План занятия

1. Задача
2. Лента постов
3. RecyclerView
4. DiffUtil & ListAdapter
5. Итоги

ЗАДАЧА

ЗАДАЧА

Мы научились работать с коллекциями элементов (а также Sequence) в Kotlin. А также научились отображать один элемент в Android-приложении.

Естественная задача: научиться отображать коллекцию элементов, т.е. сделать так называемую ленту постов.

ЛЕНТА ПОСТОВ

ЛЕНТА ПОСТОВ

С первого взгляда идея достаточно простая:

1. Модернизируем ViewModel и Repository для хранения коллекции элементов.
2. Создаём отдельный layout для карточки поста.
3. Итерируемся по элементам нашей коллекции.
4. Для каждого элемента из XML ресурса с layout для карточки создаём View.
5. Добавляем View на экран Activity.

REPOSITORY

```
interface PostRepository {
    fun getAll(): LiveData<List<Post>>
    fun likeById(id: Long)
}
```

теперь лайкаем по id (как в курсе по Kotlin)

REPOSITORY

```
class PostRepositoryInMemoryImpl : PostRepository {
    private var posts = listOf(
        Post(
            id = 2,
            author = "Нетология. Университет интернет-профессий будущего",
            content = "Знаний хватит на всех: на следующей неделе разбираемся с р
            published = "18 сентября в 10:12",
            likedByMe = false
        ),
        Post(
            id = 1,
            author = "Нетология. Университет интернет-профессий будущего",
            content = "Привет, это новая Нетология! Когда-то Нетология начиналась
            published = "21 мая в 18:36",
            likedByMe = false
        ),
    )

    private val data = MutableLiveData(posts)

    override fun getAll(): LiveData<List<Post>> = data
    override fun likeById(id: Long) {
        posts = posts.map { it: Post
            if (it.id != id) it else it.copy(likedByMe = !it.likedByMe)
        }
        data.value = posts
    }
}
```

VIEWMODEL

```
class PostViewModel : ViewModel() {
    // упрощённый вариант
    private val repository: PostRepository = PostRepositoryInMemoryImpl()
    val data = repository.getAll()
    fun likeById(id: Long) = repository.likeById(id)
}
```

ACTIVITY LAYOUT

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    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:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".activity.MainActivity">
</androidx.constraintlayout.widget.ConstraintLayout>
```

Обратите внимание, мы убрали всё содержимое.

POST CARD LAYOUT

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    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:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="@dimen/common_spacing">

    <ImageView
        android:id="@+id/avatar"
        app:layout_constraintBottom_toBottomOf="@id/header"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:srcCompat="@sample/posts.json/data/authorAvatar"
        android:layout_width="@dimen/posts_avatar_size"
        android:layout_height="@dimen/posts_avatar_size"
        android:layout_marginBottom="@dimen/common_spacing"
        android:contentDescription="@string/description_post_author_avatar" />

    <TextView
        android:id="@+id/author"
```

… (аналогично предыдущей лекции)

