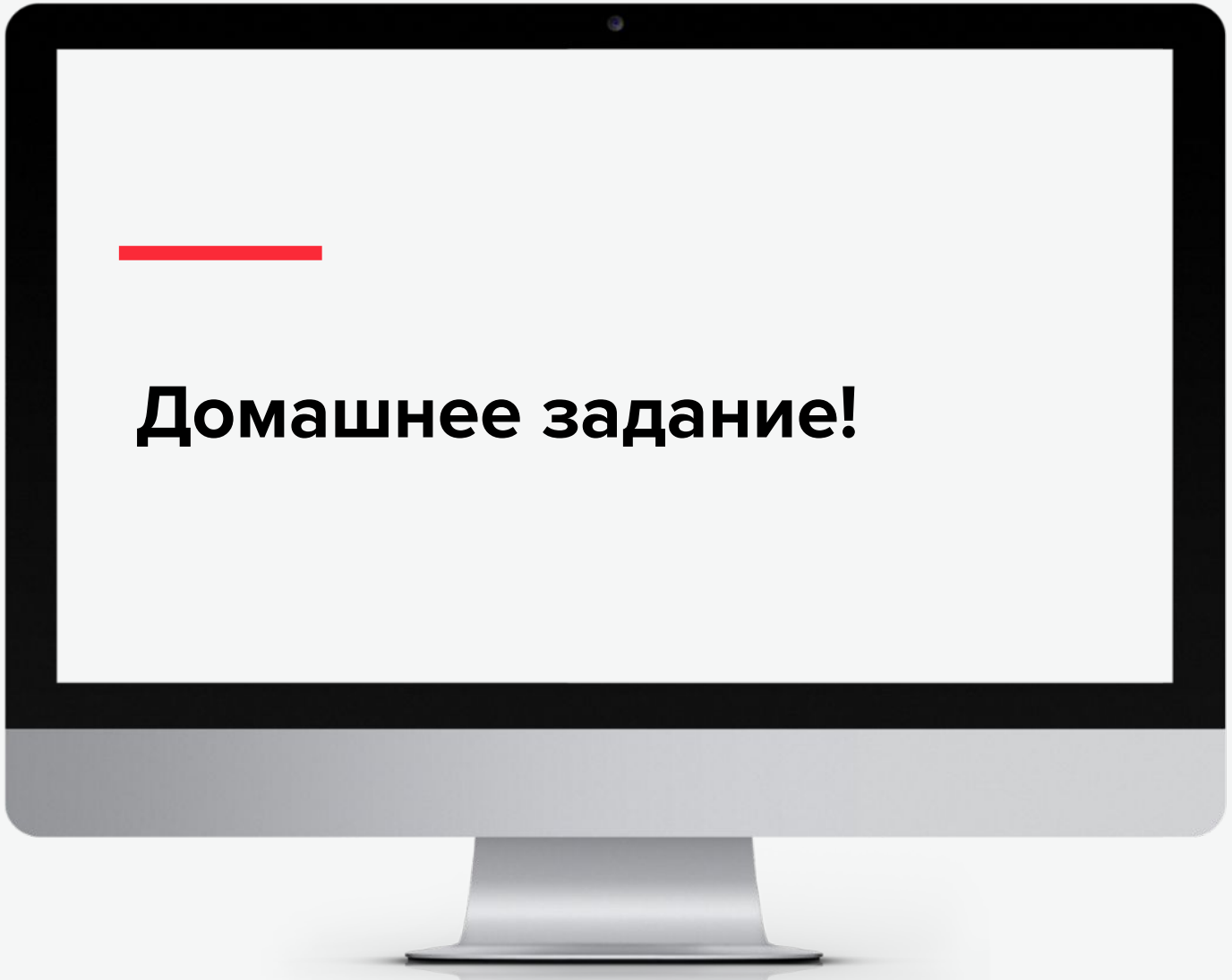
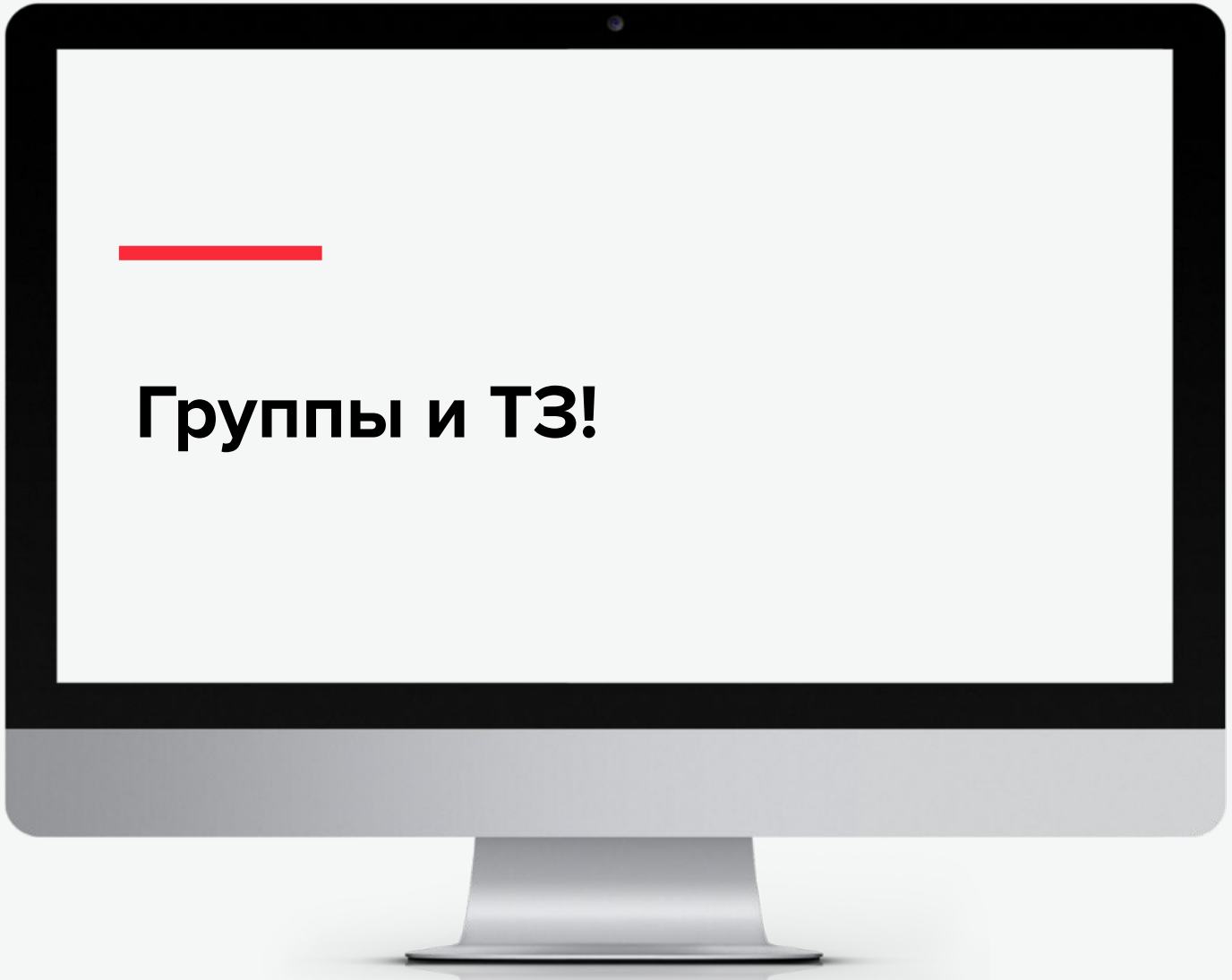
x mail.ru group

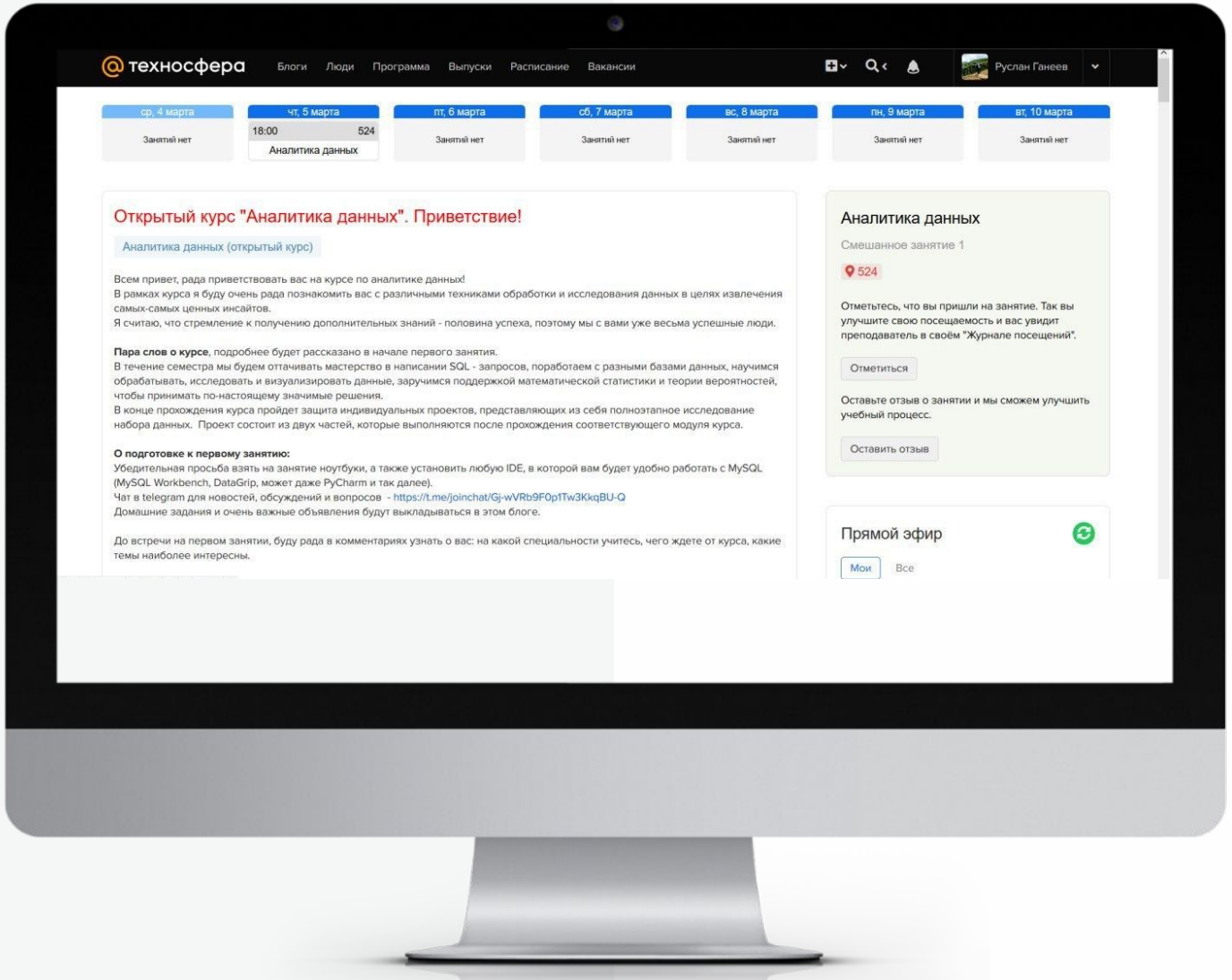
Фрагменты, Состояния и Жизненный цикл

Клещин Никита


Домашнее задание!


