Андроид разработка это классно

Но прежде чем начать, давайте поиграем в игру: «Поднимите руки кто…»

«Поднимите руки кто…»

- Писал что нибудь на Java/Kotlin

«Поднимите руки кто…»

- Писал что нибудь на Java/Kotlin
- Уже пробовал что нибудь писать под Android

«Поднимите руки кто…»

- Писал что нибудь на Java/Kotlin
- Уже пробовал что нибудь писать под Android
- Проходил где-то стажировку или стажируется сейчас

3+ млрд Устройств

Android – самая популярная операционная система для мобильных устройств

Доля на мировом рынке …

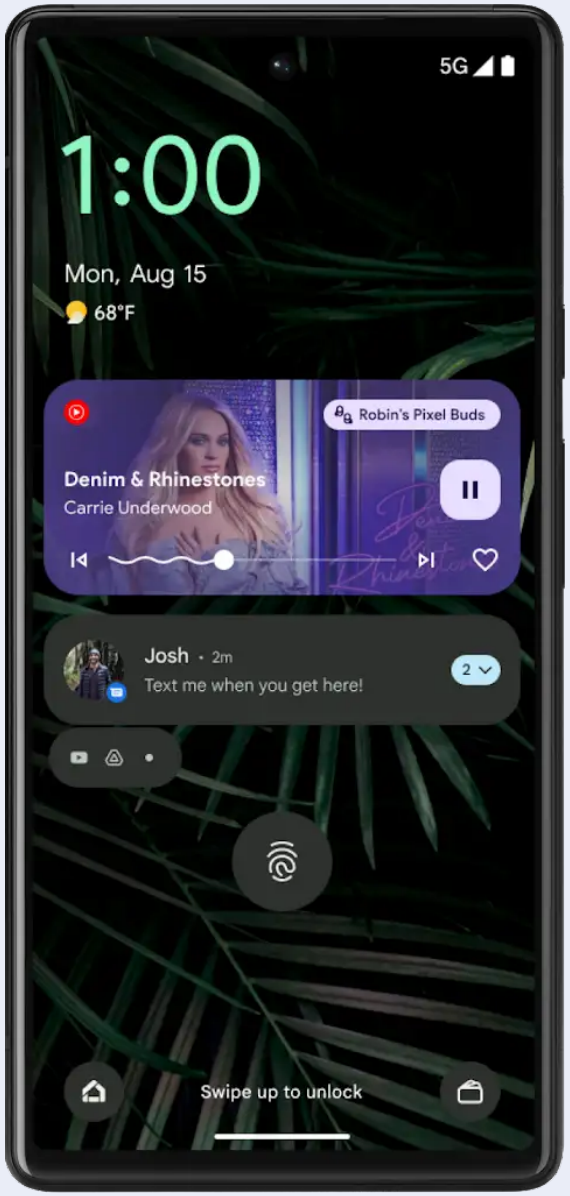
ОК

Доля на мировом рынке ~**72,26%**

https://gs.statcounter.com/os-market-share/mobile/worldwide

Android это

- Смартфоны



Android это

- Смартфоны
- Телевизоры

Android это

- Смартфоны
- Телевизоры
- Авто

Android это

- Смартфоны
- Телевизоры
- Авто
- Часы

Android это

- Смартфоны
- Телевизоры
- Авто
- Часы
- И ещё много различных устройств (умные колонки, холодильники, 3D-принтеры и т.д.)

Андроид основы

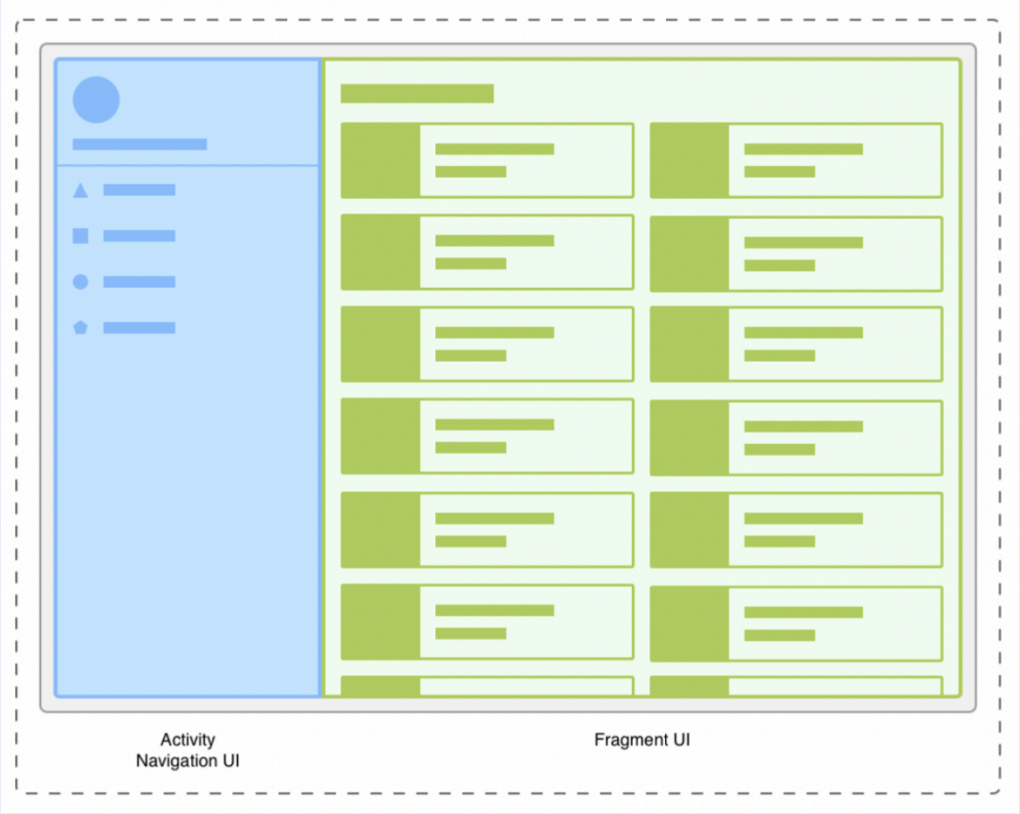
OK

Но перед началом давайте запустим

- Нужно удостоверится что у всех собирается проект
- Если у кого-то проблемы, то поднимите руки и мы поможем

