BAKI ALİ NEFT MƏKTƏBİ
BAKU HIGHER OIL SCHOOL

Mobile development and security

Session 4

Karim Karimov
Lecturer

ListView

Displays a vertically-scrollable collection of views, where each view is positioned immediately below the previous view in the list.
To display a list, you can include a list view in your layout XML file:

```
<ListView
    android:id="@+id/list_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
/>
```

ListView

A list view is an **adapter view** that does not know the details, such as type and contents, of the views it contains. Instead list view requests views on demand from a **ListAdapter** as needed, such as to display new views as the user scrolls up or down.
In order to display items in the list, call ***setAdapter(android.widget.ListAdapter)*** to associate an adapter with the list.

```
val adapter = ArrayAdapter<String>(this, android.R.layout.simple_list_item_1,
myStringArray)

val listView: ListView = findViewById(R.id.listview)
listView.adapter = adapter
```

ListView

To display a more custom view for each item in your dataset, implement a ListAdapter. For example, extend **BaseAdapter** and create and configure the view for each data item in ***getView(...)***:

```
private class MyAdapter extends BaseAdapter {

    // override other abstract methods here

    @Override
    public View getView(int position, View convertView,
ViewGroup container) {
        if (convertView == null) {
            convertView =
getLayoutInflater().inflate(R.layout.list_item, container, false);
        }

        ((TextView) convertView.findViewById(android.R.id.text1))
                .setText(getItem(position));
        return convertView;
    }
}
```

RecyclerView

RecyclerView makes it easy to efficiently display large sets of data. You 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 onscreen. RecyclerView improves performance and your app's responsiveness, and it reduces power consumption.

Key classes

Several classes work together to build your dynamic list.

- **RecyclerView** is the ViewGroup that contains the views corresponding to your data. It's a view itself, so you add RecyclerView to 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 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.

Steps for implementing

If you're going to use RecyclerView, there are a few things you need to do. They are explained in detail in the following sections.

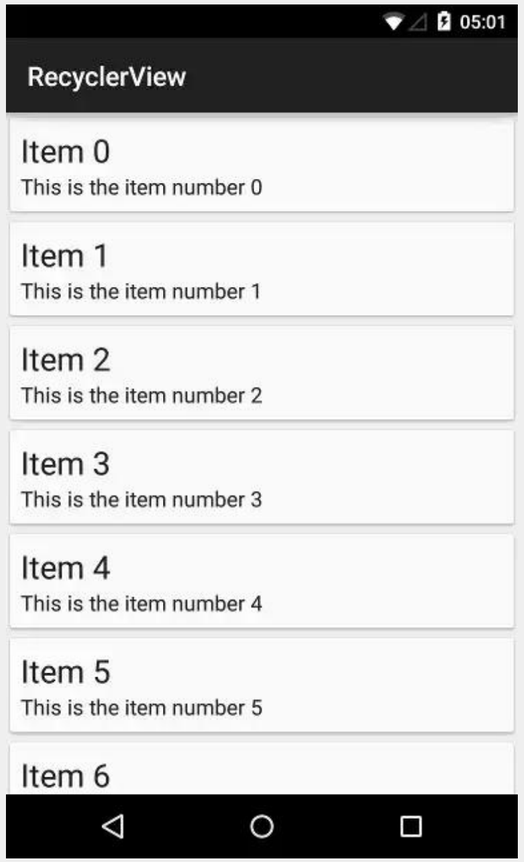
1. Decide **how the list or grid looks**. Ordinarily, you can use one of the RecyclerView library's standard layout managers.
2. Design how each element in the list looks and behaves. Based on this design, extend the **ViewHolder** class. Your version of ViewHolder provides all the functionality for your list items. Your view holder is a wrapper around a View, and that view is managed by RecyclerView.
3. Define the **Adapter** that associates your data with the ViewHolder views.