ACTIVITY

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        val viewModel: PostViewModel by viewModels()
        viewModel.data.observe( owner: this) { posts ->
            posts.map { post ->
                CardPostBinding.inflate(layoutInflater, binding.container, attachToParent: false).apply {
                    author.text = post.author
                    published.text = post.published
                    content.text = post.content
                    like.setImageResource(
                        if (post.likedByMe) R.drawable.ic_liked_24 else R.drawable.ic_like_24
                    )
                    like.setOnClickListener { it: View!
                        viewModel.likeById(post.id)
                    }
                }.root
            }.forEach { it: ConstraintLayout
                binding.container.addView(it)
            }
        }
    }
}
```

это уже List<Post>

добавим View во View Activity позже

Обязательно указываем родительский layout, чтобы применились его LayoutParams

добавляем настроенное View во View Activity

ACTIVITY

Немного упростим

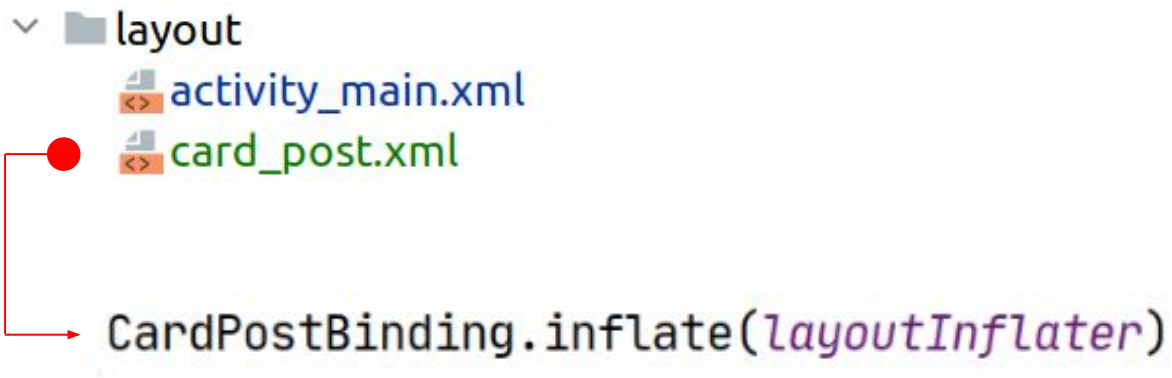
```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        val viewModel: PostViewModel by viewModels()
        viewModel.data.observe( owner: this) { posts ->
            posts.map { post ->
                CardPostBinding.inflate(layoutInflater, binding.container,  attachToParent: true).apply {
                    author.text = post.author
                    published.text = post.published
                    content.text = post.content
                    like.setImageResource(