android studio

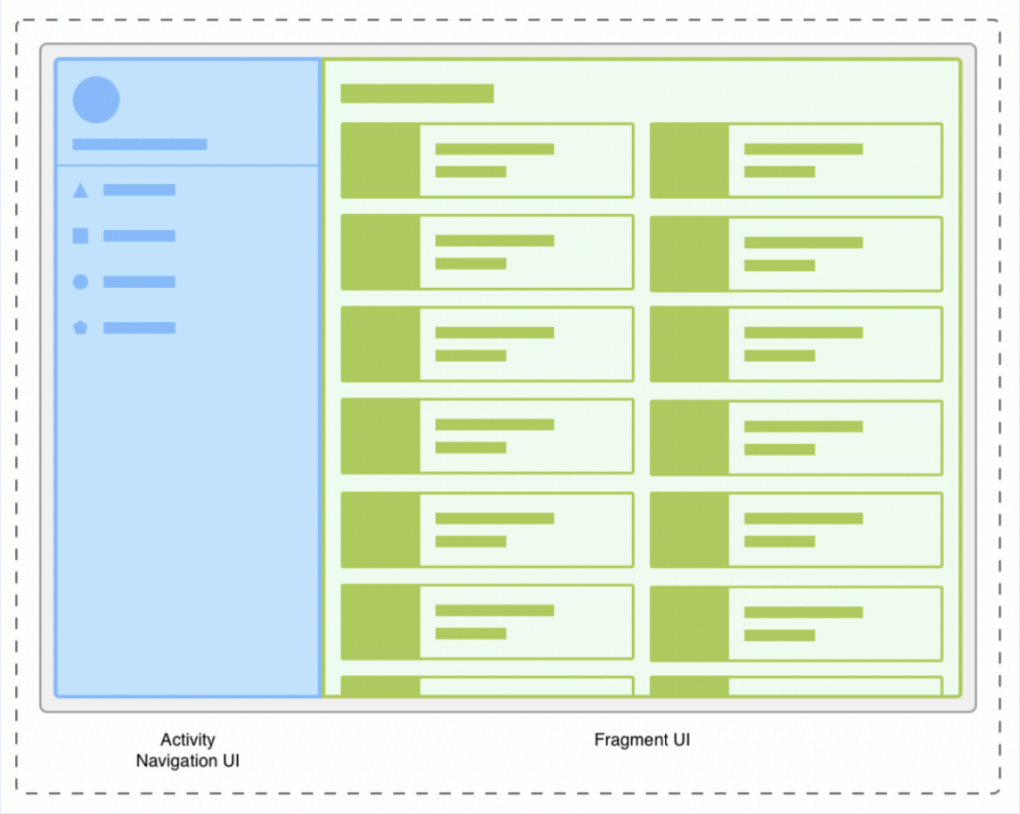
Android Fragments



https://developer.android.com/guide/fragments

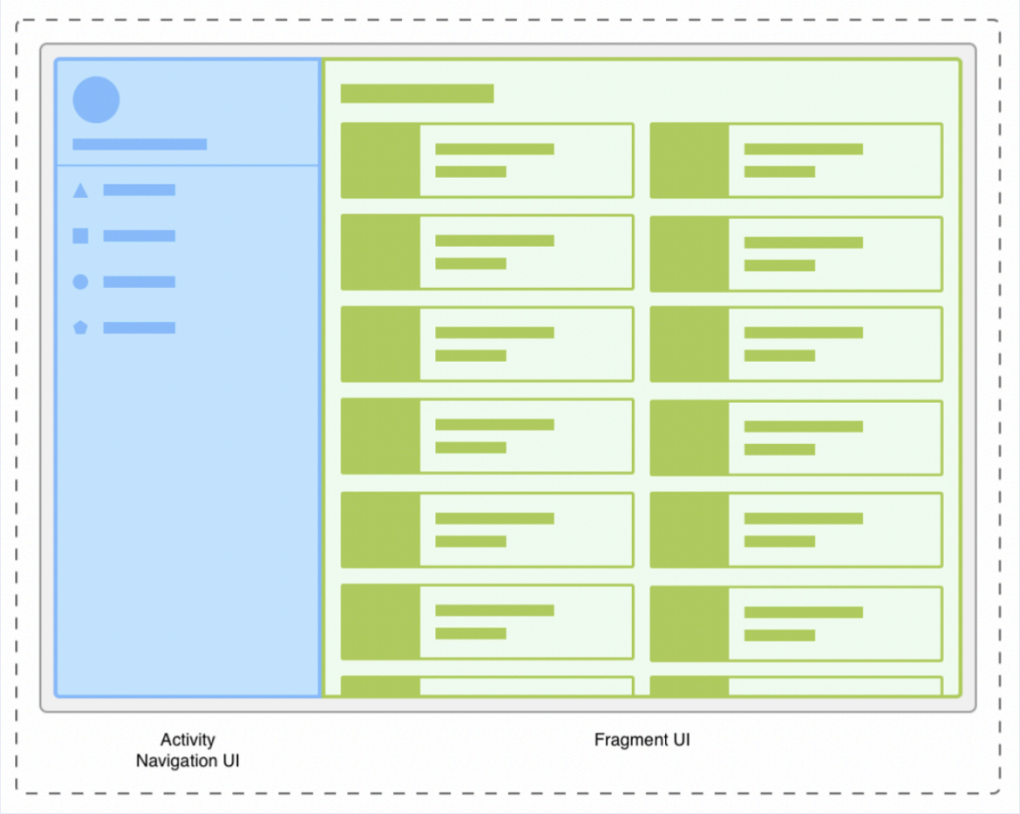
Fragments

- Модульные, переиспользуемые компоненты UI со своим lifecycle, зависящим от активити



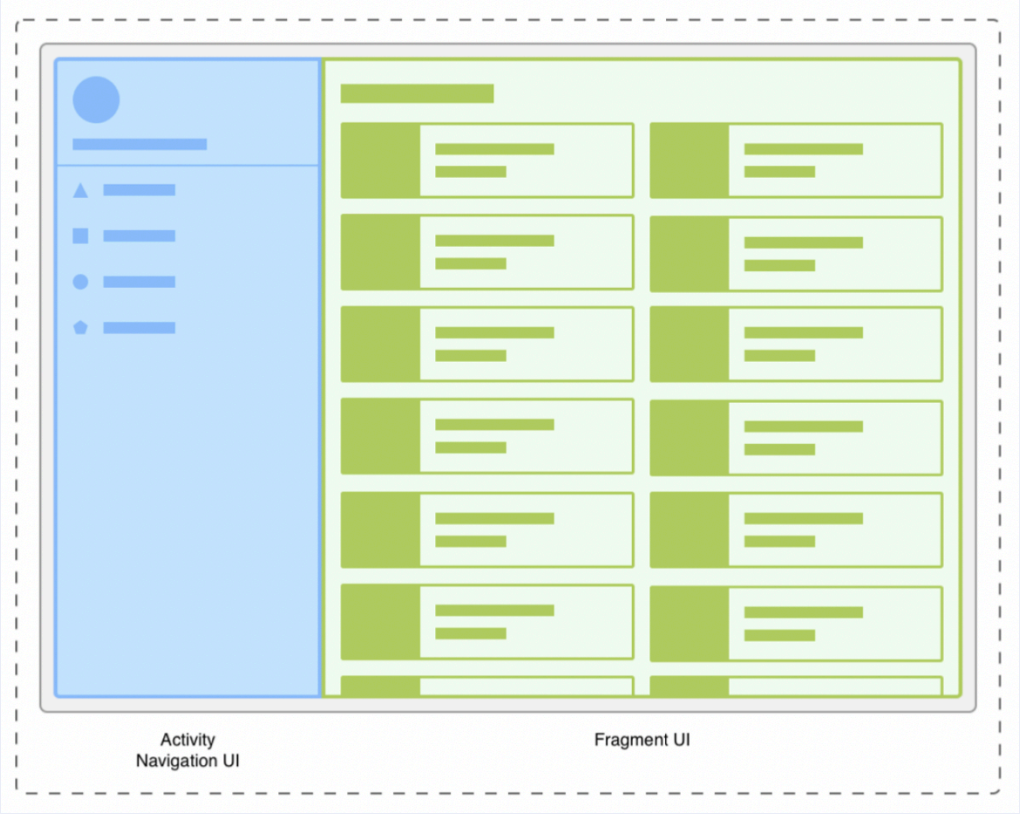
Fragments

- Модульные, переиспользуемые компоненты UI со своим lifecycle, зависящие от активити
- Более легковесные чем Activity