Группы и ТЗ!


Менторы!



Напоминание отметиться на портале

Кратко

Что помним?

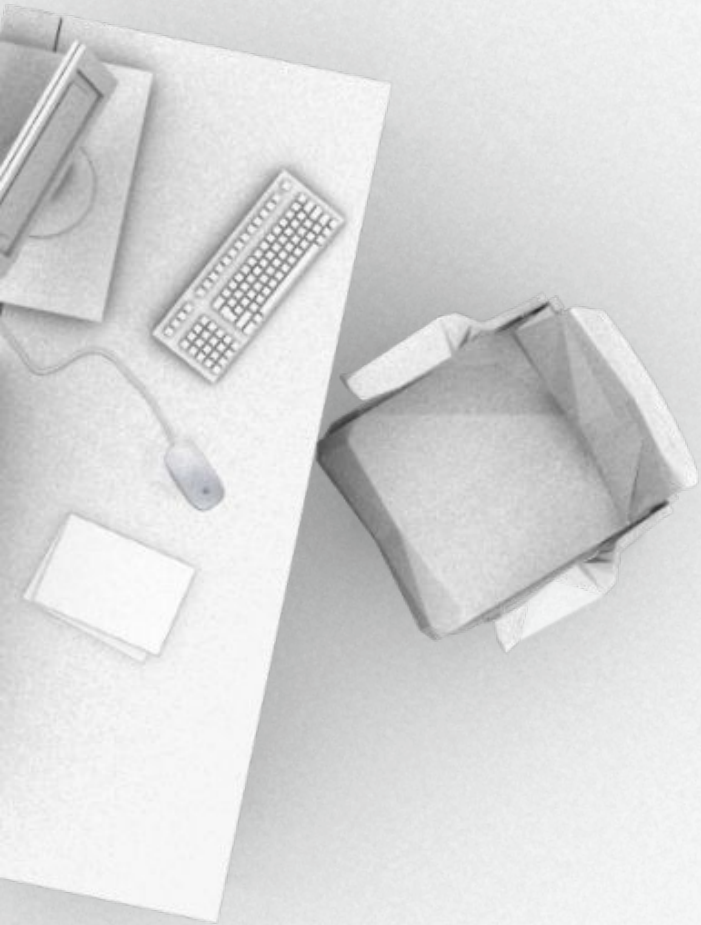
- Activity
- View/ViewGroup
- “Плотность” экрана?

Есть прогресс?

- Команды
- Проекты
- RecyclerView добили?

Содержание занятия

1. Fragment
2. Сохранение состояний
3. Работа в жизненном цикле
4. Code it!



Fragments

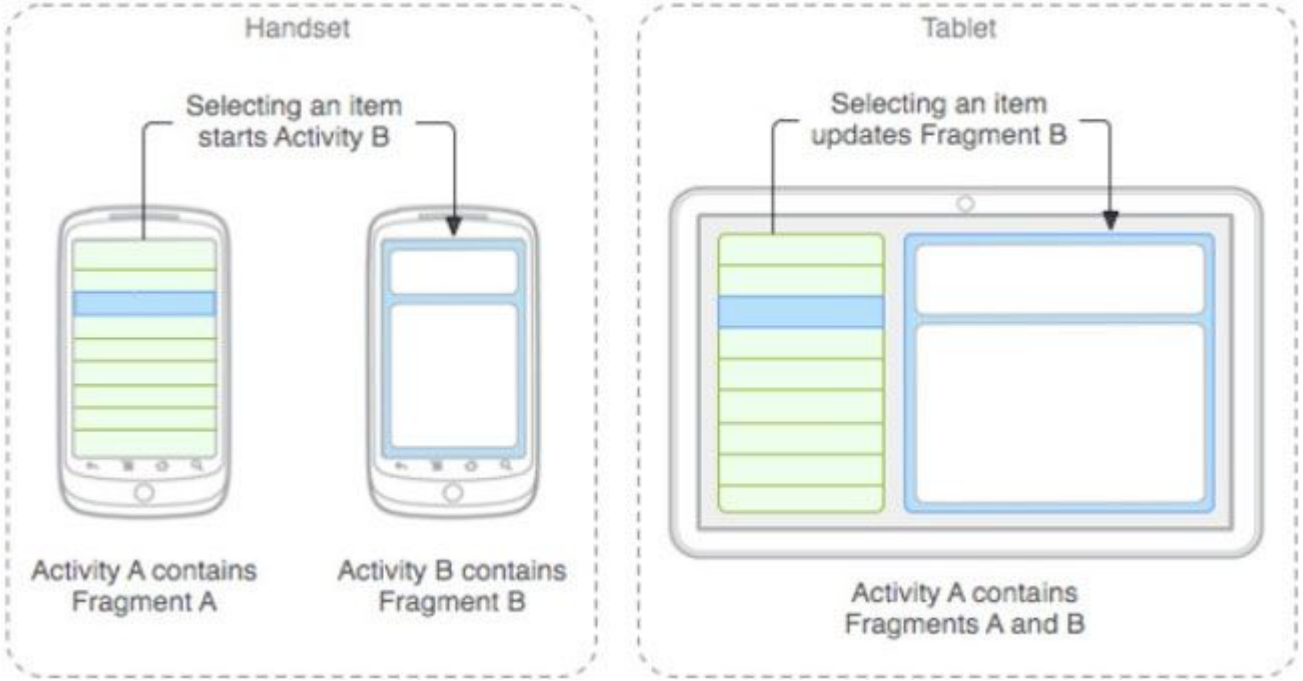
Недо-**activity** и пере-**view**

Когда и зачем?

- Появились в **API 11** (2011 год)
- Вложенные фрагменты появились в **API 17** (2013 год)
- Сейчас использовать надо фрагменты из пакета **androidx**

- Модульно и переиспользуемо **(?)**
- Умеет пересоздавать **View**

- Как и **Activity** - хранится в стэках



Как добавить фрагмент

В верстке

```
<fragment
    android:name="ru.hse.lection03.DroidListFragment"

    android:id="@+id/list"
    android:tag="TAG_LIST"

    android:layout_weight="1"
    android:layout_width="0dp"
    android:layout_height="match_parent"
    />
```

В коде

```
getSupportFragmentManager()
        .beginTransaction()
        .add(R.id.list, new DroidListFragment(), TAG_LIST)
        .commit();
```

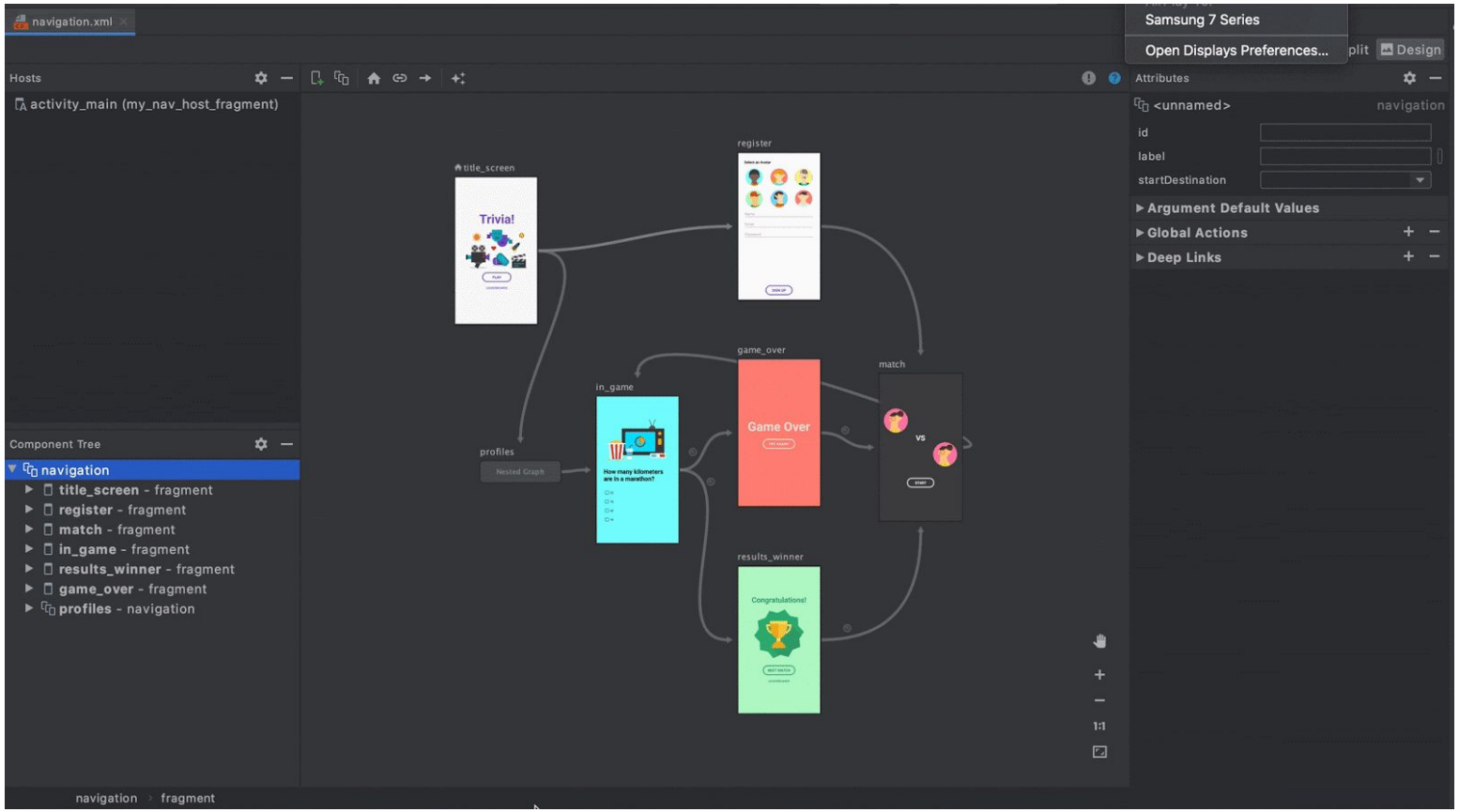
```
supportFragmentManager
        .beginTransaction()
        .add(R.id.list, DroidListFragment(), TAG_LIST)
        .commit()
```

Jetpack Navigation

Граф навигаций и свойств описывается в **xml**.

