**✅ Section 17: Planets App – Mastering ListView & Adapters**

**🔑 Key Concepts**

1. Custom ListView with item layout (ImageView + TextViews)
2. Model class for item data
3. Custom ArrayAdapter subclass
4. ViewHolder pattern for performance
5. Efficient view recycling with convertView
6. LayoutInflater to inflate XML
7. Dynamic list with ArrayList
8. Handling item clicks with OnItemClickListener
9. UI → Adapter → View → Data mapping flow

**🧱 1. Custom List Item Layout (item\_list\_layout.xml)**

**Contains:**

* ImageView: Planet icon
* TextView: Planet name
* TextView: Moon count

<androidx.constraintlayout.widget.ConstraintLayout

... >

<!-- Planet Image -->

<ImageView

android:id="@+id/planet\_image"

android:layout\_width="64dp"

android:layout\_height="64dp"

android:layout\_marginStart="16dp"

android:layout\_marginTop="16dp"

android:layout\_marginBottom="16dp"

android:contentDescription="Planet image"

app:layout\_constraintStart\_toStartOf="parent"

app:layout\_constraintTop\_toTopOf="parent" />

<!-- Planet Name -->

<TextView

android:id="@+id/planet\_name"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textSize="18sp"

android:textStyle="bold"

android:layout\_marginStart="16dp"

android:layout\_marginTop="16dp"

app:layout\_constraintStart\_toEndOf="@id/planet\_image"

app:layout\_constraintTop\_toTopOf="parent" />

<!-- Moon Count -->

<TextView

android:id="@+id/moon\_count\_text"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_marginStart="16dp"

android:layout\_marginTop="8dp"

app:layout\_constraintStart\_toEndOf="@id/planet\_image"

app:layout\_constraintTop\_toBottomOf="@id/planet\_name" />

</androidx.constraintlayout.widget.ConstraintLayout>

**🧩 2. Model Class: Planet.java**

Holds data for each planet item.

public class Planet {

private String planetName;

private String moonCount;

private int planetImage; // resource ID (R.drawable.earth)

public Planet(String planetName, String moonCount, int planetImage) {

this.planetName = planetName;

this.moonCount = moonCount;

this.planetImage = planetImage;

}

// Getters

public String getPlanetName() { return planetName; }

public String getMoonCount() { return moonCount; }

public int getPlanetImage() { return planetImage; }

}

🧠 **Best Practice**: Store resource ID (int) not the Bitmap directly.

**🛠️ 3. Custom Adapter: MyCustomAdapter.java**

Bridges ArrayList<Planet> and ListView. Uses ViewHolder pattern for performance.

public class MyCustomAdapter extends ArrayAdapter<Planet> {

private ArrayList<Planet> planetsArrayList;

private Context context;

// Adapter constructor

public MyCustomAdapter(ArrayList<Planet> planetsArrayList, Context context) {

super(context, R.layout.item\_list\_layout, planetsArrayList);

this.planetsArrayList = planetsArrayList;

this.context = context;

}

// ViewHolder class (static to prevent memory leaks)

private static class MyViewHolder {

ImageView planetImage;

TextView planetName;

TextView moonCount;

}

@Override

public View getView(int position, View convertView, ViewGroup parent) {

Planet planet = getItem(position); // Current planet item

MyViewHolder viewHolder;

if (convertView == null) {

// Inflate layout for new row

LayoutInflater inflater = LayoutInflater.from(context);

convertView = inflater.inflate(R.layout.item\_list\_layout, parent, false);

// Initialize ViewHolder and bind views

viewHolder = new MyViewHolder();

viewHolder.planetImage = convertView.findViewById(R.id.planet\_image);

viewHolder.planetName = convertView.findViewById(R.id.planet\_name);

viewHolder.moonCount = convertView.findViewById(R.id.moon\_count\_text);

// Tag ViewHolder for reuse

convertView.setTag(viewHolder);