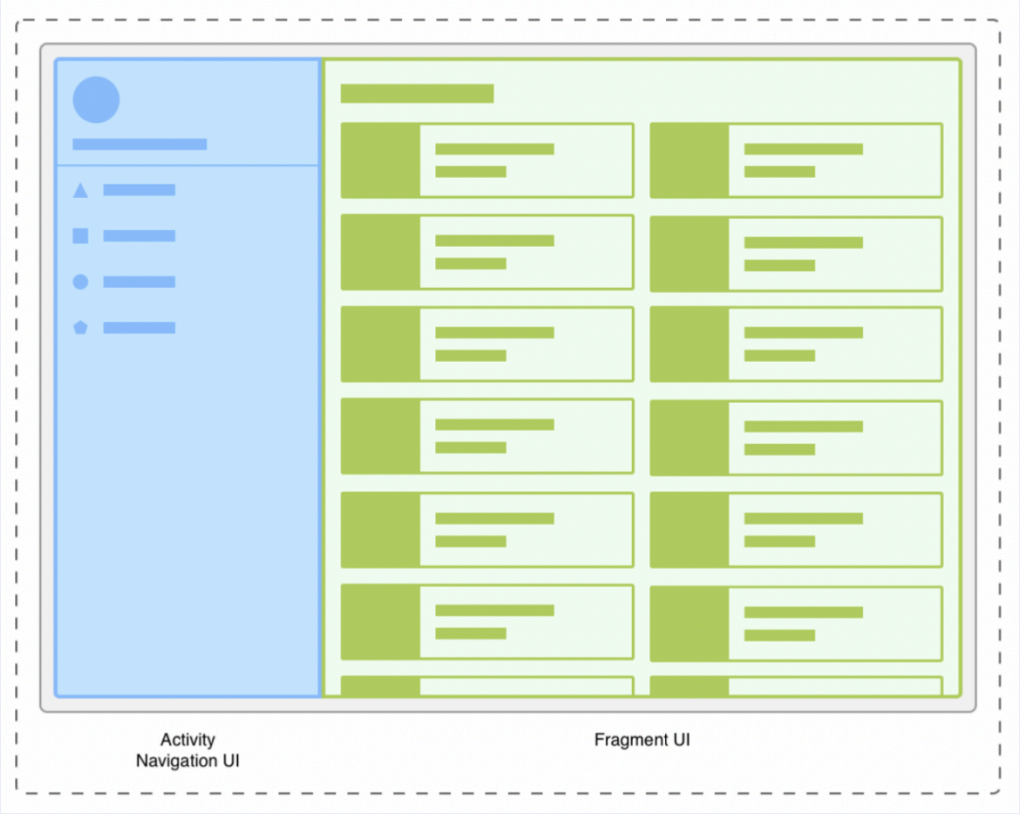
Fragments

- Модульные, переиспользуемые компоненты UI со своим lifecycle, зависящие от активити
- Более легковесные чем Activity
- Не нужно прописывать в Android manifest



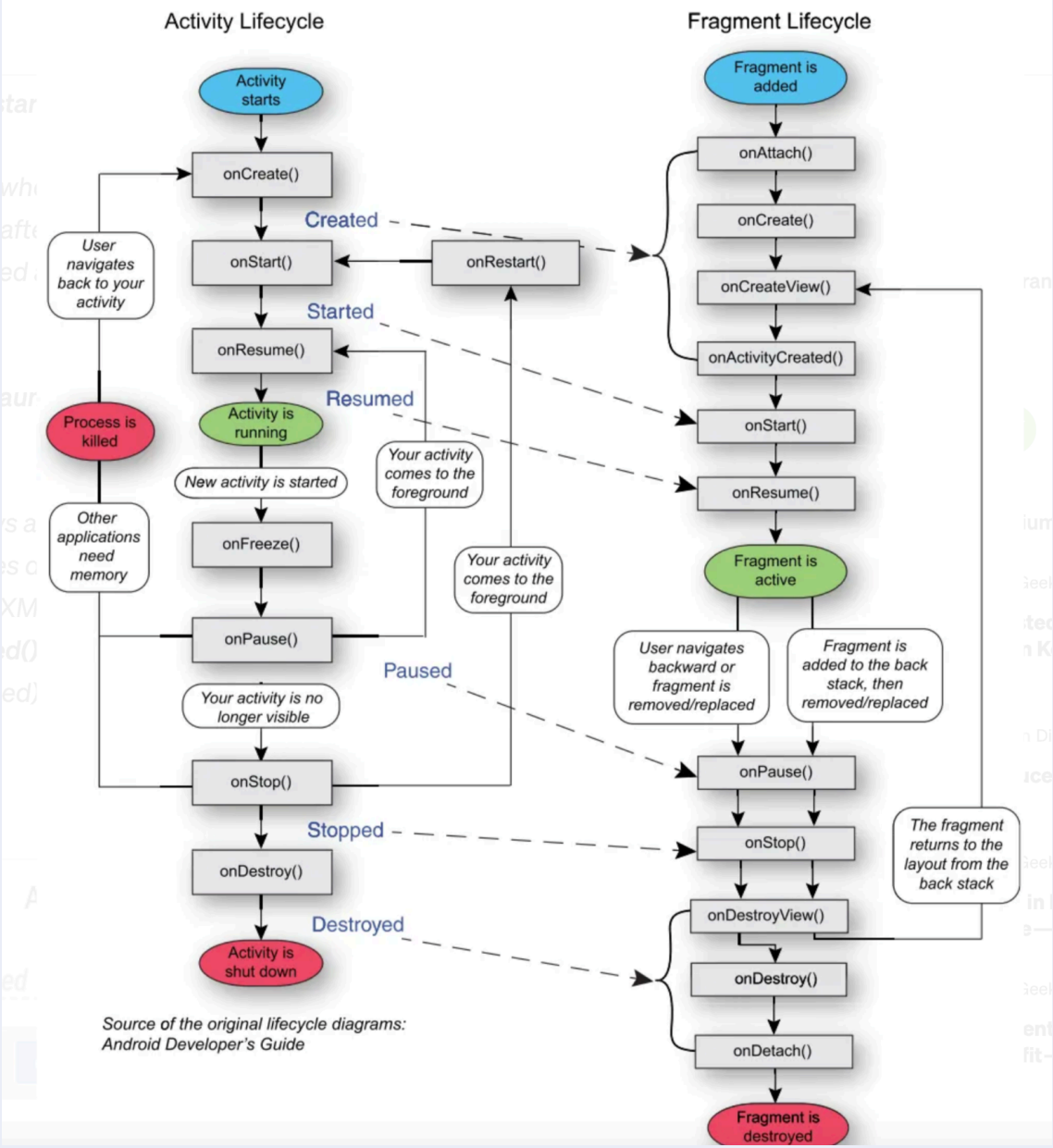
Fragments

- Модульные, переиспользуемые компоненты UI со своим lifecycle, зависящие от активити
- Более легковесные чем Activity
- Не нужно прописывать в Android manifest
- Больше возможностей для анимирования переходов между экранами



