. Implement `ViewHolder` that extends **RecyclerView.ViewHolder**

   o Holds the inflated view to be used to display individual items

5. In Activity `onCreate`, set the RecyclerView adapter and the LayoutManager

   o LayoutManager controls how individual views are laid out on the screen
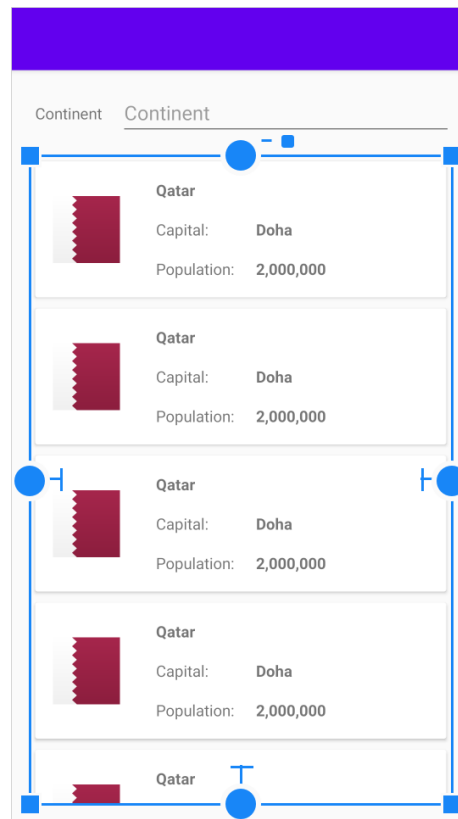
Country Explorer App
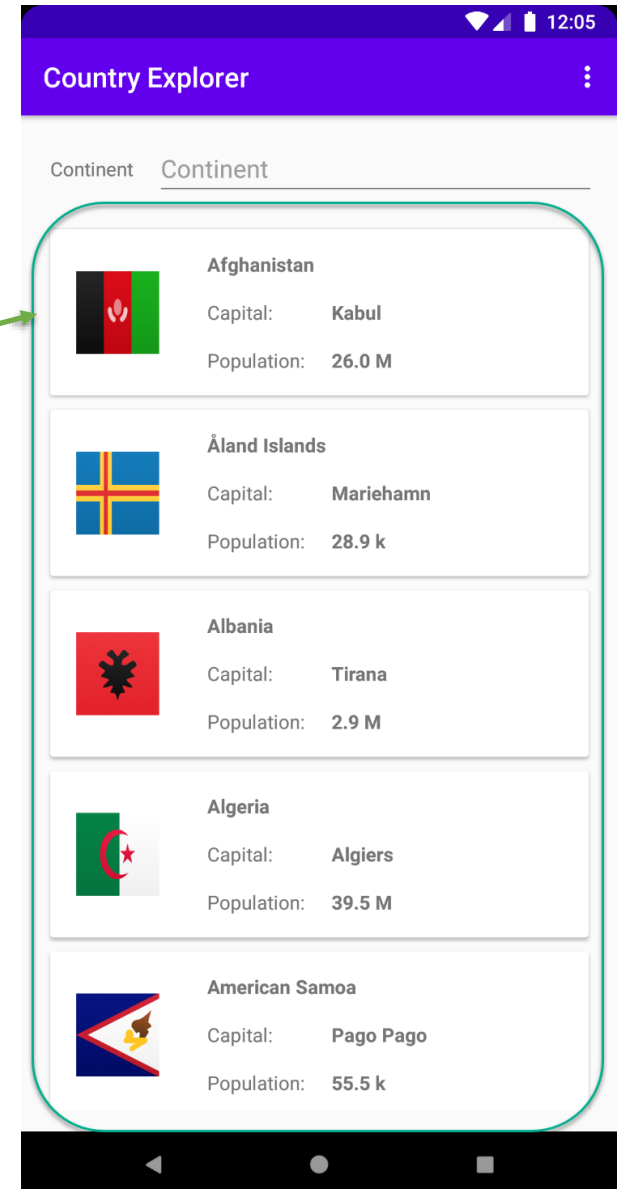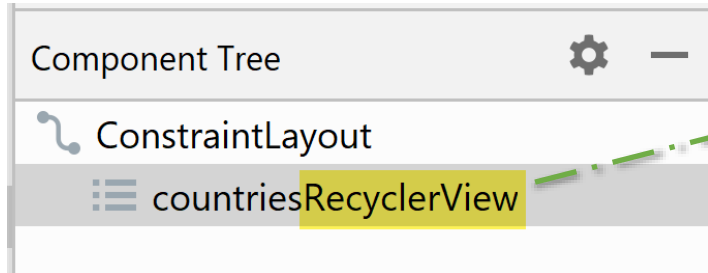
RecyclerView Summary
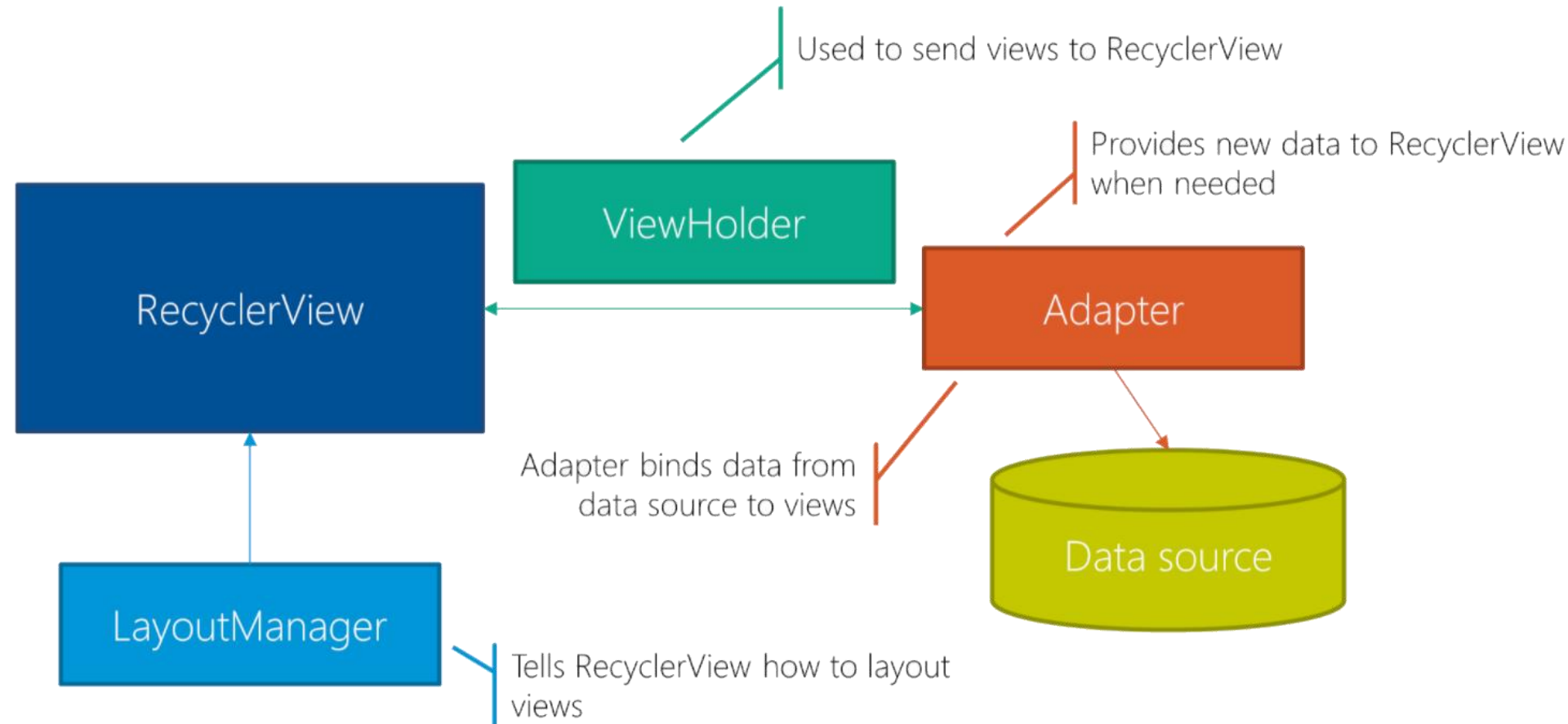
Tip: Showing List Item Views on the Layout Editor using `tools:listitem`

