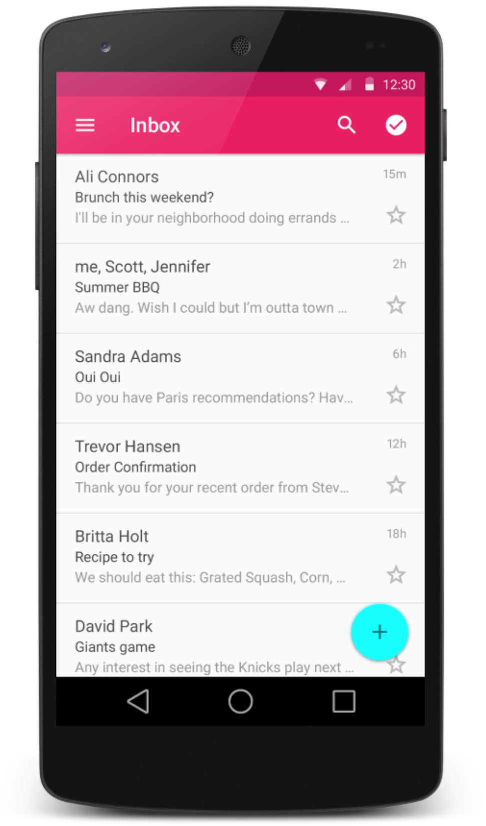
Create a List with RecyclerView

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`** (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html) as described on this page.

Tip: Start with some template code in Android Studio by clicking **File > New > Fragment > Fragment (List)**. Then simply add the fragment to your activity layout (https://developer.android.com/training/basics/fragments/creating.html#AddInLayout).



Ali Connors 15m
Brunch this weekend?
I'll be in your neighborhood doing errands ...

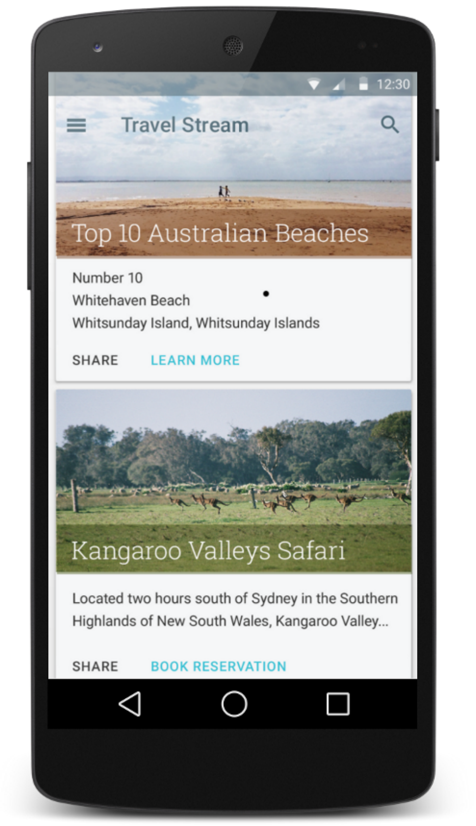
me, Scott, Jennifer 2h
Summer BBQ
Aw dang. Wish I could but I'm outta town ...

Sandra Adams 6h
Oui Oui
Do you have Paris recommendations? Hav...

Trevor Hansen 12h
Order Confirmation
Thank you for your recent order from Stev...

Britta Holt 18h
Recipe to try
We should eat this: Grated Squash, Corn, ...

David Park
Giants game
Any interest in seeing the Knicks play next ...



Number 10
Whitehaven Beach
Whitsunday Island, Whitsunday Islands

Located two hours south of Sydney in the Southern Highlands of New South Wales, Kangaroo Valley...

Figure 1. A list using `RecyclerView` **Figure 2.** A list also using `CardView`

If you'd like to create a list with cards, as shown in figure 2, also use the **`CardView`** (https://developer.android.com/reference/android/support/v7/widget/CardView.html) widget as described in Create a Card-based Layout (https://developer.android.com/guide/topics/ui/layout/cardview.html).

If you'd like to see some sample code for **`RecyclerView`**, check out the RecyclerView Sample App (https://github.com/googlesamples/android-RecyclerView/).

