**Point 1:**

package com.mastercoding.adaptersapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

import android.widget.ArrayAdapter;

import android.widget.ListView;

public class MainActivity extends AppCompatActivity {

ListView listview;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

// 1- AdapterView: ListView

listview = findViewById(R.id.listview);

// 2- Data Source: String Array

String[] countries = {"USA", "Germany", "Saudi Arabia","France"};

// 3- Adapter: acts as a bridge between the

// 'data source' and the 'AdapterView'

MyCustomAdapter adapter = new MyCustomAdapter(this, countries);

// Link Listview with the Adapter

listview.setAdapter(adapter);

}

}

\*\*\*\* \*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*

package com.mastercoding.adaptersapp;

import android.content.Context;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.BaseAdapter;

import android.widget.TextView;

public class MyCustomAdapter extends BaseAdapter {

private Context context;

private String[] items; // Data Source

public MyCustomAdapter(Context context, String[] items) {

this.context = context;

this.items = items;

}

@Override

public int getCount() {

return items.length; // Returns the number of items in your data source

}

@Override

public Object getItem(int position) {

return items[position]; // Returns the data item at the given position

}

@Override

public long getItemId(int position) {

return position; // Returns a unique Identifier for the item at the given position

}

@Override

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

ViewHolder holder;

if (convertView == null){

// convertView: is a recycled View that you can reuse to

// improve the performance of your list.

convertView= LayoutInflater.from(context)

.inflate(R.layout.my\_list\_item, parent, false);

holder = new ViewHolder();

holder.textView = convertView.findViewById(R.id.text1);

convertView.setTag(holder);

}else{

// Reusing the View (that's recycled)

holder = (ViewHolder) convertView.getTag();

}

// Set the data to the view

holder.textView.setText(items[position]);

// Binding data to views within the convertView

return convertView; // Displays the data at a position in the data set

}

static class ViewHolder{

// Holds references to the views within an item layout

TextView textView;

}

}

\*\*\*\* \*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*

<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout 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:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context=".MainActivity">

<ListView

android:id="@+id/listview"

android:layout\_width="0dp"

android:layout\_height="0dp"

app:layout\_constraintBottom\_toBottomOf="parent"

app:layout\_constraintEnd\_toEndOf="parent"

app:layout\_constraintStart\_toStartOf="parent"

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

</androidx.constraintlayout.widget.ConstraintLayout>

\*\*\*\* \*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*

<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<TextView

android:padding="10dp"

android:id="@+id/text1"

android:layout\_width="0dp"

android:layout\_height="wrap\_content"

android:text="item1"

android:textColor="#D91212"

android:textSize="48sp"

app:layout\_constraintEnd\_toEndOf="parent"

app:layout\_constraintStart\_toStartOf="parent"

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

</androidx.constraintlayout.widget.ConstraintLayout>

\*\*\*\* \*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*

**WHICH TYPE OF ADAPTER IS IN ABOVE ANDROID APP THAT IS BASEADAPTER OR CUSTOM ADAPTER OR ANY ELSE ??**

The Android app you provided uses a custom adapter, specifically extending BaseAdapter. Here's a breakdown of why it falls into this category:

**Custom Adapter**

A custom adapter is typically created by extending one of the standard adapter classes (like BaseAdapter, ArrayAdapter, CursorAdapter, etc.) and overriding necessary methods to bind your data source to the views in the adapter's item layout.

**BaseAdapter**

In this case, MyCustomAdapter extends BaseAdapter, which is a general-purpose adapter that can be used to create custom adapters for various data sources and views. This is evident from the following aspects of your code:

