Software Development Skills Mobile, Online course

Author: Kim Venho | 000430285

Lappeenrannan teknillinen yliopisto

Table of Contents

Learning Diary

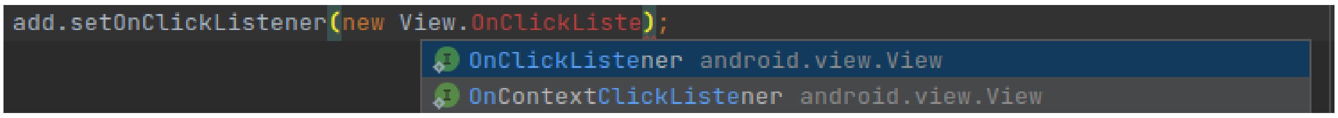
1. Introduction

10.05.2023

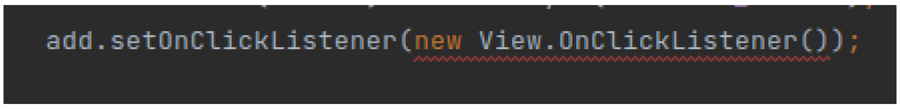
The tutorial started with installing android studio, but I happen to have it installed already since I have developed few small apps in the past. The teacher shows around the basics of android studio, like the project files view, how to add simple objects to the app and how to anchor them around. This is something I've done many times so there's not much that can go wrong. But I did learn one new thing — **sp**. It is a size unit that is recommended to be used with android studio. It stands for either scalable pixels OR scale-independent pixels (apparently documentation uses both of these terms: https://stackoverflow.com/questions/2025282/what-is-the-difference-between-px-dip-dp-and-sp). So by using this, the content that you create will look good with any user set font size.

After adding few elements on the screen, we needed to test run our application. I already had few different software emulated android versions installed on my system, but it has been a while when I last debugged my application on my actual phone — so I decided do just that. This is done by first enabling your developer settings on your phone, which is done by finding your software version from your phone settings, and tapping that 7 times. This will add developer options category to your settings menu. Next you open that and **enable USB debugging** (https://developer.android.com/studio/debug/dev-options). Depending on your operating system, you might need to install few packages so your OS can also detect your phone. After that, everything seems to work, I can launch the program we made on android studio on my phone.

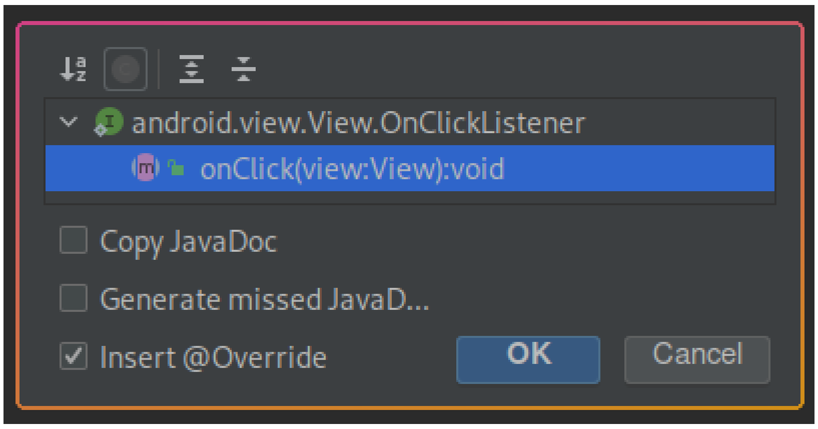
Then we jumped to writing the logic of the application in java. One thing was different on my android studio compared to the teachers — when he auto-completed the OnClickListener, the OnClick method came with it. When I did it, nothing came up:


```
add.setOnClickListener(new View.OnClickListe);
```
OnClickListener android.view.View
OnContextClickListener android.view.View

