Practical 9: RecyclerView

Letting the user display, scroll, and manipulate a list of similar data items is a common app feature. Examples of scrollable lists include contact lists, playlists, saved games, photo directories, dictionaries, shopping lists, and indexes of documents.

In the practical on scrolling views, you use ScrollView to scroll a View or ViewGroup. ScrollView is easy to use, but it's not recommended for long, scrollable lists.

RecyclerView is a subclass of ViewGroup and is a more resource-efficient way to display scrollable lists. Instead of creating a View for each item that may or may not be visible on the screen, RecyclerView creates a limited number of list items and reuses them for visible content.

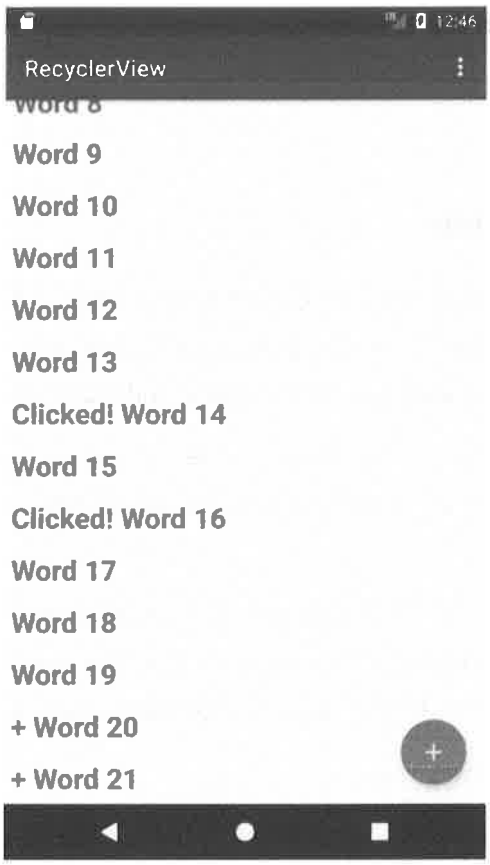
In this practical you do the following:

- Use RecyclerView to display a scrollable list.
- Add a click handler to each list item.
- Add items to the list using a floating action button (FAB), the pink button in the screenshot in the app overview section. Use a FAB for the primary action that you want the user to take.
  - Create a new app that uses a RecyclerView to display a list of items as a scrollable list and associate click behavior with the list items.
  - Use a FAB to let the user add items to the RecyclerView.

App Overview

The RecyclerView app demonstrates how to use a RecyclerView to display a long scrollable list of words. You create the dataset (the words), the RecyclerView itself, and the actions the user can take:

- Tapping a word marks it clicked.
- Tapping the floating action button (FAB) adds a new word.


12:46
RecyclerView
Word 8
Word 9
Word 10
Word 11
Word 12
Word 13
Clicked! Word 14
Word 15
Clicked! Word 16
Word 17
Word 18
Word 19
+ Word 20
+ Word 21

Task 1: Create a new project and dataset

Before you can display a **RecyclerView**, you **need data to display**. In this task, you will **create a new project** for the app and **a dataset**.

In a more sophisticated app, your data might come from **internal storage** (***a file, SQLite database, saved preferences***), from **another app** (**Contacts, Photos**), or from the **internet** (**cloud storage, Google Sheets, or any data source with an API**). Storing and retrieving data is a topic of its own covered in the data storage chapter.

For this exercise, you will **simulate data** by **creating it** in the **onCreate() method** of **MainActivity**.

1.1. Create the project and layout

1. Start Android Studio.
2. **Create** a **new project** with the name **RecyclerView**, select the **Basic Activity** template, and generate the layout file.

   The **Basic Activity template**, introduced in the chapter on using clickable images, provides a **floating action button** (FAB) and app bar in the **Activity layout** (**activity_main.xml**), and a layout for the **Activity content** (**content_main.xml**).
3. **Run** your app. You should see the RecyclerView app title and "Hello World" on the screen.

   If you encounter Gradle-related errors, sync your project as described in the practical on installing Android Studio and running Hello World.

1.2. Add code to create data

In this step you **create** a **LinkedList** of **20 word strings** that end in increasing numbers, as in ["Word 1", "Word 2", "Word 3", ...].

1. **Open MainActivity** and **add a private member variable** for the mWordList linked list.

```
public class MainActivity extends AppCompatActivity {
    private final LinkedList<String> mWordList = new LinkedList<>();
    // ... Rest of MainActivity code ...
}
```

2. **Add** code within the **onCreate() method** that **populates mWordList** with **words**:

```
// Put initial data into the word list.
for (int i = 0; i < 20; i++) {
        mWordList.addLast("Word " + i);
}
```

The code **concatenates the string "Word " with the value of i** while increasing its value. This is all you need as a **dataset** for this exercise.

