Syntax Institut

Modul 3 – Android App Entwicklung mit Kotlin

RecyclerView I

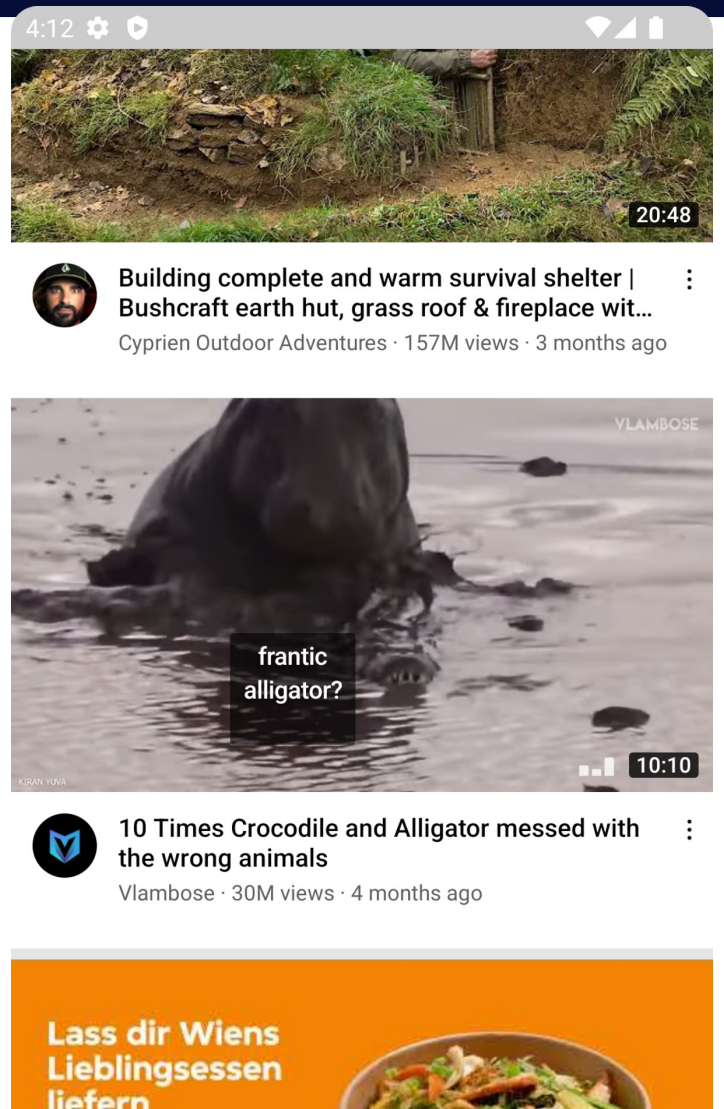
Gliederung

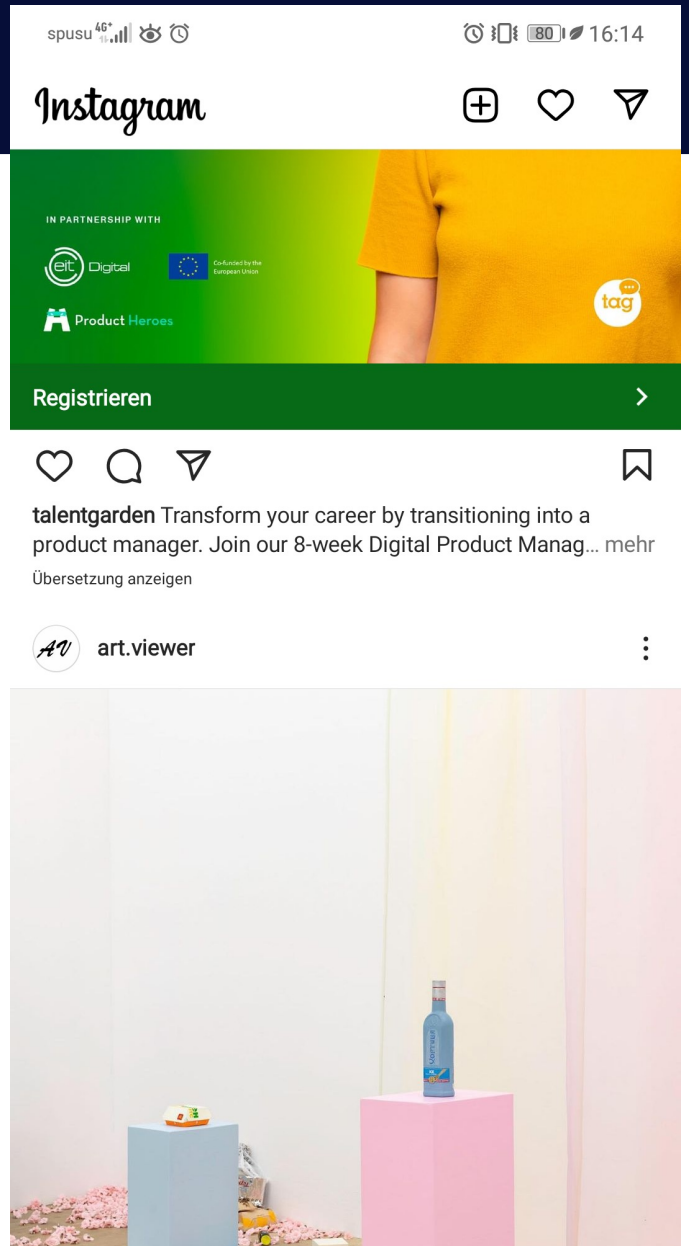
- Was ist eine RecyclerView?
- Was brauchen wir dafür?
- Wie wird sie programmiert?

Quelle: https://www.pinterest.com/pin/492581277965142379/

Was ist RecyclerView?

- Darstellung von Listen innerhalb der App
- Scrollt auch flüssig bei langen Listen mit komplexen Einträgen
- berechnet nur Einträge die gerade wirklich gezeigt werden