1. **Extending BaseAdapter**:
2. public class MyCustomAdapter extends BaseAdapter {
3. **Overriding Key Methods**:
   * getCount(): Returns the number of items in the data source.
   * getItem(int position): Returns the data item at a specified position.
   * getItemId(int position): Returns a unique ID for the item at a specified position.
   * getView(int position, View convertView, ViewGroup parent): Returns a view for each item in the data source.
4. @Override
5. public int getCount() {
6. return items.length;
7. }
8. @Override
9. public Object getItem(int position) {
10. return items[position];
11. }
12. @Override
13. public long getItemId(int position) {
14. return position;
15. }
16. @Override
17. public View getView(int position, View convertView, ViewGroup parent) {
18. ViewHolder holder;
19. if (convertView == null){
20. convertView= LayoutInflater.from(context).inflate(R.layout.my\_list\_item, parent, false);
21. holder = new ViewHolder();
22. holder.textView = convertView.findViewById(R.id.text1);
23. convertView.setTag(holder);
24. } else {
25. holder = (ViewHolder) convertView.getTag();
26. }
27. holder.textView.setText(items[position]);
28. return convertView;
29. }
30. **ViewHolder Pattern**:
    * The ViewHolder pattern is used to improve performance by avoiding frequent calls to findViewById().
31. static class ViewHolder {
32. TextView textView;
33. }

In summary, your app uses a custom adapter that extends BaseAdapter, making it a BaseAdapter-based custom adapter. This approach allows you to have complete control over how the data is displayed in the ListView by defining your own item layout and data binding logic.

**Point 2.1:**

**Types of adapters in android dev ?**

In Android development, adapters serve as a bridge between a data source and an adapter view (such as ListView, GridView, or RecyclerView). There are several types of adapters in Android, each tailored to different types of data sources and views. Here are the main types:

**1. ArrayAdapter**

* **Description**: A simple adapter used to bind an array of data (usually strings) to a ListView.
* **Usage**: Suitable for small datasets where each item is represented as a single string.
* **Example**:
* ArrayAdapter<String> adapter = new ArrayAdapter<>(this, android.R.layout.simple\_list\_item\_1, data);
* listView.setAdapter(adapter);

**2. BaseAdapter**

* **Description**: A base class for custom adapters. You must extend BaseAdapter and override its methods to create a custom adapter.
* **Usage**: Suitable for more complex data structures and custom layouts.
* **Example**:
* public class MyCustomAdapter extends BaseAdapter {
* // Implement required methods
* }

**3. CursorAdapter**

* **Description**: An adapter that binds data from a Cursor to a ListView or GridView.
* **Usage**: Suitable for displaying data from a database query.
* **Example**:
* CursorAdapter adapter = new SimpleCursorAdapter(this, layout, cursor, from, to, flags);
* listView.setAdapter(adapter);

**4. SimpleCursorAdapter**

* **Description**: A subclass of CursorAdapter that maps columns from a Cursor to TextViews or ImageViews in a layout defined in XML.
* **Usage**: Simplifies the use of a CursorAdapter.
* **Example**:
* String[] from = { "column1", "column2" };
* int[] to = { R.id.text1, R.id.text2 };
* SimpleCursorAdapter adapter = new SimpleCursorAdapter(this, R.layout.list\_item, cursor, from, to, 0);
* listView.setAdapter(adapter);

**5. RecyclerView.Adapter**

* **Description**: The primary adapter for RecyclerView. You must extend RecyclerView.Adapter and implement its methods.
* **Usage**: Suitable for complex and large datasets, providing better performance than ListView and GridView.
* **Example**:
* public class MyRecyclerViewAdapter extends RecyclerView.Adapter<MyRecyclerViewAdapter.ViewHolder> {
* // Implement required methods
* }
* RecyclerView recyclerView = findViewById(R.id.recyclerView);
* recyclerView.setAdapter(new MyRecyclerViewAdapter(data));

**6. ArrayAdapter**

* **Description**: An adapter that maps an array of data to a ListView.
* **Usage**: Simple to use for lists with single text items.
* **Example**:
* ArrayAdapter<String> adapter = new ArrayAdapter<>(this, android.R.layout.simple\_list\_item\_1, data);
* listView.setAdapter(adapter);

**7. SpinnerAdapter**

* **Description**: An adapter used for a Spinner (dropdown list).
* **Usage**: Similar to ArrayAdapter, but designed for dropdown lists.
* **Example**:
* SpinnerAdapter adapter = new ArrayAdapter<>(this, android.R.layout.simple\_spinner\_item, data);
* spinner.setAdapter(adapter);