                        if (post.likedByMe) R.drawable.ic_liked_24 else R.drawable.ic_like_24
                    )
                    like.setOnClickListener {  it: View!
                        viewModel.likeById(post.id)
                    }
                }.root
            }
        }
    }
}
```

добавим View во View Activity сразу

VIEW BINDING

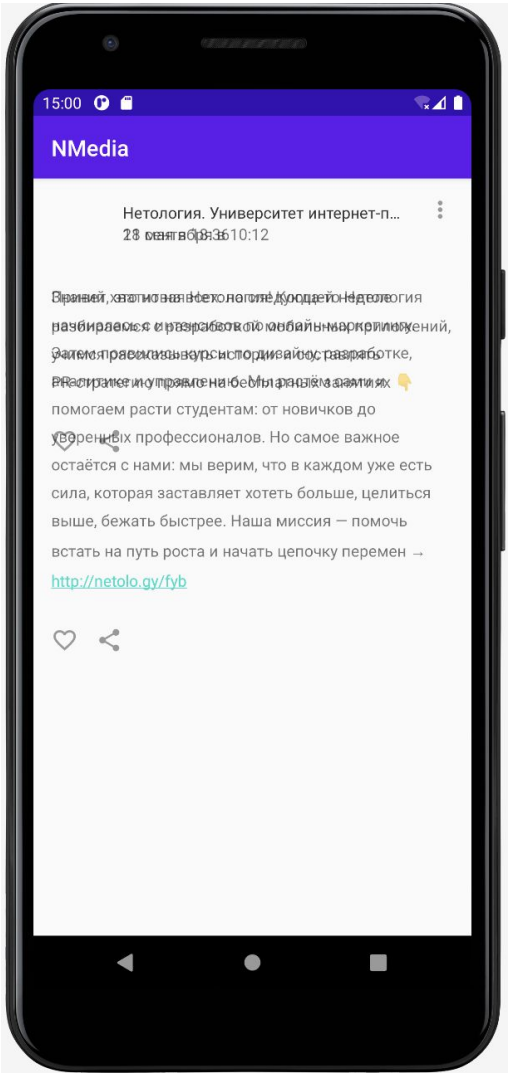
Обратите внимание: View Binding работает для всех layout'ов:



LAYOUT

Если запустим, то получим вот такую ужасную картину:

Как же понять, что происходит и почему?

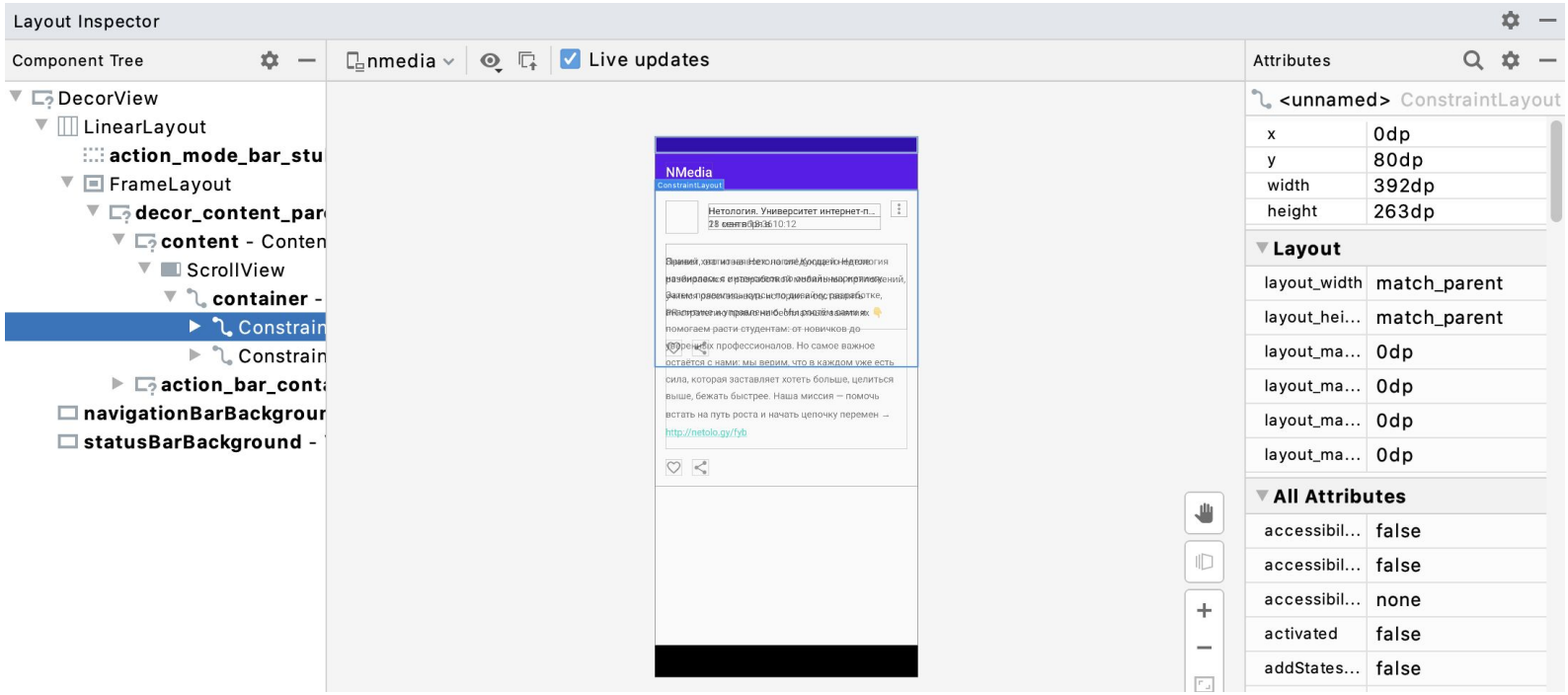


LAYOUT INSPECTOR

Для того, чтобы проанализировать, что действительно происходит, есть специальный инструмент: Layout Inspector (Tools -> Layout Inspector).

Он позволяет на работающем приложении посмотреть получившиеся View и их параметры (см. следующий слайд).

LAYOUT INSPECTOR



CONSTRAINT LAYOUT

Дело в том, что ConstraintLayout требует указания Constraint'ов, в противном случае всё «улетает» наверх и наши View накладываются друг на друга.

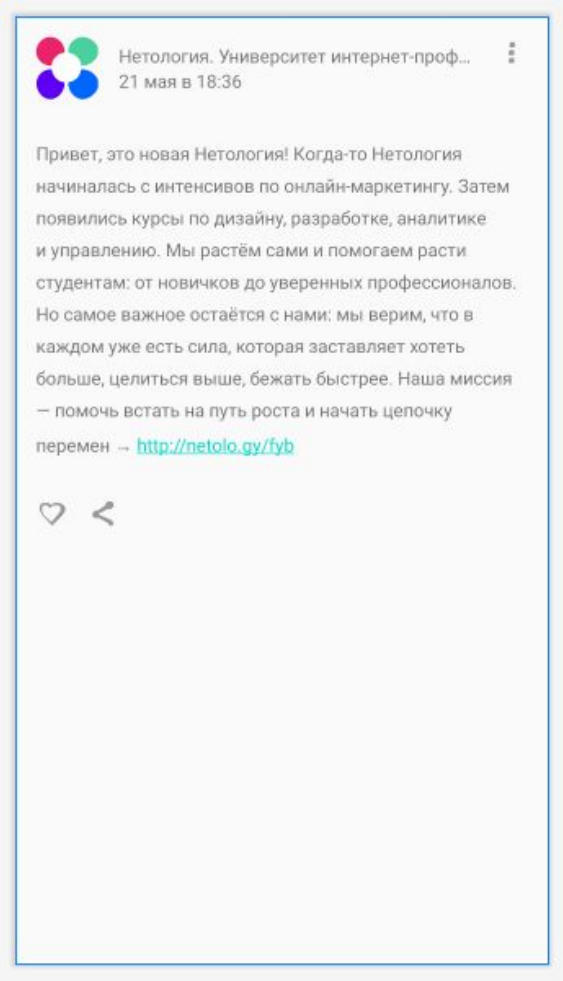
Самое простое решение — заменить родительский ConstraintLayout на более простой ViewGroup. Например, LinearLayout, который позволяет выстроить View вертикально или горизонтально.

LINEAR LAYOUT

Если заменим на LinearLayout, то на экране будет только один пост.

Всё дело в размерах нашей карточки. Если заменить match_parent на wrap_content, то получим нормальный вид:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    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:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="16dp">
```



