Mobile Applications (420-P84-AB)

John Abbott College

Recycler View

—

Aakash Malhotra

Adapter

An Adapter object acts as a bridge between an `AdapterView` and the underlying data for that view. The Adapter provides access to the data items. The Adapter is also responsible for making a `View` for each item in the data set.

View Holder

A ViewHolder describes an item view and metadata about its place within the RecyclerView.

`RecyclerView.Adapter` implementations should subclass ViewHolder and add fields for caching potentially expensive `findViewById(int)` results.

While `RecyclerView.LayoutParams` belong to the `RecyclerView.LayoutManager`, `ViewHolders` belong to the adapter. Adapters should feel free to use their own custom ViewHolder implementations to store data that makes binding view contents easier. Implementations should assume that individual item views will hold strong references to `ViewHolder` objects and that `RecyclerView` instances may hold strong references to extra off-screen item views for caching purposes

Recycler View

If your app needs to display a scrolling list of elements based on large data sets (or data that frequently changes), you should use `RecyclerView`

The `RecyclerView` widget is a more advanced and flexible version of `ListView`.

In the `RecyclerView` model, several different components work together to display your data. The overall container for your user interface is a `RecyclerView` object that you add to your layout. The `RecyclerView` fills itself with views provided by a *layout manager* that you provide. You can use one of our standard layout managers (such as `LinearLayoutManager` or `GridLayoutManager`), or implement your own.

Recycler View

The views in the list are represented by *view holder* objects. These objects are instances of a class you define by extending `RecyclerView.ViewHolder`. Each view holder is in charge of displaying a single item with a view. For example, if your list shows music collection, each view holder might represent a single album. The `RecyclerView` creates only as many view holders as are needed to display the on-screen portion of the dynamic content, plus a few extra. As the user scrolls through the list, the `RecyclerView` takes the off-screen views and rebinds them to the data which is scrolling onto the screen.

The view holder objects are managed by an *adapter*, which you create by extending `RecyclerView.Adapter`. The adapter creates view holders as needed. The adapter also binds the view holders to their data. It does this by assigning the view holder to a position, and calling the adapter's `onBindViewHolder()` method. That method uses the view holder's position to determine what the contents should be, based on its list position.

Recycler View

This `RecyclerView` model does a lot of optimization work so you don't have to:

- When the list is first populated, it creates and binds some view holders on either side of the list. For example, if the view is displaying list positions 0 through 9, the `RecyclerView` creates and binds those view holders, and might also create and bind the view holder for position 10. That way, if the user scrolls the list, the next element is ready to display.
- As the user scrolls the list, the `RecyclerView` creates new view holders as necessary. It also saves the view holders which have scrolled off-screen, so they can be reused. If the user switches the direction they were scrolling, the view holders which were scrolled off the screen can be brought right back. On the other hand, if the user keeps scrolling in the same direction, the view holders which have been off-screen the longest can be re-bound to new data. The view holder does not need to be created or have its view inflated; instead, the app just updates the view's contents to match the new item it was bound to.
-

Recycler View

- When the displayed items change, you can notify the adapter by calling an appropriate `RecyclerView.Adapter.notify…()` method. The adapter's built-in code then rebinds just the affected items.

Add the support library

To access the `RecyclerView` widget, you need to add the v7 Support Libraries to your project as follows:

1. Open the `build.gradle` file for your app module.
2. Add the support library to the `dependencies` section.

```
dependencies {

    implementation 'com.android.support:recyclerview-v7:27.1.1'

}
```

Add RecyclerView to your layout

Now you can add the `RecyclerView` to your layout file. For example, the following layout uses `RecyclerView` as the only view for the whole layout:

```
<android.support.v7.widget.RecyclerView

    android:id="@+id/my_recycler_view"

    android:scrollbars="vertical"

    android:layout_width="match_parent"

    android:layout_height="match_parent"/>
```

Once you have added a `RecyclerView` widget to your layout, obtain a handle to the object, connect it to a layout manager, and attach an adapter for the data to be displayed:

Add RecyclerView to your layout

```
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);
```

Add RecyclerView to your layout

```
// 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);
    }
    // ...
}
```

Add a list adapter

To feed all your data to the list, you must extend the `RecyclerView.Adapter` class. This object creates views for items, and replaces the content of some of the views with new data items when the original item is no longer visible.

The following code example shows a simple implementation for a data set that consists of an array of strings displayed using `TextView` widgets:

Add a list adapter

```
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;
```

Add a list adapter

```
public ViewHolder(TextView v) {
            super(v);
            mTextView = v;
        }
    }

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

Add a list adapter

```
// 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);
        ...
        ViewHolder vh = new ViewHolder(v);
        return vh;
    }
```

Add a list adapter

```
// 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]);

    }
```

Add a list adapter

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

Add a list adapter

The layout manager calls the adapter's `onCreateViewHolder()` method. That method needs to construct a `RecyclerView.ViewHolder` and set the view it uses to display its contents. The type of the ViewHolder must match the type declared in the Adapter class signature. Typically, it would set the view by inflating an XML layout file. Because the view holder is not yet assigned to any particular data, the method does not actually set the view's contents.

The layout manager then binds the view holder to its data. It does this by calling the adapter's `onBindViewHolder()` method, and passing the view holder's position in the `RecyclerView`. The `onBindViewHolder()` method needs to fetch the appropriate data, and use it to fill in the view holder's layout. For example, if the `RecyclerView` is displaying a list of names, the method might find the appropriate name in the list, and fill in the view holder's `TextView` widget.

Add a list adapter

If the list needs an update, call a notification method on the `RecyclerView.Adapter` object, such as `notifyItemChanged()`. The layout manager then rebinds any affected view holders, allowing their data to be updated.

References:

- https://developer.android.com/reference/android/widget/Adapter
- https://developer.android.com/guide/topics/ui/declaring-layout
- https://developer.android.com/reference/android/support/v7/widget/RecyclerView.ViewHolder
- https://developer.android.com/guide/topics/ui/layout/recyclerview