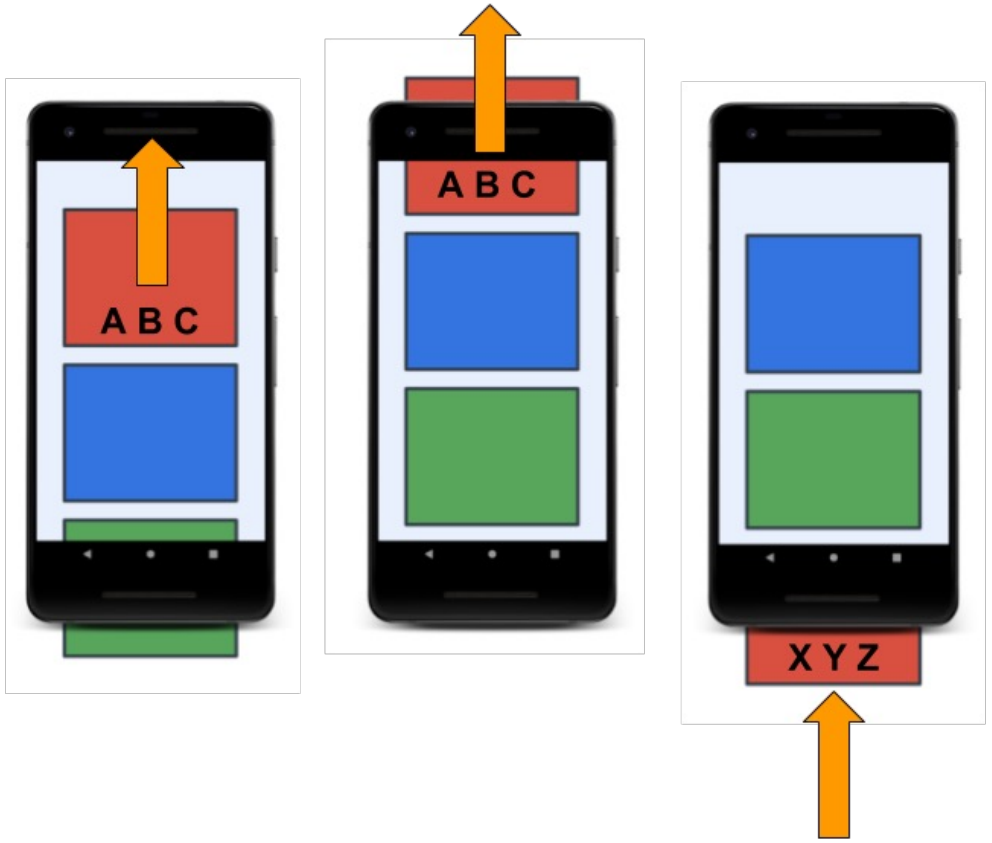




Was ist RecyclerView?

Wenn das rote Quadrat den Screen verlässt wird die View für das nächste Quadrat wiederverwendet

Nur der Text wird abgeändert



Quelle: https://developer.android.com/codelabs/basic-android-kotlin-training-recyclerview-scrollable-list/img/dcf4599789b9c2a1.png

Elemente der RecyclerView

- Items

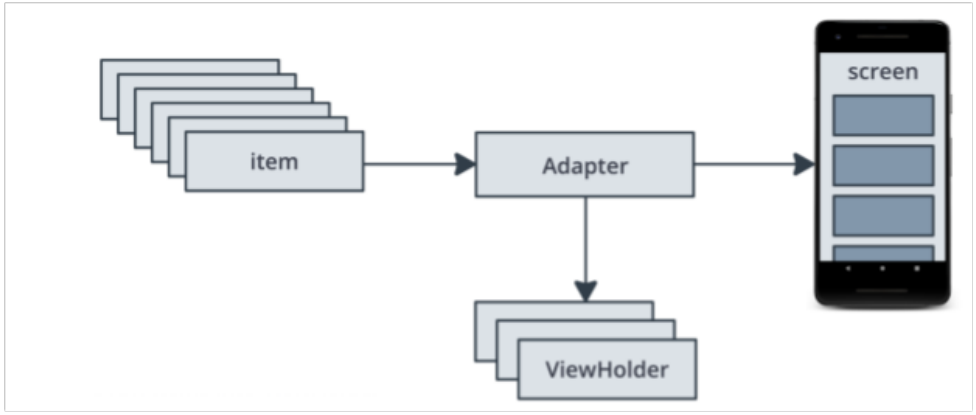
  Gegenstände einer Liste die Dargestellt werden sollen

- Adapter

  bereitet items für deren Verwendung innerhalb der RecyclerView vor

- ViewHolder

  hält items innerhalb der RecyclerView und kann immer neu befüllt werden



Quelle: https://developer.android.com/codelabs/basic-android-kotlin-training-recyclerview-scrollable-list/img/4e9c18b463f00bf7.png

