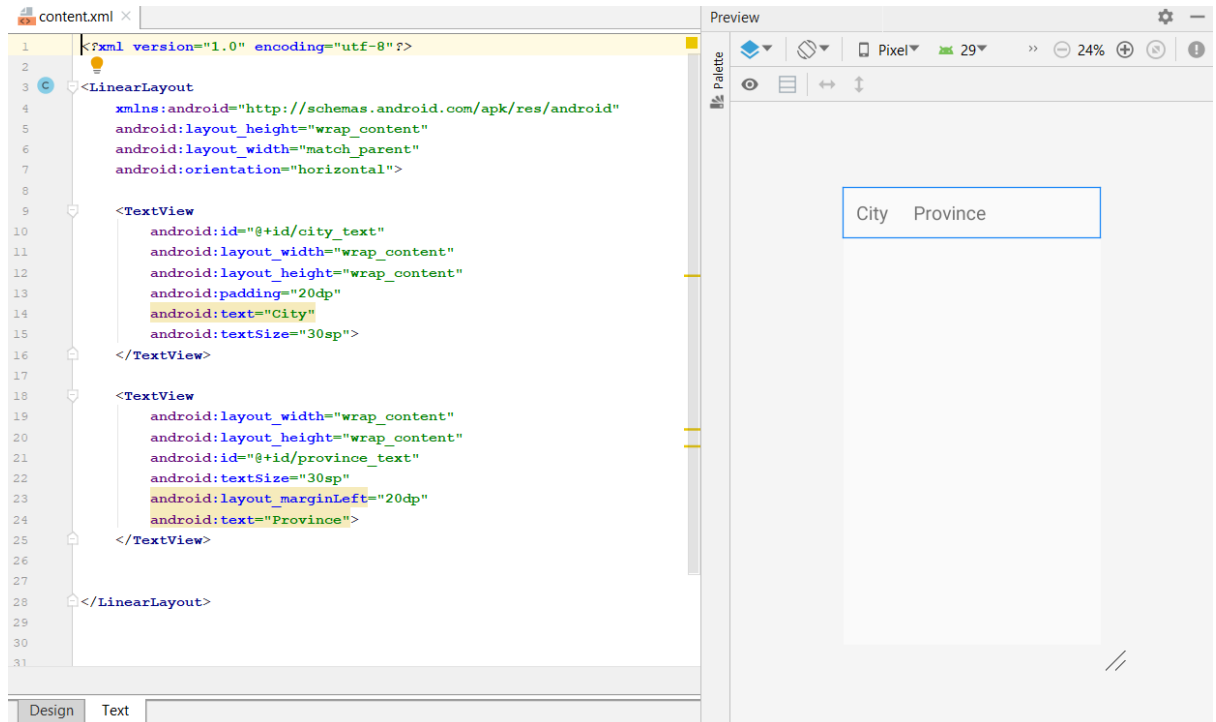


Lab 3 Instructions – CustomList

1. Download the ListyCity Project using the link below (if you don't have it on your system):
<https://drive.google.com/open?id=1qVToKQTy04CZJ9-Kf9h-sRrPOjpGh5WW>
 - a. Extract the project somewhere on the computer
 - b. Open Android Studio
 - c. Go to File-> Open -> Browse to the extract location and select the project.
2. After all that is done, go to content.xml, and then add another textView as follows:

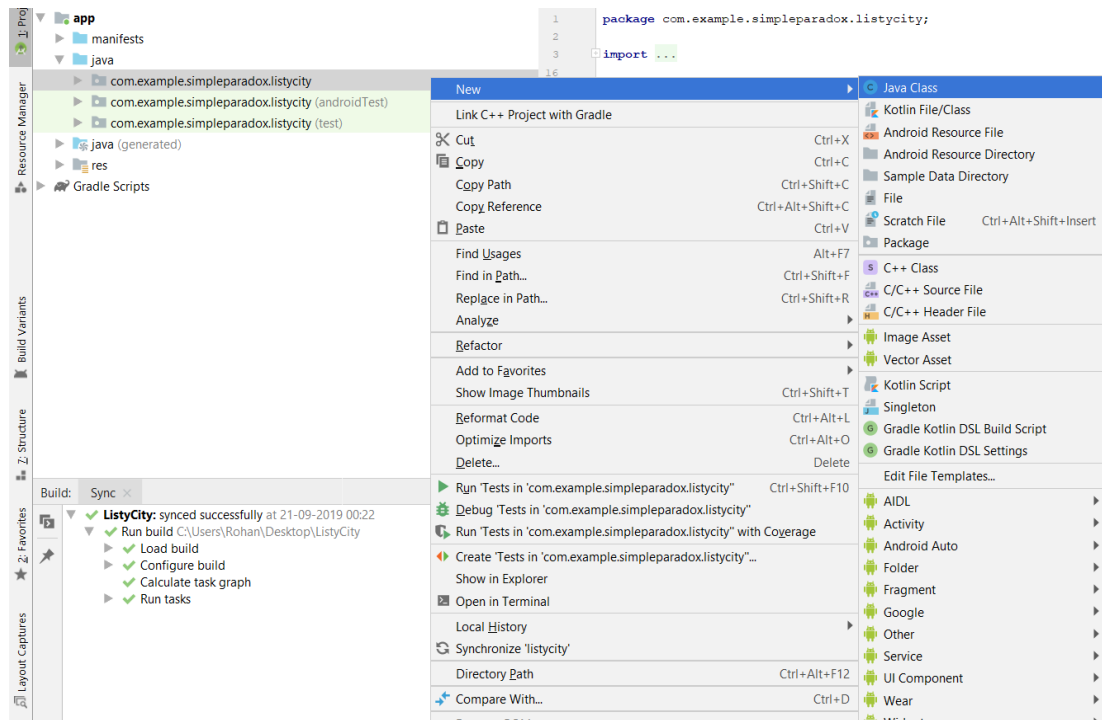


Note that the previous TextView and the new TextView are inside a LinearLayout with orientation="horizontal". Also, the ID for the first TextView is changed to 'city_text' to make it meaningful. Make sure to add an ID for the second TextView too. So that we can refer the view later while coding up the logic.

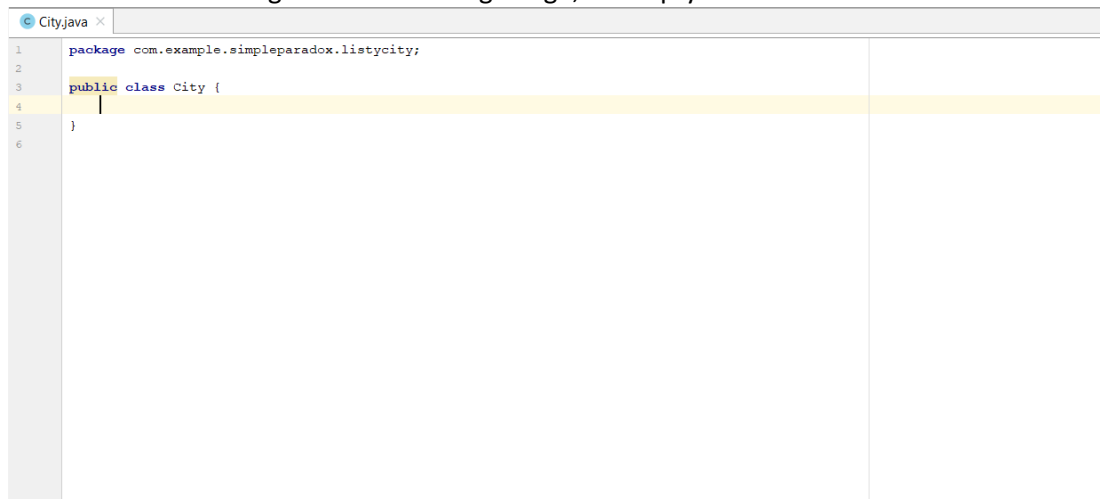
(Think about other views you can add instead of a TextView. What if you wanted to display an Image?)