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/countriesRv"
    tools:listitem="@layout/list_item_country"/>
```

Interacting with RecyclerView

Filter

1. Add **SearchView** to the top toolbar by adding an item to menu_top_toolbar.xml

```xml
<item
    android:id="@+id/searchMi"
    android:icon="@drawable/ic_search"
    android:title="Search"
    app:actionViewClass="androidx.appcompat.widget.SearchView"
    app:showAsAction="always"
/>
```

2. Handle search by grabbing the searchView from the toolbar and handling the search in Activity onCreate

```kotlin
// 2.1 Grab the searchView from the toolbar
val searchView = topToolbar.findViewById<SearchView>(R.id.searchMi)
searchView.setBackgroundColor(Color.WHITE)
// 2.2. Handle search as the user types the search text
searchView.setOnQueryTextListener(searchHandler)
```

3. Write the searchHandler in the activity

```kotlin
private val searchHandler = object : SearchView.OnQueryTextListener {

    // Ignore and do not perform any special behavior here
    override fun onQueryTextSubmit(query: String?) = false


    // As the user types filter the list based on the search text

    override fun onQueryTextChange(searchText: String): Boolean {

        Log.i("CountryListActivity", "Query: $searchText")

        countryAdapter.filter(searchText)

        return true
    }
}
```

4. Filter the data list in the RecyclerView Adapter

```kotlin
fun filter(searchText: String) {
    countryFilteredList = if (searchText.isEmpty()) {
        countries
    } else {
        countries.filter { it.continent.contains(searchText, true) or
                           it.name.contains(searchText, true) or
                           it.capital.contains(searchText, true) or
                           it.code.contains(searchText, true)
        }.toMutableList()
    }
    notifyDataSetChanged()
}
```

Sort

1. First add a toolbar to the activity layout
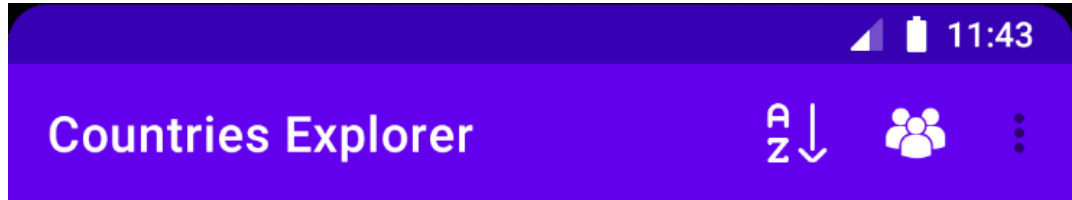

```xml
<com.google.android.material.appbar.AppBarLayout
    android:id="@+id/top_app_bar">

  <com.google.android.material.appbar.MaterialToolbar
        android:id="@+id/topToolbar"
        app:menu="@menu/menu_top_toolbar"
        app:title="@string/countries_list_title" />

</com.google.android.material.appbar.AppBarLayout>
```

```xml
<menu  ... >
    <!-- Setting app:showAsAction="never" will show the menu
    option under Overflow menu accessible from ellipses ... -->
    <item android:id="@+id/sortByNameMi"
        android:title="Sort by name"
        android:icon="@drawable/ic_sort"
        app:showAsAction="ifRoom" />

    <item android:id="@+id/sortByPopulationMi"
        android:title="Sort by population"
        android:icon="@drawable/ic_population"
        app:showAsAction="ifRoom" />

    <item android:id="@+id/sortByPopulationDescendingMi"
        android:title="Sort by population - Descending"
        app:showAsAction="never" />

</menu>
```

3. Handle toolbar menu item clicked

- Add `topToolbar.setOnMenuItemClickListener` to the activity onCreate method

```
topToolbar.setOnMenuItemClickListener { onToolbarMenuItemClicked(it) }

private fun onToolbarMenuItemClicked(menuItem: MenuItem): Boolean {
    val sortBy = when (menuItem.itemId) {
        R.id.sortByNameMi -> SortBy.NAME
        R.id.sortByPopulationMi -> SortBy.POPULATION
        R.id.sortByPopulationDescendingMi -> SortBy.POPULATION_DESC
        else -> null
    }
    countryAdapter.sort(sortBy!!)
    return true
}
```

4. RecyclerViewAdapter sorts the list