**8. ListAdapter**

* **Description**: A marker interface implemented by all adapters that provide data to a ListView.
* **Usage**: Common interface for all adapters used with ListView.
* **Example**:
* ListAdapter adapter = new ArrayAdapter<>(this, android.R.layout.simple\_list\_item\_1, data);
* listView.setAdapter(adapter);

**9. SectionIndexer**

* **Description**: An interface that allows fast indexing of list items.
* **Usage**: Typically used in conjunction with BaseAdapter for lists that support fast scrolling and section indexing.
* **Example**:
* public class MySectionIndexerAdapter extends BaseAdapter implements SectionIndexer {
* // Implement required methods
* }

Each adapter type serves a different purpose and is chosen based on the complexity of the data and the requirements of the view.

**Point 2.2:**

**I think you missed few ones…**

You're right; there are a few more adapters in Android development. Here are additional adapter types, along with the ones previously mentioned:

**1. ArrayAdapter**

* **Description**: A simple adapter to bind an array of data (usually strings) to a ListView.
* **Usage**: Suitable for small datasets where each item is represented as a single string.
* **Example**:
* ArrayAdapter<String> adapter = new ArrayAdapter<>(this, android.R.layout.simple\_list\_item\_1, data);
* listView.setAdapter(adapter);

**2. BaseAdapter**

* **Description**: A base class for custom adapters. You must extend BaseAdapter and override its methods to create a custom adapter.
* **Usage**: Suitable for more complex data structures and custom layouts.
* **Example**:
* public class MyCustomAdapter extends BaseAdapter {
* // Implement required methods
* }

**3. CursorAdapter**

* **Description**: An adapter that binds data from a Cursor to a ListView or GridView.
* **Usage**: Suitable for displaying data from a database query.
* **Example**:
* CursorAdapter adapter = new SimpleCursorAdapter(this, layout, cursor, from, to, flags);
* listView.setAdapter(adapter);

**4. SimpleCursorAdapter**

* **Description**: A subclass of CursorAdapter that maps columns from a Cursor to TextViews or ImageViews in a layout defined in XML.
* **Usage**: Simplifies the use of a CursorAdapter.
* **Example**:
* String[] from = { "column1", "column2" };
* int[] to = { R.id.text1, R.id.text2 };
* SimpleCursorAdapter adapter = new SimpleCursorAdapter(this, R.layout.list\_item, cursor, from, to, 0);
* listView.setAdapter(adapter);

**5. RecyclerView.Adapter**

* **Description**: The primary adapter for RecyclerView. You must extend RecyclerView.Adapter and implement its methods.
* **Usage**: Suitable for complex and large datasets, providing better performance than ListView and GridView.
* **Example**:
* public class MyRecyclerViewAdapter extends RecyclerView.Adapter<MyRecyclerViewAdapter.ViewHolder> {
* // Implement required methods
* }
* RecyclerView recyclerView = findViewById(R.id.recyclerView);
* recyclerView.setAdapter(new MyRecyclerViewAdapter(data));

**6. SpinnerAdapter**

* **Description**: An adapter used for a Spinner (dropdown list).
* **Usage**: Similar to ArrayAdapter, but designed for dropdown lists.
* **Example**:
* SpinnerAdapter adapter = new ArrayAdapter<>(this, android.R.layout.simple\_spinner\_item, data);
* spinner.setAdapter(adapter);

**7. ListAdapter**

* **Description**: A marker interface implemented by all adapters that provide data to a ListView.
* **Usage**: Common interface for all adapters used with ListView.
* **Example**:
* ListAdapter adapter = new ArrayAdapter<>(this, android.R.layout.simple\_list\_item\_1, data);
* listView.setAdapter(adapter);

**8. SectionIndexer**

* **Description**: An interface that allows fast indexing of list items.
* **Usage**: Typically used in conjunction with BaseAdapter for lists that support fast scrolling and section indexing.
* **Example**:
* public class MySectionIndexerAdapter extends BaseAdapter implements SectionIndexer {
* // Implement required methods
* }