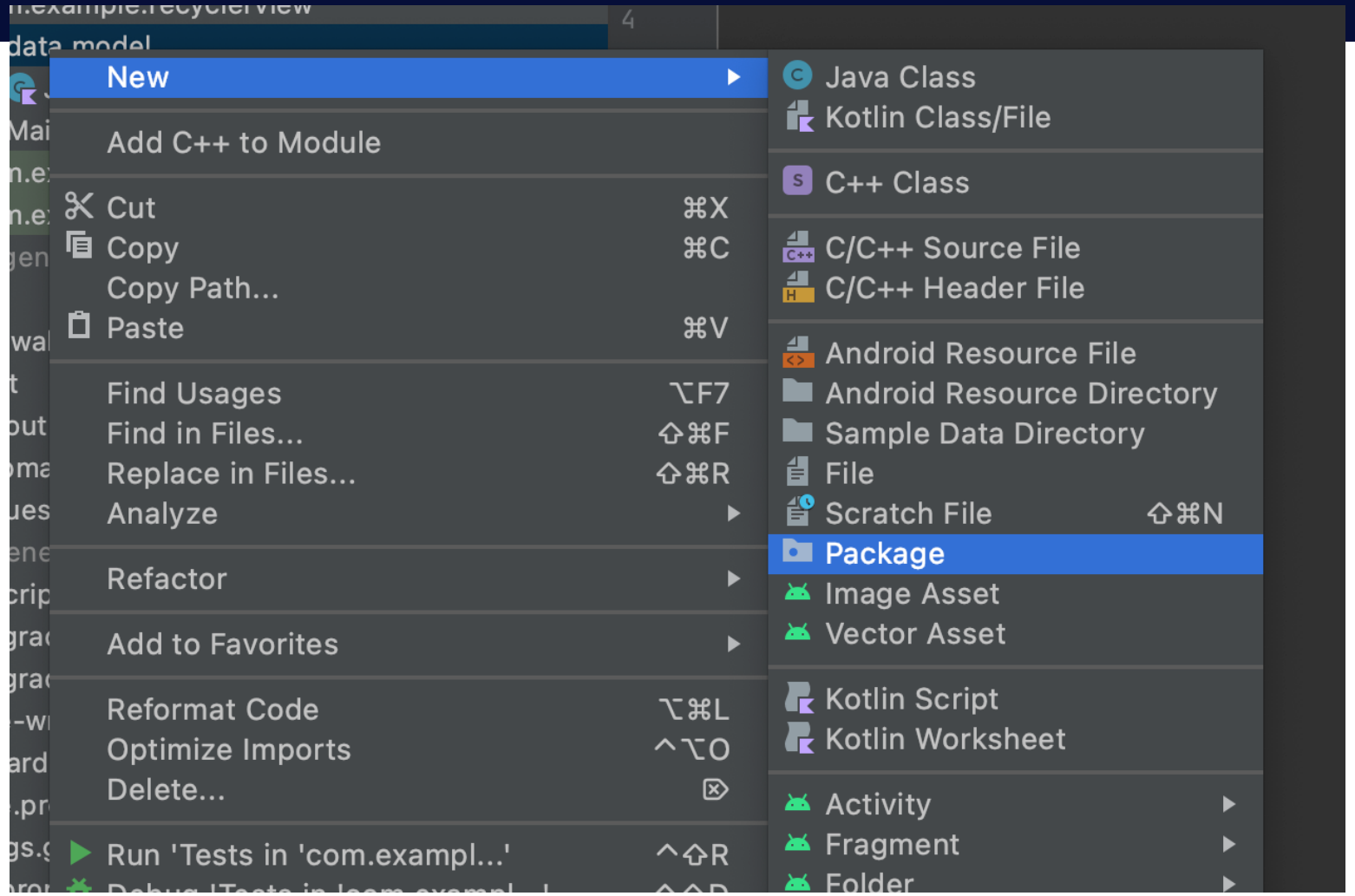
Bespiel: Anti-Witz Liste

- Packages um Code zu organisieren
- Liste an Witzen
- Layout mit RecyclerView
- Layout für die Darstellung eines Witzes
- RecyclerView-Adapter samt ViewHolder

