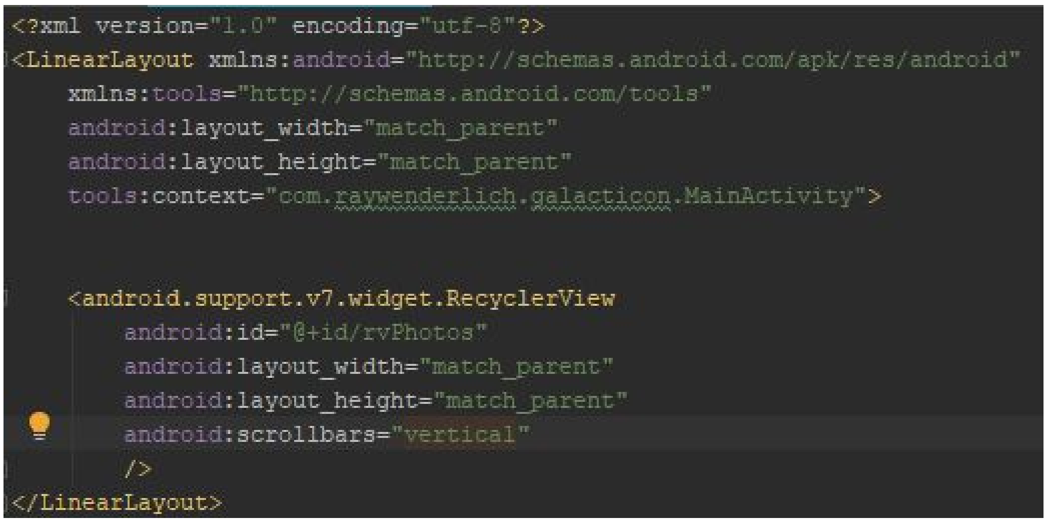
- It’s important to know that we can use different LayoutManager in the RecyclerView to set how we want the view of the items, they are
    a. **LinearLayoutManager** positions items to look like a standard ListView
    b. **GridLayoutManager** positions items in a grid format similar to a GridView
    c. **StaggeredGridLayoutManager** positions tems in a staggered grid format.
    d. Create your own LayoutManager
- So if we want to implement a RecyclerView in our project we need to create 4 different codes that together make work the RecyclerView, they are
    a. Declare the RecyclerView in an activity layout and reference it in the activity Kotlin file.
    b. Create a custom item XML layout for RecyclerView for its items.
    c. Create the view holder for view items, connect the data source of the RecyclerView and handle the view logic by creating a RecyclerView Adapter.
    d. Attach the adapter to the RecyclerView.

Starting with the steps

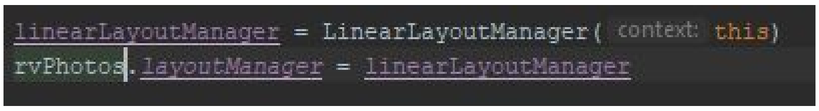
1. First We need to declare the recyclerView in the activity/fragment where we want to show it
    a. Open the xml of you activity/fragment. And move inside a new child RecyclerView (Remember RecyclerView will need to import some library) usually you’ll finish with something like this



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.raywenderlich.galacticon.MainActivity">


    <android.support.v7.widget.RecyclerView
        android:id="@+id/rvPhotos"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scrollbars="vertical"
        />
</LinearLayout>
```

    b. Every RecyclerView need a Layout Manager, in this case we’ll use a LinearLayout Manager, we need to declare it in the Activity, and then call this RecyclerView and set the layout manager, so in the top the activity we’ll add the code : `private lateinit var linearLayoutManager: LinearLayoutManager`

c. To set this LayaoutManager with the RecyclerView we need to use the follow code inside the **onCrete**()

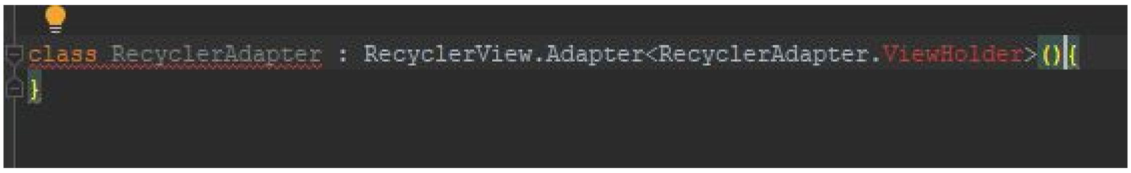