ЖЦ Fragments

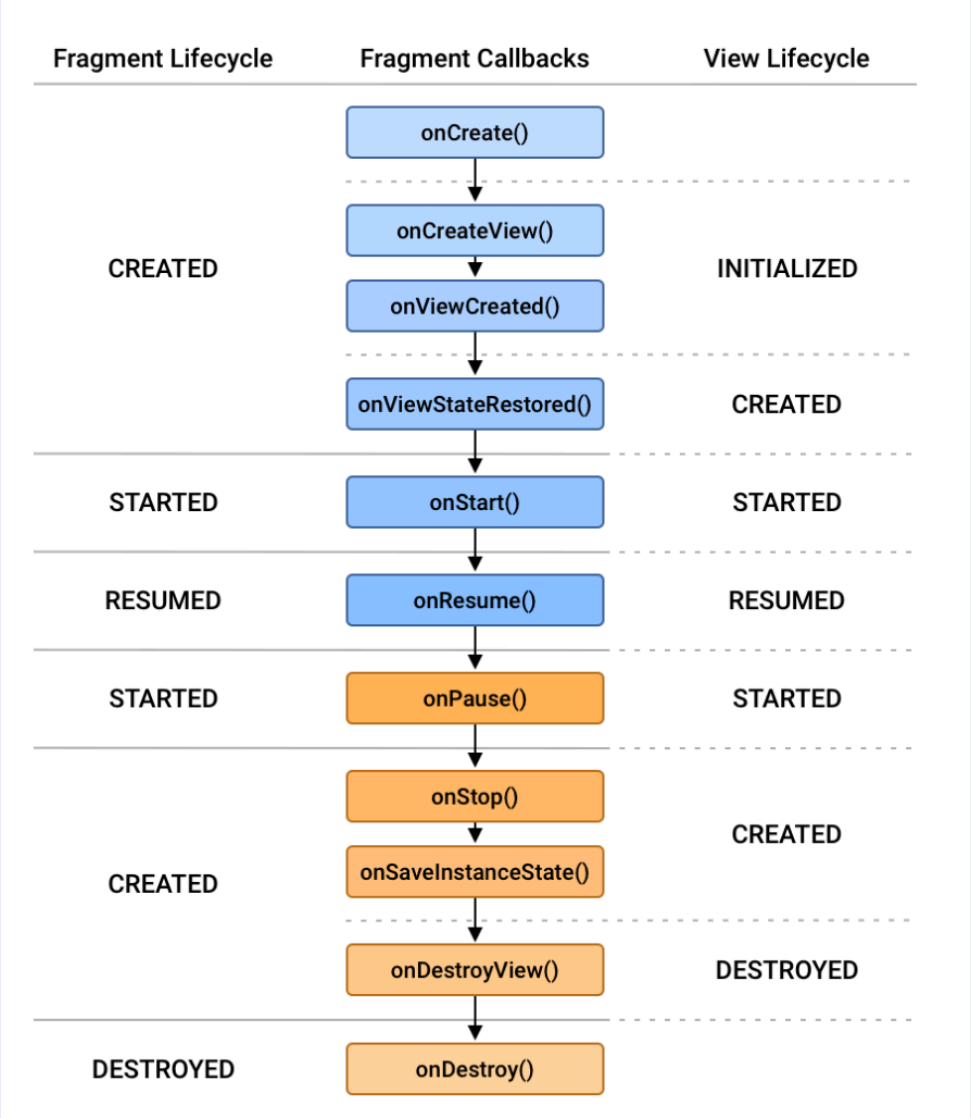
- Более сложный чем у Activity



https://basaransuleyman.medium.com/android-activity-lifecycle-and-fragment-lifecycle-states-and-method-descriptions-136efc3c2ff3

ЖЦ Fragments

- Более сложный чем у Activity

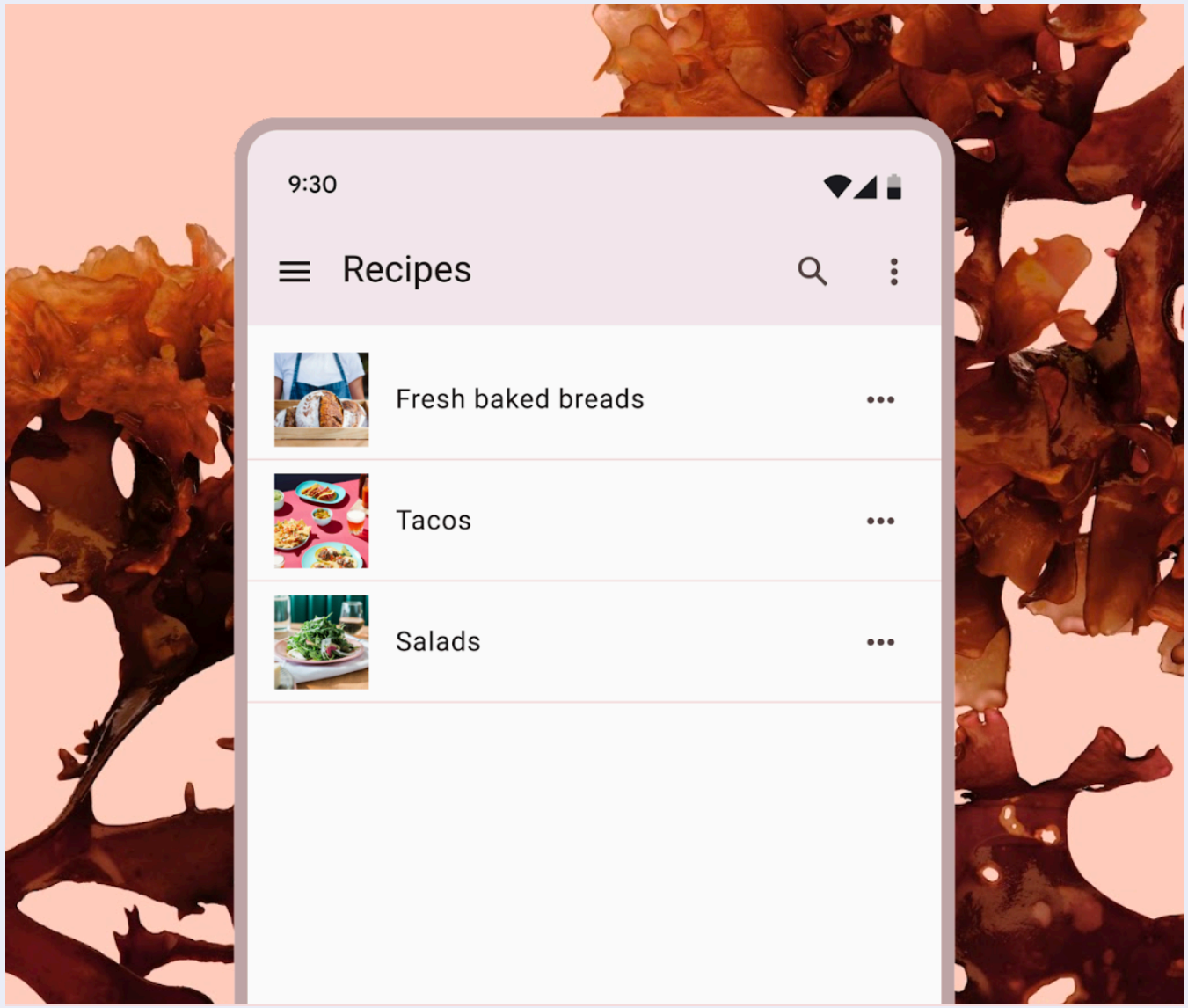


FragmentManager

- класс, отвечающий за выполнение действий над фрагментами вашего приложения, таких как добавление, удаление или замена, а также добавление их в стек.