1.3. Change the FAB icon

For this practical, you will **use a FAB to generate a new word to insert into the list**. The **Basic Activity template provides a FAB**, but you may want to **change its icon**. As you learned in another lesson, you can **choose an icon from the set of icons in Android Studio** for the FAB. Follow these steps:

1. Expand **res** in the **Project > Android** pane, and right-click (or Control-click) the **drawable** folder.
2. Choose **New > Image Asset**. The Configure Image Asset dialog appears.
3. Choose **Action Bar and Tab Items** in the drop-down menu at the top of the dialog.
4. Change **ic_action_name** in the **Name** field to **ic_add_for_fab**.
5. Click the clip art image (the Android logo next to **Clipart:**) to select a clip art image as the icon. A page of icons appears. Click the icon you want to use for the FAB, such as the plus (**+**) sign.
6. Choose **HOLO_DARK** from the **Theme** drop-down menu. This sets the icon to be white against a dark-colored (or black) background. Click **Next**.
7. Click **Finish** in the Confirm Icon Path dialog.

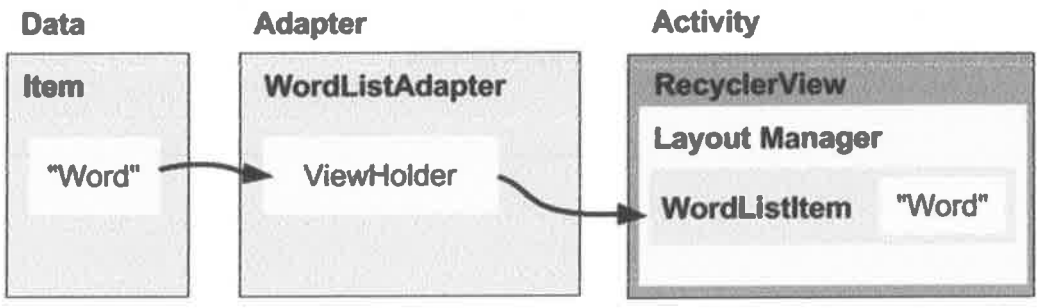
Tip: For a complete description of adding an icon, see Create app icons with Image Asset Studio.

Task 2: Create a RecyclerView

In this practical, you **display data** in a **RecyclerView**. You need the following:

- **Data to display**: Use the **mWordList**.
- **A RecyclerView** for the scrolling list that contains the **list items**.
- **Layout for one item of data**. All list items look the **same**.
- **A layout manager**. **RecyclerView.LayoutManager** handles the hierarchy and layout of **View** elements.
  - **RecyclerView** requires an **explicit layout manager** to manage the arrangement of list items contained within it.
  - This layout could be **vertical, horizontal**, or a grid.
  - You will use **a vertical LinearLayoutManager**.
- **An adapter**.
  - **RecyclerView.Adapter connects your data to the RecyclerView**.
  - It **prepares the data** in a **RecyclerView.ViewHolder**.
  - You will create an adapter that inserts into and updates your generated words in your views.
- **A ViewHolder**.
  - **Inside your adapter**, you will **create a ViewHolder** that contains the **View information** for **displaying one item** from the item's layout.

- The diagram below shows the relationship between the **data**, the **adapter**, the **ViewHolder**, and the **layout manager**.



To implement these pieces, you will need to :

1. **Add** a **RecyclerView element** to the **MainActivity XML content layout** (**content_main.xml**) for the RecyclerView app.
2. **Create** an XML layout file (**wordlist_item.xml**) for **one list item**, which is **WordListItem**.
3. **Create an adapter** (**WordListAdapter**) with a **ViewHolder** (**WordViewHolder**). **Implement** the **method that takes the data**, places it in the **ViewHolder**, and **lets the layout manager know to display it**.
4. In the **onCreate() method** of **MainActivity**, **create** a **RecyclerView** and **initialize it with the adapter and a standard layout manager**.

Let's do these one at a time.

2**.1. Modify the layout in content_main.xml**

To **add a RecyclerView element to the XML layout**, follow these steps:

1. Open **content_main.xml** in your **RecyclerView app**. It shows a "Hello World" **TextView** at the center of a ConstraintLayout.
2. Click the **Text** tab to show the **XML code**.
3. **Replace** the entire **TextView** element with the following:

```
<android.support.v7.widget.RecyclerView
    android:id="@+id/recyclerview"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
</android.support.v7.widget.RecyclerView>
```

You need to specify the full path (**android.support.v7.widget.RecyclerView**), because RecyclerView is part of the Support Library.