```
linearLayoutManager = LinearLayoutManager( context: this)
rvPhotos.layoutManager = linearLayoutManager
```

We can do it this way thanks to kotlin because now use `kotlinx.android.synthetic.main.activity_main.*` and with this we can directly call the views by the ID

2. Now we need to Laying Out RecyclerView Items so now we are going to create a custom layout for the item you want the RecyclerView to use.
   a. We need to create a new XML in our Res folder, we can set a name like `recyclerview_item_row` then we'll set all the views that we need to show in each item in the recyclerView
   b. Not next steps .. this is pretty much the second step, all depend how many views you want to put inside

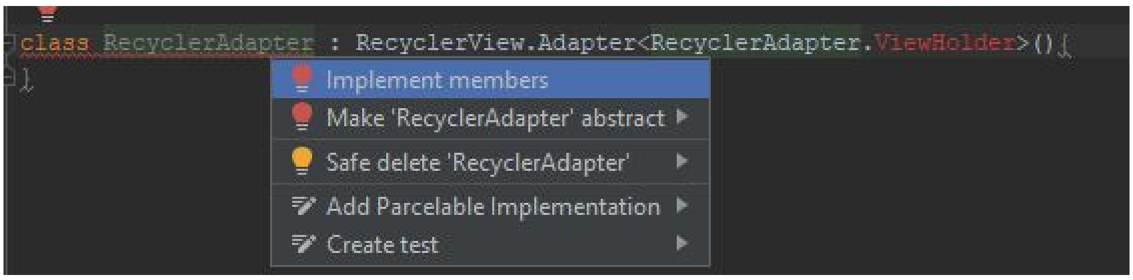
3. Now we need to create our **Adapter** *pum pum puuuum*…
   a. For this we going to our kt code and we'll Right-click there, select New ‣ Kotlin File ‣ Class, name it **RecyclerAdapter** and select Class for Kind.
   b. Now we need to make the class to extend from adapter, in kotling we do it like in the next code