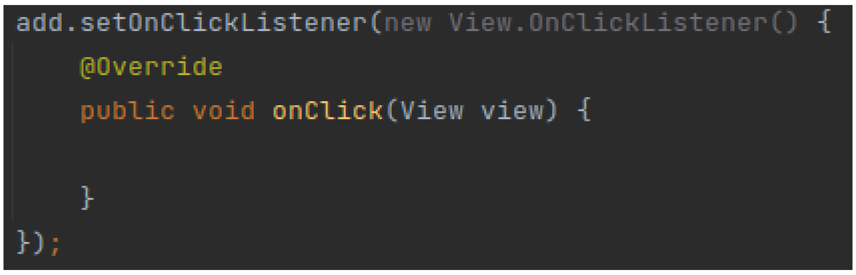
... after auto-completing:


```
add.setOnClickListener(new View.OnClickListener());
```

... no OnClick method as you can see. Now you could write it by hand, but there's another way. If you right-click that OnClickListener, select the 'Show Context Actions', and then select the 'Implement methods'. Then select the following:



... this did the same action that was automatically done on the teachers end when he auto-completed the OnClickListener. Result for reference:



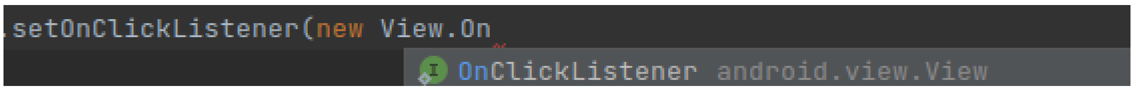
```
add.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

    }
});
```

Lastly the teacher showed the basics of debugging in basically any JetBrains IDSs. That debugger is very powerful tool, in which I have spent a lot of time .. is that a good or bad thing?

2. Core Elements

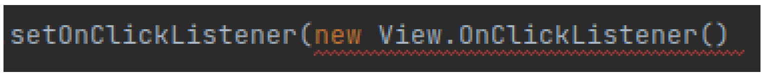
11.05.2023

Today our goal is to create an app that uses Intents and multiple Activities. We started the lecture by creating new project. As this is just simple demo of the features of android, we just put few buttons on the screen. But while coding the OnClickListener like in last part in this diary, I tried the auto-completion. But this time it worked differently and auto-inserted the OnClick method also? Why is this? Well, it depends on what part you made the auto-completion:



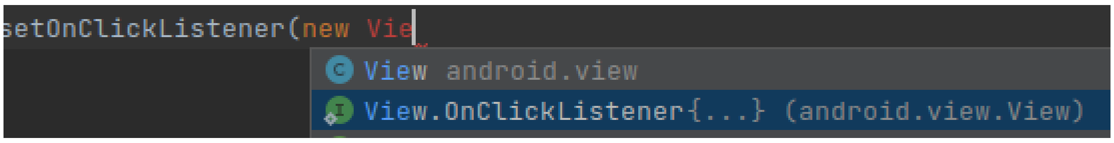
```
.setOnClickListener(new View.On
                            OnClickListener android.view.View
```

... clicking the auto-completion here will result to this:



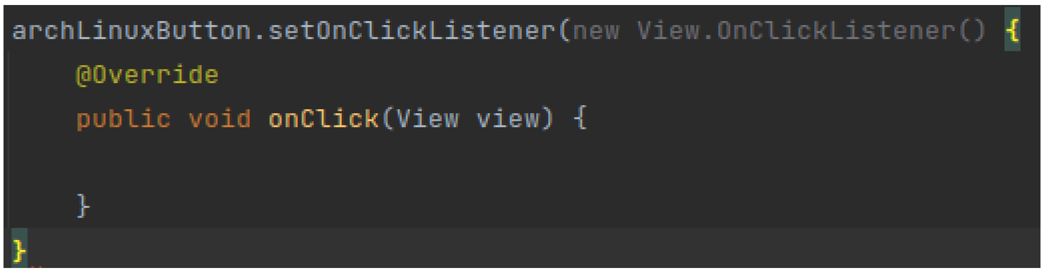
```
setOnClickListener(new View.OnClickListener()
```

... but auto-completing on here:



```
setOnClickListener(new Vie
                      View android.view
                      View.OnClickListener{...} (android.view.View)
```

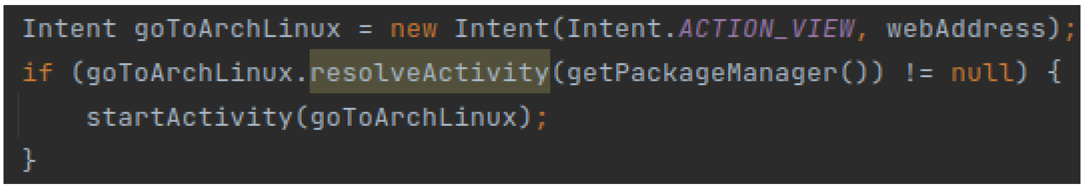
... results in this:



```
archLinuxButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

    }
}
```

... this was good thing to learn!