2**.2. Create the layout for one list item**

The adapter needs the layout for one item in the list. All the items use the **same layout**. You need to **specify that list item layout in a separate layout resource file**, because it is **used by the adapter**, separately from the RecyclerView.

Create a simple word item layout using a **vertical LinearLayout** with a **TextView**:

1. Right-click the **app > res > layout** folder and choose **New > Layout resource file**.
2. Name the file **wordlist_item** and click **OK**.
3. **In the new layout file**, click the **Text** tab to show the **XML code**.
4. **Change** the **ConstraintLayout** that was created with the file to a **LinearLayout** with the following attributes (extract resources as you go):

LinearLayout attribute	Value
android:layout_width	"match_parent"
android:layout_height	"wrap_content"
android:orientation	"vertical"
android:padding	"6dp"

5. **Add** a **TextView** for the word to the **LinearLayout.** Use **word** as the **ID** of the word:

Attribute	Value
android:id	"@+id/word"
android:layout_width	"match_parent"
android:layout_height	"wrap_content"
android:textSize	"24sp"
android:textStyle	"bold"

2.3 Create a style from the TextView attributes

You can use **styles** to allow elements to share groups of display attributes. An easy way to create a style is to **extract the style of a UI element** that you already created. To extract the style information for the **word TextView** in **wordlist_item.xml**:

1. Open **wordlist_item.xml** if it is not already open.
2. **Right-click** (or Control-click) the **TextView** you just created in **wordlist_item.xml**, and choose **Refactor > Extract > Style**. The **Extract Android Style** dialog appears.

3. **Name** your style **word_title** and leave all other options selected. Select the **Launch 'Use Style Where Possible'** option. Then click **OK**.
4. When prompted, **apply the style** to the **Whole Project**.
5. Find and examine the **word_title** style in **values > styles.xml**.
6. Reopen **wordlist_item.xml** if it is not already open. The **TextView** now uses the style in place of individual styling properties, as shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:padding="6dp">

    <TextView
        android:id="@+id/word"
        style="@style/word_title" />

</LinearLayout>
```

2**.4. Create an adapter**

Android uses adapters (from the Adapter **class) to connect data with View items in a list.**

- There are many different kinds of adapters available, and you can also write custom adapters.
- In this task you will create an adapter that associates your list of words with word list **View** items.

To connect data with **View items**, the **adapter needs to know about the View items**. The adapter uses a ViewHolder that **describes a View item** and **its position** within the **RecyclerView**.

First, you will **build an adapter** that **bridges** the gap between the **data** in your **word list** and the **RecyclerView that displays it**:

1. Right-click **java/com.android.example.recyclerview** and select **New > Java Class**.
2. Name the **class WordListAdapter**.

3. Give **WordListAdapter** the following signature:

```
public class WordListAdapter extends
    RecyclerView.Adapter<WordListAdapter.WordViewHolder> {}
```

WordListAdapter extends a **generic adapter for RecyclerView** to use a **View holder** that is **specific for your app** and defined inside **WordListAdapter. WordViewHolder** shows an error, because you **have not yet defined it**.

4. Click the **class declaration (WordListAdapter)**, then **click** the **red light bulb** on the left side of the pane. Choose **Implement methods**.

   A dialog appears that asks you to choose which methods to implement. Choose **all three methods** and click **OK**.

   Android Studio creates empty placeholders for all the methods. Note how **onCreateViewHolder** and **onBindViewHolder** both reference the **WordViewHolder**, which **hasn't been implemented yet**.

2.5 Create the ViewHolder for the adapter

To **create the ViewHolder**, follow these steps:

1. Inside the **WordListAdapter class**, **add a new WordViewHolder inner class** with this signature:

```
class WordViewHolder extends RecyclerView.ViewHolder {}
```

You will see an **error about a missing default constructor**. You can see details about the errors by hovering your mouse cursor over the red-underlined code or over any red horizontal line on the right margin of the editor pane.

2. **Add variables** to the **WordViewHolder inner class** for the **TextView** and the adapter:

```
public final TextView wordItemView;
final WordListAdapter mAdapter;
```

3. In the **inner class WordViewHolder**, **add** a **constructor** that initializes the **ViewHolder TextView** from the **word** XML resource, and sets its adapter:

```
public WordViewHolder(View itemView, WordListAdapter adapter) {
  super(itemView);
  wordItemView = itemView.findViewById(R.id.word);
  this.mAdapter = adapter;
}
```

4. Run your app to make sure that you have **no errors**. Your will still see only a **blank view**.
5. Click the **Logcat** tab to see the **Logcat** pane, and note the **E/RecyclerView: No adapter attached;** skipping layout warning. **You will attach the adapter to the RecyclerView in another step.**

2.6 Storing your data in the adapter

You need to **hold your data in the adapter**, and **WordListAdapter needs a constructor that initializes the word list** from the **data**. Follow these steps:

1. To **hold your data in the adapter**, create a **private linked list of strings** in **WordListAdapter** and call it **mWordList**.