- Since we want to display more than one element in a row of the text view, we will need a class to hold the data.

In the project pane, right click on the directory containing MainActivity.java and select New -> Java Class. Name the class 'City' and then press 'Ok'.



You will see something like the following image; an empty class.



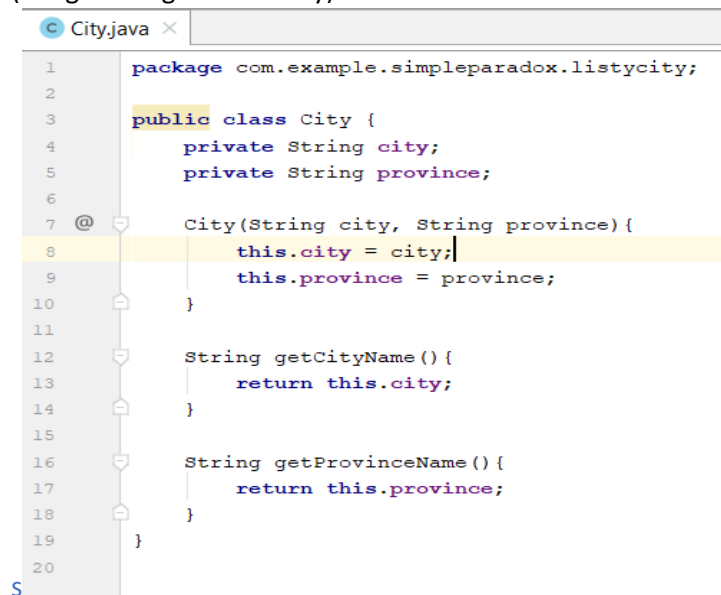
4. Create private two variables
private String city;
private String province;

Create a constructor with two parameters, city and province.

Assign the values to the variables you declared earlier. (Use the 'this' keyword).

The last thing to add to this class are getters for the city and province. [Can think of why we are using getters to access the attributes?](#)

(Image enlarged for clarity)

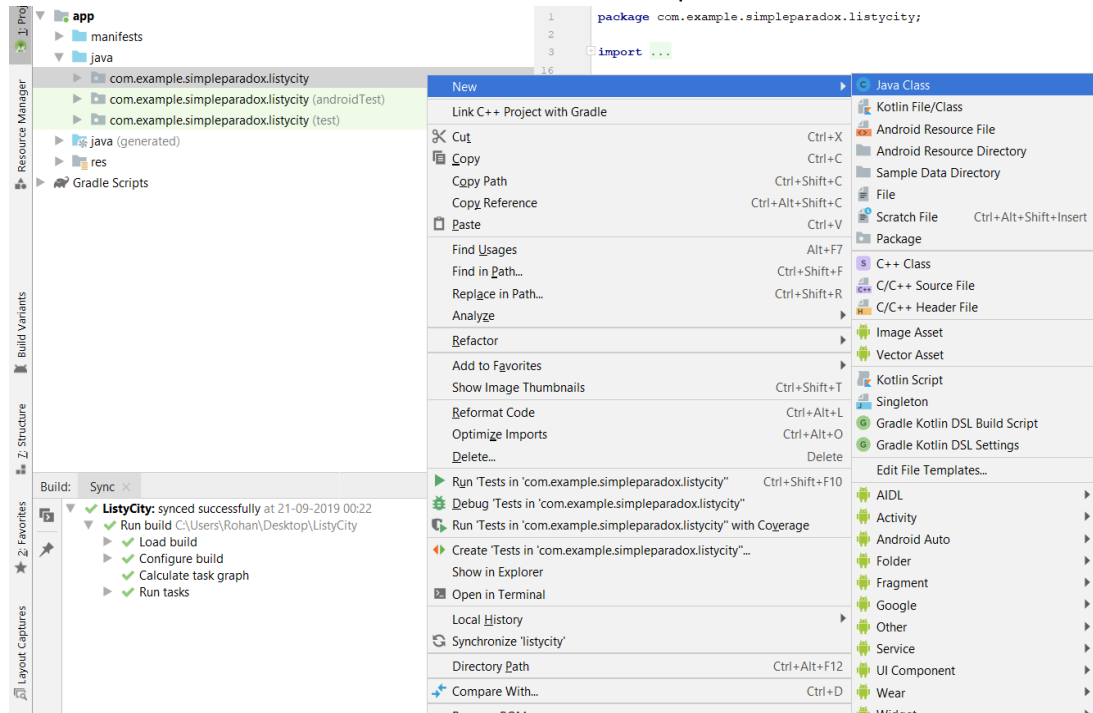


```
1 package com.example.simpleparadox.listcity;  
2  
3 public class City {  
4     private String city;  
5     private String province;  
6  
7     @City(String city, String province) {  
8         this.city = city;  
9         this.province = province;  
10    }  
11  
12    String getCityName() {  
13        return this.city;  
14    }  
15  
16    String getProvinceName() {  
17        return this.province;  
18    }  
19 }  
20
```

