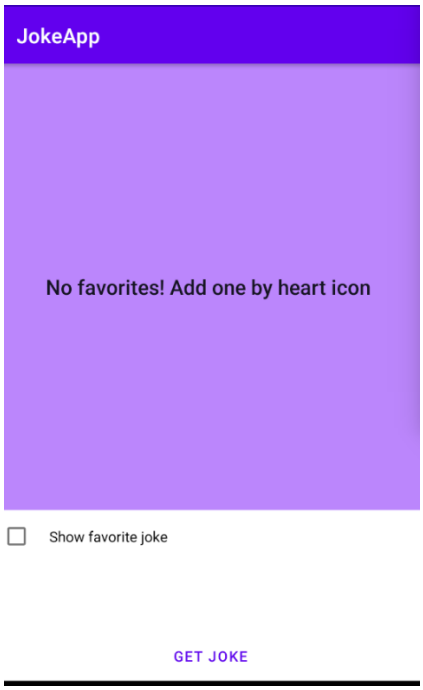
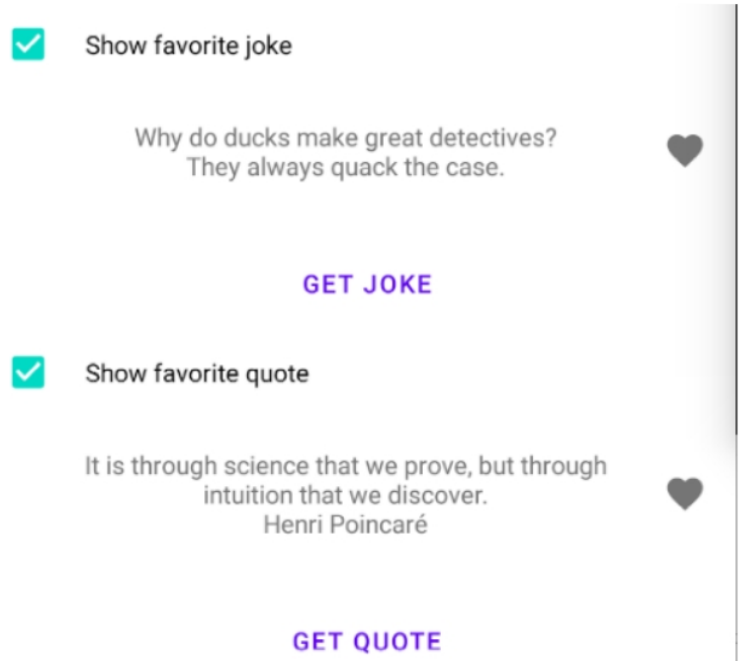
ViewPager, Fragments

Отображаем обе фичи одновременно



На данный момент наше приложение выглядит как-то так. У нас есть список избранных шуток и кастомвью для получения новых или просмотра избранных рандомно.

Если помните, то ранее (17-ая лекция) у нас приложение выглядело вот таким образом



У нас кроме шуток есть еще и цитаты. Но вот незадача: на одном экране умещается или 2 блока кастомвью или список шуток и кастомвью.

Вопрос: как отобразить больше контента одновременно?

Было бы круто иметь возможность смотреть и список избранных шуток и избранных цитат, так же получать новые шутки и новые цитаты.

Конечно же можно все положить в один контейнер и скролить остальное, но места получается слишком мало.

И у андроид есть решение, которое называется ViewPager, которое работает на Fragments.

Для начала нам нужно вынести в отдельную разметку контент

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager" />

    <com.github.johnnysc.jokeapp.presentation.FavoriteDataView
        android:id="@+id/favoriteDataView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```

Как видите у меня 1 разметка, которая подойдет как для шуток так и для цитат. Я убрал установку значений в кастомвью прямо из хмл, мы сделаем это из кода. Также я переименовал айдишник для вью.

Теперь перейдем к самому важному. У нас до сих пор была 1 активити и весь код был в ней. Но как мы уже сказали нам нужно отобразить всю информацию, но опять же на разных как бы экранах. Для этого есть фрагменты. Давайте напишем для начала базовый фрагмент в котором вся логика и после уже 2 наследника.