```
private final LinkedList<String> mWordList;
```

2. You can now **fill in the getItemCount() method** to **return the size** of **mWordList**:

```
@Override
public int getItemCount() {
  return mWordList.size();
}
```

3. **WordListAdapter** needs a **constructor** that **initializes the word list from the data**.
   - To create a View for a list item, the WordListAdapter needs to inflate the XML for a list item.
   - You use a *layout inflator* for that job. **LayoutInflator** reads a layout XML description and converts it into the corresponding View items.
   - Start by creating a **member variable for the inflater** in **WordListAdapter**:

```
private LayoutInflater mInflater;
```

4. **Implement the constructor** for WordListAdapter.

   - The constructor needs to have a **context parameter**, and a **linked list of words** with the app's data.
   - The method needs to **instantiate a LayoutInflator** for **mInflater** and set **mWordList** to the **passed in data**:

```
public WordListAdapter(Context context,
                       LinkedList<String> wordList) {
  mInflater = LayoutInflater.from(context);
  this.mWordList = wordList;
}
```

5. Fill out the **onCreateViewHolder() method** with this code:

```
@Override
public WordViewHolder onCreateViewHolder(ViewGroup parent,
                                         int viewType) {
  View mItemView = mInflater.inflate(R.layout.wordlist_item,
                                     parent, false);
  return new WordViewHolder(mItemView, this);
}
```

   - The **onCreateViewHolder()** method is similar to the onCreate() method.
   - It **inflates the item layout**, and **returns** a **ViewHolder with the layout** and the **adapter**.

6. Fill out the onBindViewHolder() method with the code below:

```
@Override
public void onBindViewHolder(WordViewHolder holder, int position) {
  String mCurrent = mWordList.get(position);
  holder.wordItemView.setText(mCurrent);
}
```

   The **onBindViewHolder() method connects** your **data to the view holder**.

7. Run your app to make sure that there are no errors.

2**.7. Create the RecyclerView in the Activity**

Now that you have an adapter with a ViewHolder, you can **finally create a RecyclerView and connect all the pieces to display your data.**

1. Open **MainActivity**.

2. **Add member variables** for the RecyclerView and the **adapter**.

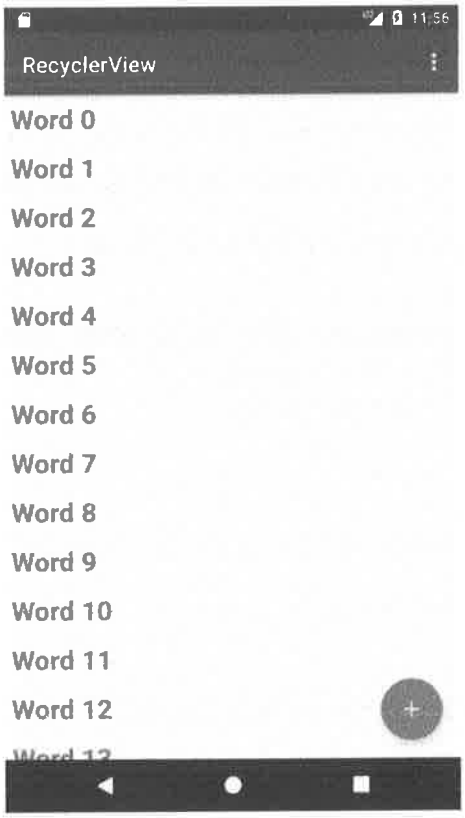
```
private RecyclerView mRecyclerView;
private WordListAdapter mAdapter;
```

3. In the **onCreate() method** of **MainActivity**, add the following code that **creates the RecyclerView and connects it with an adapter and the data**. The comments explain each line. You must **insert this code *after* the mWordList initialization**.