Plan your layout

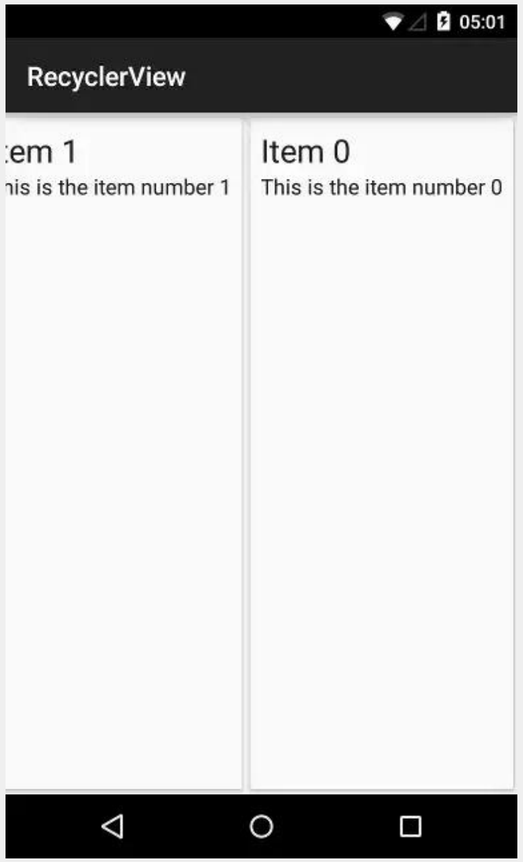
The items in your RecyclerView are arranged by a **LayoutManager** class. The RecyclerView library provides three layout managers, which handle the most common layout situations:

- **LinearLayoutManager** arranges the items in a one-dimensional list.
- **GridLayoutManager** arranges the items in a two-dimensional grid:
  - If the grid is arranged vertically, GridLayoutManager tries to make all the elements in each row have the same width and height, but different rows can have different heights.
  - If the grid is arranged horizontally, GridLayoutManager tries to make all the elements in each column have the same width and height, but different columns can have different widths.
- **StaggeredGridLayoutManager** is similar to GridLayoutManager, but it does not require that items in a row have the same height (for vertical grids) or items in the same column have the same width (for horizontal grids). The result is that the items in a row or column can end up offset from each other.

