Practical class # 9 – RecyclerView

1) Create an empty activity project

2) Add affirmation strings

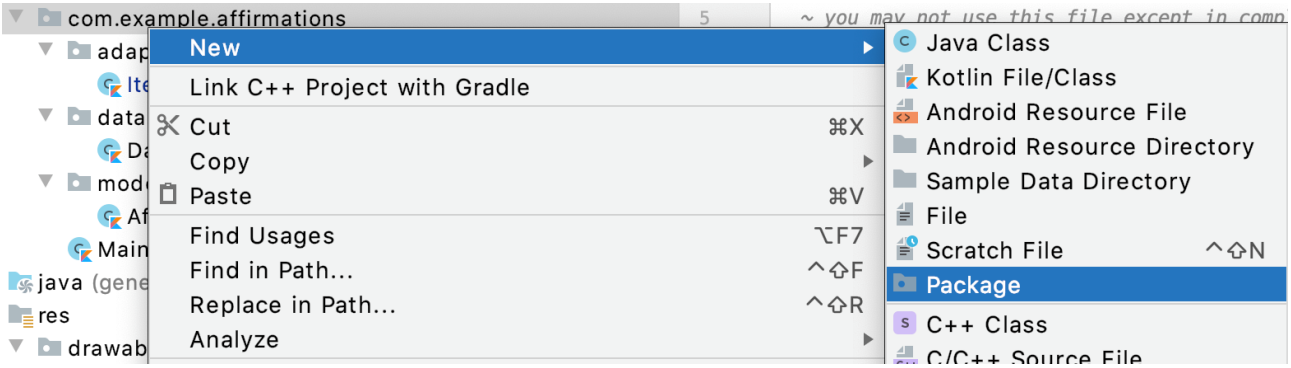
1. In the **Project** window, open **app > res > values > strings.xml**. This file currently has a single resource which is the name of the app.
2. In strings.xml, add the following affirmations as individual string resources. Name them affirmation1, affirmation2, and so on.

```
I am strong.
I believe in myself.
Each day is a new opportunity to grow and be a better version of myself.
Every challenge in my life is an opportunity to learn from.
I have so much to be grateful for.
Good things are always coming into my life.
New opportunities await me at every turn.
I have the courage to follow my heart.
Things will unfold at precisely the right time.
I will be present in all the moments that this day brings.
```

<string name="affirmation1">I am strong.</string> and so on

3) Create a package

1. In Android Studio, in the **Project** pane, right-click **app > java > com.example.affirmations** and select **New > Package**.

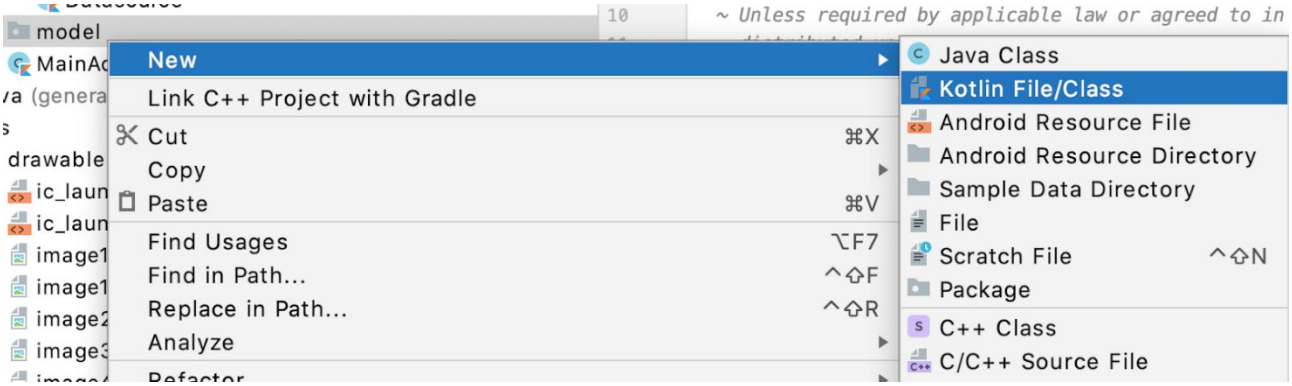


2. In the **New Package** popup, notice the suggested package name prefix. The suggested first part of the package name is the name of the package you right-clicked. While package names do not create a hierarchy of packages, reusing parts of the name is used to indicate a relationship and organization of the content!
3. In the popup, append **model** to the end of the suggested package name. Developers often use **model** as the package name for classes that model (or represent) the data.