Просто создаем класс и наследуем от фрагмента андроид

```
import androidx.fragment.app.Fragment

/** @author Asatryan on 21.07.2021 ...*/
class BaseFragment : Fragment() {
```

Грубо говоря : активити у вас один, а в нем может жить более 1 фрагмента одновременно. И сами фрагменты могут заполнять весь экран. Мы увидим чуть позже каким образом. Итак, давайте напишем дальше базовый фрагмент. В активити мы читали дерево вью из хмл файла в onCreate с помощью setContentView, но в фрагменте все немного иначе. Здесь есть другой метод, который создает вью onCreateView

```
override fun onCreateView(
    inflater: LayoutInflater,
    container: ViewGroup?,
    savedInstanceState: Bundle?
): View? {
    return inflater.inflate(R.layout.data_fragment, container,  attachToRoot: false)
}
```

Запомните раз и навсегда, вот так должен выглядеть ваш onCreateView и никак иначе (за исключением очень редких кейсов, но почти всегда вот так). В 1 линию. Берем inflater и из хмл файла парсим вью. Теперь встает другой вопрос : а если здесь только чтение из хмл файла, то где я буду работать с вью? Для этого уже есть метод onViewCreated

```
abstract class BaseFragment<T> : Fragment() {

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup

    protected abstract fun getViewModel(): BaseViewModel<T>

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        val viewModel = getViewModel()

    }
```

Если помните, то у нас есть базовая вьюмодель дженерик типа и потому я переписал базовый фрагмент чтобы он был абстрактным и типизированным. Далее я буду писать метод уже внутри onViewCreated

Но мне особо ничего не нужно придумывать: все уже готово и просто лежит в активити.

Потому я просто перенесу в базовый фрагмент код из активити

Первое отличие фрагмента от активити в том, что в активит вы просто имеете доступ к вью через findViewById но в фрагменте мы создали вью в отдельном методе и потому пробрасываем его в метод onViewCreated в аргументе и теперь нужно у этого обьекта вызывать метод поиска вью по айди.

Все остальное в принципе одинаковое. Но мы забыли еще об одном. Раз в хмл файле я убрал установку текстов кастомвью, то мне нужно делать это в коде. Поэтому давайте добавим 2 метода в кастомвью

```
val communication = getCommunication()

val favoriteDataView = view.findViewById<FavoriteDataView>(R.id.favoriteDataView)
favoriteDataView.linkWith(viewModel)
viewModel.observe( owner: this, { state ->
    favoriteDataView.show(state)
})
val recyclerView = view.findViewById<RecyclerView>(R.id.recyclerView)
val adapter = CommonDataRecyclerAdapter(object :
    CommonDataRecyclerAdapter.FavoriteItemClickListener<T> {
    override fun change(id: T) {
        Snackbar.make(
            favoriteDataView,
            R.string.remove_from_favorites,
            Snackbar.LENGTH_SHORT
        ).setAction(R.string.yes) { it: View!
            viewModel.changeItemStatus(id)
        }.show()
    }
}, communication)
recyclerView.adapter = adapter

viewModel.observeList( owner: this, { it: List<CommonUiModel<T>>!
    adapter.update()
})
viewModel.getItemList()
```

Так как тексты у меня идут из ресурсов, то я сделаю аргумент метода айди интовый

```
fun checkBoxText(@StringRes id:Int) = checkBox.setText(id)
fun actionButtonText(@StringRes id:Int) = actionButton.setText(id)
```

Но с аннотацией @StringRes чтобы не сетили любые числа. Теперь вернемся к фрагменту

```
favoriteDataView.checkBoxText(checkBoxText())
favoriteDataView.actionButtonText(actionButtonText())
```

И 2 абстрактный метода для получения в наследниках

```
@StringRes
protected abstract fun checkBoxText() : Int
@StringRes
protected abstract fun actionButtonText() : Int
```