LINEAR LAYOUT

Если мы попробуем запустить полученное решение, то вроде как ничего работать не будет.

На самом деле, причин несколько:

1. Скролл автоматически не добавляется (хотя можно завернуть всё в ScrollView).
2. Мы добавляем View каждый раз вниз, поэтому и не видим, что что-то происходит (это можно увидеть в Layout Inspector).

SCROLL VIEW

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView 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:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".activity.MainActivity">
    <LinearLayout
        android:id="@+id/container"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

    </LinearLayout>
</ScrollView>
```

ACTIVITY

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        val viewModel: PostViewModel by viewModels()
        viewModel.data.observe( owner: this) { posts ->
            binding.container.removeAllViews()
            posts.map { post ->
                CardPostBinding.inflate(layoutInflater, binding.container, attachToParent: true).apply {
                    author.text = post.author
                    published.text = post.published
                    content.text = post.content
                    like.setImageResource(
                        if (post.likedByMe) R.drawable.ic_liked_24 else R.drawable.ic_like_24
                    )
                    like.setOnClickListener { it: View!
                        viewModel.likeById(post.id)
                    }
                }.root
            }
        }
    }
}
```

ПРОБЛЕМЫ

Решение, полученное нами, может и рабочее, но не очень уж элегантное. Наверняка проблема, которую мы пытаемся решить, уже была решена.

Среди готовых компонентов вы можете найти ListView, предназначенный для отображения списков. Вы можете его использовать для небольших списков.
Для больших, таких как ленты новостей и т.д., рекомендуется использовать другой компонент — RecyclerView.

RECYCLERVIEW

RECYCLERVIEW

RecyclerView — этот способ позволяет отображать данные «лимитировано».

Q: Что значит лимитировано?

A: Если мы возьмём нашу предыдущую реализацию и представим, что в репозитории у нас будет 500 элементов, то мы под них создадим 500 View (имеется в виду View, полученный из card_post).