```
class RecyclerAdapter : RecyclerView.Adapter<RecyclerAdapter.ViewHolder>(){
}
```

Don't worry for the red line and word, I'll explain it later

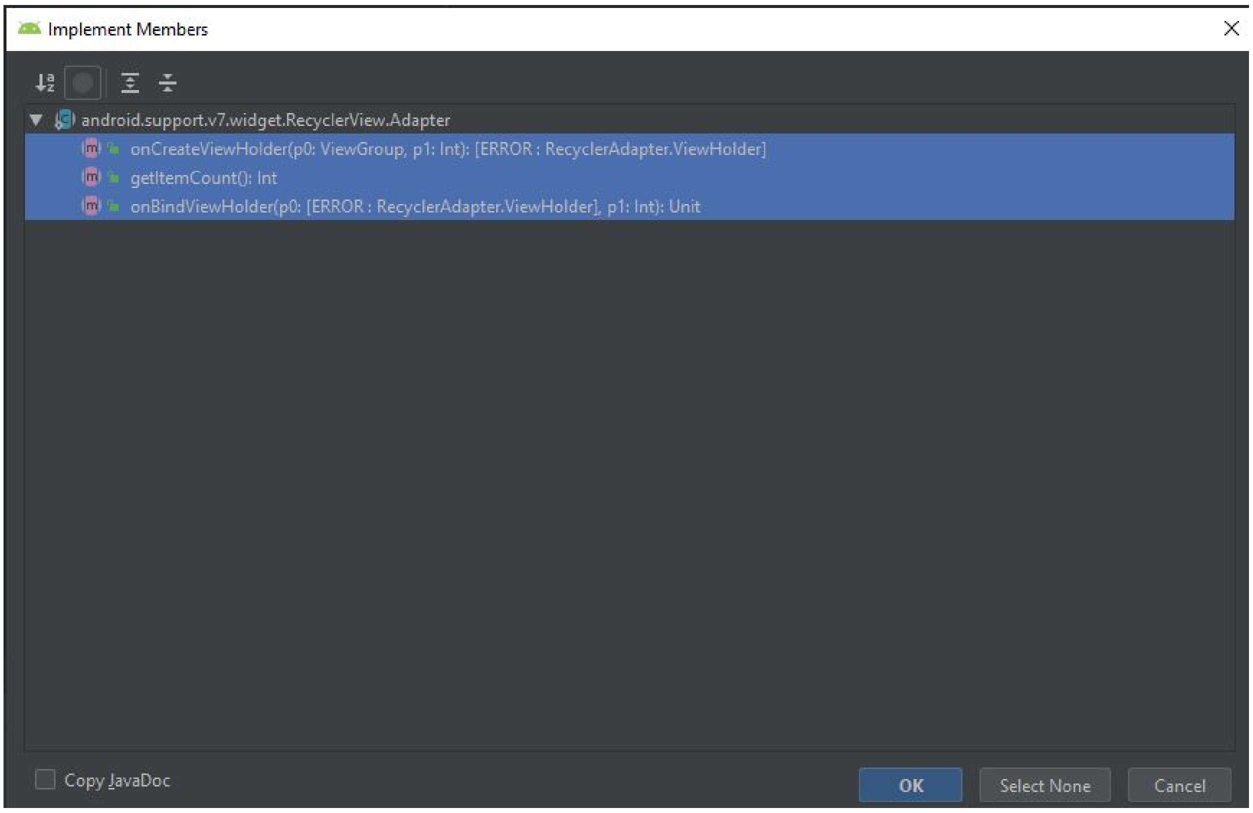
4. The red line is because as Adapter we need to implement three Functions (onCreateViewHolder,getItemCOunt,onBindViewHolder) so we can set the mouse on the line and use alt+enter, select the option implement members



```
class RecyclerAdapter : RecyclerView.Adapter<RecyclerAdapter.ViewHolder>(){
}
```

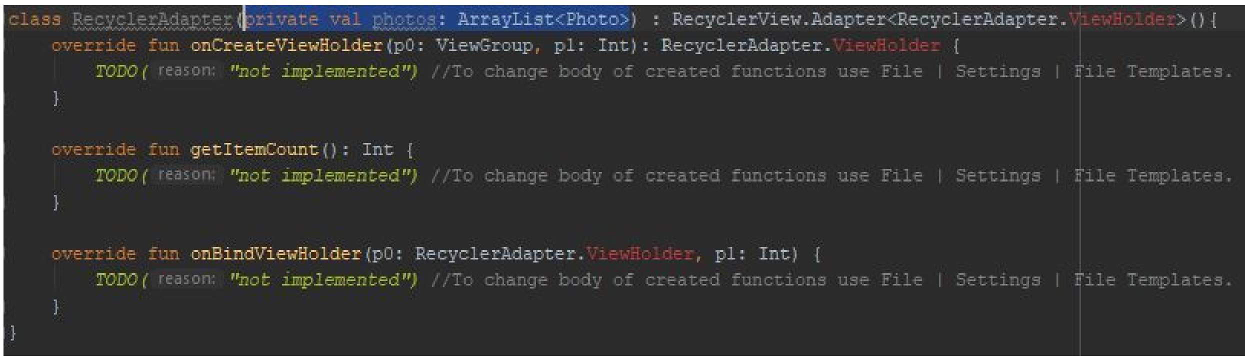
Implement members
Make 'RecyclerAdapter' abstract
Safe delete 'RecyclerAdapter'
Add Parcelable Implementation
Create test

Select all the three options and ok



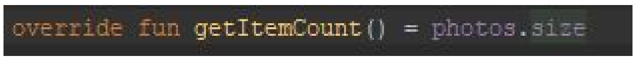
```
android.support.v7.widget.RecyclerView.Adapter
    onCreateViewHolder(p0: ViewGroup, p1: Int): [ERROR : RecyclerAdapter.ViewHolder]
    getItemCount(): Int
    onBindViewHolder(p0: [ERROR : RecyclerAdapter.ViewHolder], p1: Int): Unit
```

a. Now we need to add the parameter of the list that we'll need to show in the RecyclerView, so we'll have something like this