Package

wird mittels Rechtsklick erstellt

Und Dient zur Organisation des Codes

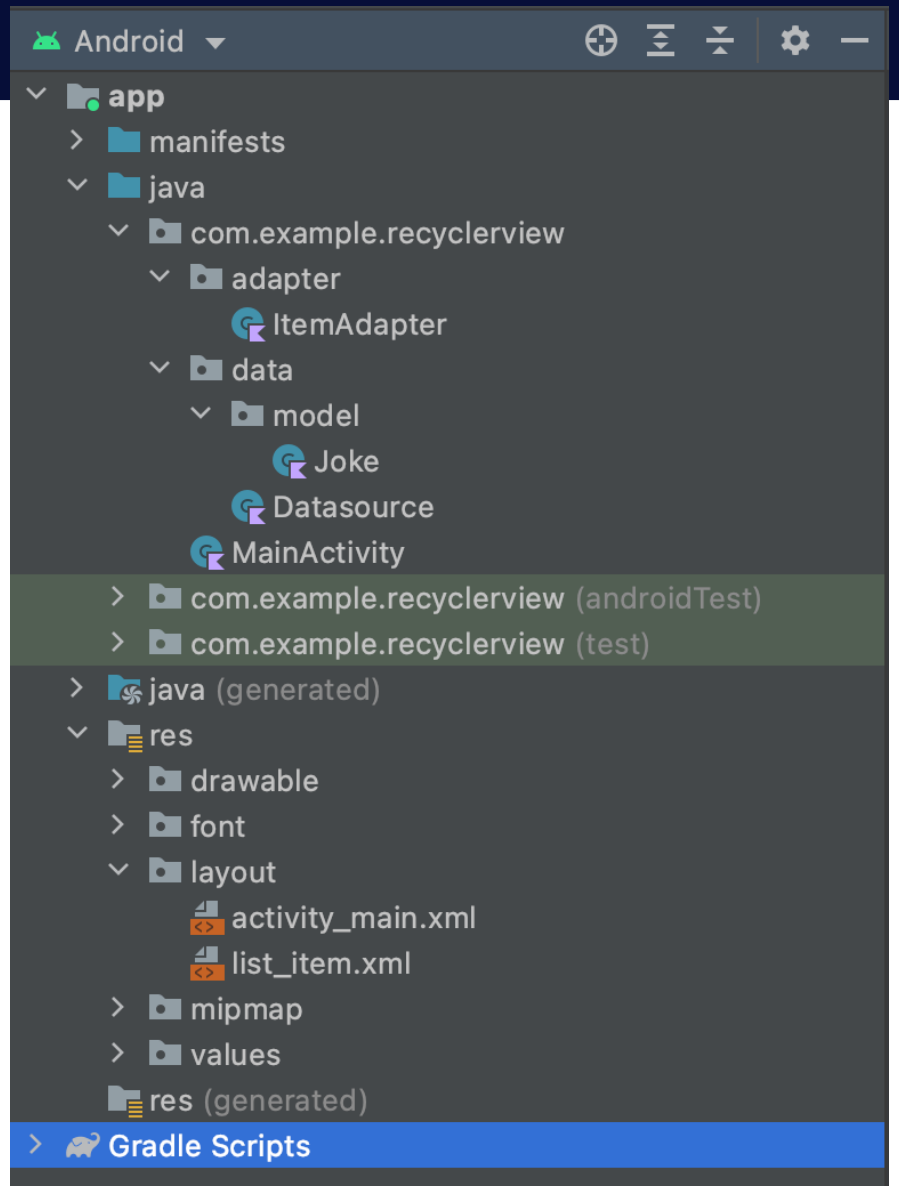


Packages

in diesem Fall wurde

- ein adapter package für den ItemAdapter
- ein data package für die Datasource
- ein model package im daten package für die Joke Klasse

erstellt



Data Class

Spezielle Klasse zum Speichern von Daten

- besitzt automatisch toString(), equals(), copy() Methoden
- kann keine Kinder haben

```
package com.example.recyclerview.data.model

data class Joke(val stringResource: Int)
```