И вроде как теперь можно приступить к написанию наследников для шуток и цитат.

```
class JokesFragment : BaseFragment<Int>() {
    override fun getViewModel() = (requireActivity().application as JokeApp).jokeViewModel
    override fun getCommunication() = (requireActivity().application as JokeApp).jokeCommunication
    override fun checkBoxText() = R.string.show_favorite_joke
    override fun actionButtonText() = R.string.get_joke
}
```

Если помните, то наши инстансы вьюмоделей и комуникаций лежат в апликейшн классе. И здесь встает вопрос, а как добраться до него из фрагмента? Ну, из фрагмента можно получить активити и дальше уже все известно. Но у нас тут нарушается DRY don’t repeat yourself и потому я бы переписал прямо сейчас код чтобы методы в аргументах получали апликейшн

```
protected abstract fun getViewModel(app:JokeApp): BaseViewModel<T>
protected abstract fun getCommunication(app:JokeApp): BaseCommunication<T>

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    val application = requireActivity().application as JokeApp
    val viewModel = getViewModel(application)
    val communication = getCommunication(application)
```

И уже в наследниках будет намного проще, смотрите

```
class JokesFragment : BaseFragment<Int>() {
    override fun getViewModel(app: JokeApp) = app.jokeViewModel
    override fun getCommunication(app: JokeApp) = app.jokeCommunication
    override fun checkBoxText() = R.string.show_favorite_joke
    override fun actionButtonText() = R.string.get_joke
}
```

Вот и все. Весь код фрагмента – конкретные реализации вьюмодели и комуникаций и строковые ресурсы. Все остальное должно работать само собой в базовом фрагменте.

Точно так же легко и просто напишем второй фрагмент для цитат

```
class QuotesFragment : BaseFragment<String>() {
    override fun getViewModel(app: JokeApp) = app.quoteViewModel
    override fun getCommunication(app: JokeApp) = app.quoteCommunication
    override fun checkBoxText() = R.string.show_favorite_quote
    override fun actionButtonText() = R.string.get_quote
}
```

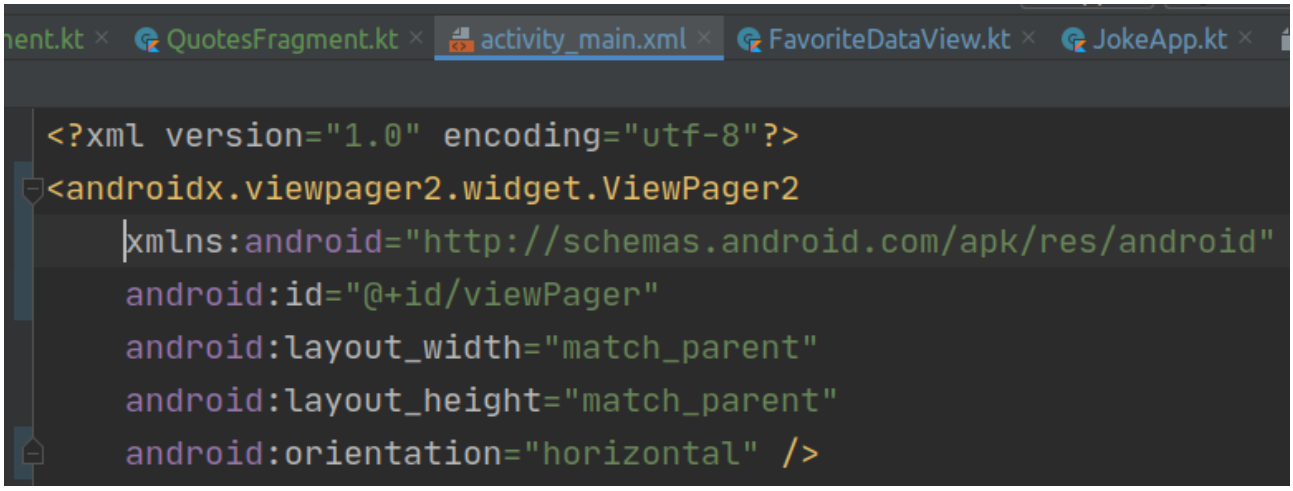
Вот и все. Я просто создал в апликейшн классе нужные инстансы и дал ресурсы цитаты.