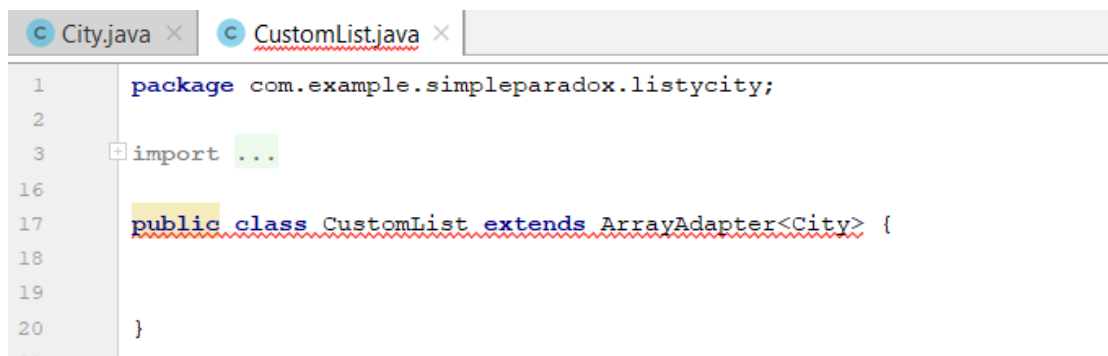
5. Since now we also have the province as an extra information to be displayed with the city name, the default ListAdapter will not work. Therefore, we need a custom ListAdapter that we can modify it as per our requirements.

Again, in the project pane, right click on the directory containing MainActivity.java and select

New -> Java Class. Name the class 'CustomList' and then press 'Ok'.



- When you have the empty 'CustomList' class, then extend it from the ArrayAdapter class so that you can customize the ArrayAdapter as per your requirements. Make sure that ArrayAdapter has a type of <City>. Something like the following image.



Android Studio will give you an error. This is because you will have to implement the constructor present in the parent class.

