

# Activity. Fragment. Lifecycle

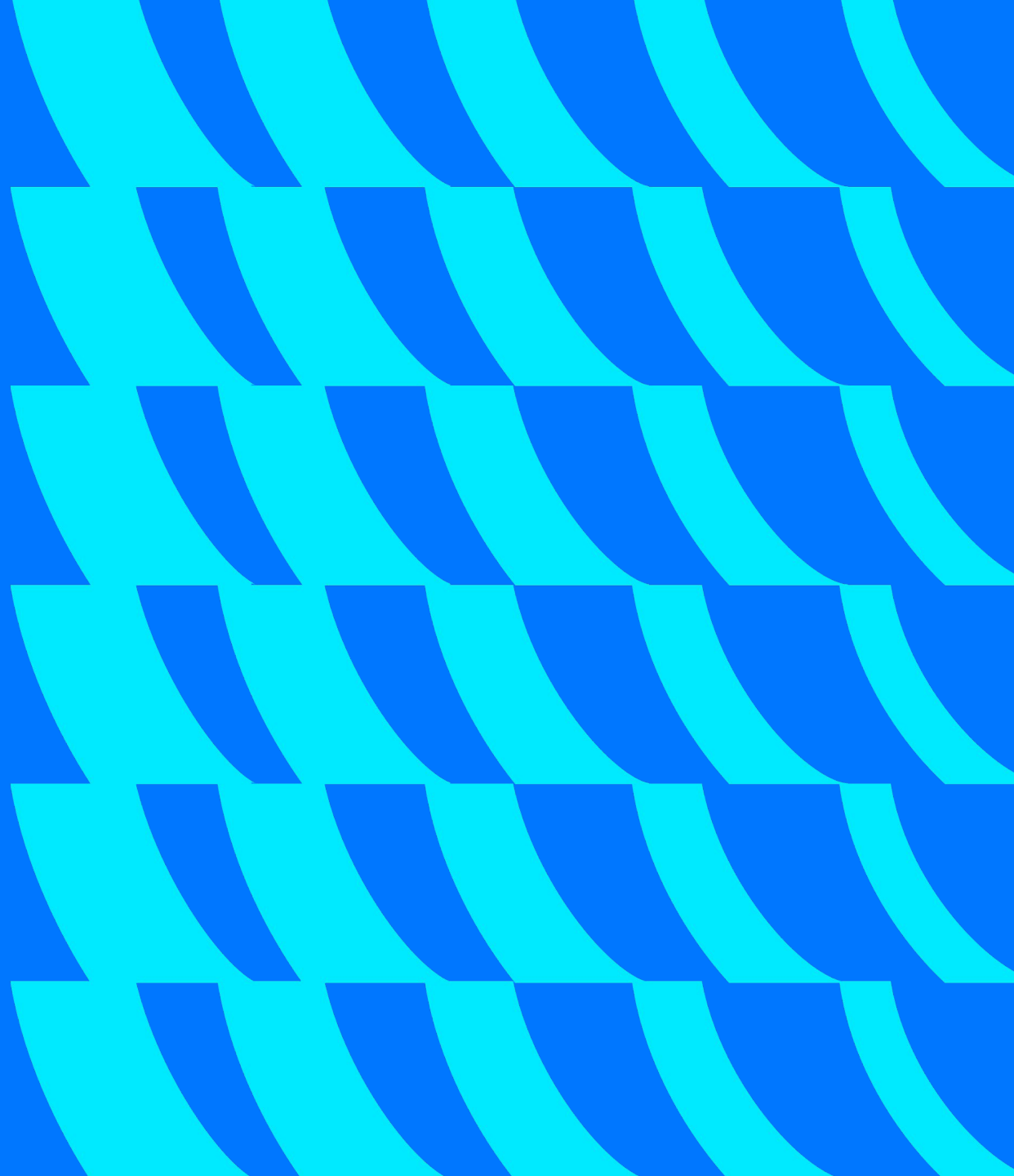
Червяков Алексей



# Наш план

- Что такое activity
- Как запустить activity
- Что такое фрагмент fragment
- Что будем делать с fragment
- Как они живут в проекте
- Пишем немного кода