RECYCLERVIEW

RecyclerView же сможет обойтись одним или несколькими десятками, в зависимости от того, сколько их умещается за один просмотр на экране.

Т.е. ключевая идея: recycling (переиспользование) существующих View для отображения новых данных при скроллинге (поскольку у нас на экран умещается всего 2 карточки, нам не нужно сразу 500).

КЛЮЧЕВЫЕ ПОНЯТИЯ

- Adapter — класс, отвечающий за предоставление View, соответствующего конкретному элементу в наборе данных (посту в нашем случае).
- Position — позиция элемента в Adapter'е.
- Index — индекс View в Layout'е.
- ViewHolder — класс, содержащий информацию о визуальном отображении конкретного элемента списка.
- Binding — процесс подготовки View для отображения данных, соответствующих определённой позиции в адаптере.

RECYCLERVIEW

RecyclerView уже должен быть подключен в вашем build.gradle:

```
def recyclerview_version = "1.1.0"
implementation "androidx.recyclerview:recyclerview:$recyclerview_version"
```

RECYCLERVIEW

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout 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:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".activity.MainActivity">
    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/list"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"
        tools:listitem="@layout/card_post"
        />
</FrameLayout>
```

← как отображать элементы

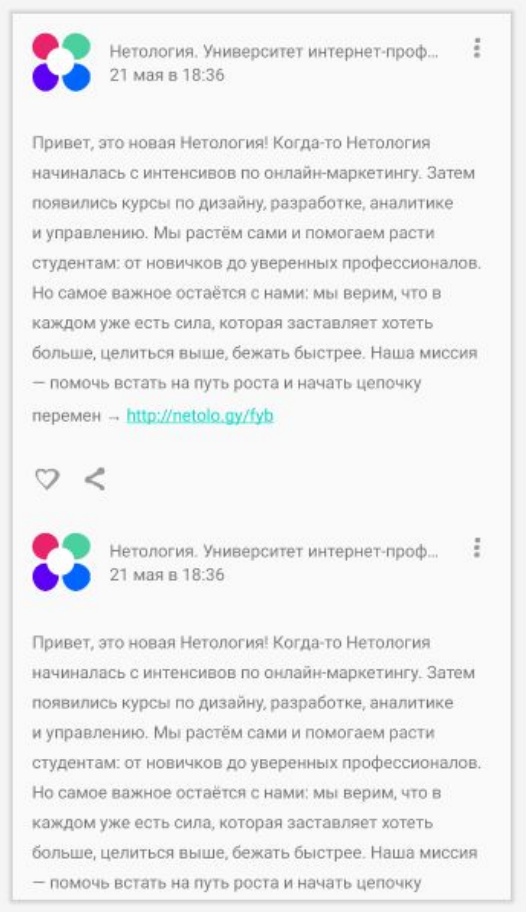
FrameLayout — простейший Layout. Поскольку мы не собираемся пока отображать ничего, кроме списка, вполне нам подходит.

TOOLS:LISTITEM

Как вы помните, tools нужен, чтобы отображать элементы в дизайнере:

```
tools:listitem="@layout/card_post"
```

С его помощью мы можем увидеть уже в режиме предпросмотра, как будет выглядеть наш список.



ACTIVITY

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        val viewModel: PostViewModel by viewModels()
        val adapter = PostsAdapter {  it: Post
            viewModel.likeById(it.id)
        }
        binding.list.adapter = adapter
        viewModel.data.observe( owner: this) { posts ->
            adapter.list = posts
        }
    }
}
```

callback

обновление данных

ADAPTER

```
typealias OnLikeListener = (post: Post) -> Unit
```

тип для callback'а

```
class PostsAdapter(private val onLikeListener: OnLikeListener) : RecyclerView.Adapter<PostViewHolder>() {
    var list = emptyList<Post>()
        set(value) {
            field = value
            notifyDataSetChanged()
        }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): PostViewHolder {
        val binding = CardPostBinding.inflate(LayoutInflater.from(parent.context), parent, attachToParent: false)
        return PostViewHolder(binding, onLikeListener)
    }

    override fun onBindViewHolder(holder: PostViewHolder, position: Int) {
        val post = list[position]
        holder.bind(post)
    }

    override fun getItemCount(): Int = list.size
}
```

данные

(notifyDataSetChanged уведомляет об изменении данных)

ADAPTER

Вызывая соответствующие методы (в нашем примере notifyDataSetChanged), мы оповещаем адаптер об изменении данных, что приводит к перерисовке списка.