https://developer.android.com/guide/fragments/fragmentmanager

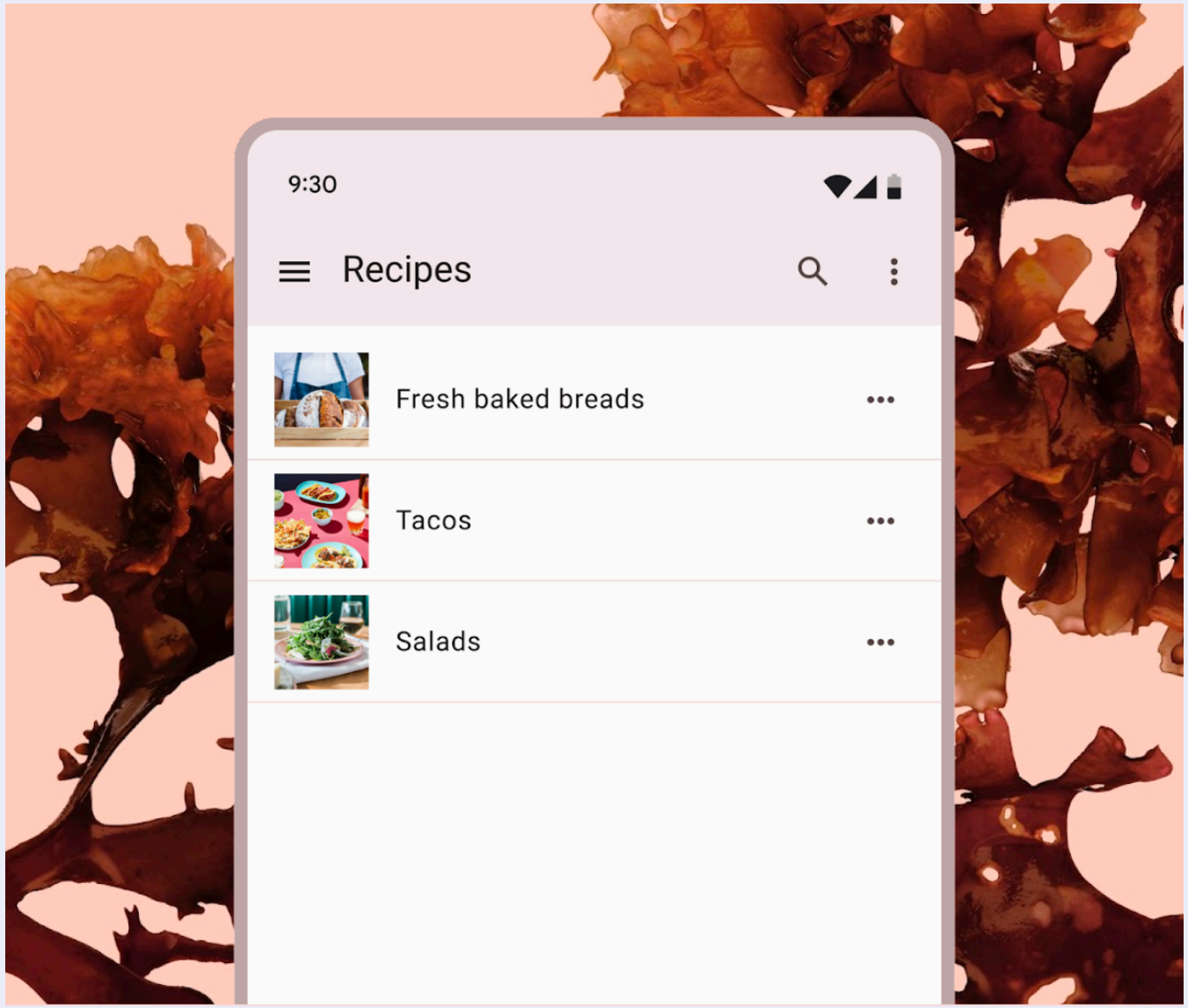
Recycler View



https://developer.android.com/develop/ui/views/layout/recyclerview

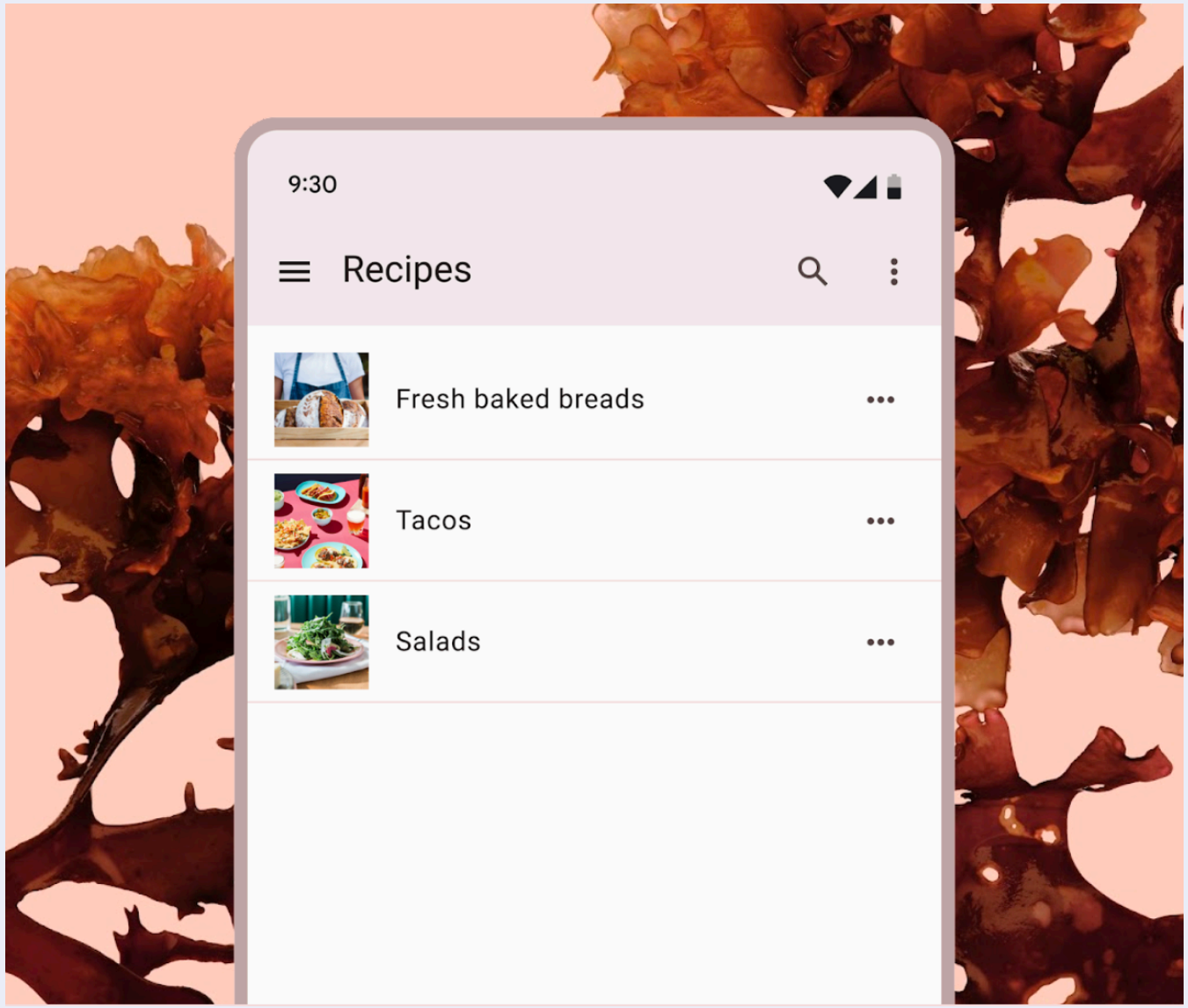
Recycler View

- Используется для отображение больших наборов данных. Вы предоставляете данные и определяете, как выглядит каждый элемент, а библиотека RecyclerView динамически создает элементы, когда они необходимы.
- Является наследником ViewGroup