4. Press **Enter**. This creates a new package under the **com.example.affirmations** (root) package. This new package will contain any data-related classes defined in your app.

4) Create the Affirmation class

5. Right-click on the **com.example.affirmations.model** package and select **New > Kotlin File/Class.**



6. In the popup, select **Class** and enter Affirmation as the name of the class. This creates a new file called Affirmation.kt in the model package.
7. Make Affirmation a data class by adding the data keyword before the class definition. This leaves you with an error, because data classes must have at least one property defined.

Change the code to solve the error

```
package com.example.affirmations.model

data class Affirmation(val stringResourceId: Int)
```

5) Create a class for being a Datasource

Data displayed in your app may come from different sources (e.g. within your app project or from an external source that requires connecting to the internet to download data). As a result, data may not be in the exact format that you need. The rest of the app should not concern itself with where the data originates from or in what format it is originally. You can and should hide away this data preparation in a separate Datasource class that prepares the data for the app.

Since preparing data is a separate concern, put the Datasource class in a separate **data** package.

1. In Android Studio, in the **Project** window, right-click **app > java > com.example.affirmations** and select **New > Package**.
2. Enter data as the last part of the package name.
3. Right click on the data package and select **new Kotlin File/Class.**
4. Enter Datasource as the class name.
5. Inside the Datasource class, create a function called loadAffirmations().

The loadAffirmations() function needs to return a list of Affirmations. You do this by creating a list and populating it with an Affirmation instance for each resource string.

6. Declare List<Affirmation> as the return type of the method loadAffirmations().
7. In the body of loadAffirmations(), add a return statement.
8. After the return keyword, call listOf<>() to create a List.
9. Inside the angle brackets <>, specify the type of the list items as Affirmation. If necessary, import com.example.affirmations.model.Affirmation.
10. Inside the parentheses, create an Affirmation, passing in R.string.affirmation1 as the resource ID as shown below.

Affirmation(R.string.affirmation1)

11. Add the remaining Affirmation objects to the list of all affirmations, separated by commas. The finished code should look like the following.

```
package com.example.affirmations.data

import com.example.affirmations.R
import com.example.affirmations.model.Affirmation


class Datasource {

    fun loadAffirmations(): List<Affirmation> {
        return listOf<Affirmation>(
            Affirmation(R.string.affirmation1),
            Affirmation(R.string.affirmation2),
            Affirmation(R.string.affirmation3),
            Affirmation(R.string.affirmation4),
            Affirmation(R.string.affirmation5),
            Affirmation(R.string.affirmation6),
            Affirmation(R.string.affirmation7),
            Affirmation(R.string.affirmation8),
            Affirmation(R.string.affirmation9),
            Affirmation(R.string.affirmation10)
        )
    }
}
```

Display the size of the list in a textview

```
val textView: TextView = findViewById(R.id.textview)
textView.text = Datasource().loadAffirmations().size.toString()
```

6) Add a RecyclerView to the app

1. Open activity_main.xml
2. Delete the TextView and the ConstraintLayout
3. Use the FrameLayout:

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

4. Switch to Design view.
5. In the Palette, select Containers, and find the RecyclerView.
6. Drag a RecyclerView into the layout.
7. If it appears, read the Add Project Dependency popup and click OK. (If the popup doesn't appear, no action is needed.)
8. Wait for Android Studio to finish and the RecyclerView to appear in your layout.
9. If necessary, change the layout_width and layout_height attributes of the RecyclerView to match_parent so that the RecyclerView can fill the whole screen.
10. Set the resource ID of the RecyclerView to recycler_view.
11. Switch back to Code view. In the XML code, inside the RecyclerView element, add LinearLayoutManager as the layout manager attribute of the RecyclerView and add a scrollbar

```
app:layoutManager="LinearLayoutManager"
android:scrollbars="vertical"
```