```
class RecyclerAdapter(private val photos: ArrayList<Photo>) : RecyclerView.Adapter<RecyclerAdapter.ViewHolder>() {
    override fun onCreateViewHolder(p0: ViewGroup, p1: Int): RecyclerAdapter.ViewHolder {
        TODO( reason: "not implemented") //To change body of created functions use File | Settings | File Templates.
    }

    override fun getItemCount(): Int {
        TODO( reason: "not implemented") //To change body of created functions use File | Settings | File Templates.
    }

    override fun onBindViewHolder(p0: RecyclerAdapter.ViewHolder, p1: Int) {
        TODO( reason: "not implemented") //To change body of created functions use File | Settings | File Templates.
    }
}
```

b. So now we going to make the right code for getItemCount, this one is for show how many items we have in the array, so the code will be

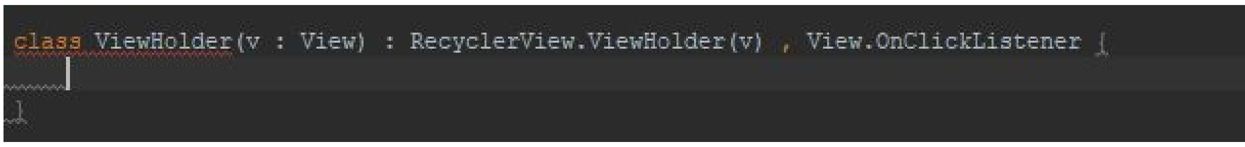


```
override fun getItemCount() = photos.size
```

for the other two we'll need the ViewHolder and that it's the next big step

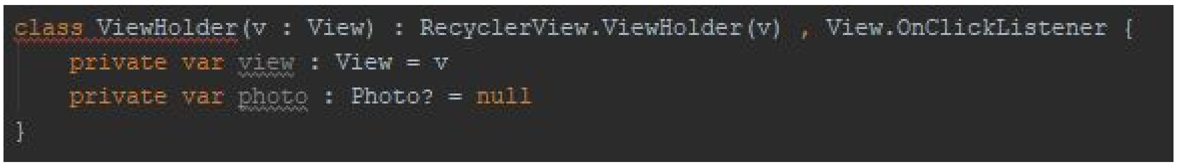
5. To create a ViewHolder for your views reference we'll create a nested class in your adapter, but it will be a different class to the adapter
    a. We're going to create a new class inside the adapter with the name ViewHolder, this one will need a view as parameter, and extend ViewHolder from RecyclerView and send inside the parameter view, we also need to implement

the onClick because we'll need it



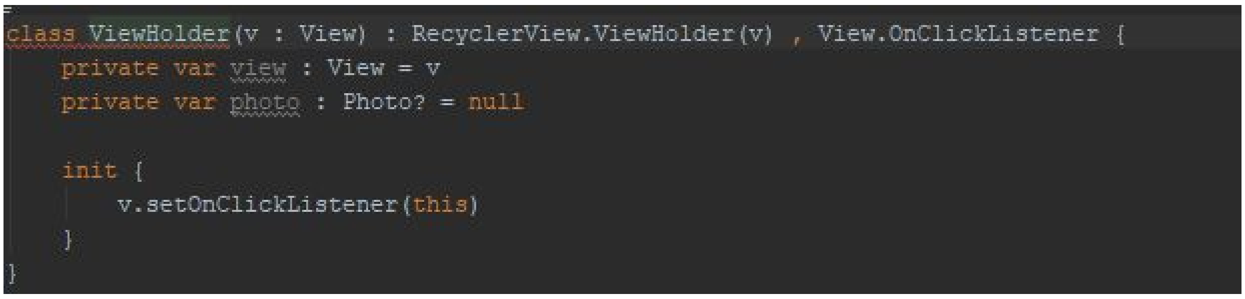
```
class ViewHolder(v : View) : RecyclerView.ViewHolder(v) , View.OnClickListener {