} else {

// Reuse existing ViewHolder

viewHolder = (MyViewHolder) convertView.getTag();

}

// Populate data into views

viewHolder.planetName.setText(planet.getPlanetName());

viewHolder.moonCount.setText(planet.getMoonCount());

viewHolder.planetImage.setImageResource(planet.getPlanetImage());

return convertView;

}

}

**🎯 4. MainActivity.java – Bind Adapter to ListView**

public class MainActivity extends AppCompatActivity {

ListView listView;

ArrayList<Planet> planetsArrayList;

MyCustomAdapter adapter;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

// Step 1: Find ListView

listView = findViewById(R.id.list\_view);

// Step 2: Create data source

planetsArrayList = new ArrayList<>();

planetsArrayList.add(new Planet("Earth", "1", R.drawable.earth));

planetsArrayList.add(new Planet("Mars", "2", R.drawable.mars));

planetsArrayList.add(new Planet("Jupiter", "79", R.drawable.jupiter));

// Add more as needed...

// Step 3: Initialize Adapter

adapter = new MyCustomAdapter(planetsArrayList, getApplicationContext());

// Step 4: Bind Adapter to ListView

listView.setAdapter(adapter);

// Step 5: Set item click listener

listView.setOnItemClickListener((parent, view, position, id) -> {

Planet clickedPlanet = adapter.getItem(position);

Toast.makeText(MainActivity.this,

"Planet: " + clickedPlanet.getPlanetName(),

Toast.LENGTH\_SHORT).show();

});

}

}

**🧪 Tools, APIs & Concepts Used**

| **Concept** | **Description** |
| --- | --- |
| ListView | Displays list of scrollable items |
| ArrayAdapter<T> | Binds ArrayList to ListView |
| LayoutInflater | Converts XML into View |
| ViewHolder | Optimizes view lookup |
| setTag()/getTag() | Stores ViewHolder with row |
| getApplicationContext() | Adapter context |

**💡 Best Practices & Alternatives**

| **Feature** | **Traditional** | **Modern Best Practice** |
| --- | --- | --- |
| List UI | ListView | ✅ RecyclerView |
| Adapter | ArrayAdapter | ✅ ListAdapter + DiffUtil |
| Click Handling | In Activity | ✅ Interface callback in Adapter |
| Image Handling | setImageResource | ✅ Glide / Coil (for URLs) |
| UI Design | XML + Manual Code | ✅ Jetpack Compose / DataBinding |

**🚀 Part B – Crucial Topics Not Covered in Course**

**🧭 RecyclerView (Modern ListView Alternative)**

* More efficient, flexible, modular.
* Handles animations, view reuse, multiple layouts.
* Use: RecyclerView.Adapter, ViewHolder, LayoutManager.

**⚡ DiffUtil + ListAdapter**

* Optimized list updates.
* Only redraws changed rows.
* Use DiffUtil.ItemCallback.

**🧰 Data Binding**

* Avoid findViewById() entirely.
* Bind XML directly to data.
* Enabled via build.gradle:
* buildFeatures {
* dataBinding true
* }

**🌐 Image Loading (From URL)**

// Glide (URL image loading)

Glide.with(context).load(imageUrl).into(viewHolder.planetImage);

**🧪 Adapter Unit Testing**

* Validate view binding logic.
* Use mock data and assert correctness.

**🌍 Accessibility**

* Add android:contentDescription to images.
* Maintain contrast and text size.

**🔁 Pagination (For large lists)**

* Use Paging Library or load in chunks.

**🔄 State Restoration**

* Save list state in onSaveInstanceState() during rotation.

**📌 Summary**

This section teaches how to build a custom list UI using ListView, ArrayAdapter, model class, and the ViewHolder pattern. While it's useful for fundamentals, modern Android apps should migrate to **RecyclerView**, **Glide**, and **Jetpack Compose** for better performance, testability, and architecture.