ANDROID

Agenda

- List View
- Recycle View

RECYCLERVIEW

LISTVIEW

Introduction

In Android development, we often need to display a large number of items in a structured way. Instead of manually creating multiple views, ListView and RecyclerView provide an efficient way to handle lists of data while optimizing memory and performance. Both are used in applications like contact lists, messaging apps, and e-commerce product catalogs to provide a smooth user experience with minimal resource consumption.

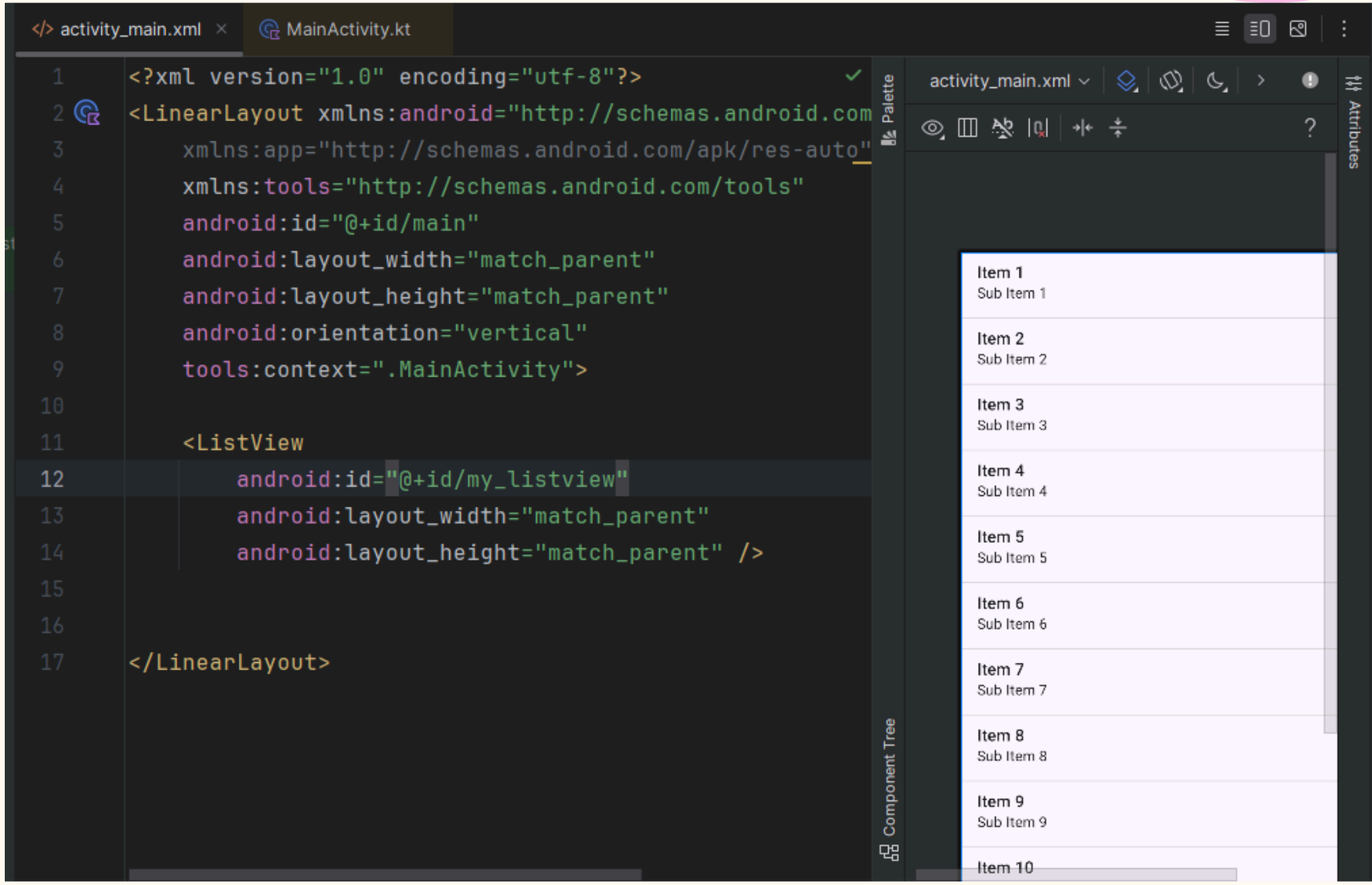
What is ListView

- A UI component in Android used to display a vertically scrollable list of items.
- **Used in:**
    - Displaying lists of contacts, messages, or products.
    - Showing dynamic data from an API or database.
    - Creating simple menus or navigation lists.