}
```

b. Now we're going to add a reference to the view you've inflated to allow the ViewHolder to access the ImageView and TextView as an extension property. Kotlin Android Extensions plugin adds hidden caching functions and fields to prevent the constant querying of views.



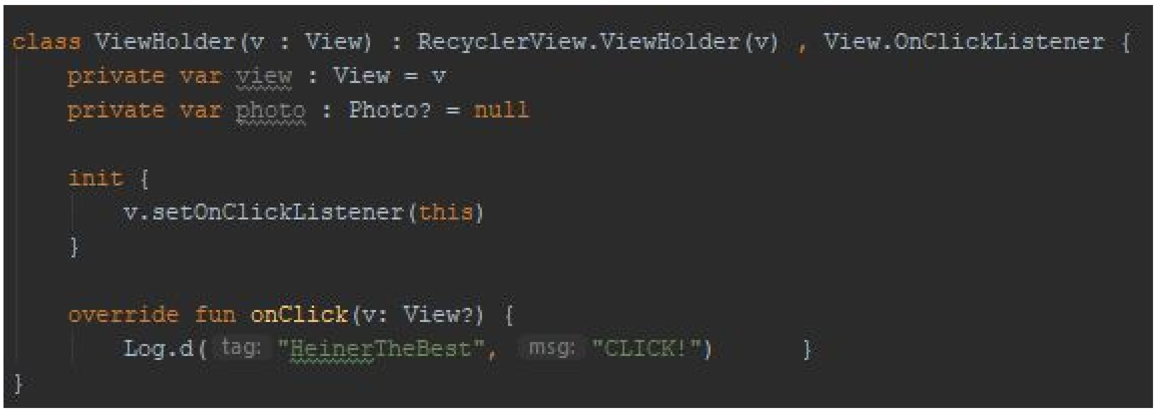
```
class ViewHolder(v : View) : RecyclerView.ViewHolder(v) , View.OnClickListener {
    private var view : View = v
    private var photo : Photo? = null
}
```

c. We need to add the init to initialize the View.OnClickListener.



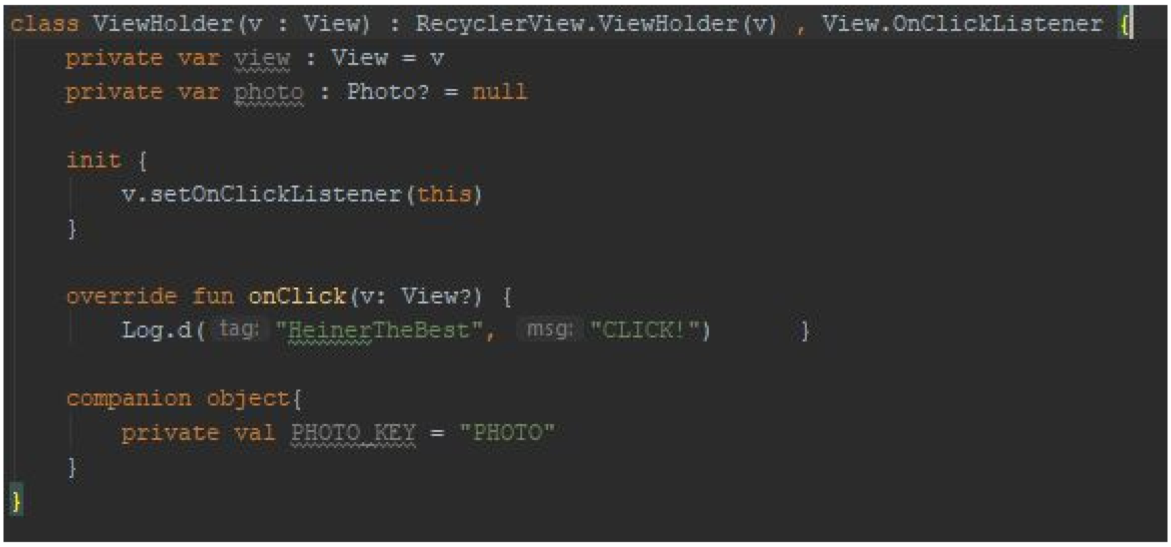
```
class ViewHolder(v : View) : RecyclerView.ViewHolder(v) , View.OnClickListener {
    private var view : View = v
    private var photo : Photo? = null