RecyclerView overview

The `RecyclerView` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html) widget is a more advanced and flexible version of `ListView` (https://developer.android.com/reference/android/widget/ListView.html).

In the `RecyclerView` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html) model, several different components work together to display your data. The overall container for your user interface is a `RecyclerView` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html) object that you add to your layout. The `RecyclerView` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html) fills itself with views provided by a *layout manager* that you provide. You can use one of our standard layout managers (such as `LinearLayoutManager` (https://developer.android.com/reference/android/support/v7/widget/LinearLayoutManager.html) or `GridLayoutManager` (https://developer.android.com/reference/android/support/v7/widget/GridLayoutManager.html)), or implement your own.

The views in the list are represented by *view holder* objects. These objects are instances of a class you define by extending `RecyclerView.ViewHolder` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.ViewHolder.html). 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` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html) 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` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html) 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` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter.html). 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()` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter.html#onBindViewHolder(VH, int)) method. That method uses the view holder's position to determine what the contents should be, based on its list position.

This `RecyclerView` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html) 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` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html) 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` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html) 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.
- When the displayed items change, you can notify the adapter by calling an appropriate `RecyclerView.Adapter.notify…()` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter.html#notifyItemChanged(int)) method. The adapter's built-in code then rebinds just the affected items.

Add the support library

To access the `RecyclerView` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html) widget, you need to add the v7 Support Libraries (https://developer.android.com/tools/support-library/features.html#v7) 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** (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html) to your layout file. For example, the following layout uses **RecyclerView** (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html) as the only view for the whole layout:

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

Once you have added a **RecyclerView** (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html) 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:

KOTLIN JAVA

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

Add a list adapter

To feed all your data to the list, you must extend the **`RecyclerView.Adapter`** (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter.html) 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`** (https://developer.android.com/reference/android/widget/TextView.html) widgets:

KOTLIN JAVA

```
public class MyAdapter extends RecyclerView.Adapter<MyAdapter.ViewHolde
    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);
        ...
        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;
    }
}
```

The layout manager calls the adapter's `onCreateViewHolder()` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter.html#onCreateViewHolder(android.view.ViewGroup, int)) method. That method needs to construct a `RecyclerView.ViewHolder` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.ViewHolder.html) 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()` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter.html#onBindViewHolder(VH, int)) method, and passing the view holder's position in the `RecyclerView` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html). The `onBindViewHolder()` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter.html#onBindViewHolder(VH, int)) method needs to fetch the appropriate data, and use it to fill in the view holder's layout. For example, if the `RecyclerView` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html) is displaying a list of names, the method might find the appropriate name in the list, and fill in the view holder's `TextView` (https://developer.android.com/reference/android/widget/TextView.html) widget.

If the list needs an update, call a notification method on the `RecyclerView.Adapter` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter.html) object, such as `notifyItemChanged()` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter.html#notifyItemChanged(int))
. The layout manager then rebinds any affected view holders, allowing their data to be updated.

Tip: You might find the `ListAdapter` (https://developer.android.com/reference/android/support/v7/recyclerview/extensions/ListAdapter.html)
class useful for determining which items in your list need to be updated when the list changes.

Customize your RecyclerView

You can customize the `RecyclerView` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html) objects to meet your specific needs. The standard classes provide all the functionality that most developers will need; in many cases, the only customization you need to do is design the view for each view holder and write the code to update those views with the appropriate data. However, if your app has specific requirements, you can modify the standard behavior in a number of ways. The following sections describe some of the other common customizations.

Modifying the layout

The `RecyclerView` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html) uses a layout manager to position the individual items on the screen and determine when to reuse item views that are no longer visible to the user. To reuse (or *recycle*) a view, a layout manager may ask the adapter to replace the contents of the view with a different element from the dataset. Recycling views in this manner improves performance by avoiding the creation of unnecessary views or performing expensive `findViewById()` (https://developer.android.com/reference/android/app/Activity.html#findViewById(int)) lookups. The Android Support Library includes three standard layout managers, each of which offers many customization options:

- `LinearLayoutManager` (https://developer.android.com/reference/android/support/v7/widget/LinearLayoutManager.html) arranges the items in a one-dimensional list. Using a `RecyclerView` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html) with `LinearLayoutManager` (https://developer.android.com/reference/android/support/v7/widget/LinearLayoutManager.html) provides functionality like the older `ListView` (https://developer.android.com/reference/android/widget/ListView.html) layout.
- `GridLayoutManager` (https://developer.android.com/reference/android/support/v7/widget/GridLayoutManager.html) arranges the items in a two-dimensional grid, like the squares on a checkerboard. Using a `RecyclerView` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html) with `GridLayoutManager` (https://developer.android.com/reference/android/support/v7/widget/GridLayoutManager.html) provides functionality like the older `GridView` (https://developer.android.com/reference/android/widget/GridView.html) layout.
- `StaggeredGridLayoutManager` (https://developer.android.com/reference/android/support/v7/widget/StaggeredGridLayoutManager.html) arranges the items in a two-dimensional grid, with each column slightly offset from the one before, like the stars in an American flag.

If none of these layout managers suits your needs, you can create your own by extending the `RecyclerView.LayoutManager` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.LayoutManager.html) abstract class.

Add item animations

Whenever an item changes, the `RecyclerView` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html) uses an *animator* to change its appearance. This animator is an object that extends the abstract `RecyclerView.ItemAnimator` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.ItemAnimator.html) class. By default, the `RecyclerView` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html) uses `DefaultItemAnimator` (https://developer.android.com/reference/android/support/v7/widget/DefaultItemAnimator.html) to provide the animation. If you want to provide custom animations, you can define your own animator object by extending `RecyclerView.ItemAnimator` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.ItemAnimator.html) .

Enable list-item selection

The `recyclerview-selection` (https://developer.android.com/reference/androidx/recyclerview/selection/package-summary.html) library enables users to select items in `RecyclerView` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html) list using touch or mouse input. You retain control over the visual presentation of a selected item. You can also retain control over policies controlling selection behavior, such as items that can be eligible for selection, and how many items can be selected.

To add selection support to a `RecyclerView` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html) instance, follow these steps:

1. Determine which selection key type to use, then build a `ItemKeyProvider` (https://developer.android.com/reference/androidx/recyclerview/selection/ItemKeyProvider.html) .

   There are three key types that you can use to identify selected items: `Parcelable` (https://developer.android.com/reference/android/os/Parcelable.html) (and all subclasses like `Uri` (https://developer.android.com/reference/android/net/Uri.html)), `String` (https://developer.android.com/reference/java/lang/String.html), and `Long` (https://developer.android.com/reference/java/lang/Long.html). For detailed information about selection-key types, see `SelectionTracker.Builder` (https://developer.android.com/reference/androidx/recyclerview/selection/SelectionTracker.Builder.html) .

2. Implement `ItemDetailsLookup` (https://developer.android.com/reference/androidx/recyclerview/selection/ItemDetailsLookup.html)
   .

   `ItemDetailsLookup` (https://developer.android.com/reference/androidx/recyclerview/selection/ItemDetailsLookup.html)
   enables the selection library to access information about `RecyclerView` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html) items given a `MotionEvent` (https://developer.android.com/reference/android/view/MotionEvent.html). It is effectively a factory for `ItemDetails` (https://developer.android.com/reference/androidx/recyclerview/selection/ItemDetailsLookup.ItemDetails.html)
   instances that are backed up by (or extracted from) a `RecyclerView.ViewHolder` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.ViewHolder.html)
   instance.

3. Update item `Views` (https://developer.android.com/reference/android/view/View.html) in `RecyclerView` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html) to reflect that the user has selected or unselected it.

   The selection library does not provide a default visual decoration for the selected items. You must provide this when you implement `onBindViewHolder()` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter.html#onBindViewHolder(VH, int))
   . The recommended approach is as follows:

   - In `onBindViewHolder()` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter.html#onBindViewHolder(VH, int))
     , call `setActivated()` (https://developer.android.com/reference/android/view/View.html#setActivated(boolean)) (not `setSelected()` (https://developer.android.com/reference/android/view/View.html#setSelected(boolean))) on the `View` (https://developer.android.com/reference/android/view/View.html) object with `true` or `false` (depending on if the item is selected).
   - Update the styling of the view to represent the activated status. We recommend you use a color state list resource (https://developer.android.com/guide/topics/resources/color-list-resource.html) to

configure the styling.