LISTVIEW COMPONENTS

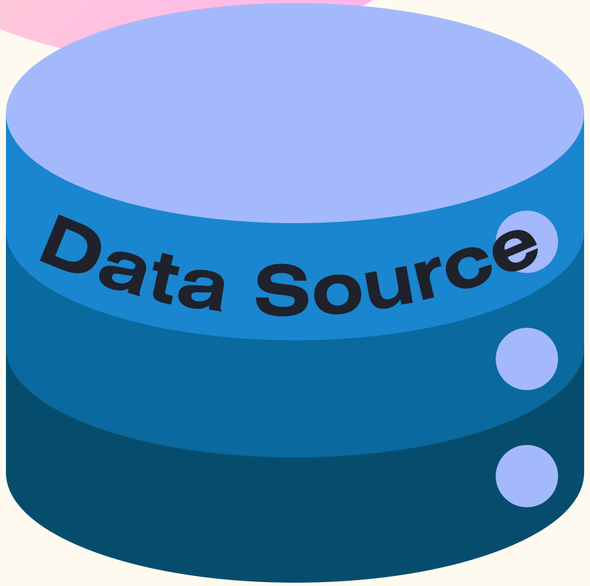
ListView

The main container that displays the list of items.



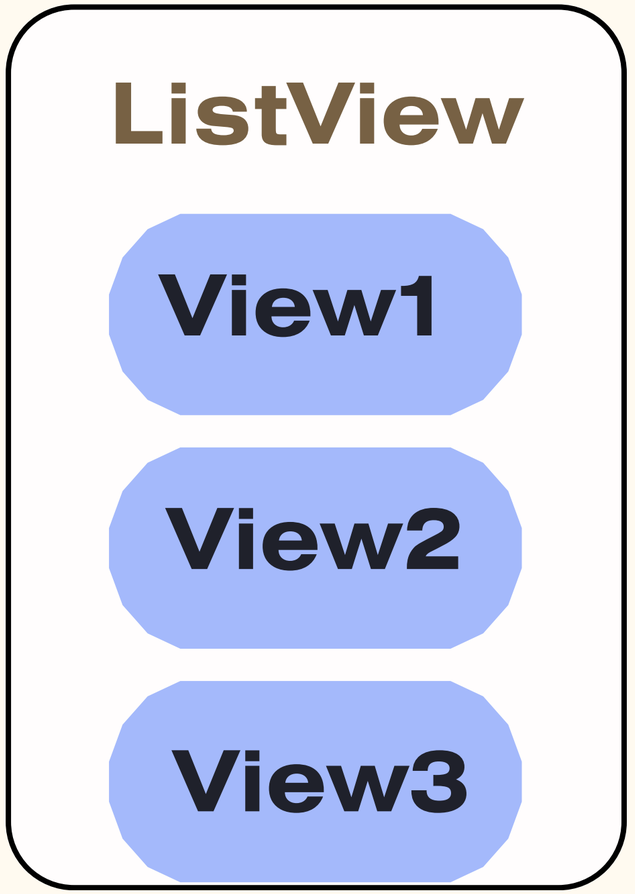
Adapter

What is it !??



LIST ADAPTER

Create a view from data



Adapter

- An adapter is a **bridge** between the data source (a list or database) and the UI component (ListView or RecyclerView).
- It binds data to views and manages how data is displayed.

Types of Adapter

ArrayAdapter	• used if the model data is an array of objects
BaseAdapter	• used if the model data is some other in-memory data structure
SimpleAdapter	• used for static data.
HeaderViewListAdapter	• used when a ListView has header views.

How to create simple listView ?!

- Define data source

```
val listContent= arrayListOf("one","two","three","four","five","six","seven","eight","nine","ten")
```

- Define Adapter

```
val adapter : ArrayAdapter<String> =
    ArrayAdapter( context: this,android.R.layout.simple_list_item_1, listContent)
val listView:ListView=findViewById(R.id.list_view)
listView.adapter=adapter
```

How to create Custom listView ?!

- **Define Single-row in separate Layout**

```
    <ImageView
        android:id="@+id/imageView"
        android:layout_width="135dp"
        android:layout_height="163dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:src="@drawable/cat"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="TextView"
        android:textSize="20sp"
        app:layout_constraintBottom_toBottomOf="@+id/imageView"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toEndOf="@+id/imageView"
        app:layout_constraintTop_toTopOf="@+id/imageView" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

How to create Custom listView ?!

- **Tell the Adapter how to display data**

```
val adapter: ArrayAdapter<String> =
    ArrayAdapter(applicationContext,
                 R.layout.single_row,
                 R.id.textView,
                 values)
listView.adapter = adapter
```

Layout Folder

Text View Id

How to create Complex listView ?!

1. **Create a Class that Extends ArrayAdapter**
2. **Define a Constructor that Calls a Superclass Constructor**
3. **Override the getView() Method**

How to create Complex listView ?!