7) Create an adapter

<u>Create a layout to display the elements</u>

In res > layout, create a new empty File called list_item.xml.
Open list_item.xml in Code view.
Add a TextView with id item_title.
Add wrap_content for the layout_width and layout_height

<u>Create the adapter</u>

In Android Studio in the Project pane, right-click app > java > com.example.affirmations and select New > Package.
Enter adapter as the last part of the package name.
Right-click on the adapter package and select New > Kotlin File/Class.
Enter ItemAdapter as the class name, finish, and the ItemAdapter.kt file opens.

Add a parameter to the ItemAdapter constructor that is a val called dataset of type List<Affirmation>. Import Affirmation, if necessary.

Since the dataset will be only used within this class, make it private

```
class ItemAdapter(private val dataset: List<Affirmation>) {

}
```

Add a parameter to the ItemAdapter constructor that is a val called context of type Context. Position it as the first parameter in the constructor.

```
class ItemAdapter(private val context: Context, private val dataset: List<Affirmation>) {

}
```

Create a ViewHolder

RecyclerView does not interact directly with item views, but deals with ViewHolders instead. A ViewHolder represents a single list item view in RecyclerView, and can be reused when possible. A ViewHolder instance holds references to the individual views within a list item layout (hence the name "view holder"). This makes it easier to update the list item view with new data. View holders also add information that RecyclerView uses to efficiently move views around the screen.

Inside the ItemAdapter class, before the closing curly brace for ItemAdapter, create an ItemViewHolder class.

```
class ItemAdapter(private val context: Context, private val dataset: List<Affirmation>) {

        class ItemViewHolder()
}
```

Add a private val view of type View as a parameter to the ItemViewHolder class constructor.

Make ItemViewHolder a subclass of RecyclerView.ViewHolder and pass the view parameter into the superclass constructor.

Inside ItemViewHolder, define a val property textView that is of type TextView. Assign it the view with the ID item_title that you defined in list_item.xml.

```
class ItemAdapter(private val context: Context, private val dataset: List<Affirmation>) {

        class ItemViewHolder(private val view: View) : RecyclerView.ViewHolder(view) {
                val textView: TextView = view.findViewById(R.id.item_title)
        }
}
```

Override Adapter Methods

Add the code to extend your ItemAdapter from the abstract class RecyclerView.Adapter. Specify ItemAdapter.ItemViewHolder as the view holder type in angle brackets.

```
class ItemAdapter(
                private val context: Context,
```

```
        private val dataset: List<Affirmation>
        ) : RecyclerView.Adapter<ItemAdapter.ItemViewHolder>()
```

Put your cursor on ItemAdapter and press **Command+I** (**Control+I** on Windows). This shows you the list of methods you need to implement: getItemCount(), onCreateViewHolder(), and onBindViewHolder().

Select all three functions using **Shift+click** and click **OK**

Implement getItemCount()

```
override fun getItemCount() = dataset.size
```

Implement onCreateViewHolder()

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ItemViewHolder {
   // create a new view
   val adapterLayout = LayoutInflater.from(parent.context)
      .inflate(R.layout.list_item, parent, false)

   return ItemViewHolder(adapterLayout)
}
```

Implement onBindViewHolder()

```
override fun onBindViewHolder(holder: ItemViewHolder, position: Int) {
   val item = dataset[position]
   holder.textView.text =  context.resources.getString(item.stringResourceId)
}
```

8) Modify the MainActivity to use the RecyclerView

```
class MainActivity : AppCompatActivity() {


   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_main)


      // Initialize data.
      val myDataset = Datasource().loadAffirmations()


      val recyclerView = findViewById<RecyclerView>(R.id.recycler_view)
      recyclerView.adapter = ItemAdapter(this, myDataset)
```

```
        // Use this setting to improve performance if you know that changes
        // in content do not change the layout size of the RecyclerView
        recyclerView.setHasFixedSize(true)
    }
}
```