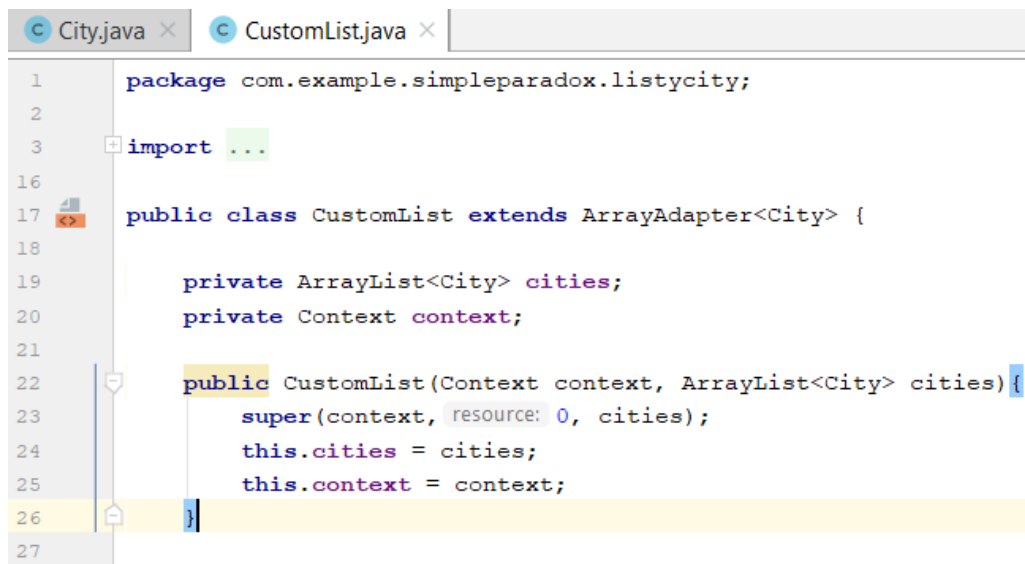
- Before defining a constructor, declare two private variables as follows:

```
private ArrayList<City> cities;  
private Context context;
```

This two variables will hold the cities (city name and province) and the activity context respectively.

- Implement the constructor now as follows:

```
public CustomList(Context context, ArrayList<City> cities){  
    super(context,0, cities);  
    this.cities = cities;  
    this.context = context;  
}
```



```
1 package com.example.simpleparadox.listcity;
2
3 import ...
4
5
6
7
8
9
10
11
12
13
14
15
16
17 public class CustomList extends ArrayAdapter<City> {
18
19     private ArrayList<City> cities;
20     private Context context;
21
22     public CustomList(Context context, ArrayList<City> cities) {
23         super(context, resource: 0, cities);
24         this.cities = cities;
25         this.context = context;
26     }
27 }
```

9. Now you have to implement the 'getView' method which allows you to set the values for the views in your 'listview'. While typing 'getView' the autocomplete should suggest you with the method 'getView'. However, it is extremely important that you understand what each parameter represents. For this tutorial, we will omit this as it is out of scope for this tutorial.

When you get the empty getView method, remove or comment the default statement.

```
// return super.getView(position, convertView, parent);
```

10. Now we have an empty method. Let's write some code so that get references to the TextViews in the content.xml layout file and fill them with values (city and province).

```
public View getView(int position, @Nullable View convertView, @NonNull ViewGroup parent) {
    // return super.getView(position, convertView, parent);
    View view = convertView;

    if(view == null){
        view = LayoutInflater.from(context).inflate(R.layout.content, parent,false);
    }

    City city = cities.get(position);

    TextView cityName = view.findViewById(R.id.city_text);
    TextView provinceName = view.findViewById(R.id.province_text);

    cityName.setText(city.getCityName());
    provinceName.setText(city.getProvinceName());

    return view;
}
```

The 'convertView' object is a way to recycle old views inside the 'ListView' ultimately increasing the performance of the 'ListView'. First we get the reference to the current

‘convertView’ object.

If the ‘convertView’ object holds nothing, then we inflate the ‘content.xml’.

The next step is to extract the information from the ‘cities’ list and the set the name of the city and the province to each ‘TextView’.

```
@NonNull
@Override
public View getView(int position, @Nullable View convertView, @NonNull ViewGroup parent) {
    /
    return super.getView(position, convertView, parent);
    View view = convertView;

    if(view == null){
        view = LayoutInflater.from(context).inflate(R.layout.content, parent, attachToRoot: false);
    }

    City city = cities.get(position);

    TextView cityName = view.findViewById(R.id.city_text);
    TextView provinceName = view.findViewById(R.id.province_text);

    cityName.setText(city.getCityName());
    provinceName.setText(city.getProvinceName());

    return view;
}
```

11. Now it's time to make sure that all that we have built so far, takes in the data from the ‘MainActivity’ (in this case, the city names and the provinces for each city).