Data Source

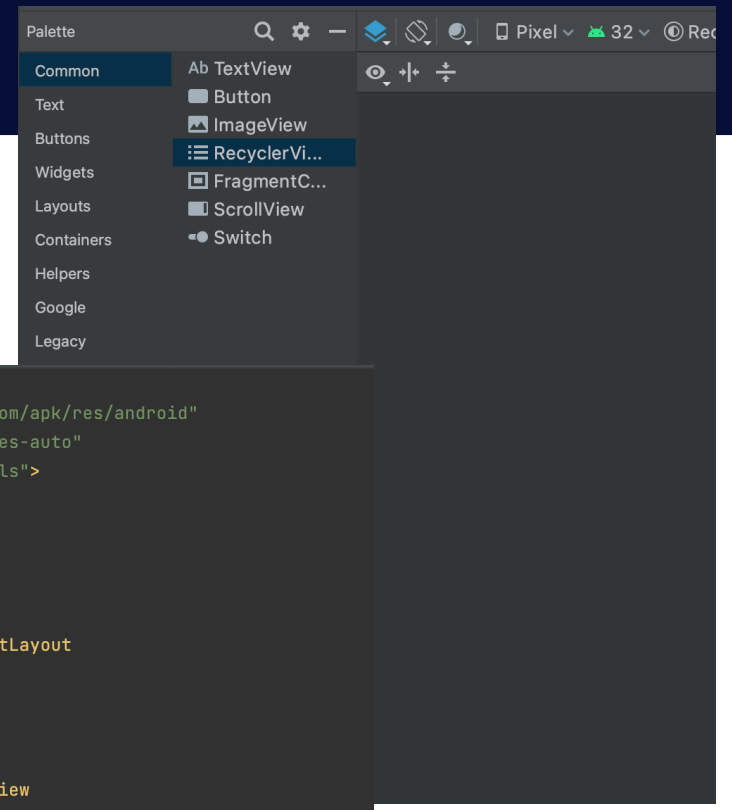
Ist für die Verwaltung von Daten zuständig

Bereitet Daten von verschiedenen Quellen und Formaten auf damit es für den Rest der App einheitlich ist

```
package com.example.recyclerview.data

import com.example.recyclerview.R
import com.example.recyclerview.data.model.Joke

class Datasource {

    fun loadJokes(): List<Joke> {
        return listOf(
            Joke(R.string.joke1),
            Joke(R.string.joke2),
            Joke(R.string.joke3),
            Joke(R.string.joke4),
            Joke(R.string.joke5),
            Joke(R.string.joke6),
            Joke(R.string.joke7),
            Joke(R.string.joke8),
            Joke(R.string.joke9)
        )
    }
}
```