While writing the part where we wanted to open some website on the device, I learned how you can ask the operating system if there is applications that can do the task you want to do. Like if we want to open some website, we of course need web browser for that. It is done like this:



```
Intent goToArchLinux = new Intent(Intent.ACTION_VIEW, webAddress);
if (goToArchLinux.resolveActivity(getPackageManager()) != null) {
    startActivity(goToArchLinux);
}
```

... this will either open the site on web browser on users phone, or if they have multiple browsers, it will ask on which one to open the site.

3. Lists, Layouts, and Images

12.05.2023

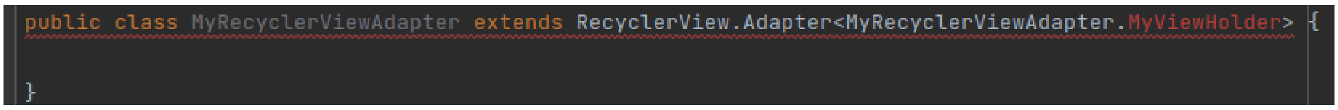
As per usual, we started by creating a new project. Simple project again, and we are just putting few components to our app. One of which is ListView — but I did not have it. Well it appears that the tutorial is 5 years old and there is new way to create the list like result that we are aiming for — it's called RecyclerView (https://developer.android.com/reference/androidx/recyclerview/widget/RecyclerView.html). I will be using that (since it gives the same results, and it's the new way to do it), even the official ListView documentations says the following:

For a more modern, flexible, and performant approach to displaying lists, use `RecyclerView`.

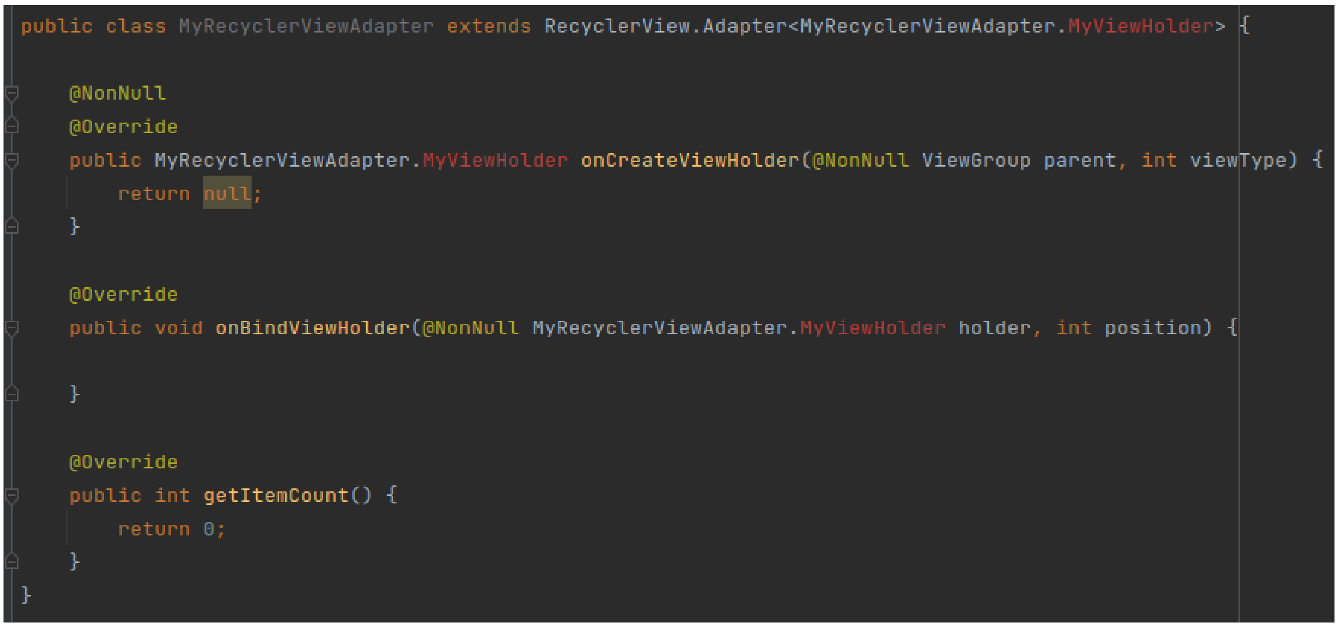
Next we did the layout file. This was very useful thing to learn. It allows you to basically use or reuse layouts where you can fill the layout elements contents afterward from some data sets!