В коде надо будет дергать **navigateTo** у контроллера.

Но бывает что стреляет:(



Транзакция

FragmentManager - главный компонент для управления фрагментами.
FragmentTransaction - транзакция, для внесения изменения стэка фрагментов

```
.beginTransaction() // Создать транзакцию

    .add() // добавить
    .remove() // удалить
    .replace() // заменить

    .setTransition() // анимация переходов (из имеющихся)
    .setCustomAnimations() // анимация перехода (своя)
    .setSharedElement() // для анимации “перемещения” View

    .addToBackStack() // добавить запись в стэк

    .commitAllowingStateLoss() // commit(), закончить транзакцию
```

Стэк

- **FragmentManager** принадлежит **Activity**
- **FragmentManager** контролит стэк фрагментов

- Можно посмотреть элементы стэка:
  - **getFragments()**
  - **findFragmentByTag()**
  - **findFragmentById()**

- Работа с записями:
  - **addOnBackStackChangedListener()** - подписаться на изменения стэка
  - **getBackStackEntryCount()** - количество записей
  - **getBackStackEntryAt(index)** - взять запись по индексу
  - **popBackStack()** - убрать верхний элемент

Сами фрагменты

Основные

Fragment - самый обычный вариант. Все что описано применимо к нему

DialogFragment - для отображения диалогов (через метод **show**). Но так же умеет все то, что и **Fragment**

Специализированные (не видел чтобы использовали их)

ListFragment - заточен под **ListView**

PreferenceFragment - заточен под **<PreferenceScreen>**

Еще был такой **WebViewFragment** - работал с **WebView**

*Для работы с **FragmentManager** нужен **FragmentActivity**

Как создать фрагмент

Информацию для инициализации в фрагмент можно передать при помощи метода **setArguments(Bundle)**

```java
public static DroidDetailsFragment newInstance(Droid droid) {
    final Bundle extras = new Bundle();
    extras.putSerializable(EXTRAS_DROID, droid);

    final DroidDetailsFragment fragment = new DroidDetailsFragment();
    fragment.setArguments(extras);

    return fragment;
}
```

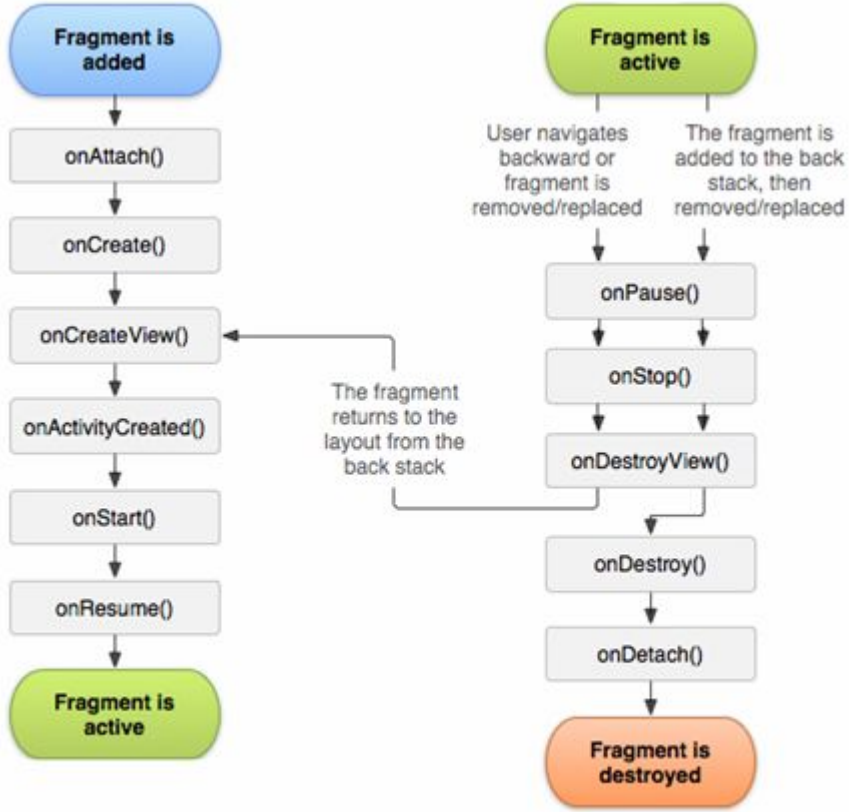
```kotlin
fun newInstance(droid: Droid): DroidDetailsFragment {
    val extras = Bundle().apply {
        putSerializable(EXTRAS_DROID, droid)
    }

    val fragment = DroidDetailsFragment().apply {
        arguments = extras
    }

    return fragment
}
```

И снова жизненный цикл

- **onAttach()** - присоединен к **Activity**
- **onCreate()** - инициализация
- **onCreateView()** - инициализации **View**
- **onViewCreated()** - когда **View** инициализировано
- **onActivityCreated()** - у **Activity** отработал **onCreate()**

- **onStart(), onResume(), onPause(), onStop(), onSaveInstanceState()** - то же что и у **Activity**

- **onDestroyView()** - очистить ссылки на **View**
- **onDestroy()** - Уничтожить все и очиститься
- **onDetach()** - отсоединился от **Activity**



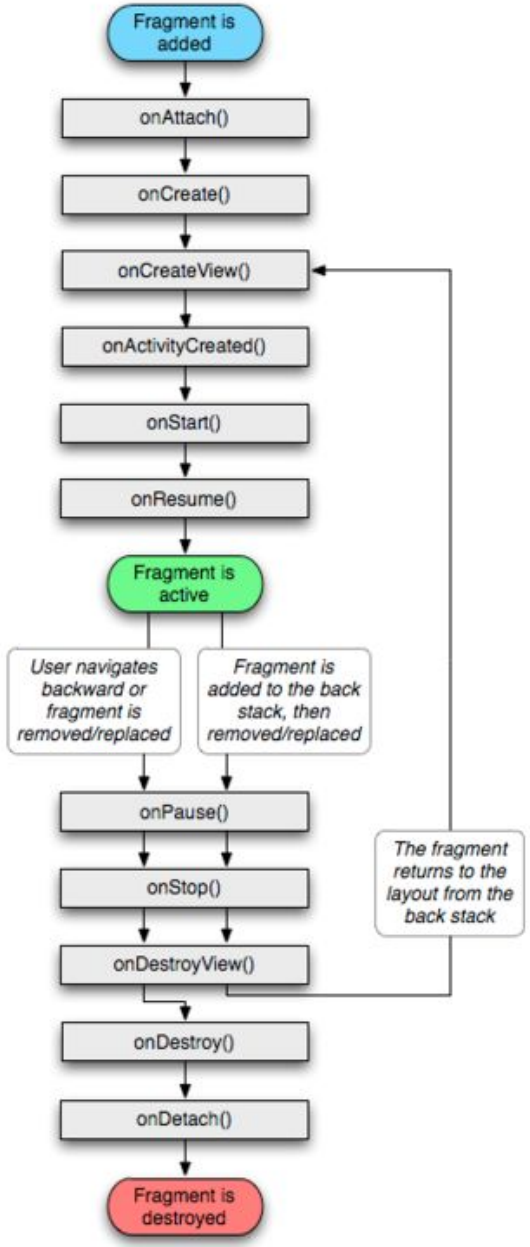
Связь с Activity

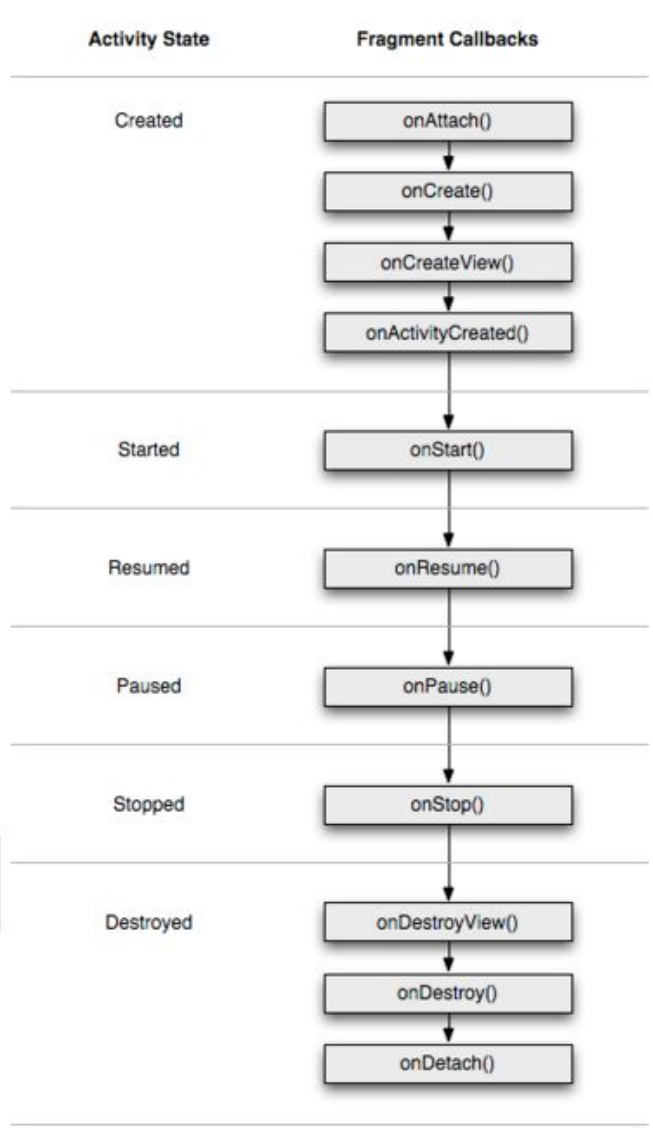
Из фрагмента можно получить **Activity**:

- **getActivity()**
- **requireActivity()**

Если фрагмент вложенный, то и на родительский фрагмент:

- **getParentFragment()**
- **requireParentFragment()**





Состояния

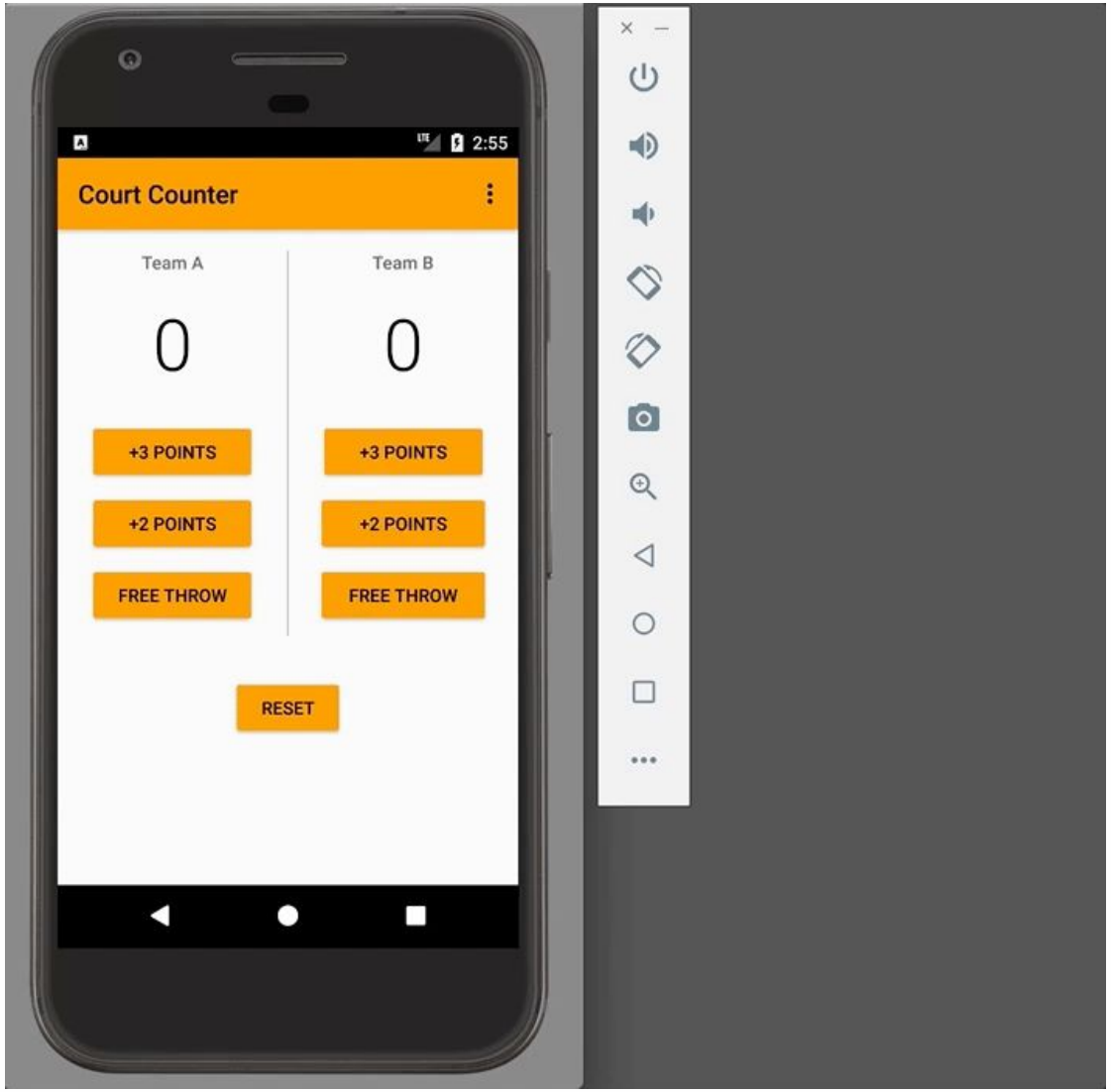
Чтобы ни случилось, не теряемся

А в чем проблема?

Система предоставляет базовый механизм сохранения состояния только для **View** (при условии что у них есть **id**).

Если данные лежат только в памяти, то со смертью процесса - они будут потеряны.

Если данные лежат только в Activity или Fragment, то с их уничтожением они тоже пропадают (они могут быть уничтожены без **onDestroy**)



Варианты состояний

	ViewModel	Saved instance state	Persistent storage
Storage location	in memory	serialized to disk	on disk or network
Survives configuration change	Yes	Yes	Yes
Survives system-initiated process death	No	Yes	Yes
Survives user complete activity dismissal/onFinish()	No	No	Yes
Data limitations	complex objects are fine, but space is limited by available memory	only for primitive types and simple, small objects such as String	only limited by disk space or cost / time of retrieval from the network resource
Read/write time	quick (memory access only)	slow (requires serialization/deserialization and disk access)	slow (requires disk access or network transaction)

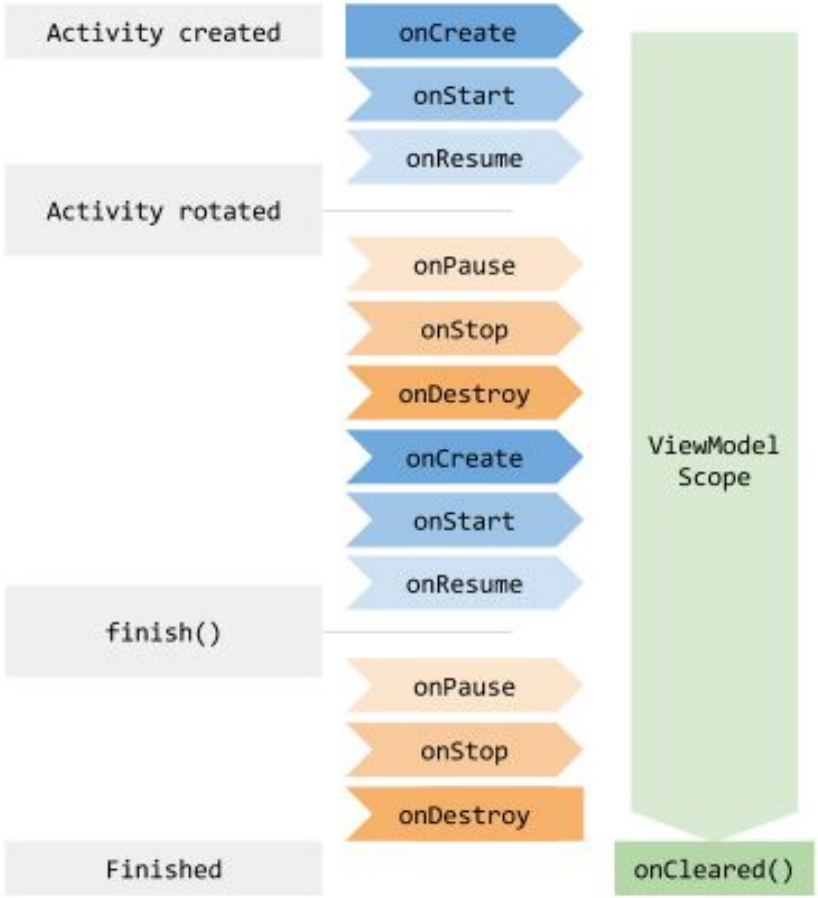
ViewModel

С подачи гугл, в Android пришел удобный **MVVM**

Это не часть **Android**, это отдельная библиотека из **Jetpack**.

Если используете фрагменты в рамках одной **Activity**, то эта **ViewModel** может быть “пошарена” между ними.

Главное, не делайте в ней ссылки на **View**:)



ViewModel в коде

```java
public class MyViewModel extends ViewModel {
    public LiveData<List<User>> getUsers() {
        ...
    }
}

public class MyActivity extends AppCompatActivity {
    public void onCreate(Bundle savedInstanceState) {
        MyViewModel model = new ViewModelProvider(this).get(MyViewModel.class);

        model.getUsers().observe(this, users -> {
            // update UI
        });
    }
}
```

```kotlin
class MyViewModel: ViewModel() {
    fun getUsers(): LiveData<List<User>> {
        ...
    }
}

class MyActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        val model: MyViewModel by viewModels()

        model.getUsers().observe(this, Observer<List<User>> { users ->
            // update UI
        })
    }
}
```

onSaveInstanceState() -> onRestoreInstanceState()

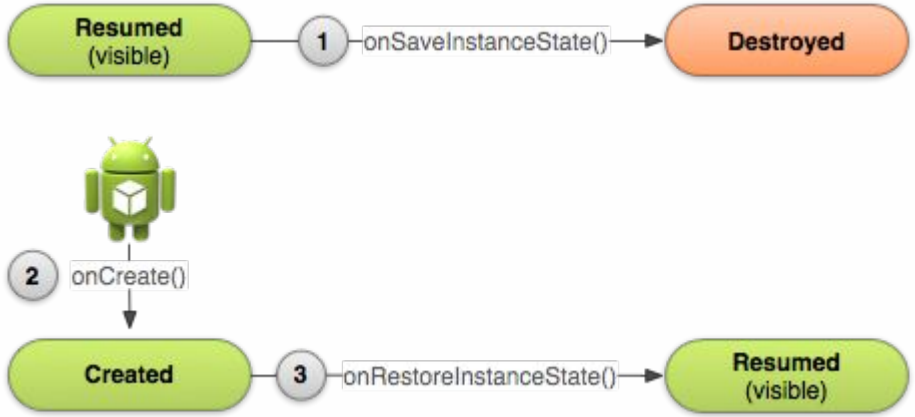
Есть и у **Activity** и у **Fragment**. Хранит состояние **View**.

У объекта **Bundle**, достаточно удобная спецификация.

Сериализуем, поэтому может пережить смерть приложения (если система его в фоне убила).