# Организационные моменты

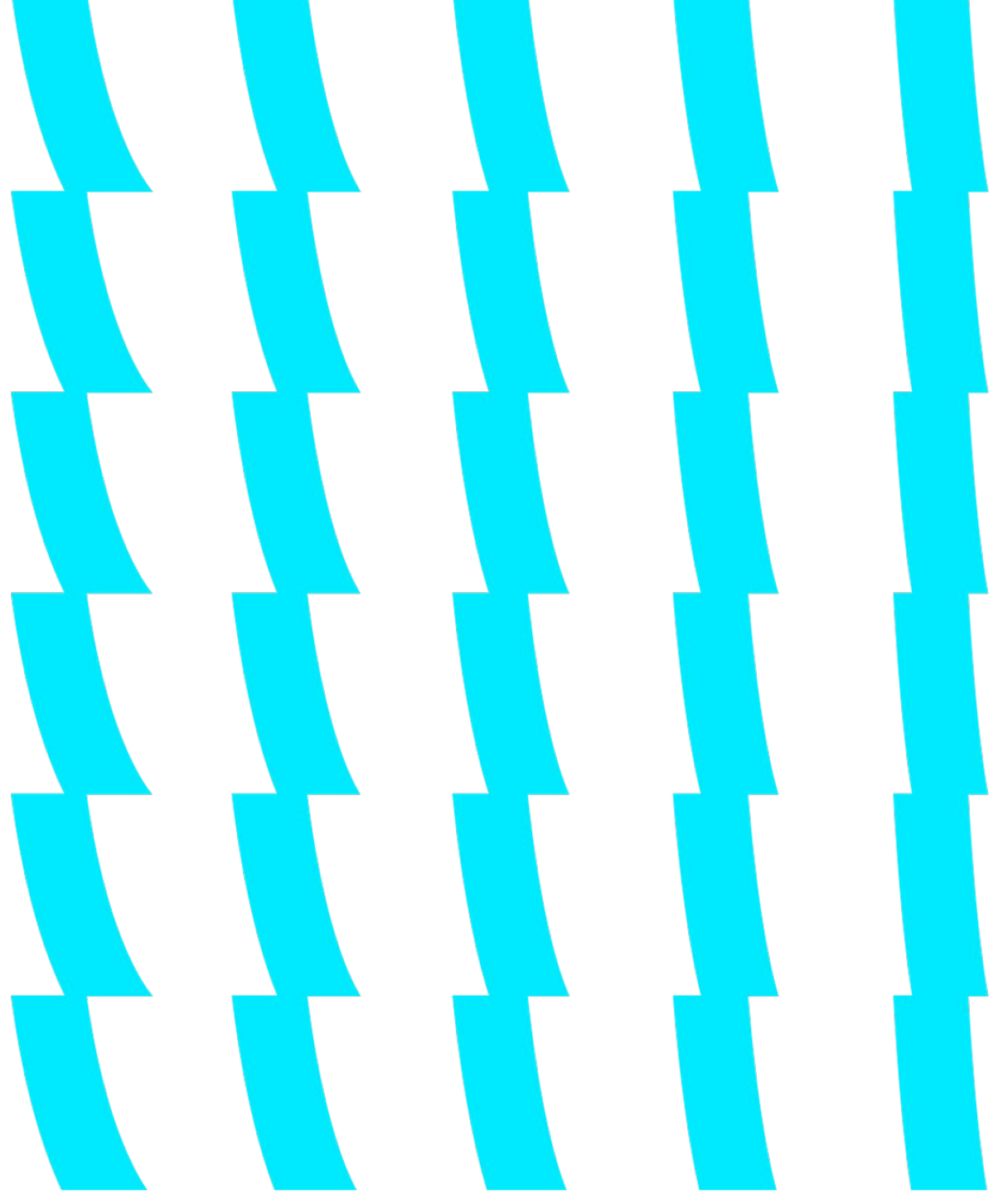


Напоминание  
отметиться на  
портале

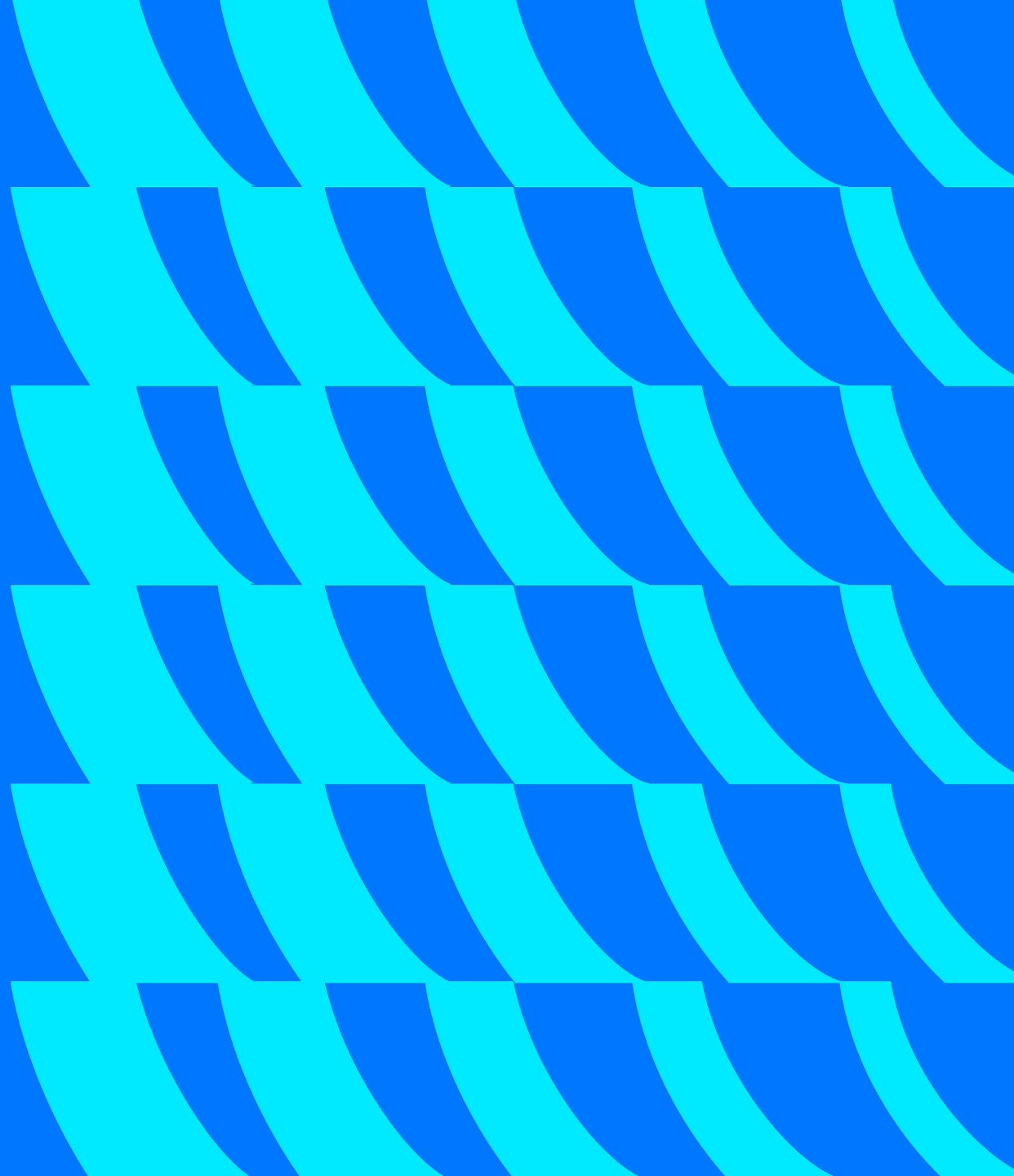
# Домашнее задание

Тест для проверки материала

- сдать до 1 ноября
- 15 минут
- 5 баллов



# Activity



# Activity

Один из главных компонентов приложения



# Activity

Входит в список основных компонентов приложения

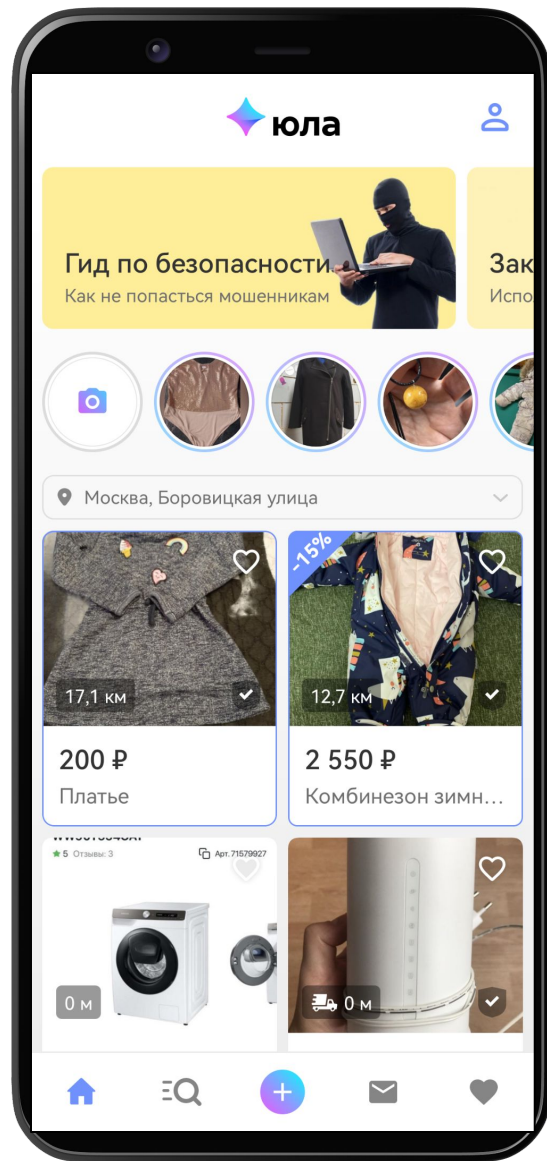
Приложение состоит из одной или нескольких Activity, которые слабо связаны друг с другом.