Хорошо, мы написали 2 фрагмента, что дальше? Нужно их отобразить на активити одновременно, но просто расположить их поделив пополам мы решили что не круто.

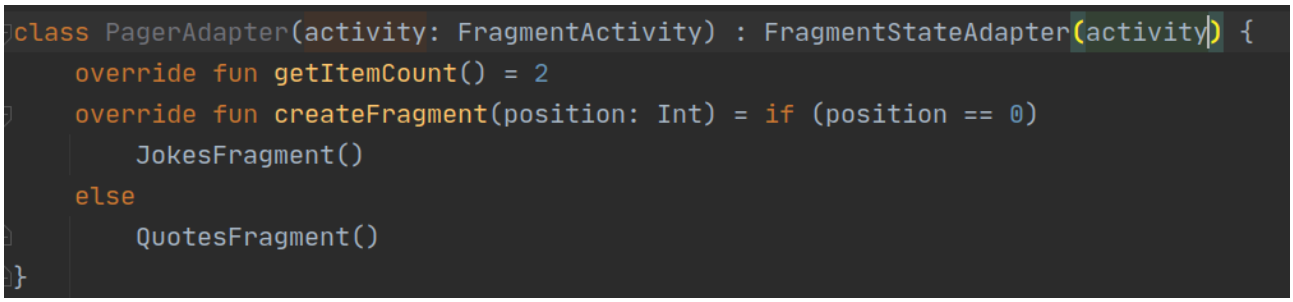
Потому мы хотим видеть каждый фрагмент полностью. Но наверно попеременно. Как?

На этот вопрос отвечает ViewPager2.

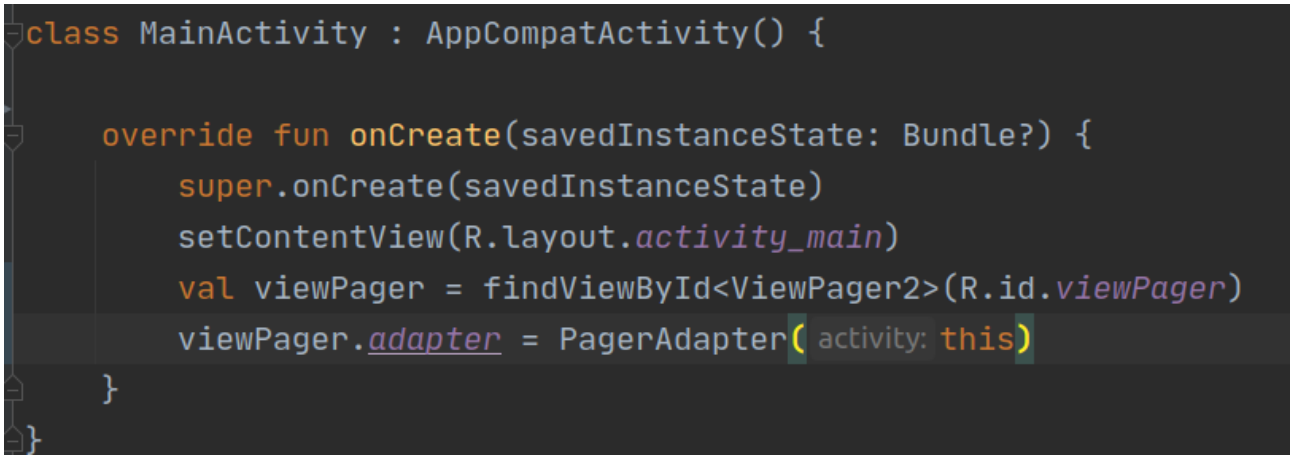
Перепишем теперь активити. Там теперь будет 1 вью



Вот и вся разметка активити. Обратите внимание что мы используем 2 версию и здесь можно указывать ориентацию как в линейном контейнере. Наверно все же нужно сказать что такое вьюпейджер – контейнер, где можно свайпом менять содержание. И ровно так же как с ресайклером, нам нужен адаптер! Давайте напишем его