Сильные ограничения на хранение:

- Непонятно от чего зависит максимальный объем **Bundle** (по опыту - от 500kb до 1Mb, но говорят разное)
- Хранит только примитивные типы, **Serializable** и **Parcelable**



InstanceState в коде

```java
@Override
public void onSaveInstanceState(@NonNull Bundle outState) {
    super.onSaveInstanceState(outState);

    outState.putString(KEY_DROID_ID, mDroidId);
    outState.putFloat(KEY_OFFSET, mListOffset);
}

@Override
public void <MethodWithState>(@Nullable Bundle savedInstanceState) {
    super.<MethodWithState>(savedInstanceState);

    if (savedInstanceState != null) {
        mDroidId = savedInstanceState.getString(KEY_DROID_ID);
        mListOffset = savedInstanceState.getFloat(KEY_OFFSET);
    }
}
```

```kotlin
override fun onSaveInstanceState(outState: Bundle) {
    super.onSaveInstanceState(outState)

    outState.putString(KEY_DROID_ID, droidId)
    outState.putFloat(KEY_OFFSET, listOffset)
}

override fun <MethodWithState>(savedInstanceState: Bundle?) {
    super.<MethodWithState>(savedInstanceState)

    if (savedInstanceState != null) {
        droidId = savedInstanceState.getString(KEY_DROID_ID)
        listOffset = savedInstanceState.getFloat(KEY_OFFSET)
    }
}
```

Persistent Storage

Хранить можно любой объем данных

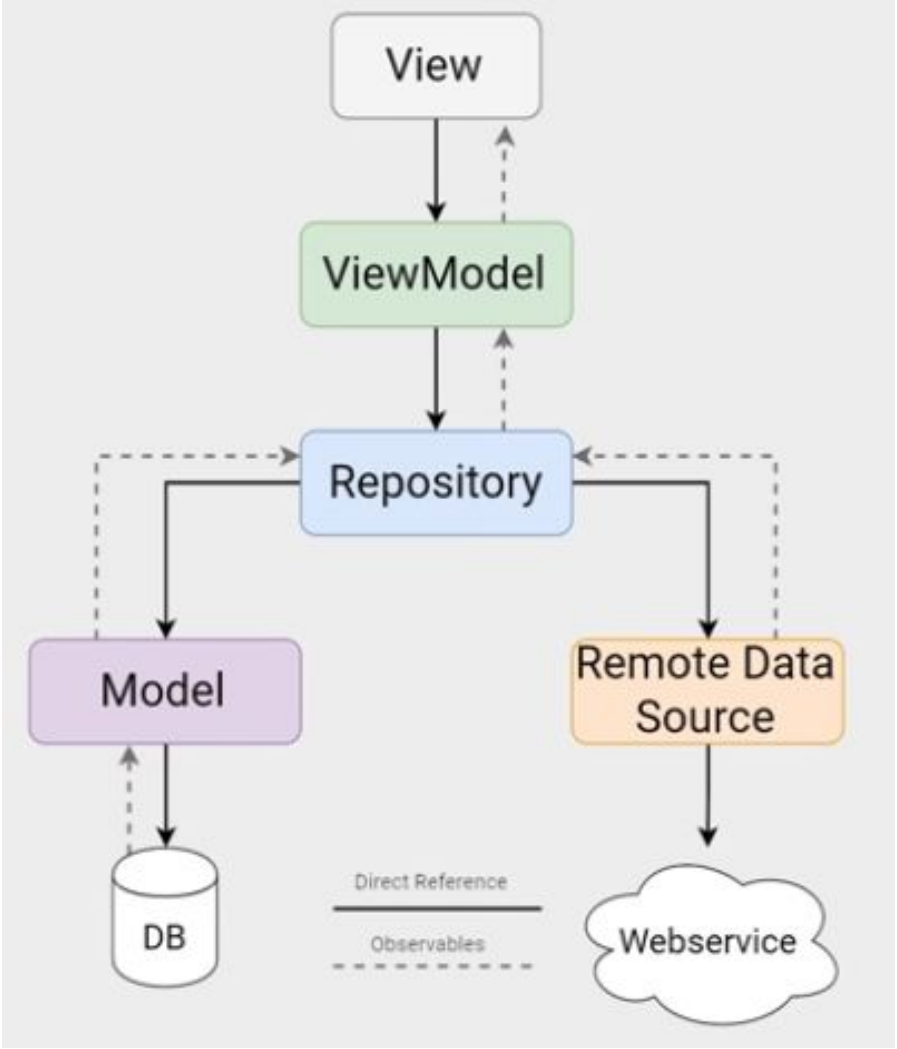
Следует учитывать - время доступа к данным, и возможные проблемы (сеть).

Реализация - разработчик сам решает как это работает, и при помощи каких библиотек

Много готовых фреймворков:

- **SharedPreferences** - “настройки”
- **Jetpack Room** - Обертка над SQLite
- **Retrofit** - для походов в сеть
- ... и т.д.

*Но не надо использовать для состояний **View**



Давайте обсудим?

Что бы вы положили **onSaveInstanceState()**?

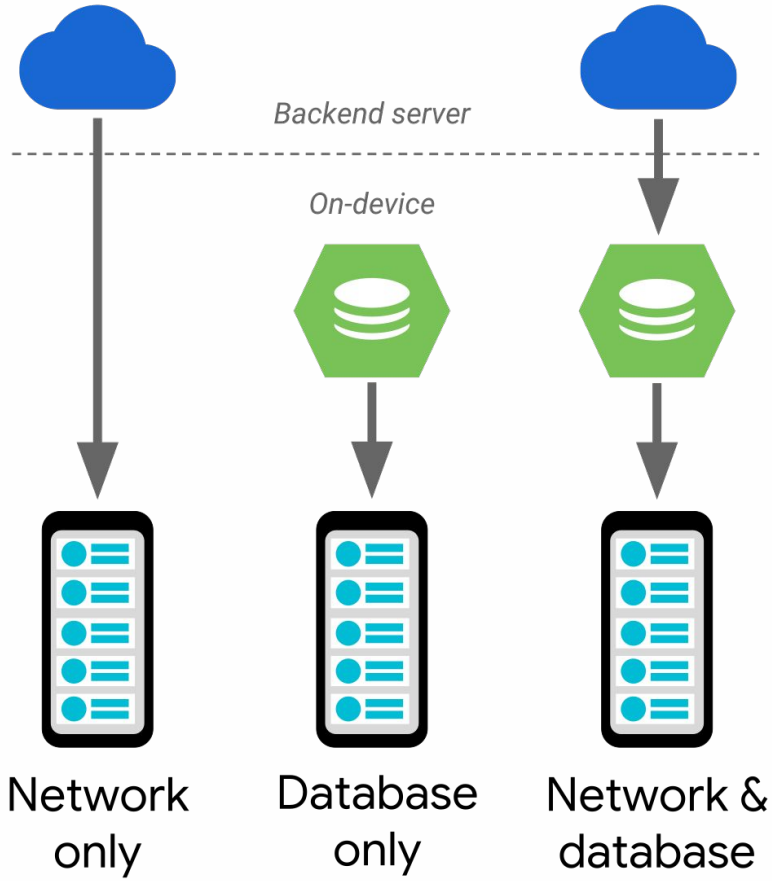
Для чего использовать **Persistent Storage**?

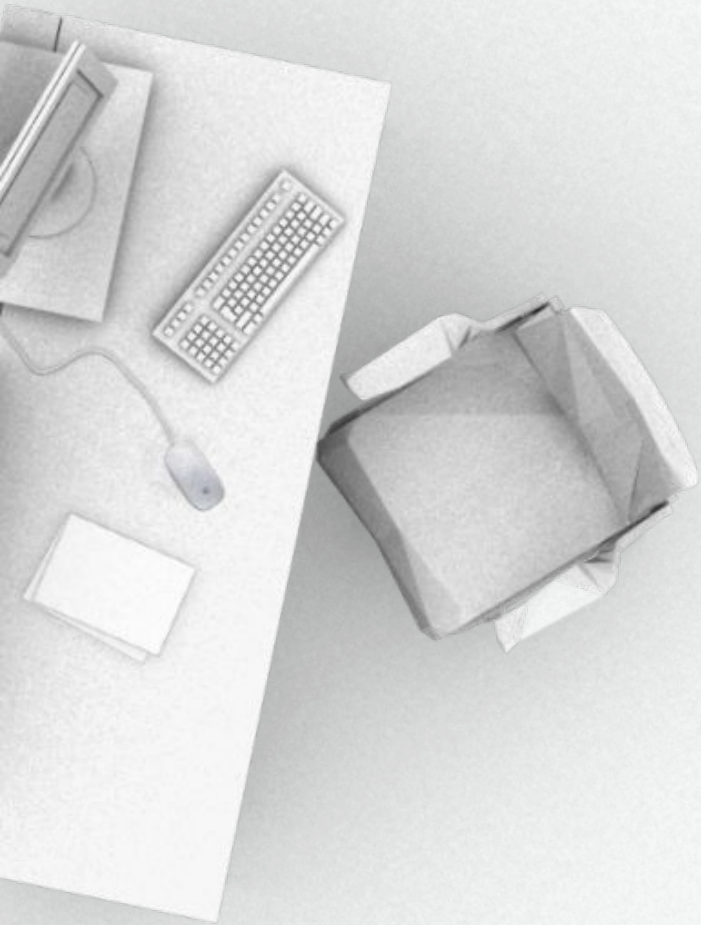
- Стоит помнить про обработку ошибок
- Не забывать про время обработки запросов

Если используете ViewModel, чем она лучше **onSaveInstanceState()**?

Кстати, мы не обсудили хранение данных в глобальных **Singleton**

- **Singleton Pattern** - это не про вид инициализации





Работа с жизненным циклом

Или проблемы

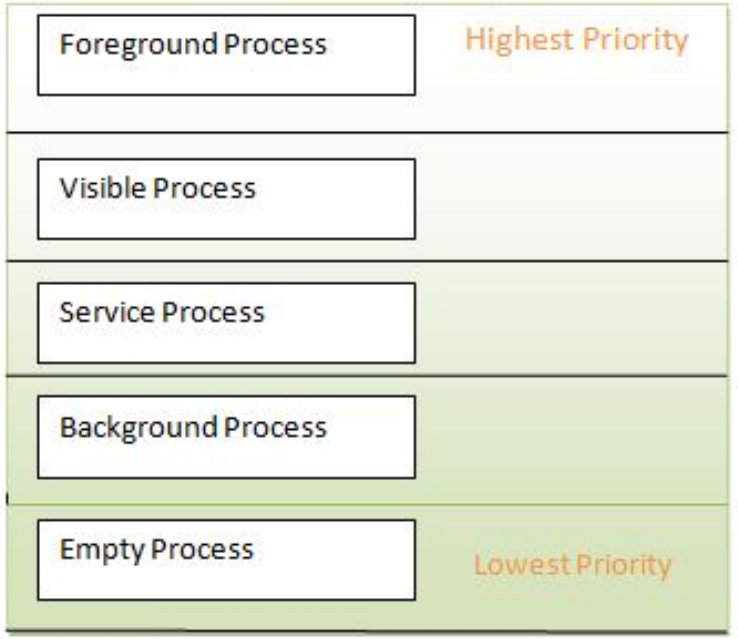
Когда происходит onSaveInstanceState

С **Api 28** - исполняется после **onStop()**

До **Api 28** - исполняется до **onStop()**, но нет гарантии исполнения ДО или ПОСЛЕ **onPause()**

Метод обязательно будет вызван до перехода вашей видимой части приложения в фон. Т.к. система может убить ваше приложение, если оно в фоне.

Google не рекомендует использовать вызов этого метода, как сигнал для сохранения “постоянных” данных. Лучше для этого использовать **onPause()**



Android Process State....

Внезапно! onConfigurationChanged

Этот метод есть и у **Activity** и у **Fragment**.

Метод вызовется, если условие срабатывания указано в **Manifest**.