ADAPTER

Методы оповещения адаптера об изменении данных:

- notifyItemChanged(int pos)
- notifyItemInserted(int pos)
- notifyItemRemoved(int pos)
- notifyItemMoved(int oldPos, int newPos)
- notifyItemRange* (несколько методов)
- notifyDataSetChanged()

T.e., вызывая notifyDataSetChanged, мы просим перерисовать все отображаемые элементы (что не очень-то логично, если мы нажали только на кнопку like и поменялся только один элемент).

VIEWHOLDER

```
class PostViewHolder(
    private val binding: CardPostBinding,
    private val onLikeListener: OnLikeListener
) : RecyclerView.ViewHolder(binding.root) {
    fun bind(post: Post) {
        binding.apply { this: CardPostBinding
            author.text = post.author
            published.text = post.published
            content.text = post.content
            like.setImageResource(
                if (post.likedByMe) R.drawable.ic_liked_24 else R.drawable.ic_like_24
            )
            like.setOnClickListener{ it: View!
                onLikeListener(post)
            }
        }
    }
}
```

Демонстрация

КЛЮЧЕВЫЕ МОМЕНТЫ

1. При увеличении количества элементов в списке количество создаваемых ViewHolder’ов будет меньше, чем размер списка.
2. В Java и Kotlin внутри интерфейсов и классов можно объявлять другие интерфейсы и классы. Доступ к именам подобных классов осуществляется через точку: RecyclerView.Adapter

Демонстрация

КЛЮЧЕВЫЕ МОМЕНТЫ

Важно: всегда в bind переназначайте все поля, а не только те, которые отличаются от значения по умолчанию (иначе получите «предыдущую» версию View).

Для примера:

```
like.setImageResource(
    if (post.likedByMe) R.drawable.ic_liked_24 else R.drawable.ic_like_24
)
```

ok

```
if (post.likedByMe) {
    like.setImageResource(R.drawable.ic_liked_24)
}
```

не ok!

Поставим лайк на первом посте и прокрутим вниз. Неожиданно лайки появятся и на нижних постах, хотя у них likedByMe = false.

CONTEXT

В Android одним из ключевых классов, является абстрактный класс Context, реализуют который все Activity.

Контекст предоставляет «программное окружение» приложения — т. е. работу с ресурсами, с системой и т.д.

CONTEXT

Один из примеров — LayoutInflater (класс, умеющий из layout делать View):

```
/**
 * Obtains the LayoutInflater from the given context.
 */
public static LayoutInflater from(Context context) {
    LayoutInflater LayoutInflater =
            (LayoutInflater) context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    if (LayoutInflater == null) {
        throw new AssertionError( detailMessage: "LayoutInflater not found.");
    }
    return LayoutInflater;
}
```

CONTEXT

Ключевое: некоторые операции возможны только с предоставлением объекта контекста, например:

1. Доступ к ресурсам.
2. Запуск других Activity.
3. И т.д.

CONTEXT

Q: Но почему раньше мы с этим не сталкивались?

A: Потому что мы работали внутри Activity , а Activity сама является контекстом. Adapter же контекстом не является, поэтому он через родительский ViewGroup (это RecyclerView) запрашивает контекст.

CONTEXT

View также нужен контекст, потому что именно он предоставляет информацию об окружении:

```
/**
 * Simple constructor to use when creating a view from code.
 *
 * @param context The Context the view is running in, through which it can
 *        access the current theme, resources, etc.
 */
public View(Context context) {
    mContext = context;
```

Соответственно, именно из родительского View мы и получаем ссылку на Context (им будет являться в данном случае Activity).