Отвечает за визуальную часть приложения

Обычно одна из Activity обозначается как основная.

Повышает шансы вашего приложения не быть убитым системой

При запуске activity помещается в backStack





# Из чего состоит activity

## Код

```
class StartActivity : AppCompatActivity() {  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.start_activity)  
    }  
}
```

# Из чего состоит activity

## Верстка

```
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        tools:text="Привет друг" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

# Android Manifest

Необходим системе Android для запуска компонента приложения.

В этом файле должны быть описаны все компоненты приложения.

В этом же файле должны быть описаны все разрешения, которые необходимы. Доступ в интернет, доступ к контактам

В нем объявляются также различные службы, которые будем использовать. Bluetooth

# AndroidManifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.hello">
    <application>
        <activity
            android:name="com.hello.StartActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

# BackStack и Tasks

## Task

- Задача, в рамках которой происходит работа с Activity

## BackStack

- Стэк, хранящий Activity
- Имеет стратегию LIFO

## Для простоты запоминания

- **Task** - как вкладка браузера
- **BackStack** - история посещений в рамках вкладки
- **Activity** - открываемая страница

# LaunchMode

- Standard Mode
- Single Top Mode
- Single Task Mode
- Single Instance Mode

# Standard

```
<activity
```

```
    ...
```

```
    android:launchMode="standard">
```

- Дефолтный режим (можно не проставлять)
- Означает, что на “намерение” запуска Activity будет всегда запущена новая Activity.



# SingleTop

```
<activity
```

```
    ...
```

```
    android:launchMode="singleTop">
```

- Если находится на вершине стека, то вместо создания экземпляра будет вызван `onNewIntent()`





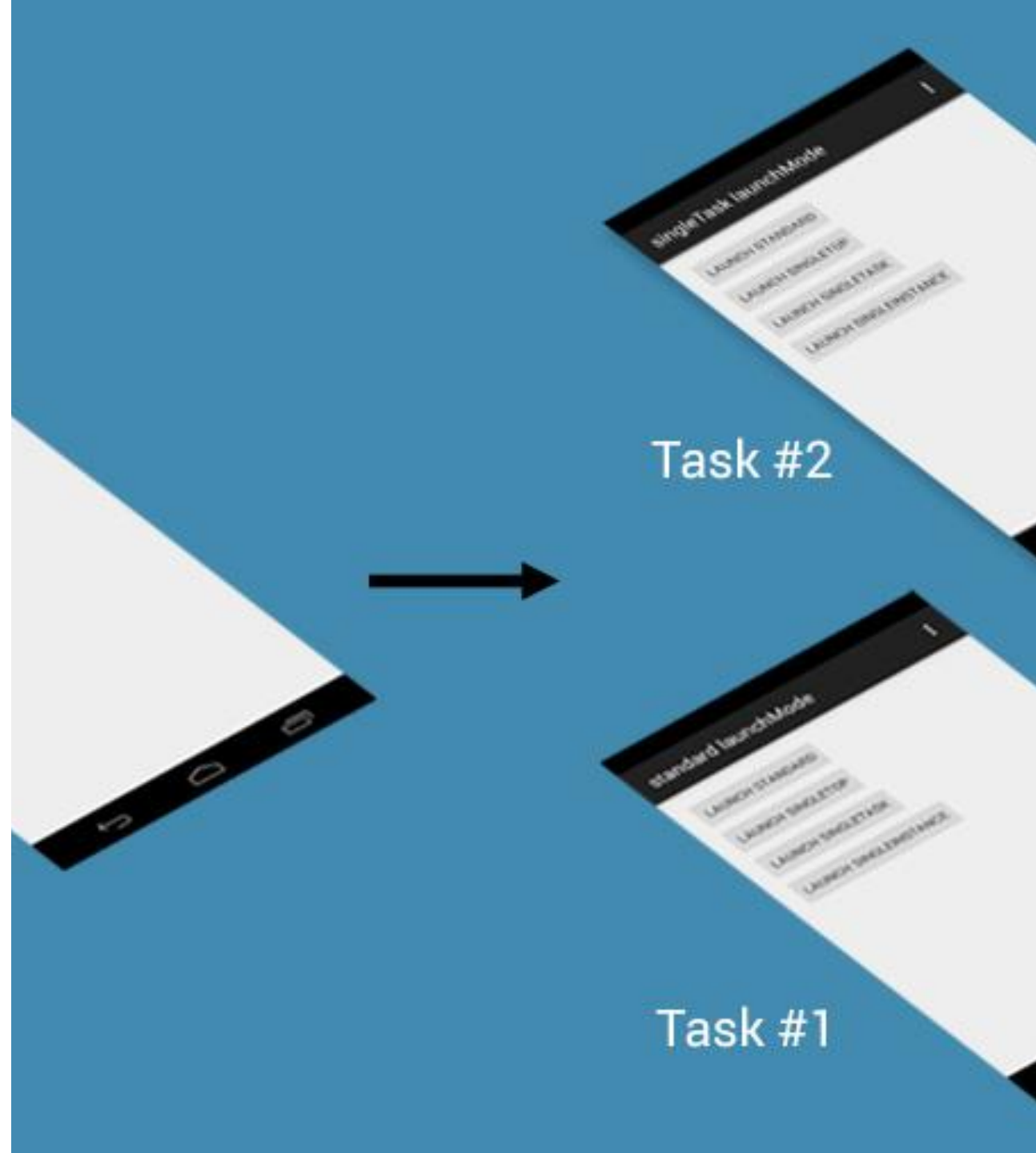
# SingleTask