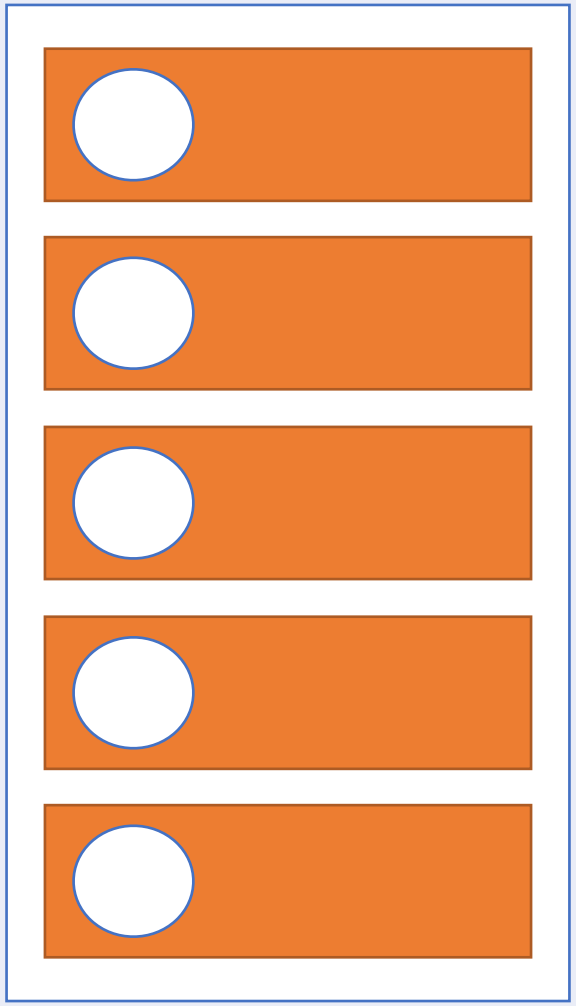
Recycler View

- `RecyclerView.Adapter`
- `RecyclerView.ViewHolder`
- `RecyclerView.LayoutManager`
- `DiffUtil`
- `ItemDecoration`



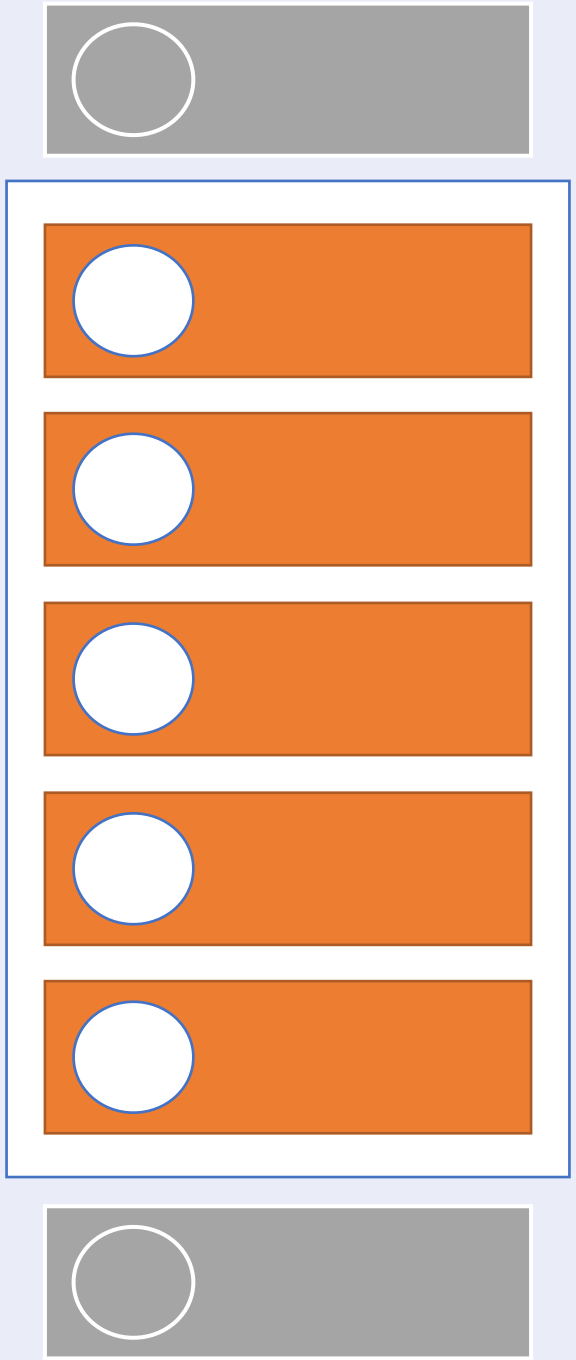
Recycler View

- Про механизм переиспользования



Recycler View

- Про механизм переиспользования



RecyclerView.Adapter

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

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
        TODO("Not yet implemented")
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        TODO("Not yet implemented")
    }

    override fun getItemCount(): Int {
        TODO("Not yet implemented")
    }

    class ViewHolder(view: View): RecyclerView.ViewHolder(view) {

    }
}
```

RecyclerView.Adapter

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

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
        TODO("Not yet implemented")
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        TODO("Not yet implemented")
    }

    override fun getItemCount(): Int {
        TODO("Not yet implemented")
    }

    class ViewHolder(view: View): RecyclerView.ViewHolder(view) {

    }
}
```

RecyclerView.Adapter

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

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
        TODO("Not yet implemented")
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        TODO("Not yet implemented")
    }

    override fun getItemCount(): Int {
        TODO("Not yet implemented")
    }

    class ViewHolder(view: View): RecyclerView.ViewHolder(view) {

    }
}
```

RecyclerView.Adapter

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

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
        TODO("Not yet implemented")
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        TODO("Not yet implemented")
    }

    override fun getItemCount(): Int {
        TODO("Not yet implemented")
    }

    class ViewHolder(view: View): RecyclerView.ViewHolder(view) {

    }
}
```

RecyclerView.ViewHolder

```
class ItemViewHolder(view: View): RecyclerView.ViewHolder(view) {

    private val tvCity: TextView
    private val tvWeather: TextView
    private val tvCountry: TextView

    init {
        tvCity = itemView.findViewById(R.id.tvCity)
        tvWeather = itemView.findViewById(R.id.tvWeather)
        tvCountry = itemView.findViewById(R.id.tvCountry)
    }

    fun bind(item: WeatherItem) {
        tvCity.text = item.city
        tvCountry.text = item.country
        tvWeather.text = item.weather
    }
}
```