    init {
        v.setOnClickListener(this)
    }
}
```

d. Now we need to implement the function onClick with this we'll stop to have the red line



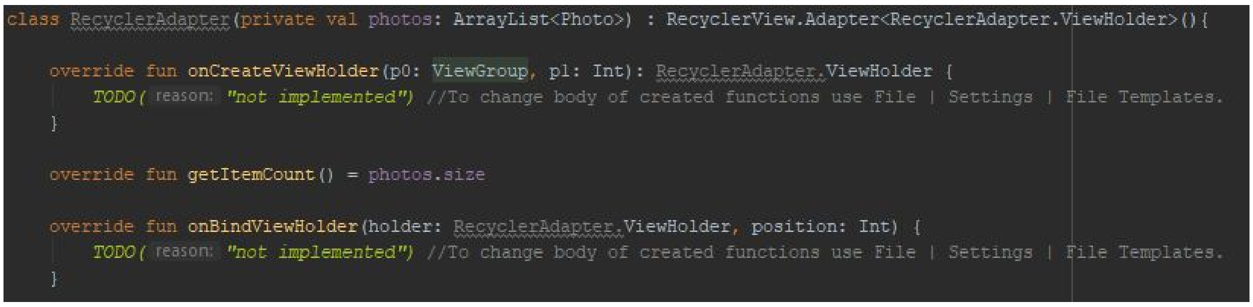
```
class ViewHolder(v : View) : RecyclerView.ViewHolder(v) , View.OnClickListener {
    private var view : View = v
    private var photo : Photo? = null

    init {
        v.setOnClickListener(this)
    }

    override fun onClick(v: View?) {
        Log.d( tag: "HeinerTheBest",  msg: "CLICK!")      }
}
```

e. Now we need to create a companion object to add a key for easy reference to the item launching the RecyclerView.



```
class ViewHolder(v : View) : RecyclerView.ViewHolder(v) , View.OnClickListener {
    private var view : View = v
    private var photo : Photo? = null

    init {
        v.setOnClickListener(this)
    }

    override fun onClick(v: View?) {
        Log.d( tag: "HeinerTheBest",  msg: "CLICK!")       }

    companion object{
        private val PHOTO_KEY = "PHOTO"
    }
}
```

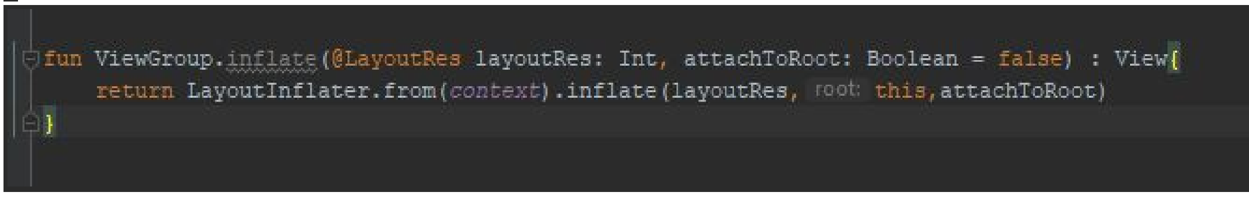
6. Now we have pretty much all the piece but before connect everything going to give more sense to the code, and change the parameters in the override of the Adapter
   a. In `onCreateViewHolder`:We will change **p0** to **parent** and **p1** to **viewType**
   b. In `onBindViewHolder`: We will change **p0** to **holder** and **p1** to **position**