For the next few part I was able to complete thanks to the developer documents (https://developer.android.com/reference/androidx/recyclerview/widget/RecyclerView.html) and one youtube tutorial (https://www.youtube.com/watch?v=Mc0XT58A1Z4).

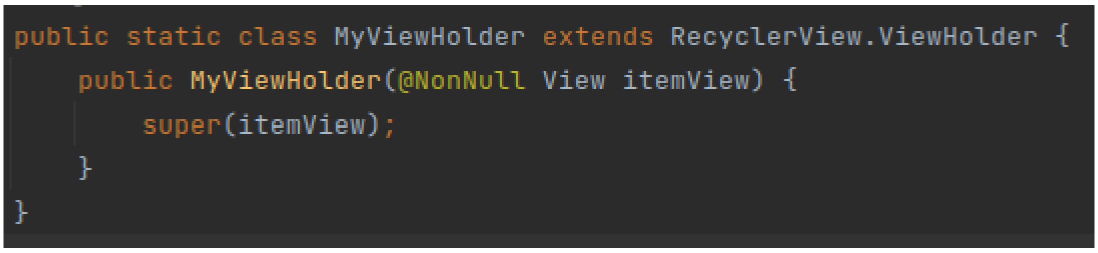
Next the recycler view needed an adaptor so it can show our data. I started by creating new adapter class for it, and set it to extend the adapter we are looking for:


```java
public class MyRecyclerViewAdapter extends RecyclerView.Adapter<MyRecyclerViewAdapter.MyViewHolder> {

}
```

... then I allowed android studio to auto-complete the missing functions:


```java
public class MyRecyclerViewAdapter extends RecyclerView.Adapter<MyRecyclerViewAdapter.MyViewHolder> {

    @NonNull
    @Override
    public MyRecyclerViewAdapter.MyViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        return null;
    }

    @Override
    public void onBindViewHolder(@NonNull MyRecyclerViewAdapter.MyViewHolder holder, int position) {

    }

    @Override
    public int getItemCount() {
        return 0;
    }
}
```

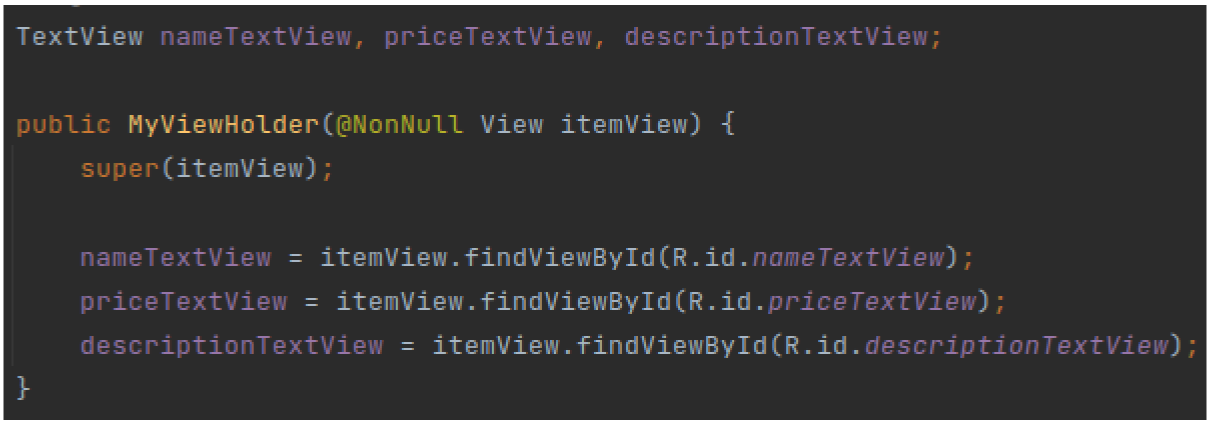
I then created the MyViewHolder that is missing (red text) in the last image, inside the adapter:



```java
public static class MyViewHolder extends RecyclerView.ViewHolder {
    public MyViewHolder(@NonNull View itemView) {
        super(itemView);
    }
}
```