```
<activity
```

```
    ...
```

```
    android:launchMode="singleTask">
```

- Если находится в стеке - то будет поднят наверх, с уничтожением всех Activity, которые находятся выше нее



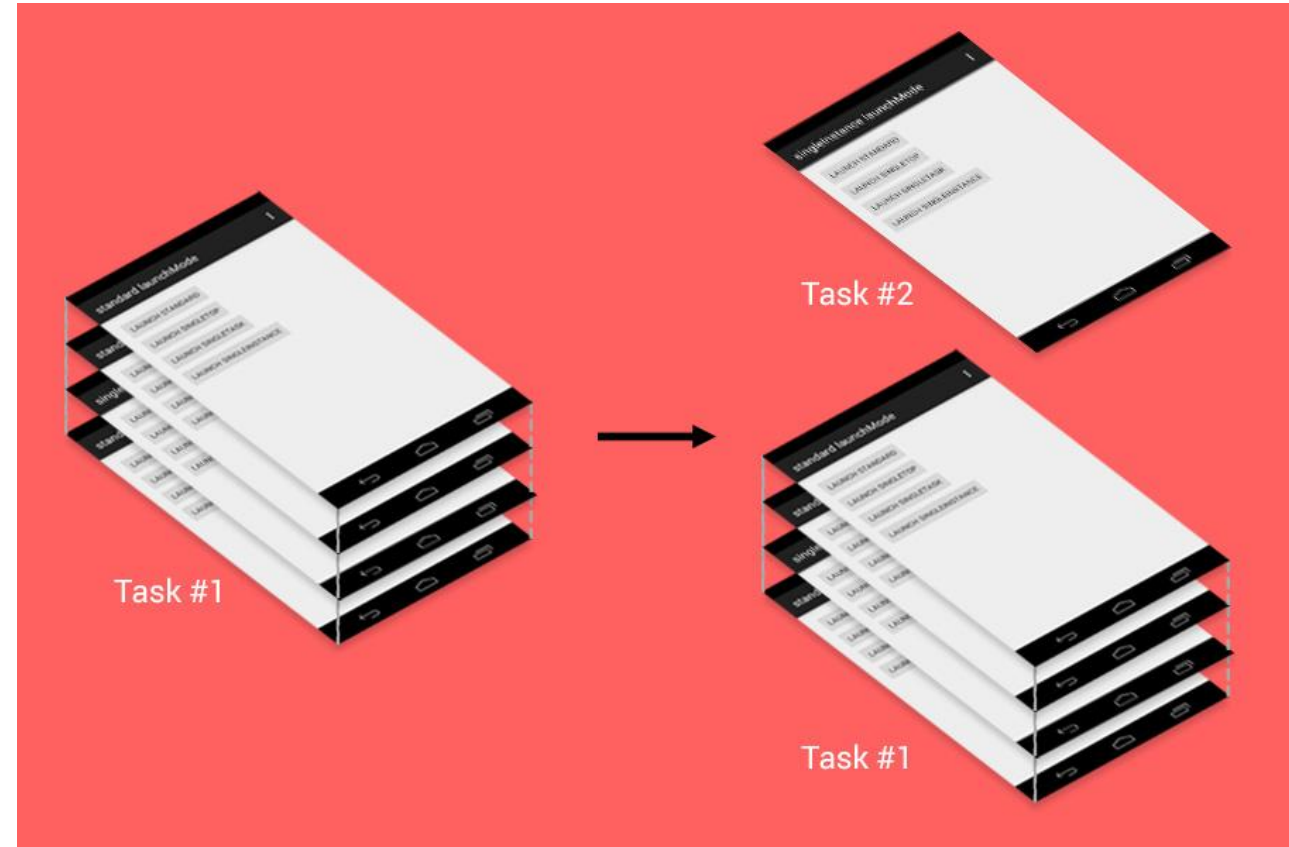
# SingleInstance

```
<activity
```

```
...
```

```
android:launchMode="singleInstance">
```

- При запуске будет создан для нее отдельный Task. Если экземпляр уже создан, то будет поднят старый экземпляр и вызван метод `onNewIntent`

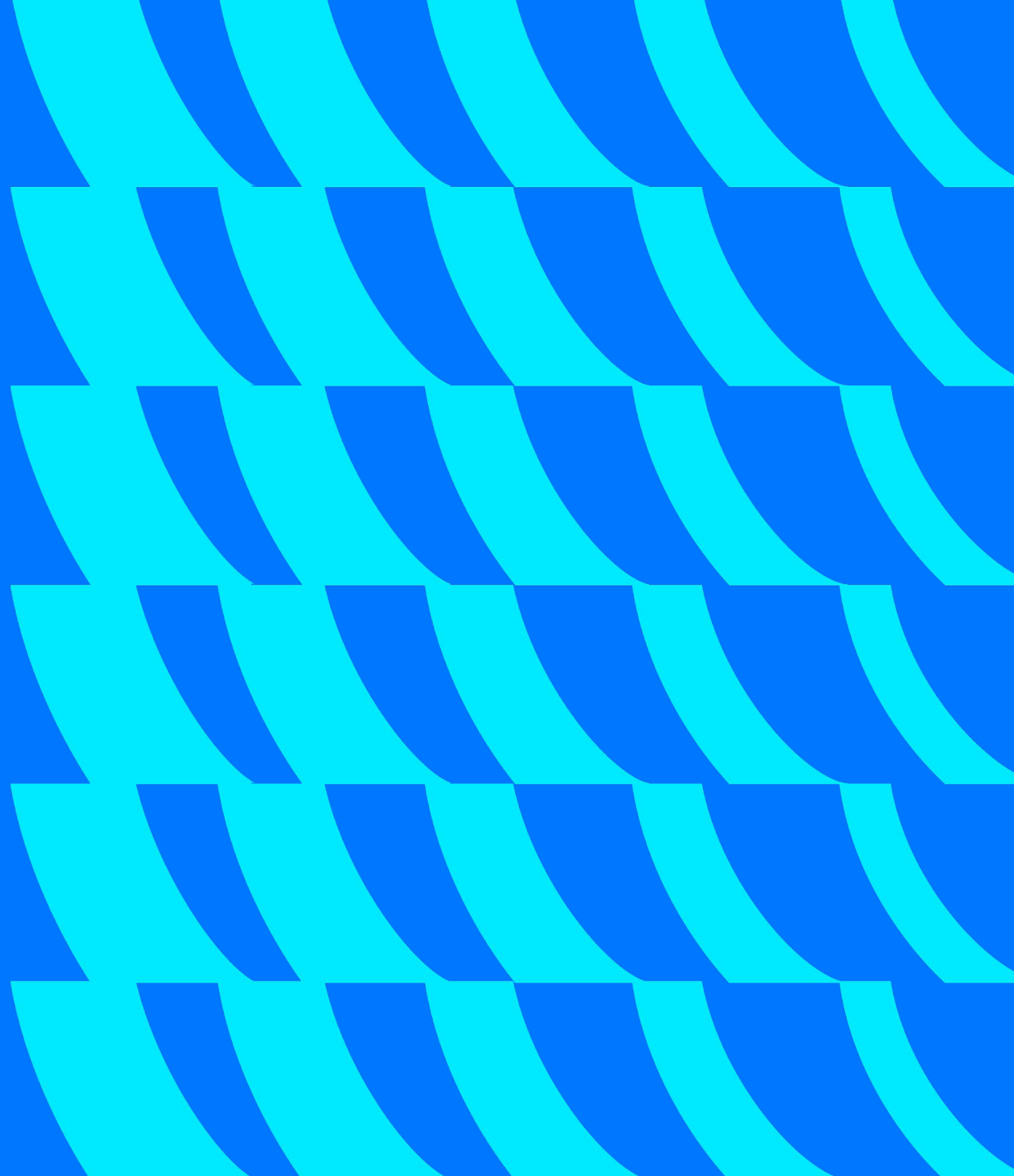


# Важно

- Activity должна быть описана как class
- Activity должна содержать верстку
- Activity должна быть определена в android Manifest

Вопросы?

Запуск  
activity



# Запуск activity

Запускаем активности через intent



# Intent

Объект для описания операции для исполнения его системой.

**Explicit intent** - Явное “намерение”.

Указываем класс, к которому хотим обратиться.