**9. ArrayWheelAdapter**

* **Description**: Used for a wheel-like UI component.
* **Usage**: Binds an array of data to a wheel view.
* **Example**:
* ArrayWheelAdapter<String> adapter = new ArrayWheelAdapter<>(this, data);
* wheelView.setViewAdapter(adapter);

**10. ResourceCursorAdapter**

* **Description**: Similar to SimpleCursorAdapter, but you supply the layout resource.
* **Usage**: Suitable for binding Cursor data to a ListView or GridView.
* **Example**:
* ResourceCursorAdapter adapter = new ResourceCursorAdapter(this, R.layout.list\_item, cursor, 0);
* listView.setAdapter(adapter);

**11. PagedListAdapter**

* **Description**: An adapter used for paginated data loading with RecyclerView.
* **Usage**: Works with LiveData and ViewModel to handle large datasets efficiently.
* **Example**:
* public class MyPagedListAdapter extends PagedListAdapter<MyDataModel, MyPagedListAdapter.ViewHolder> {
* // Implement required methods
* }
* PagedListAdapter adapter = new MyPagedListAdapter();
* recyclerView.setAdapter(adapter);

These additional adapters provide more specialized functionality for different scenarios in Android development.

**Point 2.3:**

**I think you still missed.. Please mention all in this single time only..**

Sure, here is a comprehensive list of adapters used in Android development:

**1. ArrayAdapter**

* **Description**: A simple adapter to bind an array of data (usually strings) to a ListView.
* **Usage**: Suitable for small datasets where each item is represented as a single string.
* **Example**:
* ArrayAdapter<String> adapter = new ArrayAdapter<>(this, android.R.layout.simple\_list\_item\_1, data);
* listView.setAdapter(adapter);

**2. BaseAdapter**

* **Description**: A base class for custom adapters. You must extend BaseAdapter and override its methods to create a custom adapter.
* **Usage**: Suitable for more complex data structures and custom layouts.
* **Example**:
* public class MyCustomAdapter extends BaseAdapter {
* // Implement required methods
* }

**3. CursorAdapter**

* **Description**: An adapter that binds data from a Cursor to a ListView or GridView.
* **Usage**: Suitable for displaying data from a database query.
* **Example**:
* CursorAdapter adapter = new SimpleCursorAdapter(this, layout, cursor, from, to, flags);
* listView.setAdapter(adapter);

**4. SimpleCursorAdapter**

* **Description**: A subclass of CursorAdapter that maps columns from a Cursor to TextViews or ImageViews in a layout defined in XML.
* **Usage**: Simplifies the use of a CursorAdapter.
* **Example**:
* String[] from = { "column1", "column2" };
* int[] to = { R.id.text1, R.id.text2 };
* SimpleCursorAdapter adapter = new SimpleCursorAdapter(this, R.layout.list\_item, cursor, from, to, 0);
* listView.setAdapter(adapter);

**5. RecyclerView.Adapter**

* **Description**: The primary adapter for RecyclerView. You must extend RecyclerView.Adapter and implement its methods.
* **Usage**: Suitable for complex and large datasets, providing better performance than ListView and GridView.
* **Example**:
* public class MyRecyclerViewAdapter extends RecyclerView.Adapter<MyRecyclerViewAdapter.ViewHolder> {
* // Implement required methods
* }
* RecyclerView recyclerView = findViewById(R.id.recyclerView);
* recyclerView.setAdapter(new MyRecyclerViewAdapter(data));

**6. SpinnerAdapter**

* **Description**: An adapter used for a Spinner (dropdown list).
* **Usage**: Similar to ArrayAdapter, but designed for dropdown lists.
* **Example**:
* SpinnerAdapter adapter = new ArrayAdapter<>(this, android.R.layout.simple\_spinner\_item, data);
* spinner.setAdapter(adapter);

**7. ListAdapter**

* **Description**: A marker interface implemented by all adapters that provide data to a ListView.
* **Usage**: Common interface for all adapters used with ListView.
* **Example**:
* ListAdapter adapter = new ArrayAdapter<>(this, android.R.layout.simple\_list\_item\_1, data);
* listView.setAdapter(adapter);