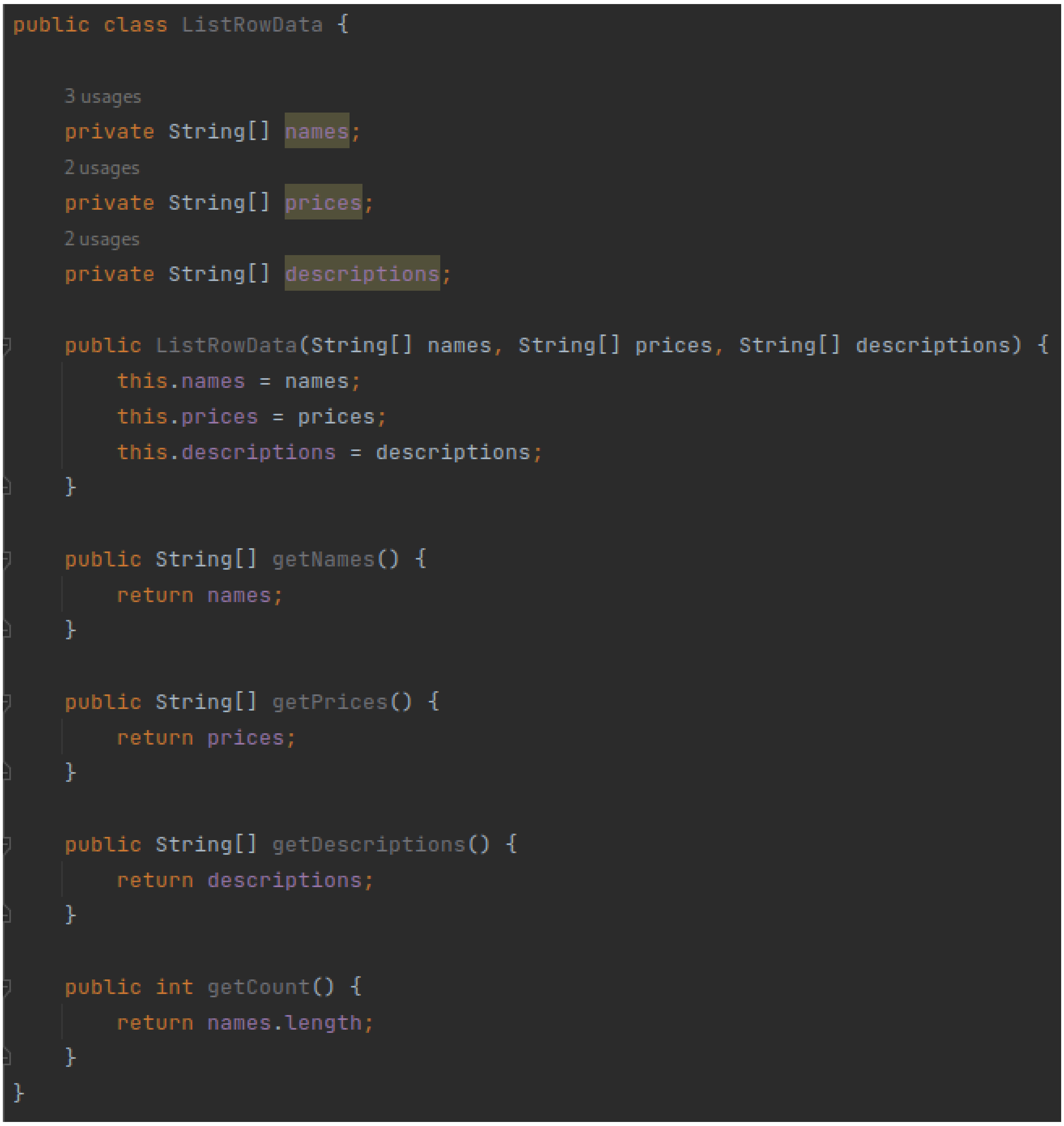
... and that is the base of our adapter done.