```
Intent(context, StartActivity::class.java)
```

**Implicit intent** - Неявное “намерение”.

Указываем данные, а далее система собирает список обработчиков

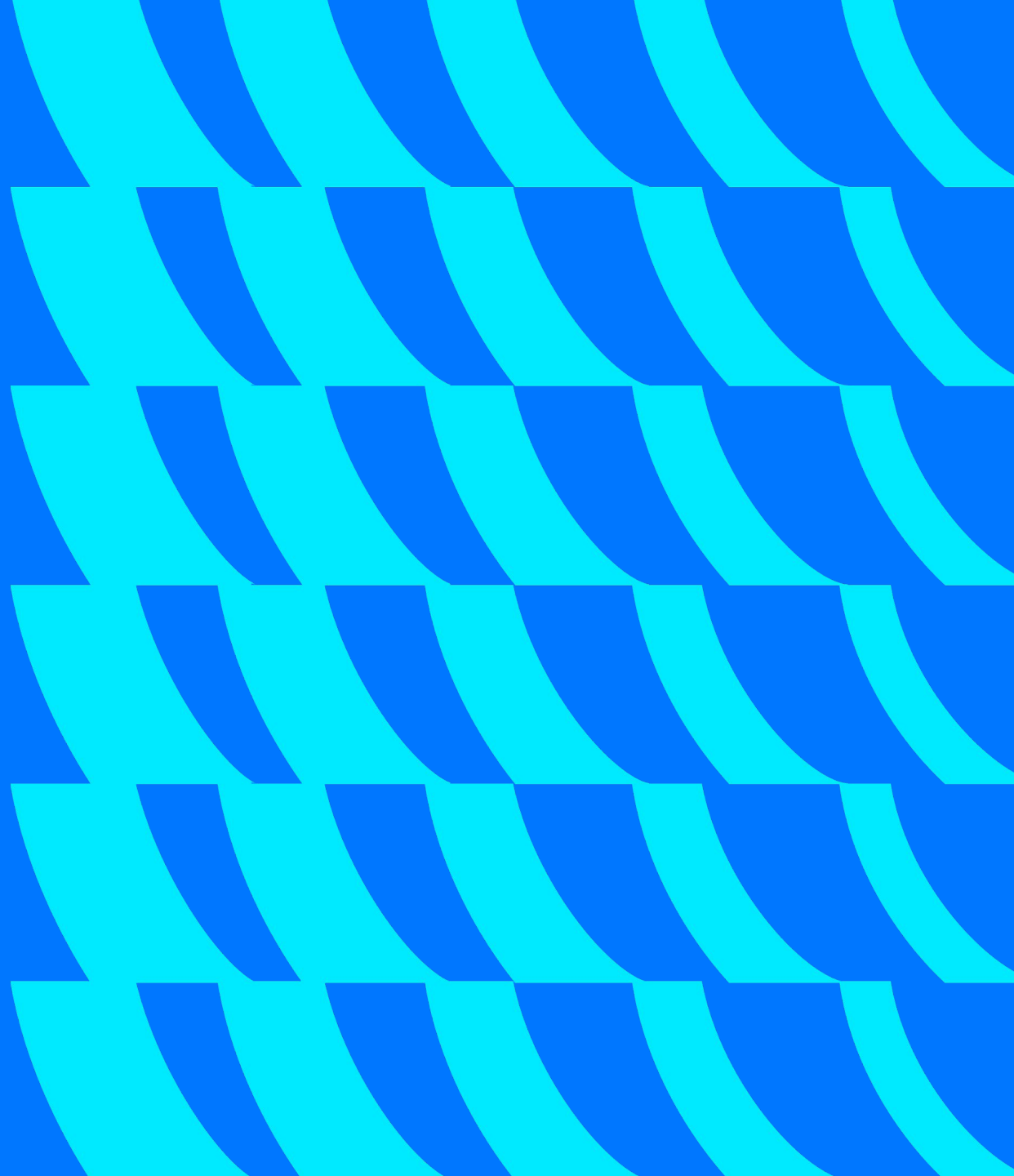
```
Intent(Intent.ACTION_VIEW, Uri.parse("https://google.com"))  
    Intent(Intent.ACTION_CALL).setData(Uri.parse("tel:555-555-5555"))
```

<b>Отправить письмо</b>	<pre>val intent = Intent(Intent.ACTION_SENDTO).apply {     data = Uri.parse("mailto:")     putExtra(Intent.EXTRA_EMAIL, addresses)     putExtra(Intent.EXTRA_SUBJECT, "subject") }</pre>
<b>Выбрать контакт</b>	<pre>val intent = Intent(Intent.ACTION_PICK).apply {     type = ContactsContract.Contacts.CONTENT_TYPE }</pre>
<b>Выбрать файл</b>	<pre>val intent = Intent(Intent.ACTION_OPEN_DOCUMENT).apply {     {         addCategory(Intent.CATEGORY_OPENABLE)     } }</pre>
<b>Вызвать звонилку</b>	<pre>val intent = Intent(Intent.ACTION_DIAL).apply {     data = Uri.parse("tel:+78001234567") }</pre>



Вопросы?

# Fragment



# Fragment

Часть интерфейса



# Fragment

Модульные, переиспользуемые части пользовательского интерфейса.

Они не самостоятельны - зависят от Activity.

## **Основные плюсы:**

- Экран можно разбить на части, и одну из этих частей, со всей логикой, можно переиспользовать на других экранах
- Можно собирать один большой экран из отдельных мелких экранов (то что на телефоне может быть разными экранами, на планшете можно сделать одним экраном)
- Чистит ресурсы при попадании в стек (в отличии от Activity)

## Из чего состоит fragment

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

# Использование

## Статическое

```
<fragment

xmlns:android="http://schemas.android.com/apk/res/andro
id"

    android:name="ru.example.myapplication.MyFragment"
    android:layout_height="match_parent"
    android:layout_width="match_parent"
/>
```

## Динамическое

```
supportFragmentManager

    .beginTransaction()

    .replace(R.id.container, MyFragment())
    .commit()
```

В данном случае, создаем транзакцию, для того, чтобы подменить фрагмент, который находится во View с идентификатором container

# Управление фрагментами

- Fragment - часть UI, который мы можем переиспользовать
- FragmentManager - главный компонент для управления фрагментами.
- FragmentTransaction - транзакция, для внесения изменения стека фрагментов

# FragmentManager