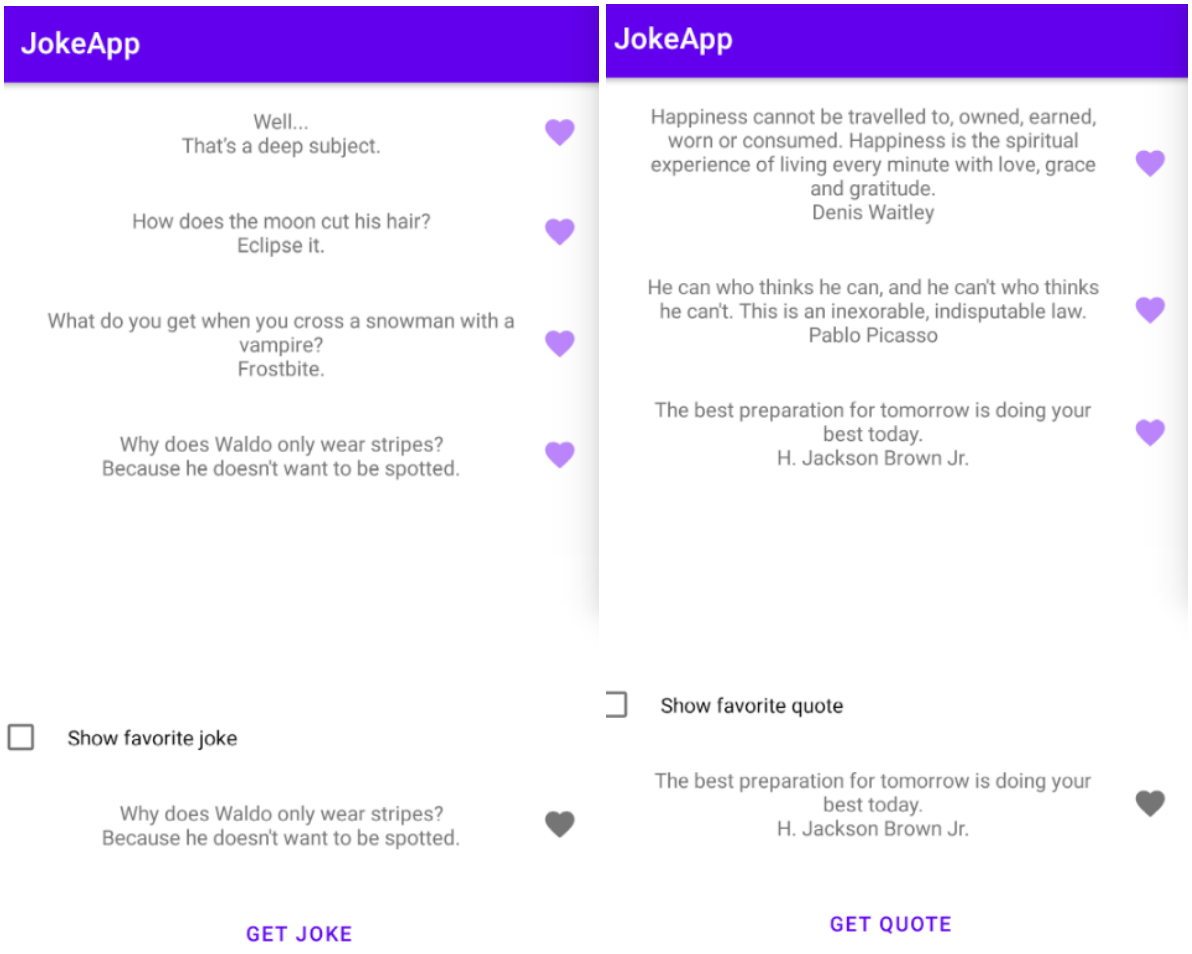
Вот так вот просто. Как видите похоже на адаптер ресайклера. Количество элементов и просто получить по позиции. Я буду показывать шутки сначала и потом цитаты. Давайте уже перепишем активити класс чтобы поскорей проверить как оно все работает



