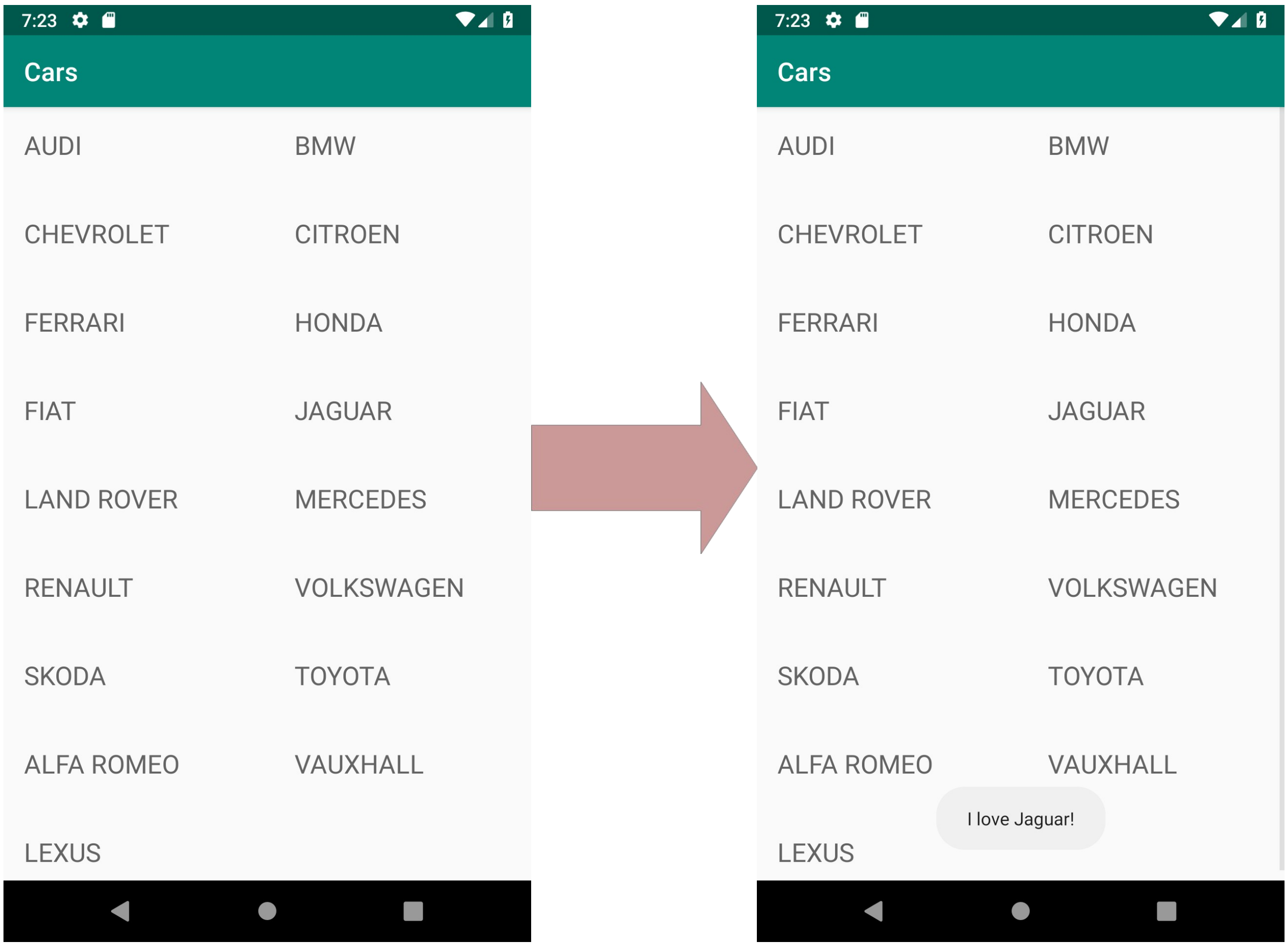
Last week we created an app that responded to the user clicking a button by displaying a toast message on the screen i.e. “The user clicked the button!”. This week our goal is to build on this and get some practice using **adapters** by creating a new app. The way the app works is simple, there is a screen with a **grid of text fields** containing the names of cars, when the user clicks on the text field a toast message appears on the screen stating that “I love” the corresponding car e.g. In the image below the user clicked on “Jaguar”.