**8. SectionIndexer**

* **Description**: An interface that allows fast indexing of list items.
* **Usage**: Typically used in conjunction with BaseAdapter for lists that support fast scrolling and section indexing.
* **Example**:
* public class MySectionIndexerAdapter extends BaseAdapter implements SectionIndexer {
* // Implement required methods
* }

**9. ArrayWheelAdapter**

* **Description**: Used for a wheel-like UI component.
* **Usage**: Binds an array of data to a wheel view.
* **Example**:
* ArrayWheelAdapter<String> adapter = new ArrayWheelAdapter<>(this, data);
* wheelView.setViewAdapter(adapter);

**10. ResourceCursorAdapter**

* **Description**: Similar to SimpleCursorAdapter, but you supply the layout resource.
* **Usage**: Suitable for binding Cursor data to a ListView or GridView.
* **Example**:
* ResourceCursorAdapter adapter = new ResourceCursorAdapter(this, R.layout.list\_item, cursor, 0);
* listView.setAdapter(adapter);

**11. PagedListAdapter**

* **Description**: An adapter used for paginated data loading with RecyclerView.
* **Usage**: Works with LiveData and ViewModel to handle large datasets efficiently.
* **Example**:
* public class MyPagedListAdapter extends PagedListAdapter<MyDataModel, MyPagedListAdapter.ViewHolder> {
* // Implement required methods
* }
* PagedListAdapter adapter = new MyPagedListAdapter();
* recyclerView.setAdapter(adapter);

**12. SimpleAdapter**

* **Description**: An easy adapter to map static data to views defined in an XML file.
* **Usage**: Typically used for simple lists or grids.
* **Example**:
* List<Map<String, String>> data = new ArrayList<>();
* String[] from = { "key1", "key2" };
* int[] to = { R.id.text1, R.id.text2 };
* SimpleAdapter adapter = new SimpleAdapter(this, data, R.layout.list\_item, from, to);
* listView.setAdapter(adapter);

**13. ExpandableListAdapter**

* **Description**: An adapter that binds data to an ExpandableListView.
* **Usage**: Used for displaying data in a tree-like structure.
* **Example**:
* ExpandableListAdapter adapter = new SimpleExpandableListAdapter(this, groupList, R.layout.group\_item,
* new String[] { "groupName" }, new int[] { R.id.groupName }, childList, R.layout.child\_item,
* new String[] { "childName" }, new int[] { R.id.childName });
* expandableListView.setAdapter(adapter);

**14. SimpleExpandableListAdapter**

* **Description**: A subclass of BaseExpandableListAdapter that maps static data to views defined in an XML file for an ExpandableListView.
* **Usage**: Simplifies the implementation of an ExpandableListAdapter.
* **Example**:
* List<Map<String, String>> groupList = new ArrayList<>();
* List<List<Map<String, String>>> childList = new ArrayList<>();
* SimpleExpandableListAdapter adapter = new SimpleExpandableListAdapter(this, groupList, R.layout.group\_item,
* new String[] { "groupName" }, new int[] { R.id.groupName }, childList, R.layout.child\_item,
* new String[] { "childName" }, new int[] { R.id.childName });
* expandableListView.setAdapter(adapter);

**15. ArrayObjectAdapter**

* **Description**: Used in the Android TV framework with the RecyclerView-based RowPresenter.
* **Usage**: Suitable for managing a list of objects in RecyclerView for TV applications.
* **Example**:
* ArrayObjectAdapter adapter = new ArrayObjectAdapter(new ListRowPresenter());
* adapter.add(new ListRow(headerItem, new ArrayObjectAdapter(cardPresenter)));

**16. CursorTreeAdapter**

* **Description**: An adapter that uses Cursor to manage data for an ExpandableListView.
* **Usage**: Suitable for displaying hierarchical data from a database.
* **Example**:
* CursorTreeAdapter adapter = new MyCursorTreeAdapter(cursor, this);
* expandableListView.setAdapter(adapter);

**17. HeaderViewListAdapter**

* **Description**: Wraps a ListAdapter allowing additional header and footer views.
* **Usage**: Typically used internally by ListView.
* **Example**:
* HeaderViewListAdapter adapter = new HeaderViewListAdapter(headerViewInfos, footerViewInfos, listAdapter);
* listView.setAdapter(adapter);

**18. ThemedSpinnerAdapter**

* **Description**: A specialized adapter for Spinner that supports theming.
* **Usage**: Used when custom themes are needed for dropdown items in a Spinner.
* **Example**:
* ThemedSpinnerAdapter adapter = new ThemedSpinnerAdapter(this, data);
* spinner.setAdapter(adapter);

These adapters provide a wide range of functionalities to meet different data presentation needs in Android development.