```
class MyAdapter( context: Context,val array: Array<String>,val images:Array<Int>):ArrayAdapter<String>(context,R.layout.row,array){
    override fun getView(position: Int, convertView: View?, parent: ViewGroup): View {
        val row:View= LayoutInflater.from(context).inflate(R.layout.row,parent, attachToRoot: false)

        val textView:TextView=row.findViewById(R.id.textView)
        val imageView:ImageView=row.findViewById(R.id.imageView)

        Log.d( tag: "Safwa", msg: "Inflate ${position}")
        textView.text=array[position]
        imageView.setImageResource(images[position])
        return row
    }
}
```

There is a problem the records increased, how to solve it ?!

ViewHolder To reuse the unused views

```
class ViewHolder(var view:View){
    private var textView:TextView?=null
    private var imageView:ImageView?=null

    fun getText():TextView?{
        return textView?:view.findViewById(R.id.textView)
    }
    fun getImage():ImageView?{
        return imageView?:view.findViewById(R.id.imageView)
    }

}
```

Update your adapter

```
class MyAdapter( context: Context,val array: Array<String>,val images:Array<Int>):ArrayAdapter<Any?>(context,R.layout.row,array){
    override fun getView(position: Int, convertView: View?, parent: ViewGroup): View {
        var rowView:View? =convertView
        val viewHolder:ViewHolder
        if (rowView==null){
            rowView= LayoutInflater.from(context).inflate(R.layout.row,parent, attachToRoot: false)
            viewHolder=ViewHolder(rowView)
            rowView.tag=viewHolder
            Log.d( tag: "Safwa", msg: "First inflation")

        }else{
            viewHolder=rowView.tag as ViewHolder
            Log.d( tag: "Safwa", msg: "reuse")

        }


        viewHolder.getText()?.text=array[position]
        viewHolder.getImage()?.setImageResource(images[position])
        return rowView !!
    }
```

RECYCLERVIEW

What is RecyclerView

- RecyclerView is a more advanced and flexible version of ListView.
- Introduced in Android Lollipop (API 21).
- Displaying large datasets with complex layouts.
- Efficient memory usage with view recycling.
- Built-in support for item animations and custom layouts.
- Supports complex layouts.
- Provides smooth scrolling with view recycling.

RecyclerView

How to create RecyclerView:

1. Define a model class to use as the data source
2. Add a RecyclerView to your layout to display the items
3. Create a custom row layout XML file to visualize the item
4. Create a RecyclerView.Adapter and ViewHolder to render the item
5. Bind the adapter to the data source to populate the RecyclerView

RecyclerView continue...

We add recyclerview in our activity in

```
<FrameLayout 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:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/card_view"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager" />
</FrameLayout>
```

RecyclerView continue...

Then as we do in listview will do here with some different

We implement adapter with viewHolder, Why?

RecyclerView does not handle displaying items on its own. Instead, it delegates this task to an Adapter, which binds the data to the list and optimizes performance using a ViewHolder.

What is ViewHolder

- A ViewHolder is a design pattern used to store references to views inside a RecyclerView item.
- Without ViewHolder, RecyclerView would call findViewById() repeatedly, which is inefficient and slows performance.
- ViewHolder holds references to item views and reuses them, reducing the number of findViewById() calls.

RecyclerView continue...

RecyclerView.Adapter Methods :

- **onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder**
  This method creates a new ViewHolder when RecyclerView needs one.
  - Inflates the list_item.xml layout.
  - Wraps it inside a ViewHolder.
  - Returns the ViewHolder to RecyclerView.
- **onBindViewHolder(holder: ViewHolder, position: Int)** This method binds data to an existing ViewHolder.
  - Gets the item at the current position.
  - Updates the TextView inside ViewHolder with the item's data.
- **getItemCount(): Int**
  - This method returns the total number of items in the dataset.

RecyclerView continue...

```
class RecyclerViewAdapter(val list: List<Number>):RecyclerView.Adapter<RecyclerViewAdapter.viewHolder>(){
    class viewHolder(row: View):RecyclerView.ViewHolder(row){
        val text=row.findViewById<TextView>(R.id.textView)
        val image:ImageView=row.findViewById(R.id.imageView)

    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): viewHolder {
        val layot=LayoutInflater.from(parent.context).inflate(R.layout.row,parent, attachToRoot: false)
        return viewHolder(layot)
    }

    override fun getItemCount(): Int =list.size

    override fun onBindViewHolder(holder: viewHolder, position: Int) {
       holder.text.text=list[position].text
        holder.image.setImageResource(list[position].image)
    }

}
```

RecyclerView continue...

Layout Managers:

A LayoutManager in RecyclerView is responsible for positioning the items and handling scrolling behavior.

RecyclerView does not position items on its own; it delegates this job to a LayoutManager.

It determines how items are arranged (linear, grid, staggered, etc.).

ListView vs. RecyclerView

Feature	ListView	RecyclerView
Performance	Slower (frequent findViewById calls)	Faster (efficient ViewHolder pattern)
Layout Support	Only supports vertical lists	Supports Linear, Grid, and Staggered layouts
View Customization	Less flexible	Highly customizable
Use Case	Simple lists with fewer items	Complex lists with better performance

Thank You