- a. Go to ‘MainActivity.java’
- b. Declare three variables
 - i. A ListView
 - ii. An ArrayAdapter<City> of type City
 - iii. An ArrayList<City> of type City

```
public class MainActivity extends AppCompatActivity {

    // Declare the variables so that you will be able to reference it later.
    ListView cityList;
    ArrayAdapter<City> cityAdapter;
    ArrayList<City> cityDataList;
```

You can name the variables anyway you want.

- c. Inside the onCreate() method, get a reference to the list view from activity_main.xml. In this case, we get a reference to it using its ID ‘city_list’.
- Also, we create two arrays of Strings containing the city names and the provinces.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    cityList = findViewById(R.id.city_list);

    String []cities ={"Edmonton", "Vancouver", "Toronto", "Hamilton", "Denver", "Los Angeles"};
    String []provinces = {"AB", "BC", "ON", "ON", "CO", "CA"};
```

You may add more cities and provinces.

- d. The next step is to create an 'ArrayList' and then add the cities in it.

```
cityDataList = new ArrayList<>();  
for(int i=0;i<cities.length;i++){  
    cityDataList.add((new City(cities[i], provinces[i])));  
}
```

- e. Now the last and final part is to pass the 'cityDataList' to the 'CustomList' custom adapter we created earlier and then set the list view adapter to the custom adapter. This will ensure that the data is being shown in the list view. Follow the image below.

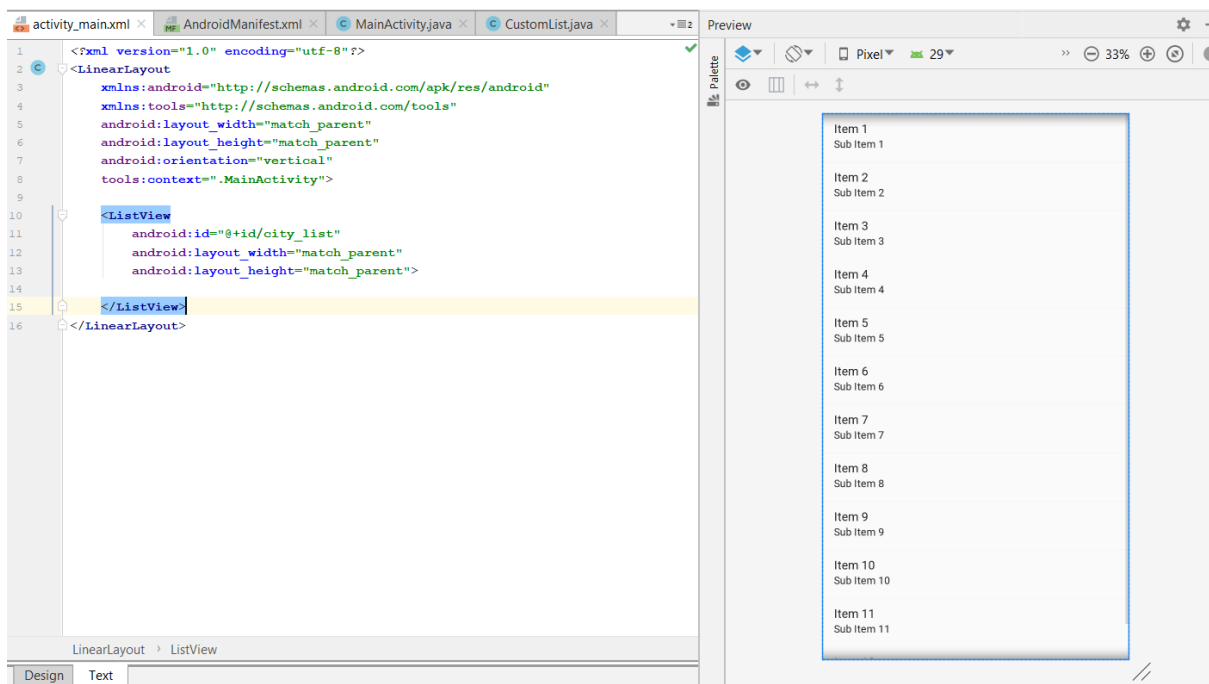
```
cityAdapter = new CustomList( context: this, cityDataList);  
  