4. Use **`ActionMode`** (https://developer.android.com/reference/android/support/v7/view/ActionMode.html) to provide the user with tools to perform an action on the selection.

   Register a **`SelectionTracker.SelectionObserver`** (https://developer.android.com/reference/androidx/recyclerview/selection/SelectionTracker.SelectionObserver.html)
   to be notified when selection changes. When a selection is first created, start **`ActionMode`** (https://developer.android.com/reference/android/support/v7/view/ActionMode.html) to represent this to the user, and provide selection-specific actions. For example, you may add a delete button to the **`ActionMode`** (https://developer.android.com/reference/android/support/v7/view/ActionMode.html) bar, and connect the back arrow on the bar to clear the selection. When the selection becomes empty (if the user cleared the selection the last time), don't forget to terminate action mode.

5. Perform any interpreted secondary actions

   At the end of the event processing pipeline, the library may determine that the user is attempting to activate an item by tapping it, or is attempting to drag and drop an item or set of selected items. React to these interpretations by registering the appropriate listener. For more information, see **`SelectionTracker.Builder`** (https://developer.android.com/reference/androidx/recyclerview/selection/SelectionTracker.Builder.html)
   .

6. Assemble everything with **`SelectionTracker.Builder`** (https://developer.android.com/reference/androidx/recyclerview/selection/SelectionTracker.Builder.html)

   The following example shows how to put these pieces together by using the **`Long`** (https://developer.android.com/reference/java/lang/Long.html) selection key:

```
SelectionTracker tracker = new SelectionTracker.Builder<>(
        "my-selection-id",
        recyclerView,
        new StableIdKeyProvider(recyclerView),
        new MyDetailsLookup(recyclerView),
        StorageStrategy.createLongStorage())
        .withOnItemActivatedListener(myItemActivatedListener)
        .build();
```

In order to build a `SelectionTracker` (https://developer.android.com/reference/androidx/recyclerview/selection/SelectionTracker.html) instance, your app must supply the same `RecyclerView.Adapter` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter.html) that you used to initialize `RecyclerView` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html) to `SelectionTracker.Builder` (https://developer.android.com/reference/androidx/recyclerview/selection/SelectionTracker.Builder.html). For this reason, you will most likely need to inject the `SelectionTracker` (https://developer.android.com/reference/androidx/recyclerview/selection/SelectionTracker.html) instance, once created, into your `RecyclerView.Adapter` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter.html) after the `RecyclerView.Adapter` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter.html) is created. Otherwise, you won't be able to check an item's selected status from the `onBindViewHolder()` (https://developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter.html#onBindViewHolder(VH, int)) method.

7. Include selection in the activity lifecycle (https://developer.android.com/guide/components/activities/activity-lifecycle.html) events.

   In order to preserve selection state across the activity lifecycle (https://developer.android.com/guide/components/activities/activity-lifecycle.html) events, your app must call the selection tracker's `onSaveInstanceState()` (https://developer.android.com/reference/androidx/recyclerview/selection/SelectionTracker.html#onSaveInstanceState(android.os.Bundle)) and `onRestoreInstanceState()` (https://developer.android.com/reference/androidx/recyclerview/selection/SelectionTracker.html#onRestoreInstanceState(android.os.Bundle)) methods from the activity's `onSaveInstanceState()` (https://developer.android.com/reference/android/app/Activity.html#onSaveInstanceState(android.os.Bundle)) and `onRestoreInstanceState()` (https://developer.android.com/reference/android/app/Activity.html#onRestoreInstanceState(android.os.Bundle)) methods respectively. Your app must also supply a unique selection ID to the

`SelectionTracker.Builder` (https://developer.android.com/reference/androidx/recyclerview/selection/SelectionTracker.Builder.html)
constructor. This ID is required because an activity or a fragment may have more than one distinct, selectable list, all of which need to be persisted in their saved state.

Additional resources

`RecyclerView` is an Android Jetpack (https://developer.android.com/jetpack/) component. See it in use in the Sunflower (https://github.com/googlesamples/android-sunflower) demo app.

Last updated July 3, 2018.