DIFFUTIL & LISTADAPTER

DIFFUTIL & LISTADAPTER

Как мы уже говорили, не совсем разумно вызывать notifyDataSetChanged, если у нас только у одного объекта изменилось одно поле.

Но и вручную высчитывать разницу между двумя списками тоже неинтересно.

DIFFUTIL & LISTADAPTER

В Android уже реализовали вспомогательные классы, которые только на основании сравнения двух элементов сделают всё остальное:

- DiffUtil — сравнивает два объекта.
- ListAdapter — реализация RecyclerView.Adapter, которая работает в связке с DiffUtil.

DIFFUTIL

```
class PostDiffCallback : DiffUtil.ItemCallback<Post>() {
    override fun areItemsTheSame(oldItem: Post, newItem: Post): Boolean {
        return oldItem.id == newItem.id
    }

    override fun areContentsTheSame(oldItem: Post, newItem: Post): Boolean {
        return oldItem == newItem
    }

}
```

Первая функция сравнивает id объектов (что это одни и те же сущности), вторая сравнивает, что их содержимое одинаково.

Пример: наш пост может быть в состоянии likedByMe = true или false. Если идентификаторы одинаковы — это одна и та же сущность, а контент разный, следовательно, мы обновляем View.

ADAPTER (БЫЛО)

```
typealias OnLikeListener = (post: Post) -> Unit

class PostsAdapter(private val onLikeListener: OnLikeListener) : RecyclerView.Adapter<PostViewHolder>() {
    var list = emptyList<Post>()
        set(value) {
            field = value
            notifyDataSetChanged()
        }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): PostViewHolder {
        val binding = CardPostBinding.inflate(LayoutInflater.from(parent.context), parent, attachToParent: false)
        return PostViewHolder(binding, onLikeListener)
    }

    override fun onBindViewHolder(holder: PostViewHolder, position: Int) {
        val post = list[position]
        holder.bind(post)
    }

    override fun getItemCount(): Int = list.size
}
```

LISTADAPTER (СТАЛО)

```
class PostsAdapter(
    private val onLikeListener: OnLikeListener
) : ListAdapter<Post, PostViewHolder>(PostDiffCallback()) {
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): PostViewHolder {
        val binding = CardPostBinding.inflate(LayoutInflater.from(parent.context), parent, attachToParent: false)
        return PostViewHolder(binding, onLikeListener)
    }

    override fun onBindViewHolder(holder: PostViewHolder, position: Int) {
        val post = getItem(position)
        holder.bind(post)
    }
}
```

Исчезло достаточно много кода.

ACTIVITY

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        val viewModel: PostViewModel by viewModels()
        val adapter = PostsAdapter {  it: Post
            viewModel.likeById(it.id)
        }
        binding.list.adapter = adapter
        viewModel.data.observe( owner: this) { posts ->
            adapter.submitList(posts)
        }
    }
}
```

SUBMITLIST

```
public void submitList(@Nullable final List<T> newList,
        @Nullable final Runnable commitCallback) {
    // incrementing generation means any currently-running diffs are discarded when they finish
    final int runGeneration = ++mMaxScheduledGeneration;

    if (newList == mList) {...}

    final List<T> previousList = mReadOnlyList;

    // fast simple remove all
    if (newList == null) {...}

    // fast simple first insert
    if (mList == null) {...}

    final List<T> oldList = mList;
    mConfig.getBackgroundThreadExecutor().execute(() -> {
            final DiffUtil.DiffResult result = DiffUtil.calculateDiff(new DiffUtil.Callback() {...});

            mMainThreadExecutor.execute(() -> {
                    if (mMaxScheduledGeneration == runGeneration) {
                        latchList(newList, result, commitCallback);
                    }
            });
    });
}
```

ИТОГИ

ИТОГИ

Сегодня мы обсудили вопросы отображения списков и даже реализовали эффективное отображение и обработку событий на базе RecyclerView.

Пока мы умеем только отображать готовый список и изменения в нём. Следующим шагом станет полноценный CRUD (Create, Read, Update, Delete) для нашего списка.