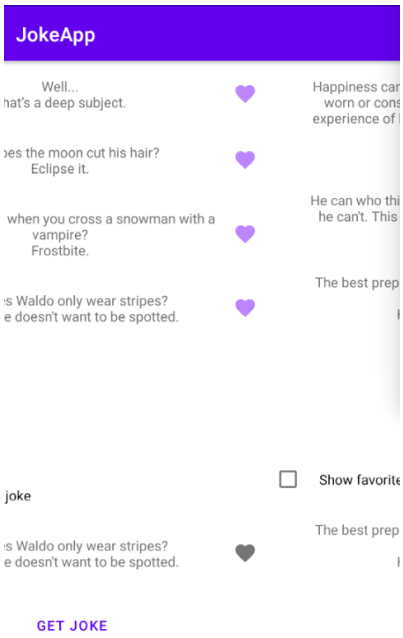
Вот и все! Весь код активити – находим вьюпейджер и сетим адаптер. Запустим проект!

ДА! Все работает!

Сначала видим шутки, свайпом вправо переходим на цитаты



Но здесь возникает вопросик : а как юзер впервые открывший приложение узнает что можно свайпом перейти на цитаты?



Ведь он открывая приложение видит лишь шутки и не зная о том, что можно свайпом перейти он никогда и не перейдет на цитаты. Здесь нам может вьюшка под названием TabLayout. Простые кнопки которые можно связать с вьюпейджером. Давайте посмотрим

Перейдем в разметку активити и добавим туда новую вью

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <com.google.android.material.tabs.TabLayout
        android:id="@+id/tabLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <androidx.viewpager2.widget.ViewPager2
        android:id="@+id/viewPager"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:orientation="horizontal" />

</LinearLayout>
```

Я покажу названия (Шутки, Цитаты) наверху, но если вы хотите, то можете переместить вниз. В принципе на них можно тапать вместо свайпов и более удобно будет тапать когда кнопки расположены на дне экрана. Но в любом случае, свайпы остаются.

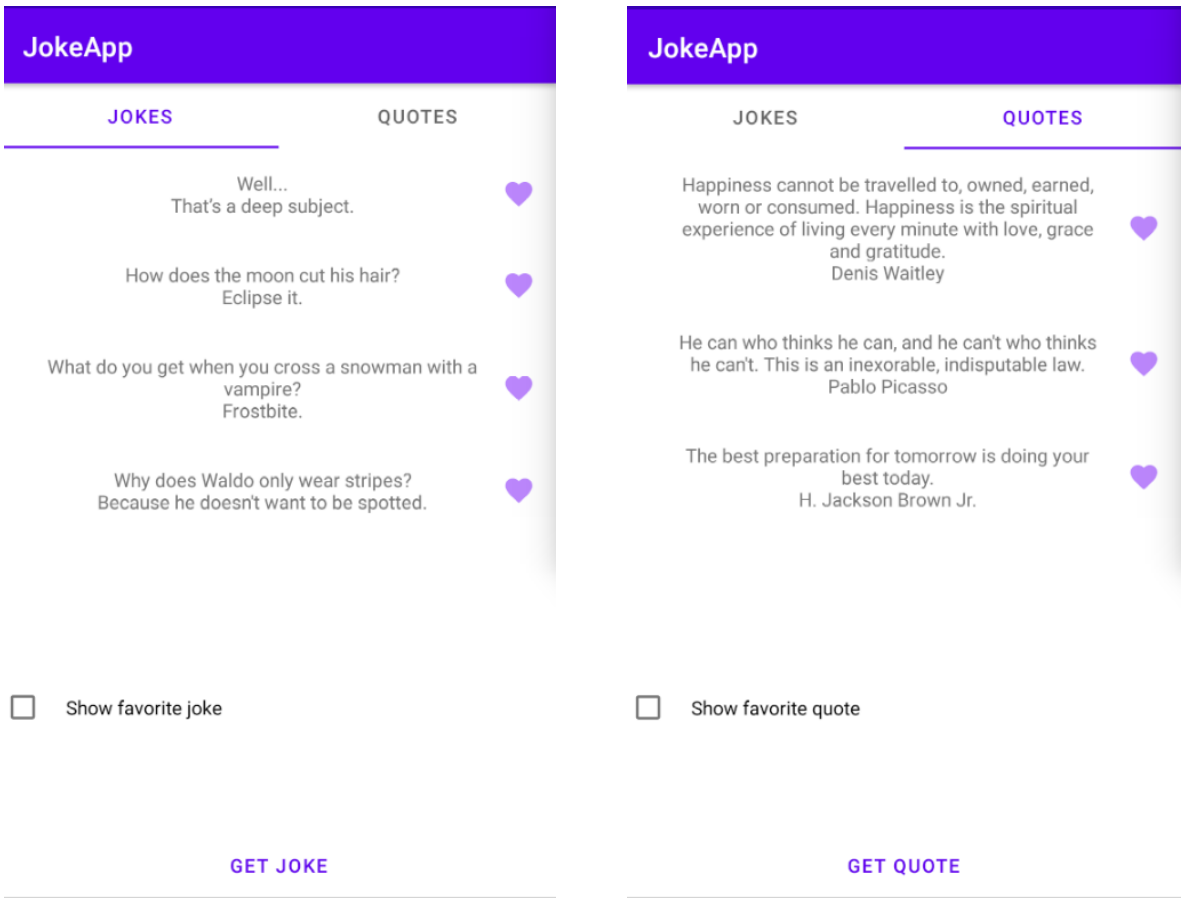
Нам нужно дать текстовые значения этим кнопкам и мы это сделаем в активити