Иначе система пересоздаст вашу **Activity** проведя ее через **onDestroy()** и **onCreate()**

Регистрация условия означает что ответственность за обработку вы берете на себя.

```
<activity
    android:name=".MainActivity"
    android:configChanges="orientation|keyboard|keyboardHidden"
    >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Динамические Fragment-ы. Когда можно?:)

Можно делать изменению стека фрагментов до вызова метода **onSaveInstanceState**. Иначе состояние стека не сохранится (а если используется метод **commit()**, то вылетит ошибка).

Так же, **commit()/commitAllowingStateLoss()** - это асинхронное изменение стэка. Можно использовать когда у вас может происходить несколько транзакций.

Можно попробовать глянуть также на методы **commitNow()/commitNowAllowingStateLoss()**

```
getSupportFragmentManager()
        .beginTransaction()
        .replace(R.id.details, detailsFragment, TAG_DETAILS)
//          .commit()
        .commitAllowingStateLoss();
```

```
supportFragmentManager
        .beginTransaction()
        .replace(R.id.details, detailsFragment, TAG_DETAILS)
//          .commit()
        .commitAllowingStateLoss()
```

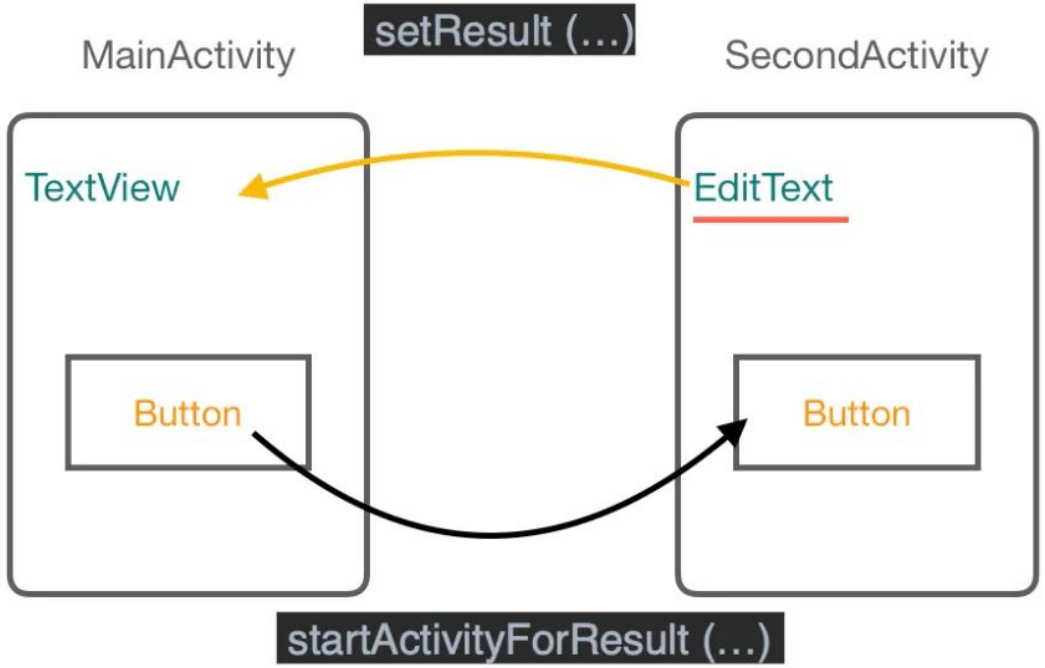
Activity умеет возвращать результат исполнения

Запустить **Activity** из кода можно двумя методами (Методы Context или его наследников):

- **startActivity(Intent)** - обычный запуск
- **startActivityForResult(Intent, code)** - запустить Activity с дальнейше обработкой ее закрытия

Установить данные для возврата, после закрытия **Activity** - **setResult(status, <data>)**

Обработка результата в методе **onActivityResult(code, status, data)**

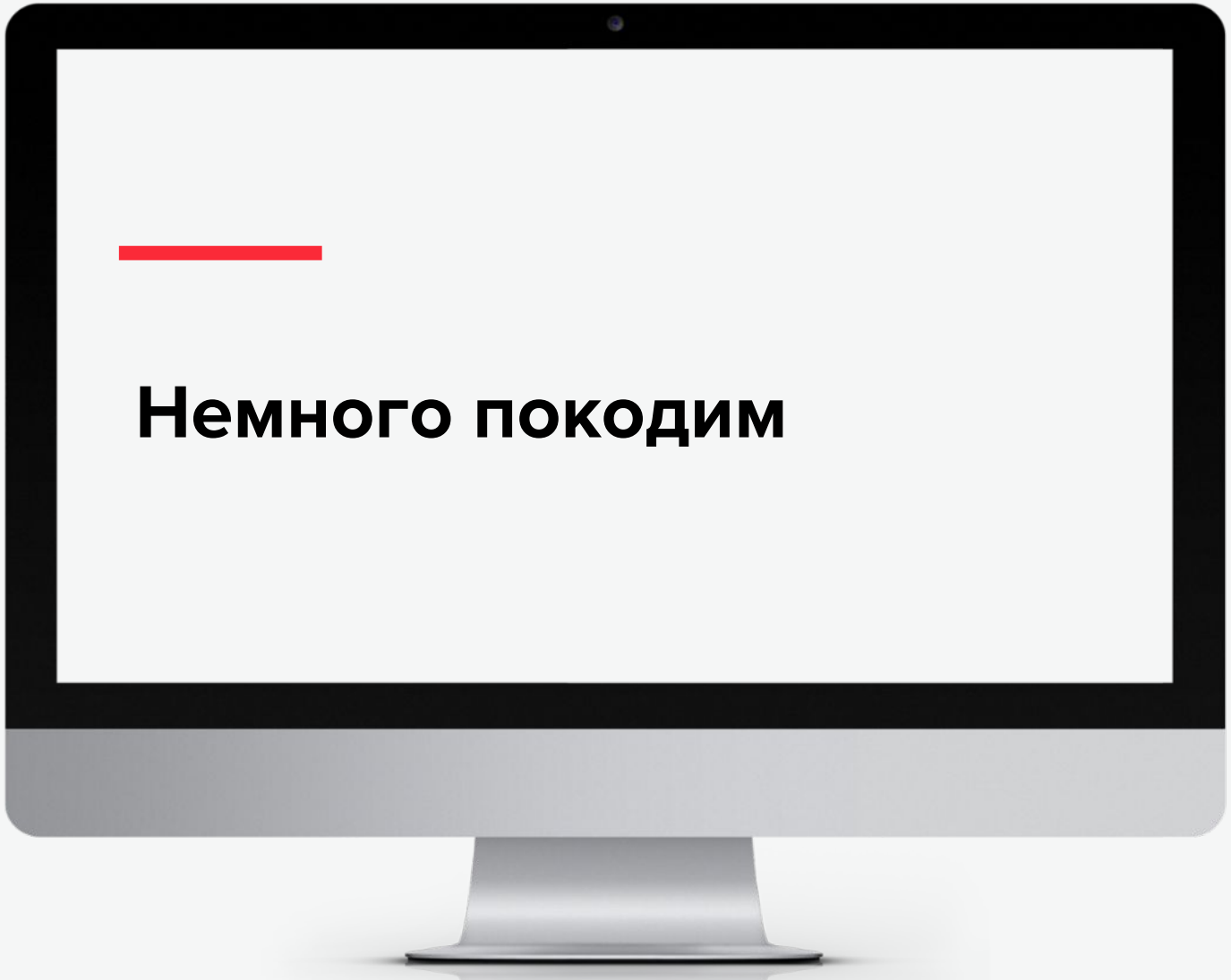


Жизненный цикл и работа с данными?

Просто напомню методы:

- **onCreate(Bundle savedState)** - Инициализируемся
- **onStart()** - Подготовка к отображению. Можно продолжить загрузку или обновление
- **onResume()** - Можно взаимодействовать
- **onPause()** - Потеряли фокус
- **onStop()** - Невидимы для пользователя. Можно синкать данные с хранилищем
- **onDestroy()** - Полное уничтожение. Данные для этой **Activity** более не нужны

Если у вас долгие операции по сохранению или загрузке данных и данные, и они мало привязаны к экрану, то скорее всего для этих операций надо будет использовать компонент **Service**.


Немного покодим

Постановка задачи

Требование

- Отображать список Дроидов. По клику на Дроида отображать его данные.

Требования к верстке

- Если устройство в вертикальном состоянии - отображать информацию о Дроидах на новом экране (или в диалоге);
- Если устройство в горизонтальном режиме - отображать список и информацию о дроидах на одном экране, в двух панельках. Отношение панелек - 1:2.

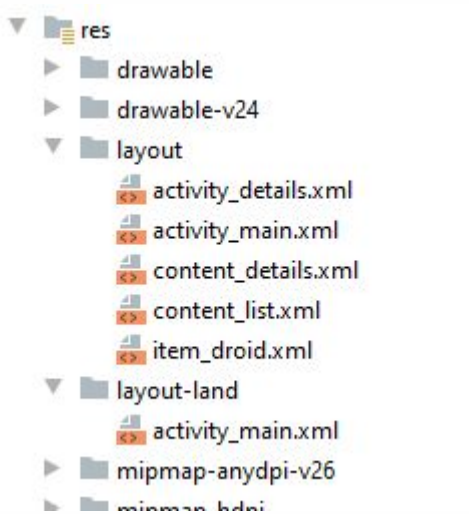
Шаг 1 - Декомпозиция

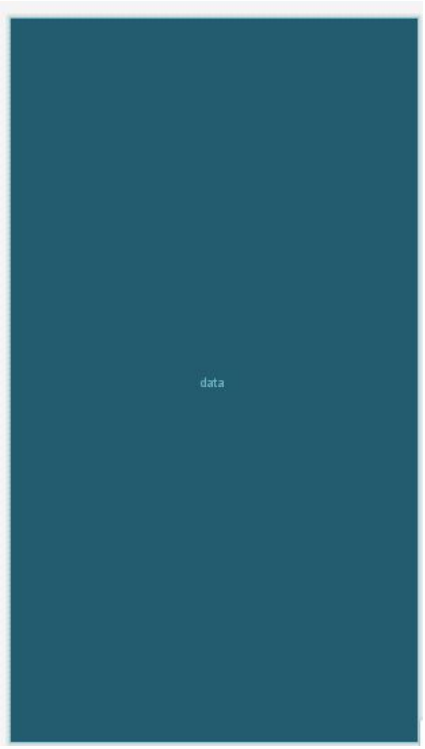
Что от нас хотят?

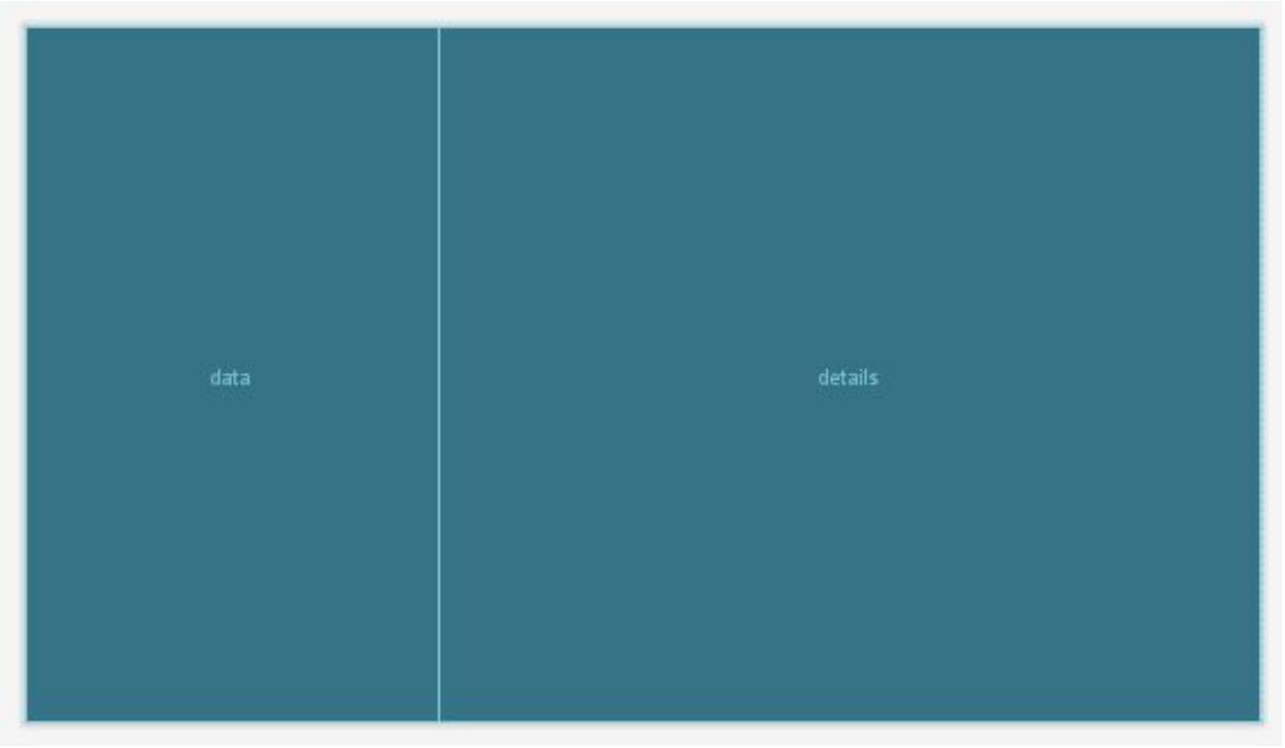
- Отображать список дроидов:
  - Если у нас вертикальный режим - отображать как экран
  - Если у нас горизонтальный режим - отображать в левой панеле
- Отображать информацию о дроиде:
  - Если у нас вертикальный режим - отображать в диалоге
  - Если у нас горизонтальный режим - отображать в правой панеле
- Клик по дроиду - отображает информацию о нем
- Должно быть понятным, что данные при перевороте не должны измениться
- В текущем примере не рассматриваем ситуацию - когда надо сохранить данные в случае смерти приложения.

Шаг 2 - Решаем задачу по частям

Можно начать с простого, в данном случае - накинуть верстку.







Шаг 2.1 - Можно сразу сверстать

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"

    android:id="@+id/item"
    android:layout_width="match_parent"
    android:layout_height="56dp"

    android:background="@color/droid_item_background"

    tools:context=".presentationlayer.MainActivity"
    >
    <ImageView
        android:id="@+id/image"
        android:layout_width="56dp"
        android:layout_height="match_parent"

        tools:src="@color/color_black"
        />
    <TextView
        android:id="@+id/name"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_toEndOf="@id/image"

        android:gravity="center_vertical"
        android:padding="8dp"

        tools:text="Random Name"
        />
</RelativeLayout>
```

```
<androidx.recyclerview.widget.RecyclerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"

    android:id="@+id/recycler"
    android:layout_width="match_parent"
    android:layout_height="match_parent"

    tools:context=".presentationlayer.MainActivity"
    />
```

```
<androidx.appcompat.widget.LinearLayoutCompat
    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"
    android:minHeight="300dp"
    android:minWidth="200dp"

    android:orientation="vertical"
    android:gravity="center"
    android:background="@color/droid_item_background"

    tools:context=".presentationlayer.MainActivity">
    <TextView
        android:id="@+id/name"
        android:layout_width="100dp"
        android:layout_height="56dp"

        android:gravity="center"

        tools:text="RandomName"
        />
    <TextView
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"

        android:text="@string/caption_droid_state"
        />
    <TextView
        android:id="@+id/state"
        android:layout_width="100dp"
        android:layout_height="56dp"

        android:gravity="center"

        tools:text="@string/caption_droid_state_unknown"
        />
</androidx.appcompat.widget.LinearLayoutCompat>
```