cityList.setAdapter(cityAdapter);
```

The 'new CustomList(...)' method will invoke the constructor you defined in the 'CustomList' class and pass the data for it to be displayed in each row of the list view.

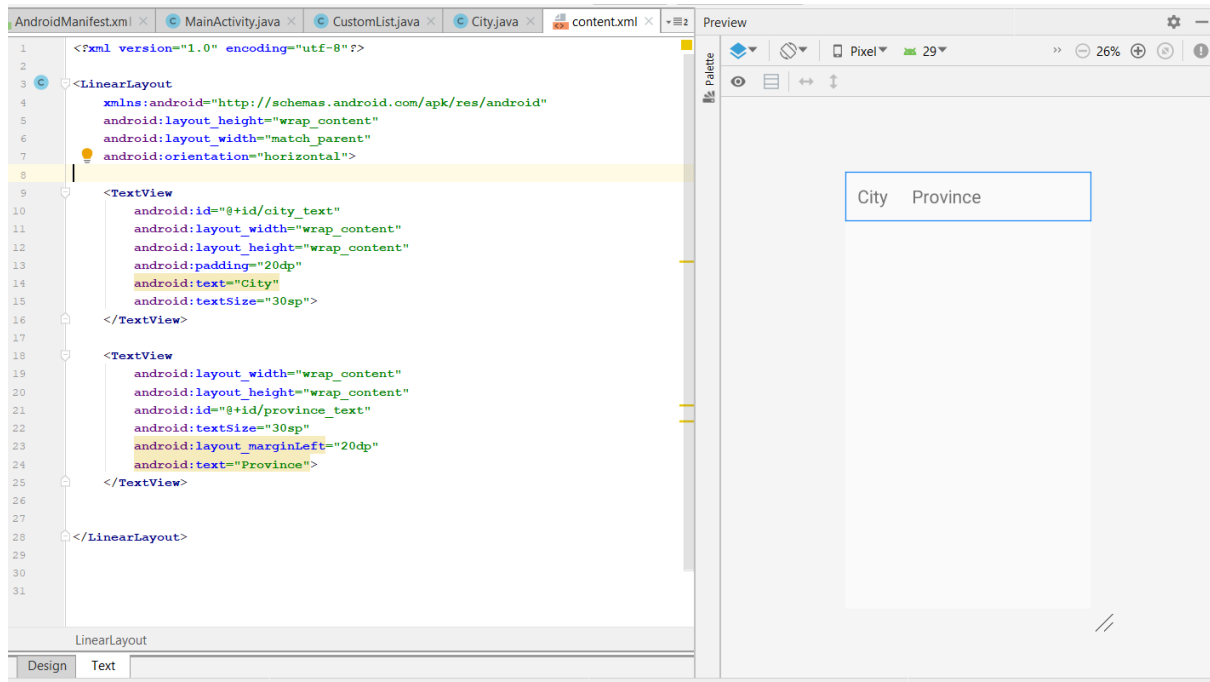
- f. Make sure you run the app and check for any errors if encountered.

After going through all the instructions above, your layout files and Java files should look like this.

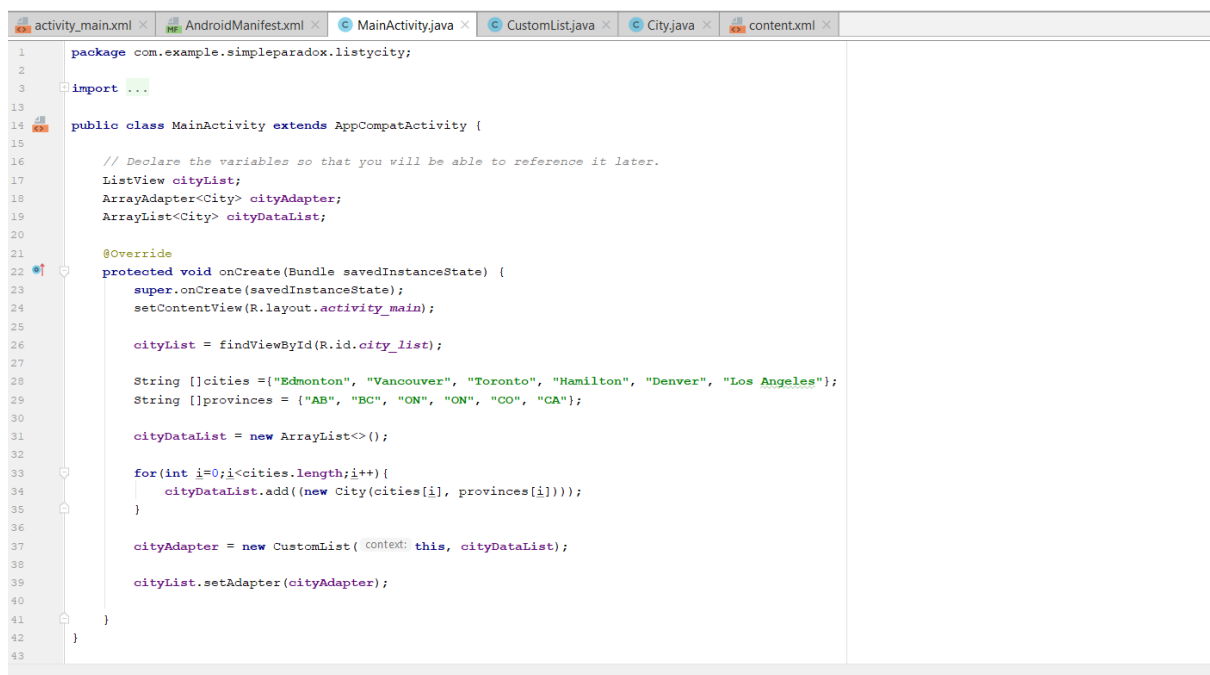
1. activity_main.xml



2. content.xml



3. MainActivity.java



4. CustomList.java