```
class RecyclerAdapter(private val photos: ArrayList<Photo>) : RecyclerView.Adapter<RecyclerAdapter.ViewHolder>(){

    override fun onCreateViewHolder(p0: ViewGroup, p1: Int): RecyclerAdapter.ViewHolder {
        TODO( reason: "not implemented") //To change body of created functions use File | Settings | File Templates.
    }

    override fun getItemCount() = photos.size

    override fun onBindViewHolder(holder: RecyclerAdapter.ViewHolder, position: Int) {
        TODO( reason: "not implemented") //To change body of created functions use File | Settings | File Templates.
    }
```

7. Now we're going to put the piece together o/ to show how amazing can we kotlin we'll use a extension funtion with the onCreatViewHolder for that we'll be doing the next steps
   a. Create a new Kt File named Extensions, and write the follow code inside:

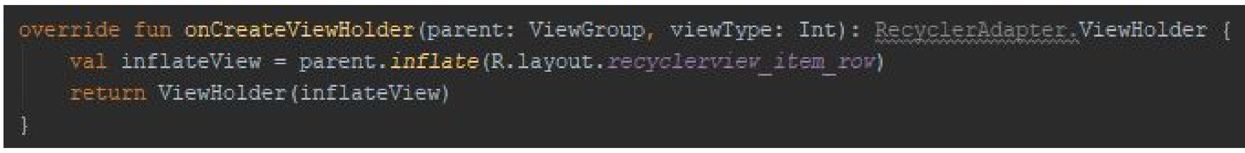
```
fun ViewGroup.inflate(@LayoutRes layoutRes: Int, attachToRoot: Boolean =
false) : View{
    return
LayoutInflater.from(context).inflate(layoutRes,this,attachToRoot)
}
```



```
fun ViewGroup.inflate(@LayoutRes layoutRes: Int, attachToRoot: Boolean = false) : View{
    return LayoutInflater.from(context).inflate(layoutRes, root: this,attachToRoot)
}
```

   b. And now from the onCreateViewHolder we just can call this extension function from the viewHolder and just sent the layout we will need, and return a new

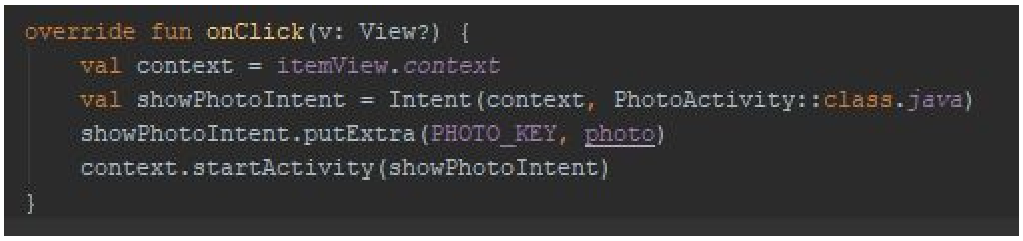
ViewHolder with this view