**Point 3:**

**Is there custom adapter missing or is covered in above 18 types of adapters ?**

Custom adapters are covered in the list above through several examples, particularly when extending BaseAdapter or other base adapter classes like RecyclerView.Adapter. Here's a detailed explanation of custom adapters and where they fit in the list:

**Custom Adapters**

Custom adapters are user-defined adapters created by extending existing adapter classes to handle complex data structures and custom layouts. They are not a distinct type of adapter but rather a customization of existing ones.

**Examples of Custom Adapters in the List:**

1. **BaseAdapter**
   * **Description**: You can create custom adapters by extending BaseAdapter and implementing its required methods.
   * **Example**:
   * public class MyCustomAdapter extends BaseAdapter {
   * private Context context;
   * private String[] data;
   * public MyCustomAdapter(Context context, String[] data) {
   * this.context = context;
   * this.data = data;
   * }
   * @Override
   * public int getCount() {
   * return data.length;
   * }
   * @Override
   * public Object getItem(int position) {
   * return data[position];
   * }
   * @Override
   * public long getItemId(int position) {
   * return position;
   * }
   * @Override
   * public View getView(int position, View convertView, ViewGroup parent) {
   * if (convertView == null) {
   * convertView = LayoutInflater.from(context).inflate(R.layout.list\_item, parent, false);
   * }
   * TextView textView = convertView.findViewById(R.id.text);
   * textView.setText(data[position]);
   * return convertView;
   * }
   * }
2. **RecyclerView.Adapter**
   * **Description**: Creating custom adapters by extending RecyclerView.Adapter for more complex and flexible data binding in RecyclerView.
   * **Example**:
   * public class MyRecyclerViewAdapter extends RecyclerView.Adapter<MyRecyclerViewAdapter.ViewHolder> {
   * private List<String> data;
   * public MyRecyclerViewAdapter(List<String> data) {
   * this.data = data;
   * }
   * @Override
   * public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
   * View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.list\_item, parent, false);
   * return new ViewHolder(view);
   * }
   * @Override
   * public void onBindViewHolder(ViewHolder holder, int position) {
   * holder.textView.setText(data.get(position));
   * }
   * @Override
   * public int getItemCount() {
   * return data.size();
   * }
   * public static class ViewHolder extends RecyclerView.ViewHolder {
   * TextView textView;
   * public ViewHolder(View itemView) {
   * super(itemView);
   * textView = itemView.findViewById(R.id.text);
   * }
   * }
   * }
3. **ArrayAdapter (Customized)**
   * **Description**: While ArrayAdapter is simple, it can be customized by overriding methods to suit specific needs.
   * **Example**:
   * public class MyArrayAdapter extends ArrayAdapter<String> {
   * public MyArrayAdapter(Context context, List<String> data) {
   * super(context, 0, data);
   * }
   * @Override
   * public View getView(int position, View convertView, ViewGroup parent) {
   * if (convertView == null) {
   * convertView = LayoutInflater.from(getContext()).inflate(R.layout.list\_item, parent, false);
   * }
   * TextView textView = convertView.findViewById(R.id.text);
   * textView.setText(getItem(position));
   * return convertView;
   * }
   * }

**Summary**

Custom adapters are essentially personalized implementations of standard adapter classes tailored to meet specific requirements. The concept of custom adapters is inherent in how you use BaseAdapter, RecyclerView.Adapter, ArrayAdapter, and others, by overriding and implementing their methods to handle your data and view requirements.

So, the concept of custom adapters is covered in the examples above, showing how you can extend and customize existing adapter classes to create your own adapters.