```
activity_main.xml x AndroidManifest.xml x MainActivity.java x CustomList.java x City.java x content.xml x
16
17 public class CustomList extends ArrayAdapter<City> {
18
19     private ArrayList<City> cities;
20     private Context context;
21
22     public CustomList(Context context, ArrayList<City> cities) {
23         super(context, resource: 0, cities);
24         this.cities = cities;
25         this.context = context;
26     }
27     @NonNull
28     @Override
29     public View getView(int position, @Nullable View convertView, @NonNull ViewGroup parent) {
30         // return super.getView(position, convertView, parent);
31         View view = convertView;
32
33         if (view == null) {
34             view = LayoutInflater.from(context).inflate(R.layout.content, parent, attachToRoot: false);
35         }
36
37         City city = cities.get(position);
38
39         TextView cityName = view.findViewById(R.id.city_text);
40         TextView provinceName = view.findViewById(R.id.province_text);
41
42         cityName.setText(city.getCityName());
43         provinceName.setText(city.getProvinceName());
44
45         return view;
46     }
47 }
48
49 CustomList
```

5. City.java

```
activity_main.xml x AndroidManifest.xml x MainActivity.java x CustomList.java x City.java x content.xml x
1 package com.example.simpleparadox.listcity;
2
3 public class City {
4     private String city;
5     private String province;
6
7     City(String city, String province) {
8         this.city = city;
9         this.province = province;
10    }
11
12    String getCityName() {
13        return this.city;
14    }
15
16    String getProvinceName() { return this.province; }
17
18 }
19
20 City > City()
```

6. The final custom list should look something like this.