RecyclerView.ViewHolder

```
class ItemViewHolder(view: View): RecyclerView.ViewHolder(view) {

    private val tvCity: TextView = itemView.findViewById(R.id.tvCity)
    private val tvWeather: TextView = itemView.findViewById(R.id.tvWeather)
    private val tvCountry: TextView = itemView.findViewById(R.id.tvCountry)

    fun bind(item: WeatherItem) {
        tvCity.text = item.city
        tvCountry.text = item.country
        tvWeather.text = item.weather
    }
}
```

RecyclerView.Adapter

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

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
        val inflater = LayoutInflater.from(parent.context)
        val view = inflater.inflate(R.layout.item_list, parent, false)
        return ViewHolder(view)
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        TODO("Not yet implemented")
    }

    override fun getItemCount(): Int {
        TODO("Not yet implemented")
    }
}
```

RecyclerView.Adapter

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

    private var items: List<WeatherItem> = ArrayList()

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
        val inflater = LayoutInflater.from(parent.context)
        val view = inflater.inflate(R.layout.item_list, parent, false)
        return ViewHolder(view)
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        TODO("Not yet implemented")
    }

    override fun getItemCount(): Int {
        return items.size
    }
}
```

RecyclerView.Adapter

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

    private var items: List<WeatherItem> = ArrayList()

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
        val inflater = LayoutInflater.from(parent.context)
        val view = inflater.inflate(R.layout.item_list, parent, false)
        return ViewHolder(view)
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        holder.bind(items[position])
    }

    override fun getItemCount(): Int {
        return items.size
    }

}
```

DiffUtil.Callback

```
public abstract abstract class Callback {

    public abstract int getOldListSize();

    public abstract int getNewListSize();

    public abstract boolean areItemsTheSame(int oldItemPosition, int newItemPosition);

    public abstract boolean areContentsTheSame(int oldItemPosition, int newItemPosition);

    @Nullable
    public Object getChangePayload(int oldItemPosition, int newItemPosition) {
        return null;
    }
}
```

DiffUtil.Callback

```
fun setData(newItems: List<WeatherItem>) {
    val oldItems = items
    items = newItems
    val diff = DiffUtil.calculateDiff(object : DiffUtil.Callback() {
        override fun getOldListSize(): Int {
            return oldItems.size
        }

        override fun getNewListSize(): Int {
            return items.size
        }

        override fun areItemsTheSame(oldItemPosition: Int, newItemPosition: Int): Boolean {
            return oldItems[oldItemPosition].city == items[newItemPosition].city
        }

        override fun areContentsTheSame(oldItemPosition: Int, newItemPosition: Int): Boolean {
            return oldItems[oldItemPosition] == items[newItemPosition]
        }
    }, false)
    diff.dispatchUpdatesTo(this)
}
```

RecyclerView.LayoutManager

- LinearLayoutManager
- GridLayoutManager
- StaggeredGridLayoutManager

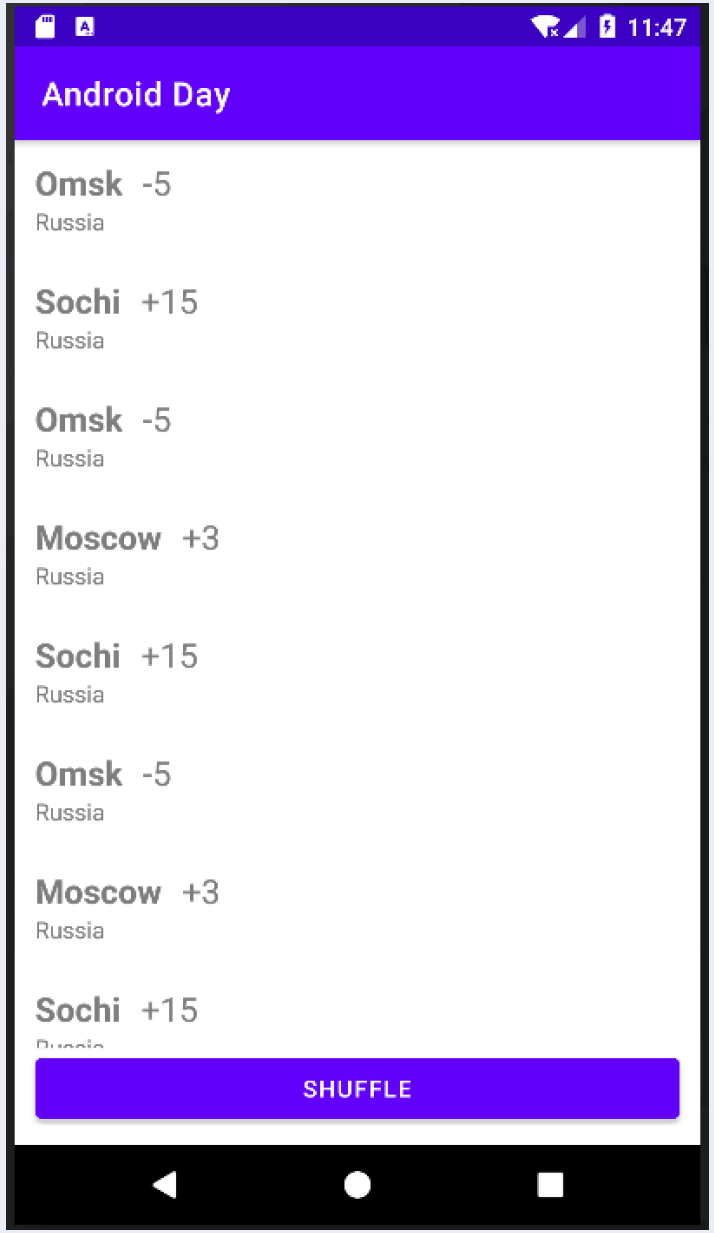
RecyclerView.LayoutManager

LinearLayout

```
recyclerView.layoutManager = LinearLayoutManager(requireContext(), LinearLayoutManager.VERTICAL, false)
```

RecyclerView.LayoutManager

LinearLayout



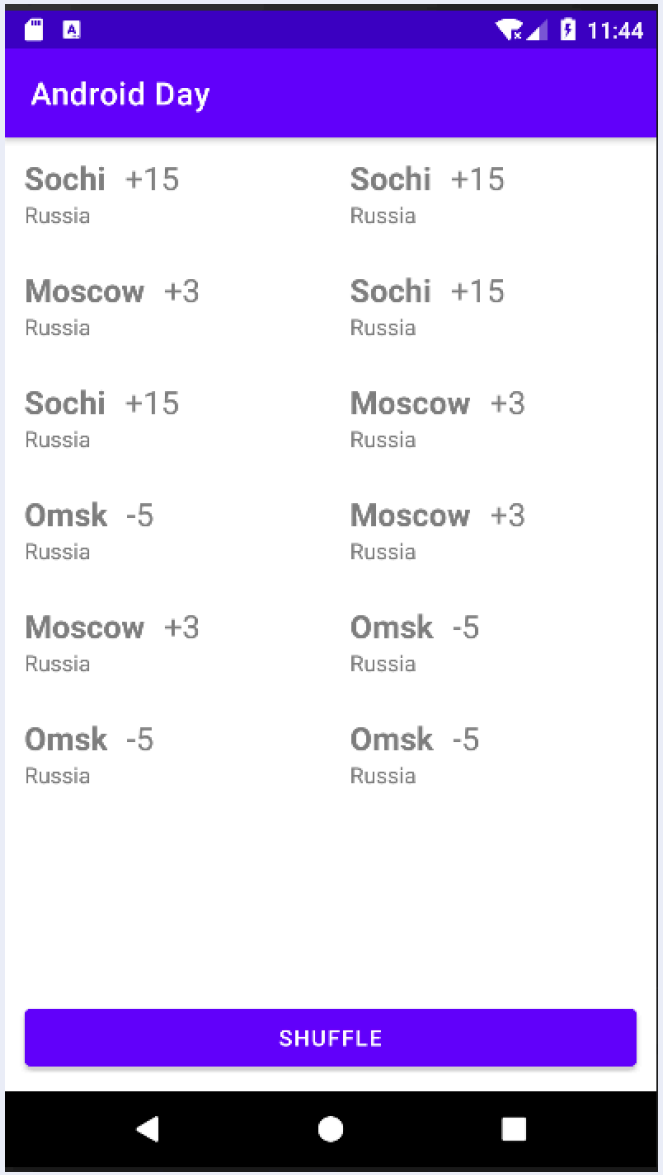
RecyclerView.LayoutManager

GridLayout