Шаг 3 - Данные

Поскольку, рассматривали хранение и обращали внимание на **Repository**, то код для генерации, хранения данных вынесем в отдельный класс - **DroidRepository**.

Для простоты, в рамках этого примера, сделаем его **Singleton**-ом.

*Можно взять объект Droid с прошлой лекции, и метод для генерации списка

```java
public class DroidRepository {
    // Объекты для реализации хардкорного синглтона в java
    private static volatile DroidRepository mInstance;

    public static DroidRepository getInstance() {
        if (mInstance == null) {
            synchronized (DroidRepository.class) {
                if (mInstance == null) {
                    mInstance = new DroidRepository();
                }
            }
        }
        return mInstance;
    }


    protected final List<Droid> mData;


    private DroidRepository() {
        mData = initializeData();
    }

    public List<Droid> list() {
        return mData;
    }

    public Droid item(int index) {
        return mData.get(index);
    }

    protected List<Droid> initializeData() {...}
}
```

```kotlin
class DroidRepository private constructor() {
    companion object {
        // простенький singleton
        val instance by lazy { DroidRepository() }

        ...
    }


    protected val droidList by lazy { initializeData() }


    fun list() = droidList

    fun item(index: Int) = droidList[index]

    protected fun initializeData(): List<Droid> { ... }
}
```

Шаг 4.1 - Сделаем отображение списка

*Если есть код с прошлой лекции - то можем перенести все объекты оттуда, но надо будет кинуть эти объекты на фрагмент.

```java
public class DroidListFragment extends Fragment {
    ...

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater
        , @Nullable ViewGroup container
        , @Nullable Bundle savedInstanceState) {

        return inflater.inflate(R.layout.content_list, container, false);
    }

    @Override
    public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);

        final RecyclerView recycler = view.findViewById(R.id.recycler);
        recycler.setAdapter(new DroidAdapter(DroidRepository.getInstance().list()));
        recycler.setLayoutManager(new LinearLayoutManager(requireContext()));
    }

    ...
}
```

```kotlin
class DroidListFragment : Fragment() {
    ...

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

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        val recycler = view.findViewById<RecyclerView>(R.id.recycler)
        recycler.apply {
            adapter = DroidAdapter(DroidRepository.instance.list())
            layoutManager = LinearLayoutManager(context)
        }
    }

    ...
}
```

Шаг 4.2 - Отображение информации

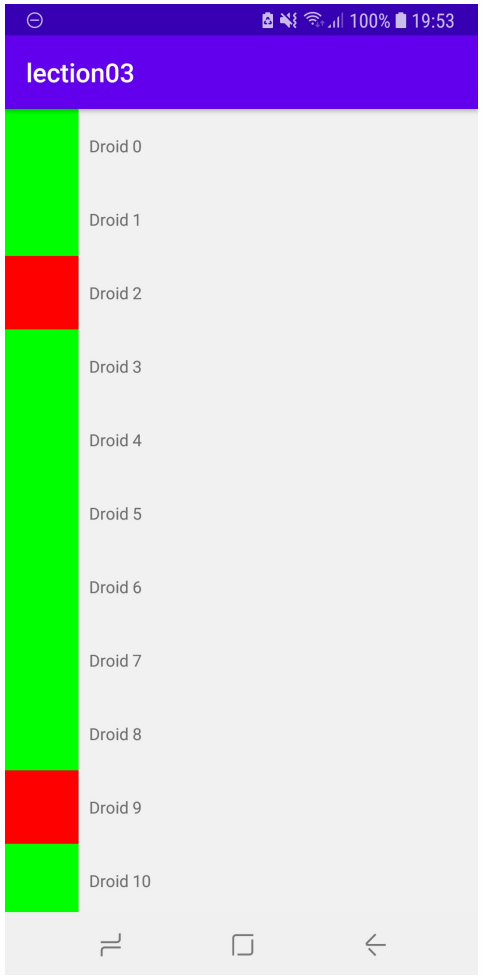
На слайд этот код не поместится. Смотрим в гитхаб:(

https://github.com/mailru-android-edu/hse-android-samples

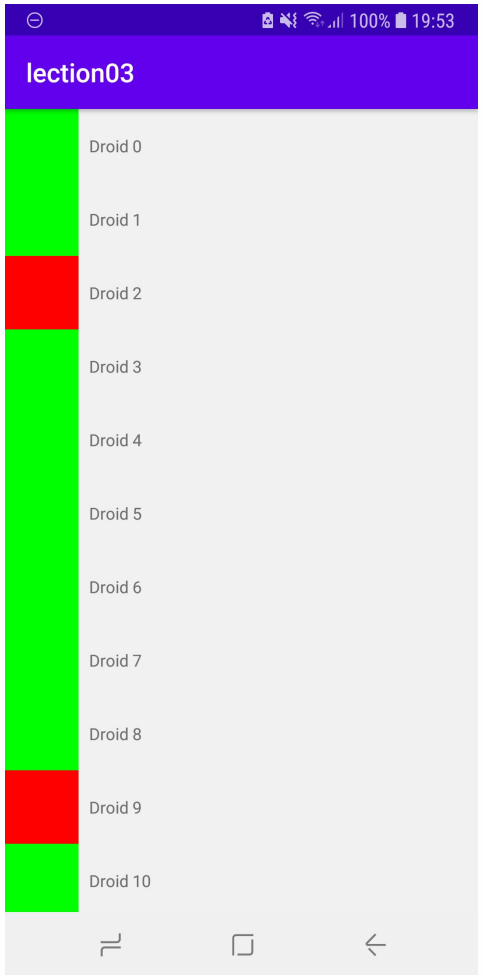
```java
public class DroidDetailsFragment extends DialogFragment {
    protected static final String EXTRAS_DROID = "DROID";

    ...
}
```

```kotlin
class DroidDetailsFragment : DialogFragment() {
    companion object {
        const val EXTRAS_DROID = "DROID"
        ...
    }

    ...
}
```

Шаг 5 - Как добавить DroidListFragment?