```
// Get a handle to the RecyclerView.
mRecyclerView = findViewById(R.id.recyclerview);

// Create an adapter and supply the data to be displayed.
mAdapter = new WordListAdapter(this, mWordList);

// Connect the adapter with the RecyclerView.
mRecyclerView.setAdapter(mAdapter);

// Give the RecyclerView a default layout manager.
mRecyclerView.setLayoutManager(new LinearLayoutManager(this));
```

4. Run your app.

   You should see your list of words displayed, and you can scroll the list.



Task 3: Make the list interactive

Looking at lists of items is interesting, but it's a lot more fun and useful if your user can interact with them. To see how the **RecyclerView** can **respond to user input**, you will **attach a click handler to each item**. When the item is tapped, the click handler is executed, and that item's text will change.

The list of items that a **RecyclerView** displays can also be **modified dynamically**—it doesn't have to be a static list.

There are several ways to add additional behaviors. One is to use the floating action button (FAB). For example, in Gmail, the **FAB** is used to compose a new email.

For this practical, you will **generate a new word to insert into the list**. For a more useful app, you would get data from your users.

3**.1. Make items respond to clicks**

1. Open **WordListAdapter**.
2. **Change** the WordViewHolder class signature to implement View.onClickListener:

```
class WordViewHolder extends RecyclerView.ViewHolder
                    implements View.OnClickListener {
```

3. **Click the class header** and on the **red light bulb** to **implement stubs for the required methods**, which in this case is just the **onClick() method**.
4. **Add** the following **code** to the body of the **onClick() method**.

```
// Get the position of the item that was clicked.
int mPosition = getLayoutPosition();

// Use that to access the affected item in mWordList.
String element = mWordList.get(mPosition);

// Change the word in the mWordList.
mWordList.set(mPosition, "Clicked! " + element);

// Notify the adapter, that the data has changed so it can
// update the RecyclerView to display the data.
mAdapter.notifyDataSetChanged();
```

5. **Connect** the **onClickListener with the View**. Add this code to the **WordViewHolder constructor** (below the this.mAdapter = adapter line):

```
itemView.setOnClickListener(this);
```

6. Run your app. Click items to see the text change.

3**.2. Add behavior to the FAB**

In this task you will implement an action for the FAB to:

- **Add a word** to the end of the list of words.
- **Notify the adapter** that the data has changed.
- **Scroll to the inserted item**.

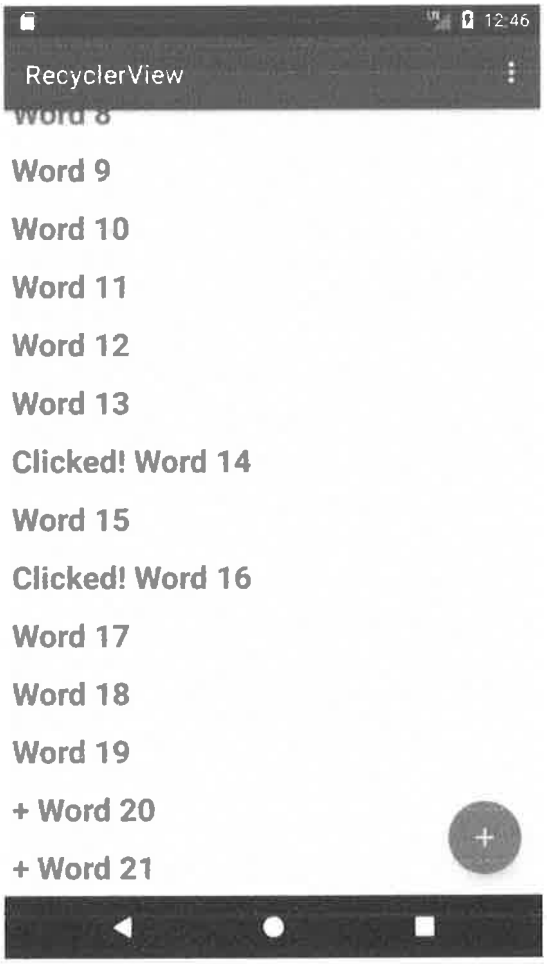
Follow these steps:

1. Open **MainActivity**. The **onCreate() method** sets an **OnClickListener()** to the **FloatingActionButton** with an **onClick() method** for taking an action.

   Change the **onClick() method** to the following:

```
@Override
public void onClick(View view) {
    int wordListSize = mWordList.size();

    // Add a new word to the wordList.
    mWordList.addLast("+ Word " + wordListSize);

    // Notify the adapter, that the data has changed.
    mRecyclerView.getAdapter().notifyItemInserted(wordListSize);

    // Scroll to the bottom.
    mRecyclerView.smoothScrollToPosition(wordListSize);
}
```

2. Run the app.
3. Scroll the list of words and click items.
4. Add items by clicking the FAB.



What happens if you rotate the screen? You will learn in a later lesson how to preserve the state of an app when the screen is rotated.