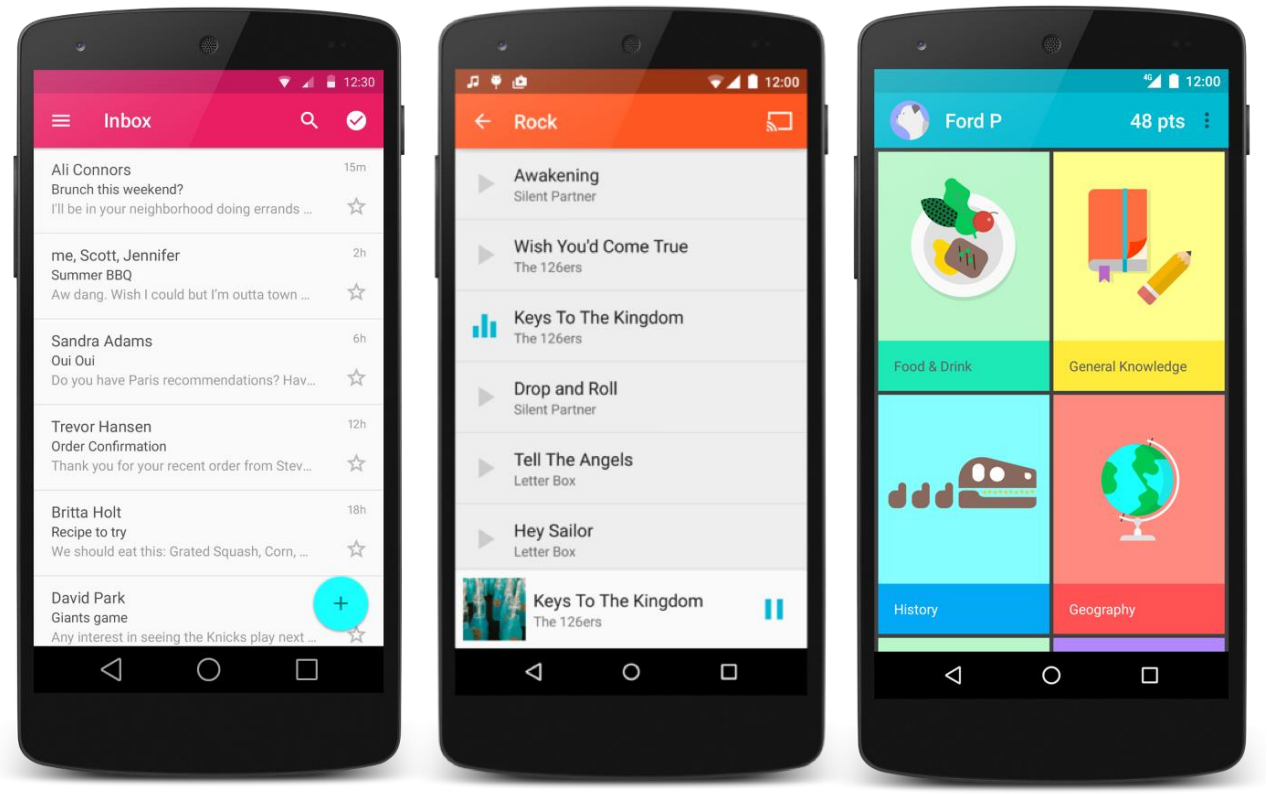
CS 426 – Mobile Application Development

Week 8: Advanced UI Components: RecyclerView

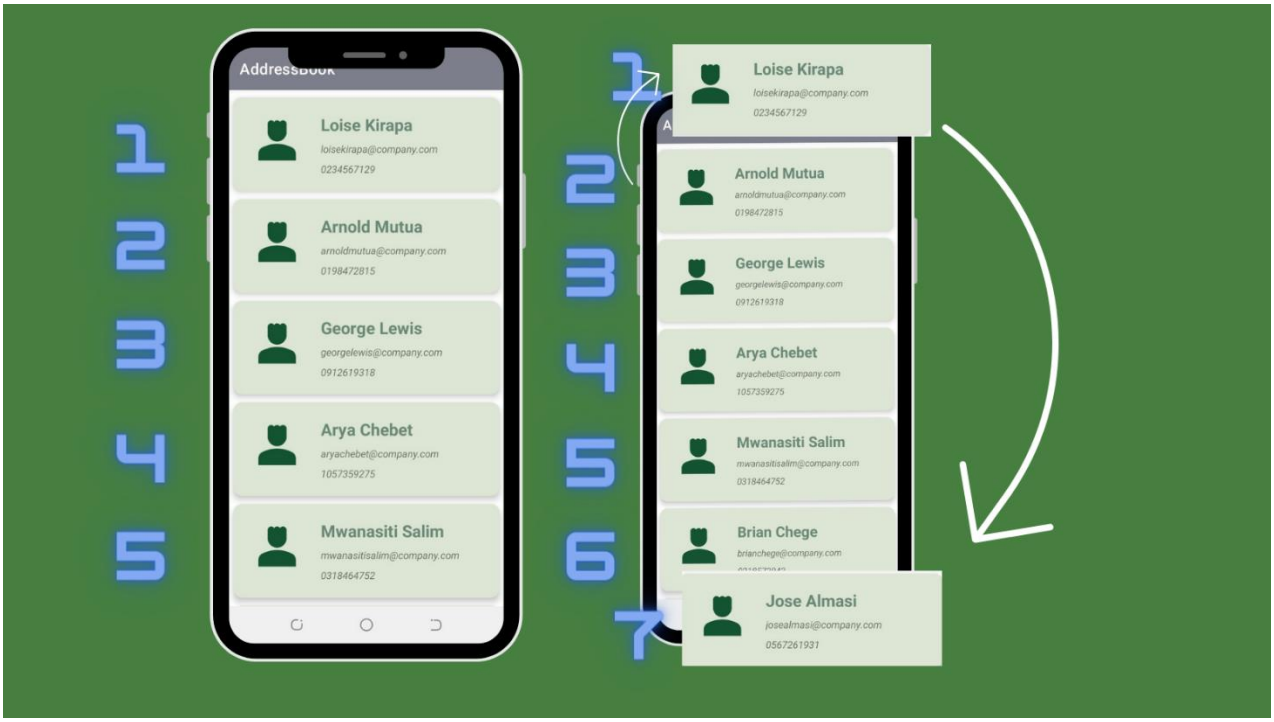
Instructor : Dr. Slim Namouchi
Date : 08 Nov 2025

Introduction

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, it reuses the view for new items that appear onscreen.



This mechanism improves performance, memory efficiency, and power consumption, especially with long or complex lists.

2. Key Classes and Concepts

Several classes work together to build a dynamic list:

Component	Role
RecyclerView	The ViewGroup that contains and displays all list elements. You add it to your XML layout like any other UI element.
ViewHolder	Represents an individual element (a single row or card). It holds references to the item's views, improving performance by avoiding repeated findViewById() calls.
Adapter	Connects your dataset to the ViewHolders. It creates them, binds data, and reports how many items exist.
LayoutManager	Decides how items are arranged (list, grid, staggered grid). You can use one of the built-in managers or implement your own.

These four parts form the core **RecyclerView architecture**.

3. How RecyclerView Works

1. RecyclerView asks the **Adapter** for a new ViewHolder when needed.
2. The **LayoutManager** places that ViewHolder in the correct position on screen.
3. When an item scrolls off screen, RecyclerView **reuses the same ViewHolder** for a new data element instead of creating one from scratch.
4. The Adapter's **onBindViewHolder()** updates the content of that ViewHolder.

This cycle repeats continuously as the user scrolls.

4. Implementation Steps

Step 1 – Plan your layout

The items in a RecyclerView are arranged by a LayoutManager.
The library provides three main implementations:

- **LinearLayoutManager** → one-dimensional vertical or horizontal list.
- **GridLayoutManager** → grid of equally sized items (rows/columns).
- **StaggeredGridLayoutManager** → grid where items may have variable size, producing a Pinterest-like staggered layout.

