# Recycler view

## Introduction

- The Recycler View class simplifies the display and handling of **large data sets** by providing:
  - **Layout managers** for positioning items.
  - Default animations for common item operations, such as **removal or addition of items**.

- Recycler View provides these **built-in layout managers**:
  - LinearLayoutManager shows items in a **vertical or horizontal** scrolling list.
  - **GridLayoutManager** shows items in a grid.
  - **StaggeredGridLayoutManager** shows items in a staggered grid.

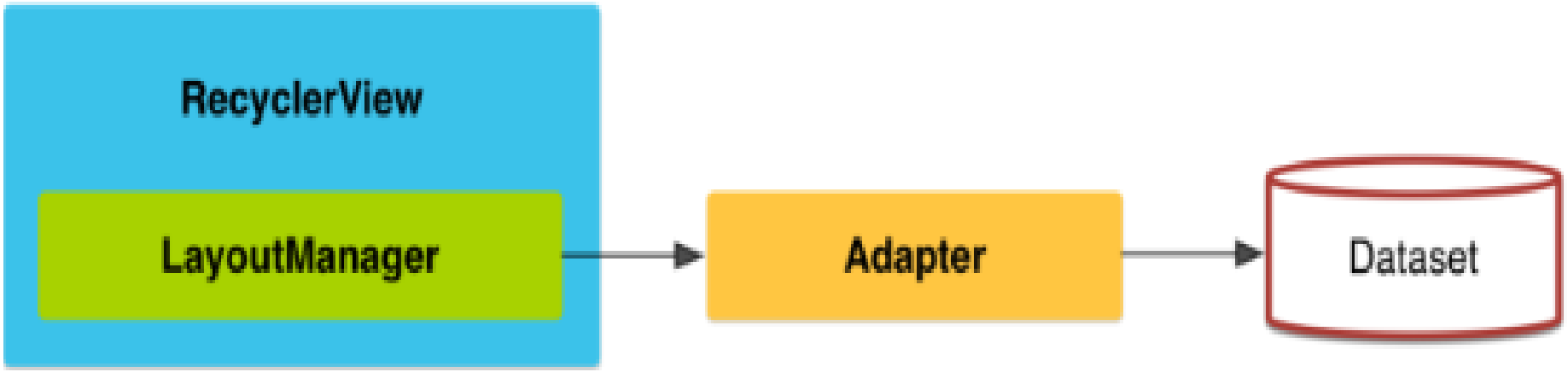
# Recycler view

## Introduction

- The Recycler View widget is a more **advanced and flexible** version of List View.
- This widget is a container for displaying large data sets that can be scrolled **very efficiently** by maintaining a limited number of views
- use the Recycler View widget when you have data **collections whose elements change at runtime** based on user action or network events.

# Recycler view

## Architecture



# Recycler view

## Animations

- Animations for adding and removing items are **enabled by default** in Recycler View.
- To customize these animations, extend the RecyclerView.ItemAnimator class and use the RecyclerView.setItemAnimator() method.
- The Recycler View uses an animator to change its **appearance**. This animator is an object.
- Recycler View  animator extends the abstract **RecyclerView.ItemAnimator** class.

# Recycler view

## Animations

- By default, the RecyclerView uses **DefaultItemAnimator** to provide the animation.
- If you want to provide custom animations, you can define your own animator object by extending **RecyclerView.ItemAnimator.**

# Recycler view

**Why ?**

- RecyclerView is **powerful** when you need to customize your list or you want better animations.
- More **Flexible**.
- Better **Memory managed**.
- Allow you to change **dynamic layout behavior**.
- **Nested Scrolling** Possible.
- It doesn't care about what is visible on the screen it care about **item placement**.

# Recycler view

## Implementation

**Step 1 :**

```
compile 'com.android.support:recyclerview-v7:25.3.1'
```

**Step 2 :**

```
<!-- A RecyclerView with some commonly used attributes -->
<android.support.v7.widget.RecyclerView
    android:id="@+id/my_recycler_view"
    android:scrollbars="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

**Step 3 :**

```
public class MyActivity extends Activity {
    private RecyclerView mRecyclerView;
    private RecyclerView.Adapter mAdapter;
    private RecyclerView.LayoutManager mLayoutManager;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.my_activity);
        mRecyclerView = (RecyclerView) findViewById(R.id.my_recycler_view);

        // use this setting to improve performance if you know that changes
        // in content do not change the layout size of the RecyclerView
        mRecyclerView.setHasFixedSize(true);

        // use a linear layout manager
        mLayoutManager = new LinearLayoutManager(this);
        mRecyclerView.setLayoutManager(mLayoutManager);

        // specify an adapter (see also next example)
        mAdapter = new MyAdapter(myDataset);
        mRecyclerView.setAdapter(mAdapter);
    }
    ...
}
```

# Recycler view

## Implementation

**Step 4 :**

```
public class MyAdapter extends RecyclerView.Adapter<MyAdapter.ViewHolder> {
    private String[] mDataset;

    // Provide a reference to the views for each data item
    // Complex data items may need more than one view per item, and
    // you provide access to all the views for a data item in a view holder
    public static class ViewHolder extends RecyclerView.ViewHolder {
        // each data item is just a string in this case
        public TextView mTextView;
        public ViewHolder(TextView v) {
            super(v);
            mTextView = v;
        }
    }

    // Provide a suitable constructor (depends on the kind of dataset)
    public MyAdapter(String[] myDataset) {
        mDataset = myDataset;
    }
```

```
    // Create new views (invoked by the layout manager)
    @Override
    public MyAdapter.ViewHolder onCreateViewHolder(ViewGroup parent,
                                                   int viewType) {
        // create a new view
        TextView v = (TextView) LayoutInflater.from(parent.getContext())
                .inflate(R.layout.my_text_view, parent, false);
        // set the view's size, margins, paddings and layout parameters
        ...
        ViewHolder vh = new ViewHolder(v);
        return vh;
    }

    // Replace the contents of a view (invoked by the layout manager)
    @Override
    public void onBindViewHolder(ViewHolder holder, int position) {
        // - get element from your dataset at this position
        // - replace the contents of the view with that element
        holder.mTextView.setText(mDataset[position]);

    }

    // Return the size of your dataset (invoked by the layout manager)
    @Override
    public int getItemCount() {
        return mDataset.length;
    }
}
```

**---------- Complete coding under Industry Expert in class room ----------**