7:23

Cars

AUDI	BMW
CHEVROLET	CITROEN
FERRARI	HONDA
FIAT	JAGUAR
LAND ROVER	MERCEDES
RENAULT	VOLKSWAGEN
SKODA	TOYOTA
ALFA ROMEO	VAUXHALL
LEXUS	

7:23

Cars

AUDI	BMW
CHEVROLET	CITROEN
FERRARI	HONDA
FIAT	JAGUAR
LAND ROVER	MERCEDES
RENAULT	VOLKSWAGEN
SKODA	TOYOTA
ALFA ROMEO	VAUXHALL
LEXUS	

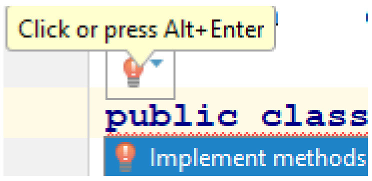
I love Jaguar!

Based on the same procedure used the last couple of weeks create a new project, you can call the application "My Third App". Use the provided **layout_main.xml** file for your layout i.e. you don't have to change anything in this file, just paste it over the one created automatically.

Part 1 Create the Adapter

As mentioned in the lecture notes, the adapter is the middleman that is going to manage the data source for the grid view

1. Create a new class called `MyAdapter` in the same folder as `MainActivity.java`
2. Tell this class to inherit from `BaseAdapter`. As a result of this action Android Studio should prompt us with a red warning (below). If you click on **implement methods** some methods will appear in the file. We'll worry about these in a minute.



3. Add the following lines to the top of `BaseAdapter` before any of the methods:

```
private Context mContext;
public MyAdapter(Context c) {
      mContext = c;
}
```

4. Define an array of `Strings` that contains the names of the cars in the screenshot for the app (an alternative approach would be to define an array of `ints` that refer to names in `strings.xml`, though this is a little more tricky).
5. Fill in the code for `getCount()` and `getItem()`. These methods have **one line of code** each, `getCount()` returns the length of the array and `getItem()` returns the string in the array at index `position`.

Part 2 Define text fields programmatically

Last week we were defining our UI widgets like text fields and buttons in xml. This week we are going to do it instead in **Java**.

1.) Create a new file called `dimensions.xml` in the **values** sub-folder of **res**

2.) The root element of this file is called **resources**. Create two child elements of type **dimen** inside this element. Give the first element a **name** attribute with value `pad_size` and the second one a **name** attribute with value text_size. These elements contain text values of `16dp` and `20sp` respectively.

3.) Put the following starter code in the `getView()` method of the `MyAdapter` class to set up the `TextView` cells of the grid.

```
TextView textView;
if (convertView == null) {

    // get the dp and sp values from dimens.xml and convert to pixels
    int space = mContext.getResources().getDimensionPixelSize(R.dimen.pad_size);
    float txtSize = mContext.getResources().getDimensionPixelSize( R.dimen.text_size );

    // create a TextView object
    textView = new TextView(mContext);
    // set the width and height of the TextView
    textView.setLayoutParams(new ViewGroup.LayoutParams(ViewGroup.LayoutParams.WRAP_CONTENT,
    ViewGroup.LayoutParams.WRAP_CONTENT));

    // your code goes here!


} else {
    textView = (TextView) convertView;
}
```

Note that the code above (lines 3 and 4) is accessing the size values that you defined in `dimensions.xml` and converting them to pixels.

4.) We would like to finish off the code above defining how a `TextView` looks in a cell of the grid. Using the `textView` variable set the **padding**, **text size**, **CAPITAL** letters and **colour**.

5.) The next step in `getView()` is to **set the text** for the `textView` variable. This can be done by calling `setText()` on `textView` with the text at **index** `position` of the array you created earlier.

6.) The final step is to return `textView`.

Part 3 Making the Toast Message

The idea here is almost exactly the same as last week with some minor differences:

1.) Set the `adapter` for the `gridView` object.

2.) Fill in the toast message and display it (**hint:** use the adapter to access the name of the car).

```
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
// Find the GridView by id
GridView gridview = (GridView) findViewById(R.id.gridView);
// create the adapter
final MyAdapter adapter = new MyAdapter(this);
// set the adapter for the gridview here
gridview.setOnItemClickListener(new AdapterView.OnItemClickListener() {

public void onItemClick(AdapterView<?> parent, View v, int position, long
id) {
     // make and display the toast message here
}});
```

Part 4 Creativity and Self-directed Learning

Now suppose after all our hard work, the client arrives and has a look at our app. They pull a strange face and say “No, no, no, that's not what I wanted at all, where are the logos for the cars that we discussed?”. Of course there was no discussion about logos but such is the life of a developer :(Anyway, using the supplied .png image files have a go at changing the UI to display the car logos instead (don't worry about displaying a toast for this one). If you can think of something better, feel free to have a go at that as well, however, in the real world it is important to remember to do what the client asked first.