```
with(supportFragmentManager) {  
    // получить все фрагменты  
    fragments  
    // найти фрагмент по id или тегу  
    findFragmentById(R.id.container)  
    findFragmentByTag("tag")  
    // отменить последнюю транзакцию  
    popBackStack()  
    // отменить все транзакции добавленные до транзакции с тегом  
    popBackStack("name", 0)  
    // отменить все транзакции вместе с транзакцией добавленные с тегом  
    popBackStack("name", FragmentManager.POP_BACK_STACK_INCLUSIVE)  
    // количество транзакций  
    backStackEntryCount  
}
```



# FragmentTransaction

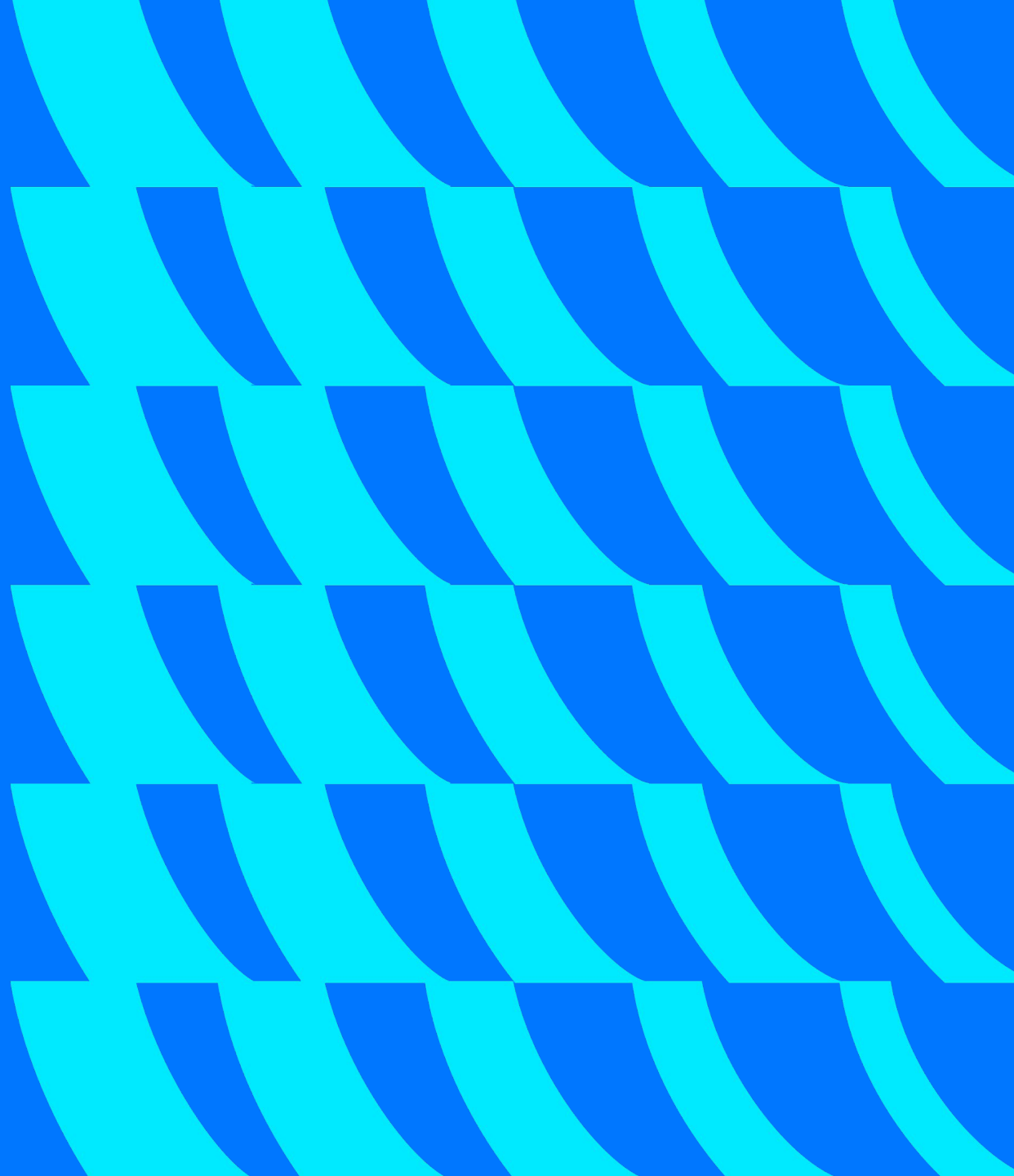
```
supportFragmentManager.beginTransaction().run {  
    // добавление нового фрагмента  
    add(R.id.container, fragment, "tag")  
    // удаление фрагмента  
    remove(fragment)  
    // замена текущего фрагмента  
    replace(R.id.container, framgnet, "tag")  
    // скрывтие фрагмента  
    hide(fragment)  
    // показываем ранее скрытый фрагмент  
    show(fragment)  
    // добавление транзакции в backStack  
    addToBackStack("transaction")  
    // завершение транзакции  
    commit()  
    // завершение транзакции с риском потерять состояние  
    commitAllowingStateLoss()  
}
```

# Создание фрагмента из кода

```
fun newInstance(data: Data): DetailsFragment {  
    val extras = Bundle().apply {  
        putSerializable(EXTRAS_DROID, data)  
    }  
    val fragment = DetailsFragment().apply {  
        arguments = extras  
    }  
    return fragment  
}
```

Вопросы?

# Передача данных



# Передача данных в activity

Можно передавать и получать данные



# Передача данных в activity

используется объект Bundle

пример для стандартизированных параметров (int, string)

```
Intent().apply {  
    putExtra("key", 1)  
    putExtra("key", "string")  
}
```

пример для не стандартизированных параметров

```
class Data() : Parcelable {}  
  
Intent().apply {  
    putExtra("key", Data())  
}
```

# Ограничения на передачу данных

Есть ограничения:

- Можно передавать только примитивы, строки, Parcelable и Serializable объекты
- Есть ограничение на вес Bundle (512kb)

# Получение данных из activity

## **ActivityResultContract**

Компонент который позволяет зарегистрироваться на получение результата из другой Activity

- Требуется создать ActivityResultContract или взять подходящий в ActivityResultContracts
- Зарегистрировать его registerForActivityResult
- Запустить - ActivityResultContract.launch()



# Создание контракта

```
class Contract : ActivityResultContract<String, Int>() {  
    override fun createIntent(context: Context, input: String): Intent =  
        Intent(context, SecondActivity::class.java).apply { putExtra("key", input) }  
  
    override fun parseResult(resultCode: Int, intent: Intent?): Int =  
        if (resultCode != Activity.RESULT_OK) 0  
        else intent?.getIntExtra("result", 0).orValue(0)  
  
    override fun getSynchronousResult(context: Context, input: String): SynchronousResult<Int>?  
    =  
        if (input.isEmpty()) SynchronousResult(0) else null  
}
```

# Регистрация контракта