```kotlin
fun sort(sortBy: SortBy) {

  countries = when (sortBy) {

    SortBy.NAME -> countries.sortedBy { it.name }

    SortBy.POPULATION ->

            countries.sortedBy { it.population }

    SortBy.POPULATION_DESC ->

            countries.sortedByDescending { it.population }

  }
  // Caused a refresh of the RecyclerView
  notifyDataSetChanged()

}
```

Handling Item Clicked Event

1. Add a parameter to the Adapter to receive the **clickListener**

```kotlin
class CountryAdapter(private val countries: List<Country>,
                     private val clickListener: (Country) -> Unit)
    : RecyclerView.Adapter<RecyclerView.ViewHolder>()
```

2. Add the **clickListener** to every List Item View

```kotlin
inner class CountryViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
    fun bind(country: Country) {  ...
        itemView.setOnClickListener { clickListener(country) }
    }
}
```

3. In the Activity create a **clickListener** and pass it to the Adapter

```kotlin
countriesRv.adapter = CountryAdapter(countries, ::onCountryClicked)
...
private fun onCountryClicked(country : Country) {
    toast("Clicked: ${country.name}", Toast.LENGTH_LONG)
}
```

Handling Item Clicked Event

- In its constructor, the *Adapter* requires a reference to the data source and a **click handler**

  - Whenever the adapter is instructed by the *RecyclerView* to bind a new *ViewHolder*, the bind method **attaches the click listener** to the item view

```
itemView.setOnClickListener { clickListener(country) }
```

  - This allows individual items in the list to handle click events

- When the user later clicks on an item in the *RecyclerView*, the click listener is executed

  - As parameter, the listener gets the data list element (e.g., country object) that was clicked, to react accordingly
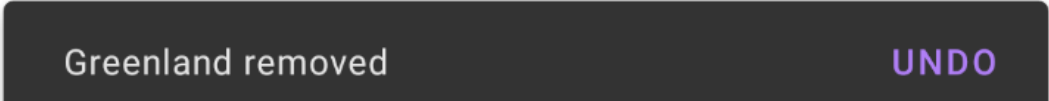
1. Swipe to Delete and Undo

```kotlin
/* 1. Create ItemTouchHelper.SimpleCallback and tell it what events to listen for.
It takes two parameters: One for drag directions and one for swipe directions.
We're only interested in swipe. Pass 0 to inform the callback not to respond to drag events. */
val swipeHandler = object :
    ItemTouchHelper.SimpleCallback(0, ItemTouchHelper.LEFT or ItemTouchHelper.RIGHT) {
    // Ignore and do not perform any special behavior here
    override fun onMove( ... ) = false

    // 2. onSwiped ask the RecyclerView adapter to delete the swiped item
    override fun onSwiped(viewHolder: RecyclerView.ViewHolder, swipeDirection: Int) {
        countryAdapter.deleteCountry(viewHolder)
    }
}

/* 3. Initialize ItemTouchHelper with the swipeHandler you defined,
        and then attach it to the RecyclerView. */
val itemTouchHelper = ItemTouchHelper(swipeHandler)
itemTouchHelper.attachToRecyclerView(countriesRv)
```

2. RecyclerViewAdapter deletes the swiped item

```kotlin
fun deleteCountry(viewHolder: RecyclerView.ViewHolder) {
    // Get the position of the item that was swiped
    val position = viewHolder.adapterPosition
    deletedCountry = countries[position]
    countries.removeAt(position)


    // Inform the RecyclerView adapter that an item has been
    // removed at a specific position.
    notifyItemRemoved(position)


    Snackbar.make(viewHolder.itemView, "${deletedCountry.name}
        removed", Snackbar.LENGTH_LONG).setAction("UNDO") {
            countries.add(position, deletedCountry)
            notifyItemInserted(position)
    }.show()
}
```

Greenland removed   UNDO

Add & Update

- Add & Update list items will be covered later

Resources

- RecyclerView codelab
  - https://codelabs.developers.google.com/codelabs/kotlin-android-training-recyclerview-fundamentals/

- Handling RecyclerView click event
  - https://www.andreasjakl.com/recyclerview-kotlin-style-click-listener-android/