```
recyclerView.layoutManager = GridLayoutManager(requireContext(), 2)
```

RecyclerView.LayoutManager

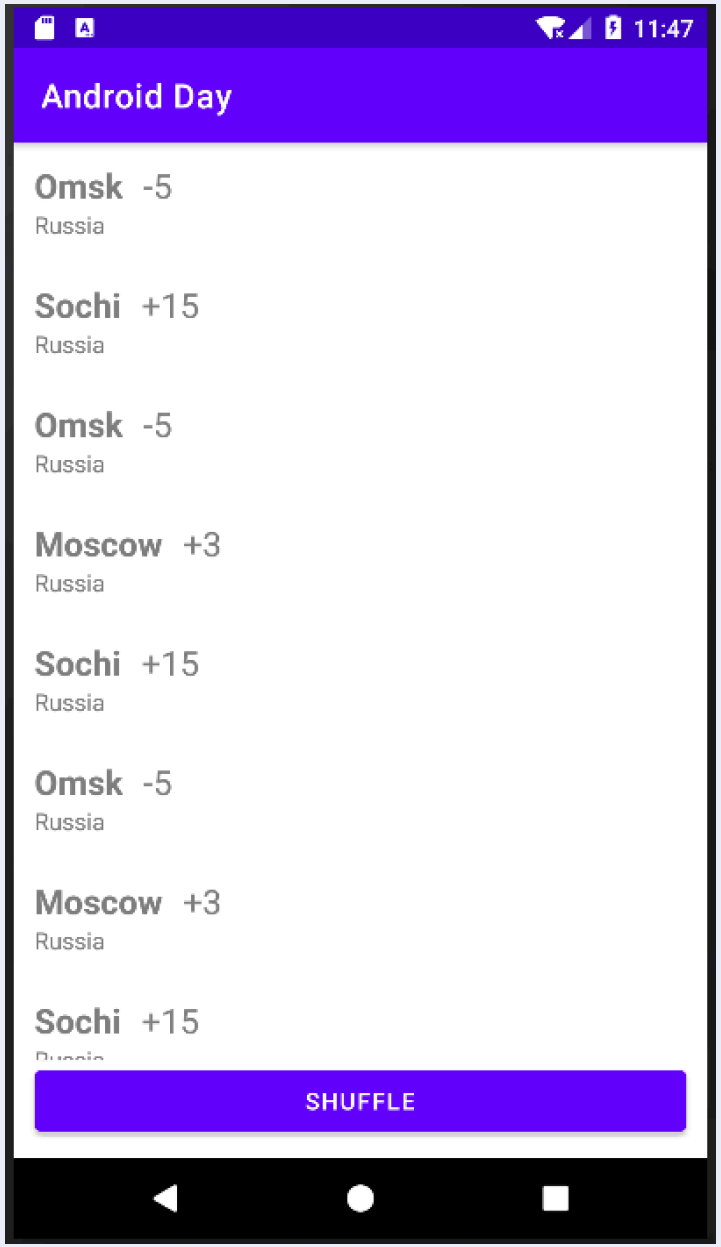
GridLayout

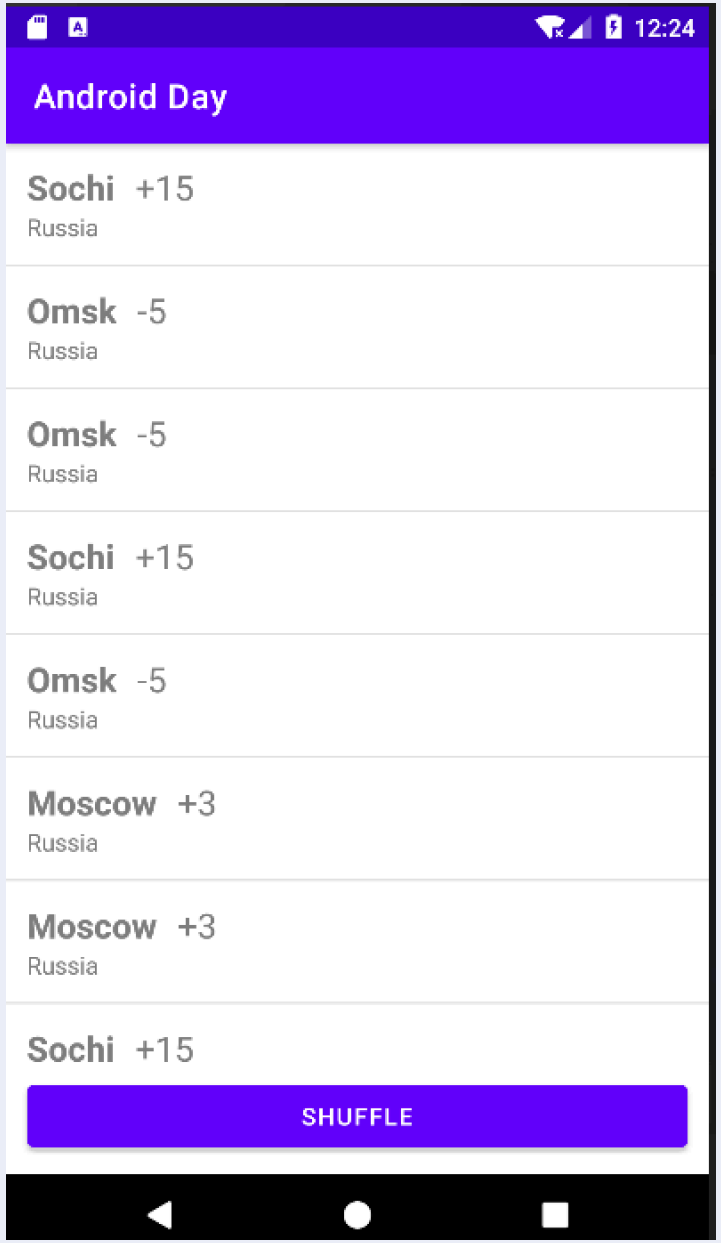


ItemDecoration

OK

ItemDecoration





ОК

Материалы для МК

А что же дальше?

ViewModel и LiveData
Retrofit
Glide/Picasso/Fresco
Room, SharedPreferences
Navigation Component
RxJava/Kotlin Flow/Kotlin coroutines
Clean architecture, MVVM/MVI
Dependency Injection (Dagger 2/Koin)
Handler, Looper, MessageQueue