Now we need to assign the views inside the layout file we created to variables, so we can actually use them:



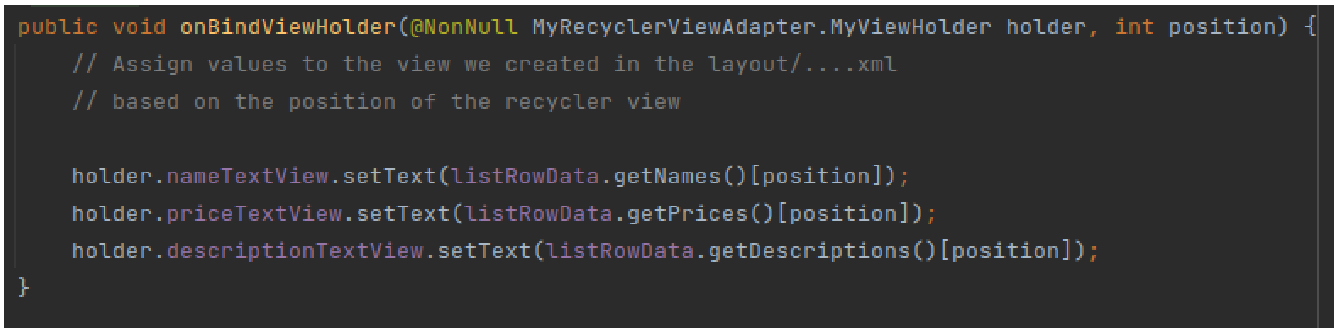
```java
TextView nameTextView, priceTextView, descriptionTextView;

public MyViewHolder(@NonNull View itemView) {
    super(itemView);

    nameTextView = itemView.findViewById(R.id.nameTextView);
    priceTextView = itemView.findViewById(R.id.priceTextView);
    descriptionTextView = itemView.findViewById(R.id.descriptionTextView);
}
```

Next I created class for holding data for each of our rows for easier data access and handling:



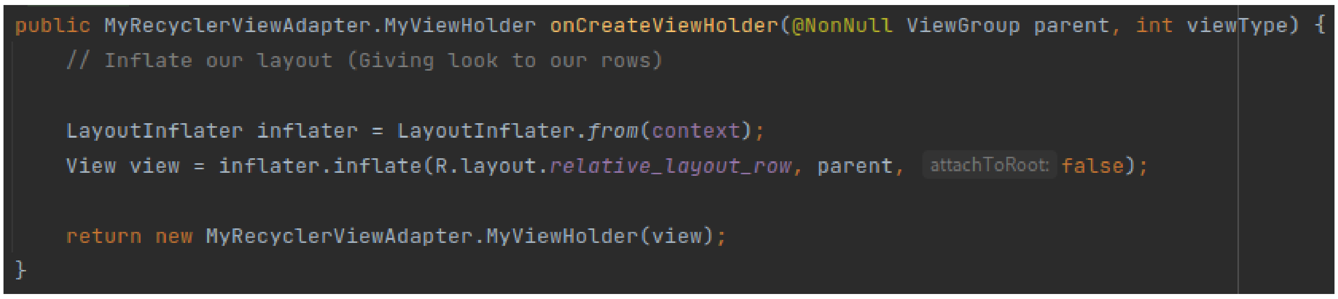
```
public class ListRowData {

    3 usages
    private String[] names;
    2 usages
    private String[] prices;
    2 usages
    private String[] descriptions;

    public ListRowData(String[] names, String[] prices, String[] descriptions) {
        this.names = names;
        this.prices = prices;
        this.descriptions = descriptions;
    }

    public String[] getNames() {
        return names;
    }

    public String[] getPrices() {
        return prices;
    }

    public String[] getDescriptions() {
        return descriptions;
    }

    public int getCount() {
        return names.length;
    }
}
```

Then I set up the binding process, which is the part where our text lists will be assigned to the on screen elements:



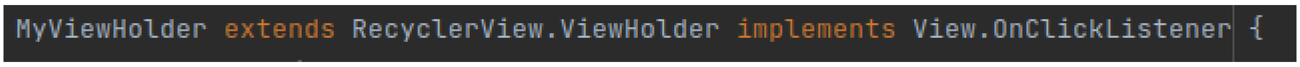
```
public void onBindViewHolder(@NonNull MyRecyclerViewAdapter.MyViewHolder holder, int position) {
    // Assign values to the view we created in the layout/....xml
    // based on the position of the recycler view

    holder.nameTextView.setText(listRowData.getNames()[position]);
    holder.priceTextView.setText(listRowData.getPrices()[position]);
    holder.descriptionTextView.setText(listRowData.getDescriptions()[position]);
}
```

Finally the inflation part that gives look to our rows:



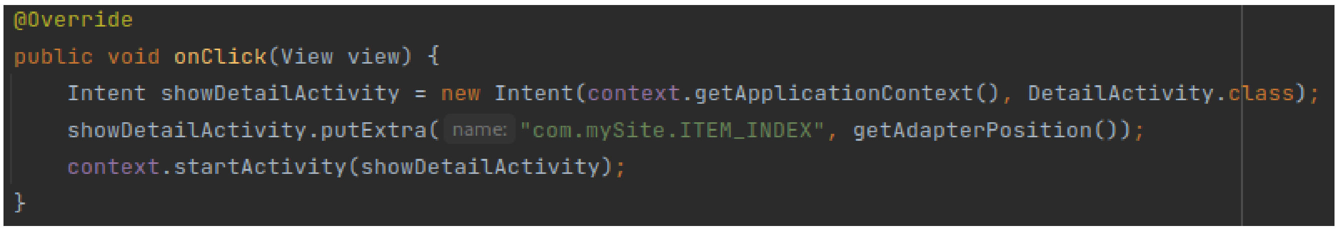
```java
public MyRecyclerViewAdapter.MyViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    // Inflate our layout (Giving look to our rows)

    LayoutInflater inflater = LayoutInflater.from(context);
    View view = inflater.inflate(R.layout.relative_layout_row, parent,  attachToRoot: false);

    return new MyRecyclerViewAdapter.MyViewHolder(view);
}
```

After all this work, everything was working! .. visually. Now we need to implement click detection. Since I'm using RecycledView instead of the ListView, I need to figure out myself how to do the detection. And by me I of course mean the results of some google searches. There were few stackoverflow threads about this and people had different opinions on what would be the best way to implement this, but I ended up using the accepted answer on this thread https://stackoverflow.com/questions/32779956/onclicklistener-for-recyclerview. So in other words, you need to implement View.OnClickListener like this:



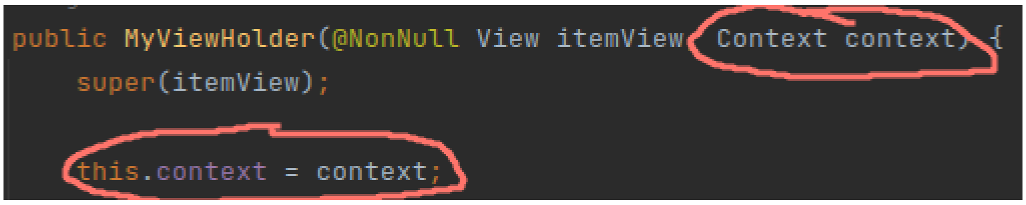
```java
MyViewHolder extends RecyclerView.ViewHolder implements View.OnClickListener {
```

... then implement the actual OnClick like so:



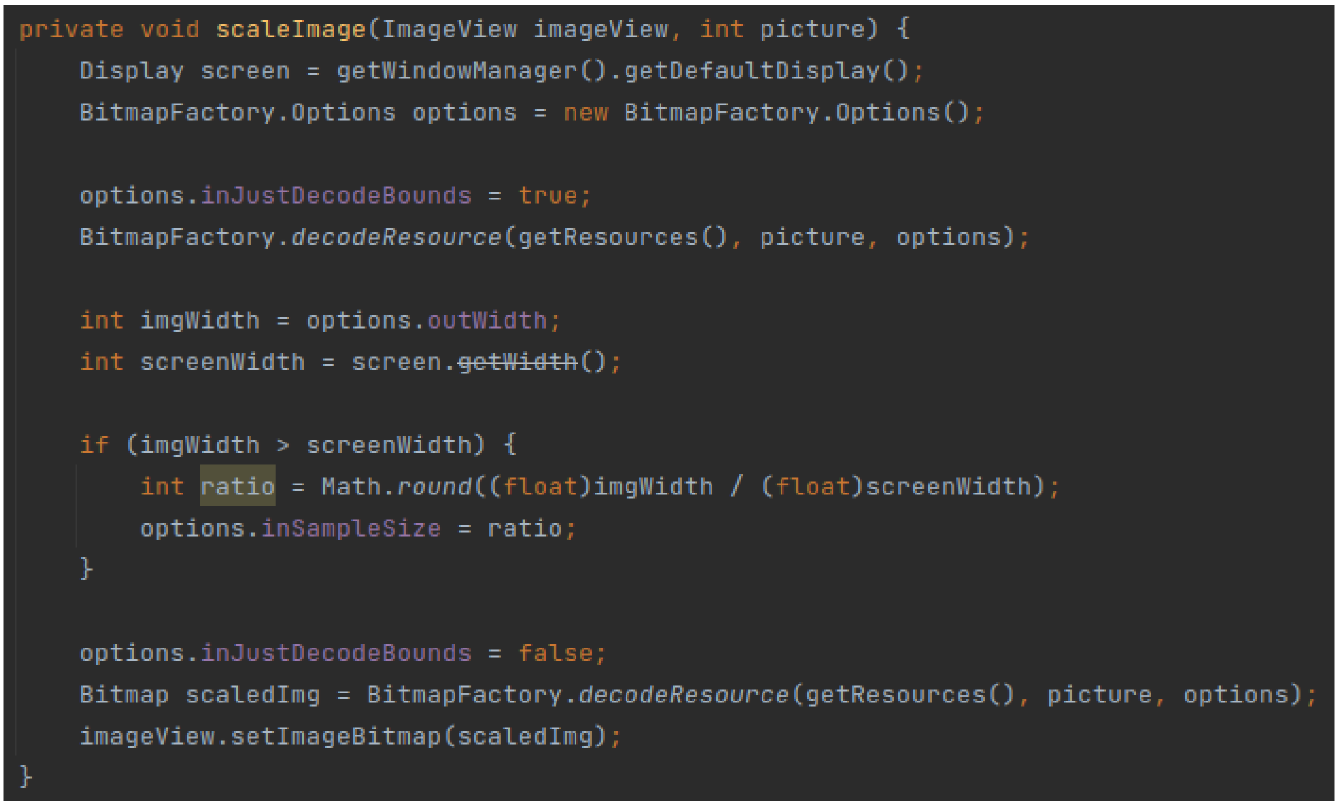
```java
@Override
public void onClick(View view) {
    Intent showDetailActivity = new Intent(context.getApplicationContext(), DetailActivity.class);
    showDetailActivity.putExtra( name: "com.mySite.ITEM_INDEX", getAdapterPosition());
    context.startActivity(showDetailActivity);
}
```

... and since we could not start activities from there without context, I'm passing the Context on MyViewHolder creation:



```java
public MyViewHolder(@NonNull View itemView, Context context) {
    super(itemView);

    this.context = context;
```

Now that we have the click detection working, the last thing we worked on was displaying images on new Activity. The Activity part was very easy since we have already done multiple of those, but the image scaling was interestingly simple:



```java
private void scaleImage(ImageView imageView, int picture) {
    Display screen = getWindowManager().getDefaultDisplay();
    BitmapFactory.Options options = new BitmapFactory.Options();

    options.inJustDecodeBounds = true;
    BitmapFactory.decodeResource(getResources(), picture, options);

    int imgWidth = options.outWidth;
    int screenWidth = screen.~~getWidth~~();

    if (imgWidth > screenWidth) {
        int ratio = Math.round((float)imgWidth / (float)screenWidth);
        options.inSampleSize = ratio;
    }

    options.inJustDecodeBounds = false;
    Bitmap scaledImg = BitmapFactory.decodeResource(getResources(), picture, options);
    imageView.setImageBitmap(scaledImg);
}
```

... basically the android itself can handle the scaling, we just tell it how to do it.