```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RecyclerAdapter.ViewHolder {
    val inflateView = parent.inflate(R.layout.recyclerview_item_row)
    return ViewHolder(inflateView)
}
```

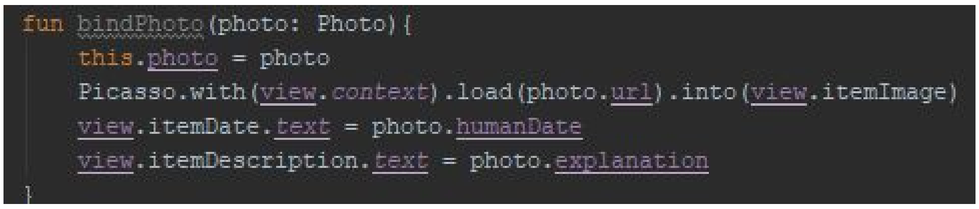
For thinks like this Kotlin is so amazing

c. Now inside the ViewHolder we can add something to do like use the context and call a second activity, when the item is clicked



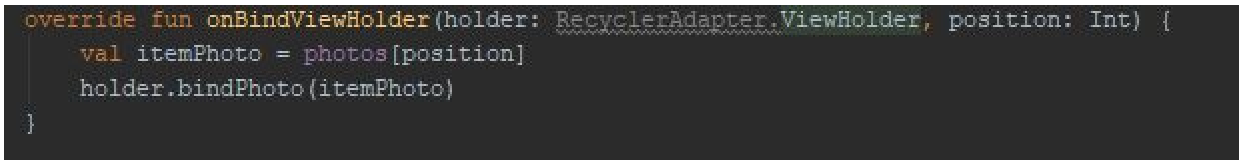
```
override fun onClick(v: View?) {
    val context = itemView.context
    val showPhotoIntent = Intent(context, PhotoActivity::class.java)
    showPhotoIntent.putExtra(PHOTO_KEY, photo)
    context.startActivity(showPhotoIntent)
}
```

d. Now we need to add also inside the ViewHolder how we want to bind the data example



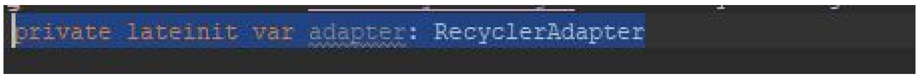
```
fun bindPhoto(photo: Photo){
    this.photo = photo
    Picasso.with(view.context).load(photo.url).into(view.itemImage)
    view.itemDate.text = photo.humanDate
    view.itemDescription.text = photo.explanation
}
```

e. And the last step is to finish the last override onBindViewHolder, here we just going to get the photo with the position from the array list, and then just call the bindFuntion from the ViewHolder and we'll going to have this



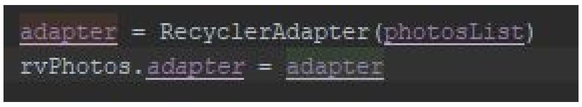
```
override fun onBindViewHolder(holder: RecyclerAdapter.ViewHolder, position: Int) {
    val itemPhoto = photos[position]
    holder.bindPhoto(itemPhoto)
}
```

8. **LAST STEP** finally now we just need to hooking up the Adapter and RecyclerView
   a. We need to create a lateinit adapter in the Activity



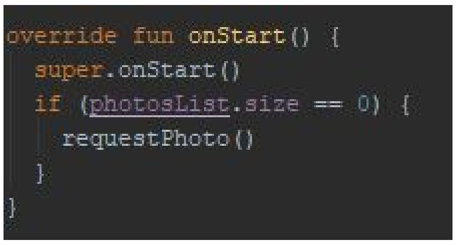
```
private lateinit var adapter: RecyclerAdapter
```

   b. When you have your list, you will set the adapter with that list and then set the RecyclerView with that adapter



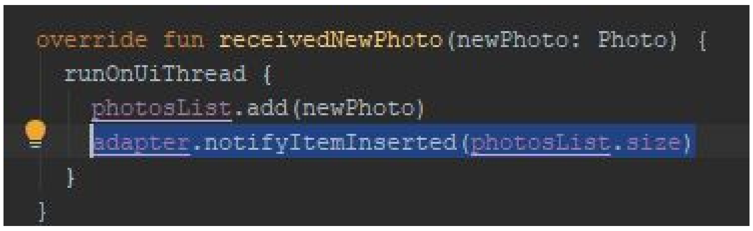
```
adapter = RecyclerAdapter(photosList)
rvPhotos.adapter = adapter
```

c. Remember always make sure to have a list, so in onStart you can set a validation



```kotlin
override fun onStart() {
  super.onStart()
  if (photosList.size == 0) {
    requestPhoto()
  }
}
```

d. If you have a function for new items, you need to add inside a notify to the adapter



```kotlin
override fun receivedNewPhoto(newPhoto: Photo) {
  runOnUiThread {
    photosList.add(newPhoto)
    adapter.notifyItemInserted(photosList.size)
  }
}
```

e.