```
val registerContract = registerForActivityResult(Contract()) { result ->  
    handleResult(result)  
}
```

# Запуск контракта

```
button.setOnClickListener {  
    registerContract.launch("Привет")  
}
```

# Передача данных

```
setResult(Activity.RESULT_OK, Intent().apply { putExtra("result", 10) })
```

# Передача данных в fragment

Можно передавать и получать данные



# Передача данных в fragment

используется объект Bundle

```
DetailFragment().apply {  
    arguments = Bundle().apply {  
        putInt("key", 1)  
        putParcelable("key", Data())  
    }  
}
```

# Получение данных из fragment

## **FragmentResultListener**

Компонент который позволяет зарегистрироваться на получение результата из другого fragment

- Требуется создать `FragmentResultListener`
- Передать данные в качестве результата из другого фрагмента

# Создание слушателя

```
setFragmentResultListener("result_key") { key, bundle ->  
    handleResult(bundle)  
}
```

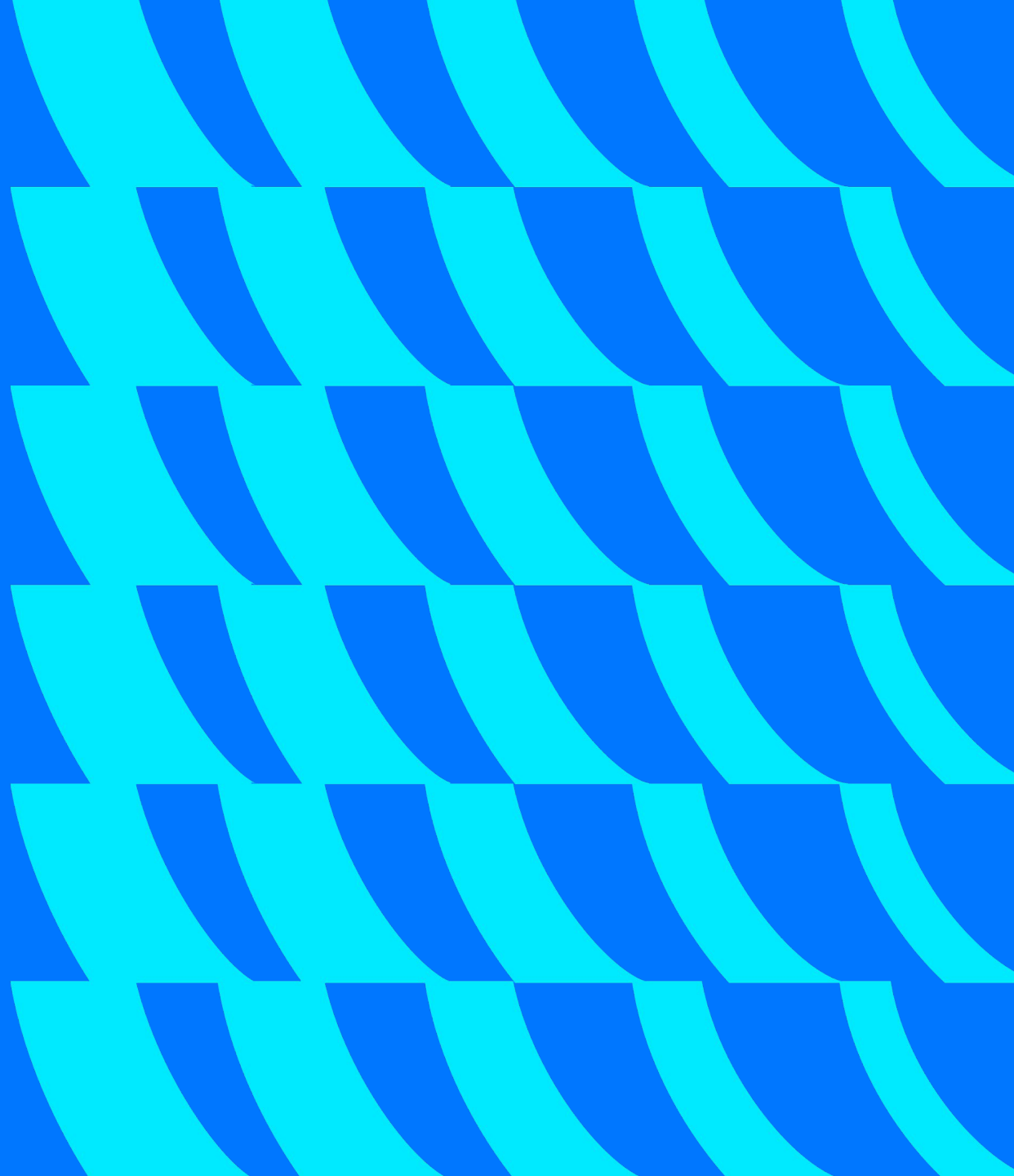


# Передача данных