Design both :

- the **overall list layout** (the RecyclerView container), and
- the **individual item layout** (how each row looks).

Step 2 – Add RecyclerView to your XML

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recycler_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Step 3 – Create the item layout

Example (res/layout/text_row_item.xml) :

```
<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>
```

Step 4 – Implement Adapter and ViewHolder

These two classes work together to define how your data is displayed.

```
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
```

```
}
```

Explanation:

- onCreateViewHolder() → Inflates the XML layout and returns a new ViewHolder.
- onBindViewHolder() → Updates the ViewHolder's contents with the data for the given position.
- getItemCount() → Tells RecyclerView how many items are available.

This design ensures maximum efficiency through view recycling.

Step 5 – Use RecyclerView in your Activity

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val dataset = arrayOf("January", "February", "March")
        val customAdapter = CustomAdapter(dataset)

        val recyclerView: RecyclerView = findViewById(R.id.recycler_view)
        recyclerView.layoutManager = LinearLayoutManager(this)
        recyclerView.adapter = customAdapter

    }

}
```

5. Enable Edge-to-Edge Display (optional for modern apps)

RecyclerView can draw under the system bars (status/navigation).

To do this safely:

```
ViewCompat.setOnApplyWindowInsetsListener(
    findViewById(R.id.my_recycler_view)
) { v, insets ->
    val innerPadding = insets.getInsets(
        WindowInsetsCompat.Type.systemBars()
                or WindowInsetsCompat.Type.displayCutout()
        // If using EditText, also add
        // "or WindowInsetsCompat.Type.ime()" to
        // maintain focus when opening the IME
    )
    v.setPadding(
        innerPadding.left,
        innerPadding.top,
        innerPadding.right,
```

```
            innerPadding.bottom)
        insets
}
```

Corresponding XML :

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

6. Advanced Topics and Customization

6.1 Item Decoration

Use RecyclerView.ItemDecoration to draw dividers, spacing, or section headers.

- **Item Animator**

Use RecyclerView.ItemAnimator to customize animations when items are added, removed, or moved.

- **DiffUtil for Efficient Updates**

Use DiffUtil to calculate list differences and update only changed items instead of reloading everything (notifyDataSetChanged() is costly).

- **Multiple View Types**

Override getItemViewType() in the adapter if you need headers, footers, or mixed layouts.

- **State Restoration**

Preserve scroll position and item state across configuration changes with:

adapter.stateRestorationPolicy =

RecyclerView.Adapter.StateRestorationPolicy.PREVENT_WHEN_EMPTY

Best Practices

- Keep onBindViewHolder() light — no heavy logic or I/O.
- Use ViewBinding to access views type-safely.
- If items have stable IDs, call setHasStableIds(true) and override getItemId() for better animations.

7. Troubleshooting

Issue	Likely Cause	Solution
RecyclerView shows nothing	No LayoutManager set	Set a LinearLayoutManager or similar
Scrolling is laggy	Work done in onBindViewHolder()	Move heavy work to background
Duplicate or wrong data	Data list not updated correctly	Update the dataset and call proper notify methods
Scroll position resets	State not saved	Enable state restoration or store position manually

8. Summary

- RecyclerView = modern, flexible replacement for ListView.
- Core components: **RecyclerView**, **Adapter**, **ViewHolder**, **LayoutManager**.
- It dynamically reuses item views → huge performance improvement.
- LayoutManagers define arrangement (Linear, Grid, StaggeredGrid).
- Adapters control data binding.
- Advanced APIs : DiffUtil, ItemDecoration, ItemAnimator.

RecyclerView is the foundation for almost all modern Android list-based UIs (chats, feeds, galleries, task lists, etc.).

📚 References

- Create dynamic lists with RecyclerView — Android Developers
- RecyclerView sample (Kotlin)
- Manage RecyclerView state
- Using the RecyclerView — CodePath guide