Recycler View

Kann ganz normal wie andere Elemente ins Layout eingefügt werden



- `scrollbars="vertical"`

  bestimmt das von oben nach unten gescrollt wird

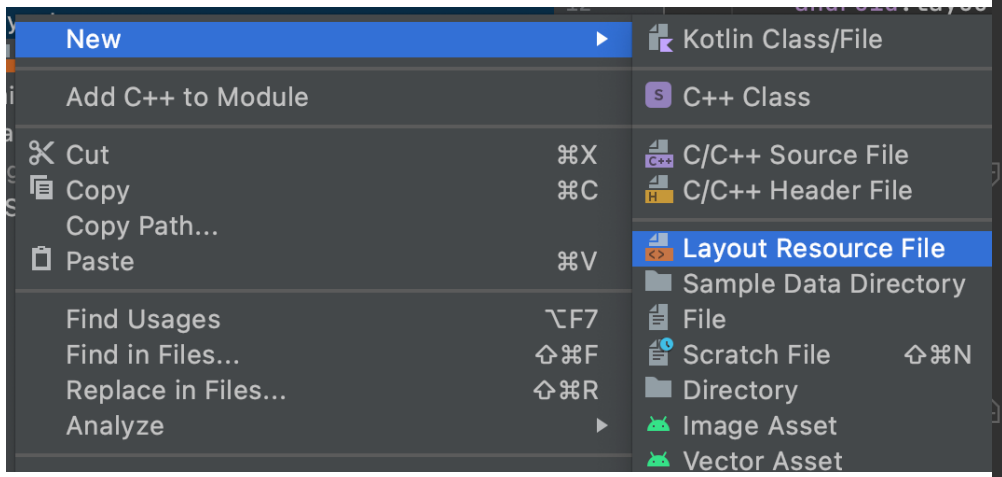
- `layoutManager="LinearLayoutManager"`

  die Elemente werden in einer Liste von oben nach unten dargestellt

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

    <data>

    </data>

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".MainActivity">

        <androidx.recyclerview.widget.RecyclerView
            android:id="@+id/recyclerView"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:scrollbars="vertical"
            app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

    </androidx.constraintlayout.widget.ConstraintLayout>
</layout>
```

Item Layout

Ein neues **list_item.xml** File bestimmt das Aussehen eines Witzes




```xml
<?xml version="1.0" encoding="utf-8"?>
<com.google.android.material.card.MaterialCardView
    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:id="@+id/item_cardView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:cardCornerRadius="10dp"
    app:cardElevation="5dp"
    app:cardPreventCornerOverlap="false"
    app:contentPadding="8dp"
    android:layout_margin="16dp">

    <TextView
        android:id="@+id/item_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        tools:text="@string/joke9" />


</com.google.android.material.card.MaterialCardView>
```

Adapter Klasse

Organisiert mit Hilfe der ViewHolder Klasse das Recycling

```
// der Adapter braucht den Context um auf String Resourcen zuzugreifen
// und die Liste an Jokes um sie für die RecyclerView vorzubereiten
class ItemAdapter(
    private val context: Context,
    private val dataset: List<Joke>
) : RecyclerView.Adapter<ItemAdapter.ItemViewHolder>() {

    // der ViewHolder weiß welche Teile des Layouts beim Recycling angepasst werden
    class ItemViewHolder(private val view: View) : RecyclerView.ViewHolder(view) {
        val textView: TextView = view.findViewById(R.id.item_text)
    }

    // hier werden neue ViewHolder erstellt
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ItemViewHolder {

        // das itemLayout wird gebaut
        val adapterLayout = LayoutInflater.from(parent.context)
            .inflate(R.layout.list_item, parent, attachToRoot: false)

        // und in einem ViewHolder zurückgegeben
        return ItemViewHolder(adapterLayout)
    }

    // hier findet der Recyclingprozess statt
    // die vom ViewHolder bereitgestellten Parameter werden verändert
    override fun onBindViewHolder(holder: ItemViewHolder, position: Int) {
        val item = dataset[position]
        holder.textView.text = context.resources.getString(item.stringResource)
    }

    // damit der LayoutManager weiß wie lang die Liste ist
    override fun getItemCount(): Int {
        return dataset.size
    }
}
```

MainActivity

- Zuerst werden die Witze in einer Variable gespeichert
- Dann wird die RecyclerView vom Layout mit dem Code verknüpft
- Unser Adapter wird erschaffen und der RecyclerView übergeben
- Da unsere RecyclerView eine fixe Größe hat kann die Performance nochmals geboostert werden

```
class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding

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

        binding = DataBindingUtil.setContentView( activity: this, R.layout.activity_main)

        // Liste an Jokes wird von der Datasource geladen
        val jokes = Datasource().loadJokes()

        // recyclerView von Layout wird mit code verknüpft
        val recyclerView = binding.recyclerView

        // ItemAdapter wird als Adapter festgelegt
        recyclerView.adapter = ItemAdapter( context: this, jokes)

        // verbesserte Performance bei fixer Größe
        recyclerView.setHasFixedSize(true)
    }
}
```

Fertig

Alles schon wieder vergessen?

Keine Angst, morgen machen wir das ganze noch einmal!

👍

Syntax Institut

Quelle: https://images.selbermachen.de/images/_aliases/1440w/0/4/6/3/153640-2-de-DE/16-07-Upcycling-1200x800.jpg

Viel Spaß!