```
val viewPager = findViewById<ViewPager2>(R.id.viewPager)
val tabLayout = findViewById<TabLayout>(R.id.tabLayout)
viewPager.adapter = PagerAdapter( activity: this)
TabLayoutMediator(tabLayout, viewPager) { tab, position ->
    tab.text = getString(if (position == 0) R.string.jokes else R.string.quotes)
}.attach()
```

Я создал 2 текстовки для кнопок и они будут отображены на экране

```xml
<string name="jokes">Jokes</string>
<string name="quotes">Quotes</string>
```

Итак, запустим теперь проект!

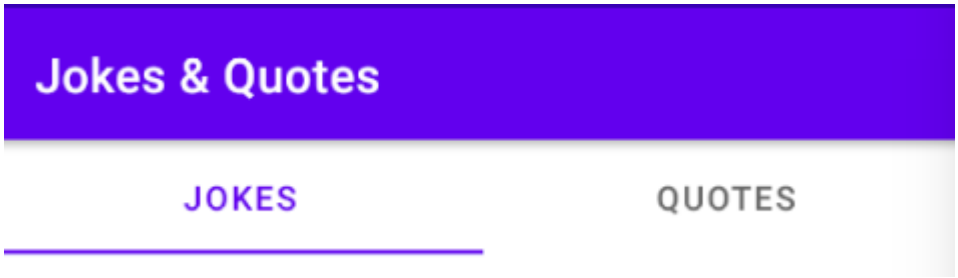


Как видите, теперь сразу понятно, что в приложении есть не только шутки, но и цитаты. Вы можете как тапать на кнопки, так и свайпать. И то и другое доступно одновременно!

Единственное конечно же что у нас имя приложения осталось все еще JokesApp. Надо бы поменять

```
<resources>
    <string name="app_name">Jokes &amp; Quotes</string>
```

И теперь намного лучше!



Хочешь – читай шутки, хочешь – цитаты.

На этом мы закончи тему вьюпдейджера и таблейаута. Вы можете самостоятельно изучить каким образом можно стилизовать кнопки, когда выбрано, когда нет. Мне кажется если человек дошел до этой лекции (а она 22-ая по счету), то он уже не новичок и умеет по крайней мере гуглить и искать на офф.сайте андроид информацию.

Спасибо всем кто читает лекции. В следующей будем улучшать код!