Шаг 5 - Его можно сделать статичным




```
<androidx.appcompat.widget.LinearLayoutCompat
    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:orientation="horizontal"

    tools:context=".presentationlayer.MainActivity"
    >
    <fragment
        android:id="@+id/data"
        android:name="ru.hse.lection03.presentationlayer.DroidListFragment"

        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        />
    <FrameLayout
        android:id="@+id/details"

        android:layout_weight="2"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        />
</androidx.appcompat.widget.LinearLayoutCompat>
```

Шаг 6 - Добавим код для отображения информации

```java
protected void showDetails(Droid droid) {
    if (droid == null) {
        return;
    }


    final DroidDetailsFragment detailsFragment = DroidDetailsFragment.newInstance(droid);

    final boolean isDual = getResources().getBoolean(R.bool.is_dual);
    if (isDual) {
        getSupportFragmentManager()
                .beginTransaction()
                .replace(R.id.details, detailsFragment, TAG_DETAILS)
                .commitAllowingStateLoss();
    } else {
        detailsFragment.show(getSupportFragmentManager(), TAG_DETAILS_DIALOG);
    }
}
```

```kotlin
protected fun showDetails(droid: Droid?) {
    if (droid == null) {
        return
    }


    val detailsFragment = DroidDetailsFragment.newInstance(droid)

    val isDual = resources.getBoolean(R.bool.is_dual)
    when(isDual) {
        true -> {
            supportFragmentManager
                    .beginTransaction()
                    .replace(R.id.details, detailsFragment, TAG_DETAILS)
                    .commitAllowingStateLoss()
        }

        false -> {
            detailsFragment.show(supportFragmentManager, TAG_DETAILS_DIALOG)
        }
    }
}
```

Шаг 7 - Инициализация Activity

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_main);


    final boolean isDual = getResources().getBoolean(R.bool.is_dual);

    if (savedInstanceState == null) {
        if (isDual) {
            final Droid droid = DroidRepository.getInstance().item(DEFAULT_DROID_INDEX);
            showDetails(droid);
        }
    } else {
        // checkDetails(isDual);
    }
}
```

```kotlin
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    setContentView(R.layout.activity_main)


    val isDual = resources.getBoolean(R.bool.is_dual)

    if (savedInstanceState == null) {
        if (isDual) {
            val droid = DroidRepository.instance.item(DEFAULT_DROID_INDEX)
            showDetails(droid)
        }
    } else {
        // checkDetails(isDual)
    }
}
```

Шаг 8.1 - Делаем список кликабельным

```java
public class DroidViewHolder extends RecyclerView.ViewHolder {
    public interface IListener {
        void onDroidClicked(int position);
    }


    protected final IListener mListener;


    public DroidViewHolder(View itemView, IListener listener) {
        super(itemView);

        mListener = listener;

        final View.OnClickListener clickListener = new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mListener.onDroidClicked(getAdapterPosition());
            }
        };

        itemView.setOnClickListener(clickListener);
    }
}
```

```kotlin
class DroidViewHolder(itemView: View, val listener: IListener)
       : RecyclerView.ViewHolder(itemView) {

       interface IListener {
         fun onDroidClicked(position: Int)
       }


    init {
        ...
        itemView.setOnClickListener {
            listener.onDroidClicked(adapterPosition)
        }
    }
}
```

Шаг 8.2 - Протягиваем IListener через Адаптер

```java
public class DroidAdapter extends RecyclerView.Adapter<DroidViewHolder> {
    protected final DroidViewHolder.IListener mListener;

    public DroidAdapter(List<Droid> data, DroidViewHolder.IListener listener) {
        mListener = listener;
        ...
    }

    @NonNull
    @Override
    public DroidViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        ...
        return new DroidViewHolder(layout, mListener);
    }

    ...
}
```

```kotlin
class DroidAdapter(val data: List<Droid>, val listener: DroidViewHolder.IListener)
    : RecyclerView.Adapter<DroidViewHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): DroidViewHolder {
        ...
        return DroidViewHolder(layout, listener)
    }

    ...
}
```

Шаг 8.3 - Вытаскиваем клик в Fragment

```java
public class DroidListFragment extends Fragment {
    public interface IListener {
        public void onDroidClicked(Droid droid);
    }


    protected IListener mListener;


    @Override
    public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);

        final RecyclerView recycler = view.findViewById(R.id.recycler);
        recycler.setAdapter(new DroidAdapter(
             DroidRepository.getInstance().list()
             , new DroidClickHandler()
             ));
        recycler.setLayoutManager(new LinearLayoutManager(requireContext()));
    }


    class DroidClickHandler implements DroidViewHolder.IListener {
        @Override
        public void onDroidClicked(int position) {
            final Droid droid = DroidRepository.getInstance().item(position);

            if (mListener != null) {
                mListener.onDroidClicked(droid);
            }
        }
    }
}
```

```kotlin
class DroidListFragment: Fragment() {
    interface IListener {
        fun onDroidClicked(droid: Droid)
    }


    protected var listener: IListener? = null


    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        val recycler = view.findViewById<RecyclerView>(R.id.recycler)
        recycler.apply {
            adapter = DroidAdapter(
                DroidRepository.instance.list()
                , DroidClickHandler()
            )
            layoutManager = LinearLayoutManager(context)
        }
    }

    inner class DroidClickHandler: DroidViewHolder.IListener {
        override fun onDroidClicked(position: Int) {
            val droid = DroidRepository.instance.item(position)

            listener?.onDroidClicked(droid)
        }
    }
}
```

Шаг 8.3 - Имплементируем слушателя фрагмента

```java
public class MainActivity extends AppCompatActivity implements DroidListFragment.IListener
{
    @Override
    public void onDroidClicked(Droid droid) {
        showDetails(droid);
    }

    ...
}
```

```kotlin
class MainActivity : AppCompatActivity(), DroidListFragment.IListener {
    override fun onDroidClicked(droid: Droid) {
        showDetails(droid)
    }

    ...
}
```

Шаг 8.4 - Получаем слушателя из Fragment

Можно на выбор:

- Запоминаем слушателя в **onAttach()** и зануляем на него ссылку в **onDetach()**
- Получаем слушателя только в нужный момент

```
listener = requireActivity() as? IListener
```

```
if (requireActivity() instanceof IListener) {
    mListener = (IListener) requireActivity();
}
```

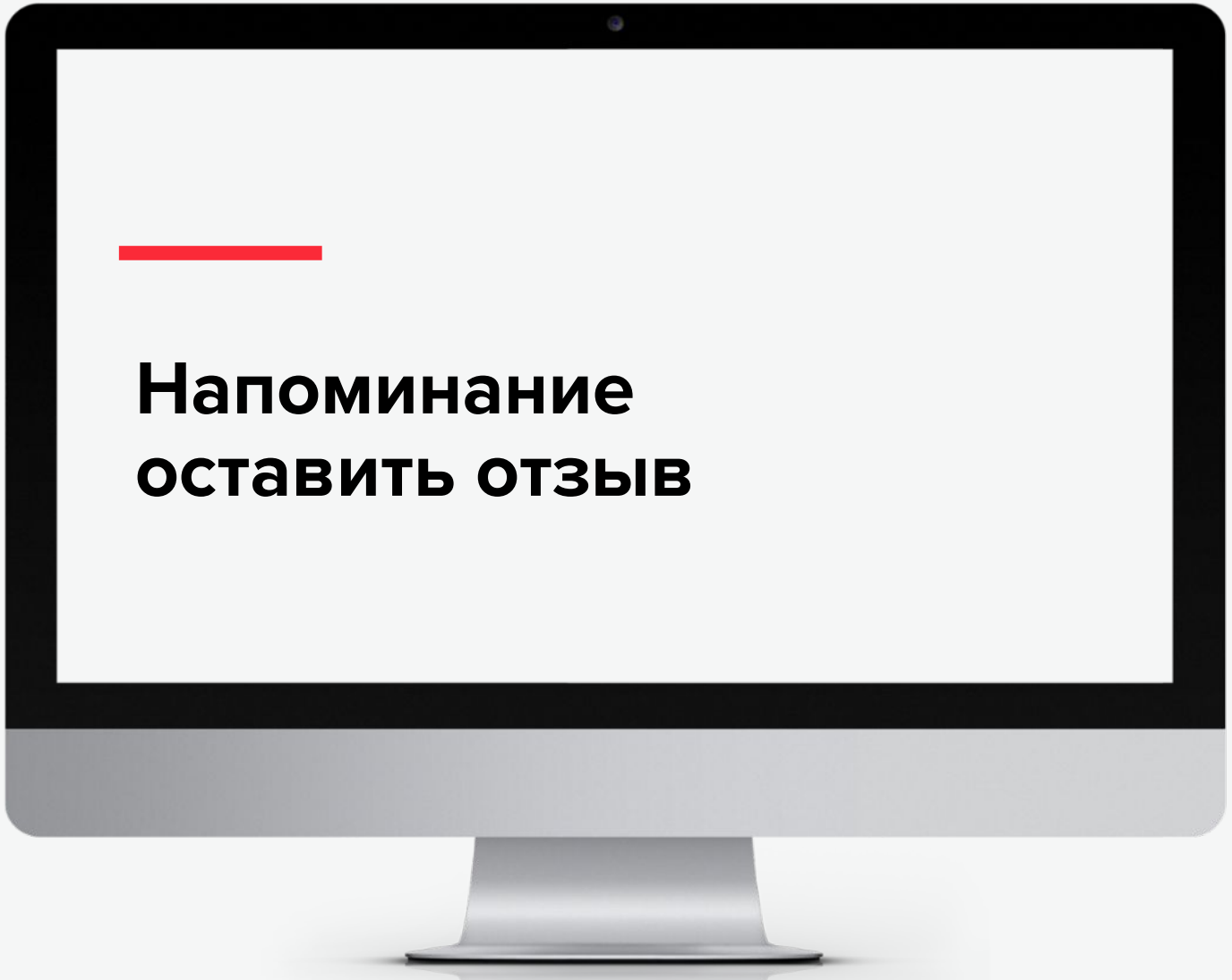
Шаг 9 - “Полировать” логику

На слайд этот код не поместится. Смотрим в гитхаб и обсуждаем!

https://github.com/mailru-android-edu/hse-android-samples

Проблемы:

- При перевороте из горизонтального режима - диалог не появляется
- При перевороте из вертикального режима - панелька не появляется


Напоминание
оставить отзыв

СПАСИБО
ЗА ВНИМАНИЕ