Linear layout manager

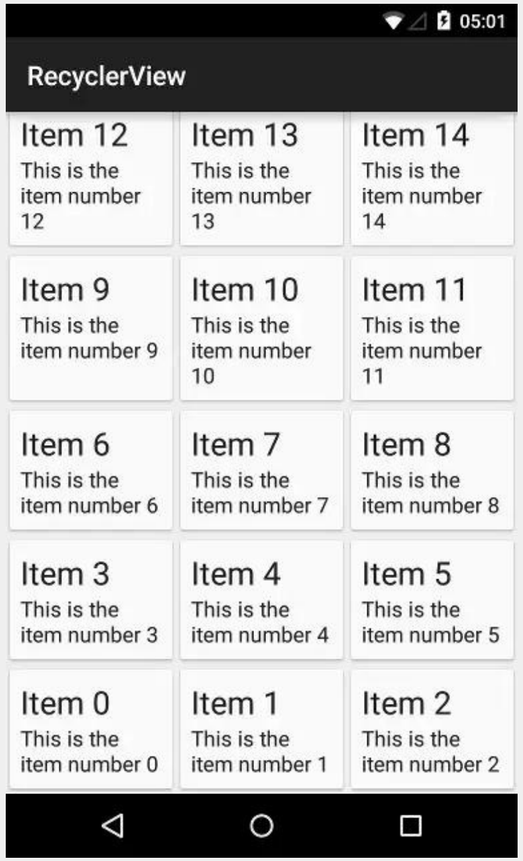


Vertical

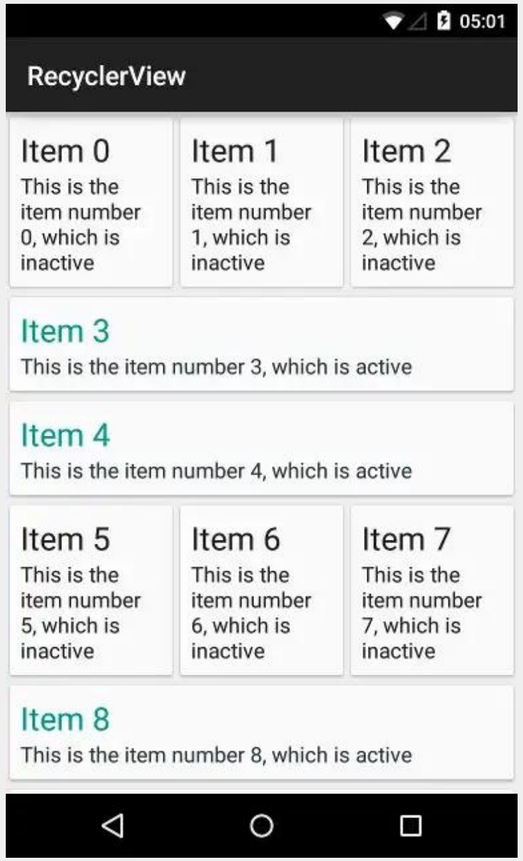


Horizontal

Grid layout manager



Staggered layout manager



Adapter and view holder

Once you determine your layout, you need to implement your Adapter and ViewHolder. These two classes work together to define how your data is displayed. The **ViewHolder** is a wrapper around a View that contains the layout for an individual item in the list. The **Adapter** creates ViewHolder objects as needed and also sets the data for those views. The process of associating views to their data is called binding.

Adapter and view holder

When you define your adapter, you override three key methods:

- **onCreateViewHolder()**: RecyclerView calls this method whenever it needs to create a new ViewHolder. The method creates and initializes the ViewHolder and its associated View, but does not fill in the view's contents—the ViewHolder has not yet been bound to specific data.
- **onBindViewHolder()**: RecyclerView calls this method to associate a ViewHolder with data. The method fetches the appropriate data and uses the data to fill in the view holder's layout. For example, if the RecyclerView displays a list of names, the method might find the appropriate name in the list and fill in the view holder's TextView widget.
- **getItemCount()**: RecyclerView calls this method to get the size of the dataset. For example, in an address book app, this might be the total number of addresses. RecyclerView uses this to determine when there are no more items that can be displayed.

Adapter and view holder

```
class CustomAdapter(private val dataSet: Array<String>) :
        RecyclerView.Adapter<CustomAdapter.ViewHolder>() {

    /**
     * Provide a reference to the type of views that you are using
     * (custom ViewHolder)
     */
    class ViewHolder(view: View) :
RecyclerView.ViewHolder(view) {
        val textView: TextView

        init {
            // Define click listener for the ViewHolder's View
            textView = view.findViewById(R.id.textView)
        }
    }
```

```
// Create new views (invoked by the layout manager)
 override fun onCreateViewHolder(viewGroup: ViewGroup, viewType: Int):
ViewHolder {
 // Create a new view, which defines the UI of the list item
 val view = LayoutInflater.from(viewGroup.context)
 .inflate(R.layout.text_row_item, viewGroup, false)

 return ViewHolder(view)
 }

 // Replace the contents of a view (invoked by the layout manager)
 override fun onBindViewHolder(viewHolder: ViewHolder, position: Int) {

 // Get element from your dataset at this position and replace the
 // contents of the view with that element
 viewHolder.textView.text = dataSet[position]
 }

 // Return the size of your dataset (invoked by the layout manager)
 override fun getItemCount() = dataSet.size
}
```

Adapter and view holder

The layout for the each view item is defined in an XML layout file, as usual. In this case, the app has a **text_row_item.xml** file like this:

```
<FrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="@dimen/list_item_height"
   android:layout_marginLeft="@dimen/margin_medium"
   android:layout_marginRight="@dimen/margin_medium"
   android:gravity="center_vertical">

   <TextView
      android:id="@+id/textView"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/element_text"/>
</FrameLayout>
```

Terms

- ***Adapter**:* A subclass of Adapter responsible for providing views that represent items in a data set.
- ***Position**:* The position of a data item within an *Adapter*.
- ***Index**:* The index of an attached child view as used in a call to getChildAt. Contrast with *Position*.
- ***Binding**:* The process of preparing a child view to display data corresponding to a *position* within the adapter.
- ***Recycle (view)**:* A view previously used to display data for a specific adapter position may be placed in a cache for later reuse to display the same type of data again later. This can drastically improve performance by skipping initial layout inflation or construction.
- ***Scrap (view)**:* A child view that has entered into a temporarily detached state during layout. Scrap views may be reused without becoming fully detached from the parent RecyclerView, either unmodified if no rebinding is required or modified by the adapter if the view was considered *dirty*.
- ***Dirty (view)**:* A child view that must be rebound by the adapter before being displayed.

References

- **ListView** - https://developer.android.com/reference/android/widget/ListView
- **RecyclerView** - https://developer.android.com/develop/ui/views/layout/recyclerview
- **RecyclerView ref doc** - https://developer.android.com/reference/androidx/recyclerview/widget/RecyclerView
- **Layout managers** - https://medium.com/@kamalvaid/recyclerview-orientation-types-9d15c3424d85

End of the session