```
setFragmentResult("result_key", Bundle().apply { putInt("result", 1) } )
```

Вопросы?

# Lifecycle



# Жизненный цикл

у activity и fragment он разный



# Activity

**onCreate** вызывается когда активити создается. В onCreate() вы должны вызвать метод setContentView().

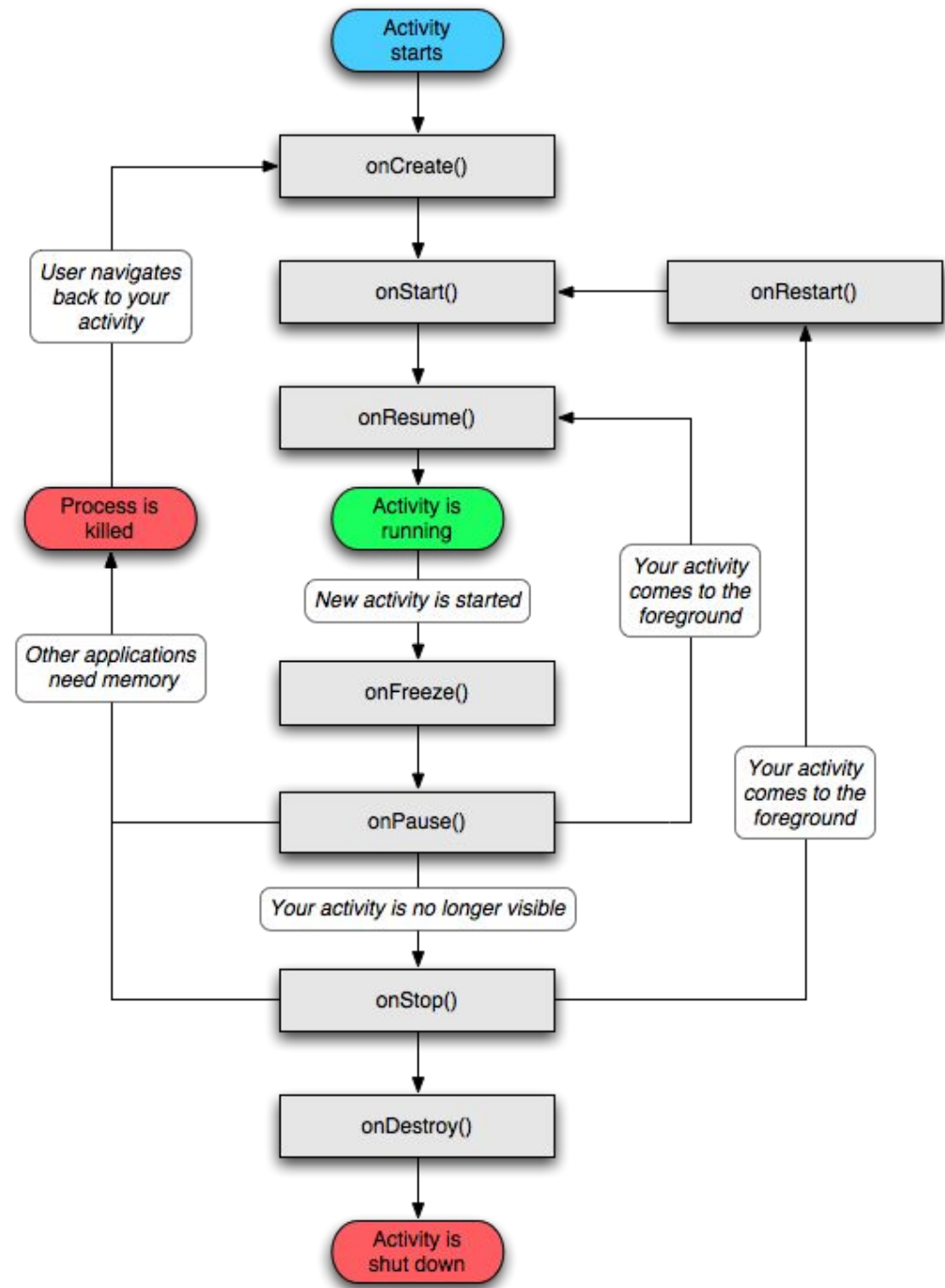
**onStart** вызывается когда активити отрисована и видима пользователю.

**onResume** вызывается перед тем как активити станет доступна для взаимодействия с пользователем.

**onPause** метод симметричный onResume(). Пользователь больше не может взаимодействовать с активити, но активити частично видна пользователю. В этом состоянии UI активити может изменяться.

**onStop** метод симметричный onStart(). Вызывается, когда активити больше не видна пользователю.

**onDestroy** метод симметричный onCreate(). Вызывается перед тем, как активити будет уничтожена системой.



# Fragment

**onAttach** – Вызывается когда фрагмент присоединяется к активити.

**onCreate** – Вызывается когда фрагмент создается.

**onCreateView** – Метод, в котором создается иерархия View, связанная с фрагментом.

**onActivityCreated** – Вызывается после того, как отработает метод Activity.onCreate().

**onStart** – Вызывается, когда фрагмент становится видим пользователю, после Activity.onStart().

**onResume** – Вызывается перед тем как фрагмент станет доступен для взаимодействия с пользователем, после Activity.onResume().

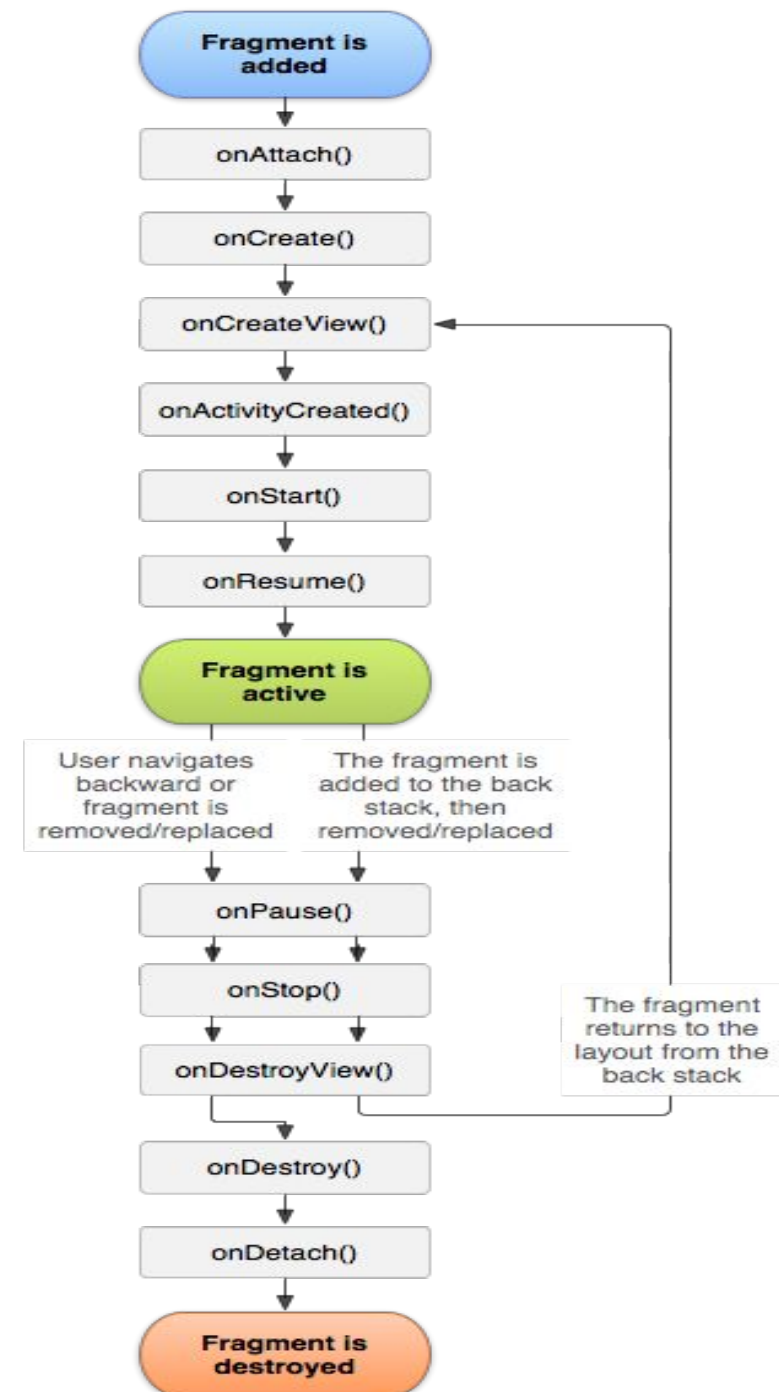
**onPause** – Пользователь не может взаимодействовать с фрагментом, но часть фрагмента видима пользователю.

**onStop** – Фрагмент становится не видим пользователю.

**onDestroyView** – Метод, в котором фрагмент очищает ресурсы, связанные с иерархией View.

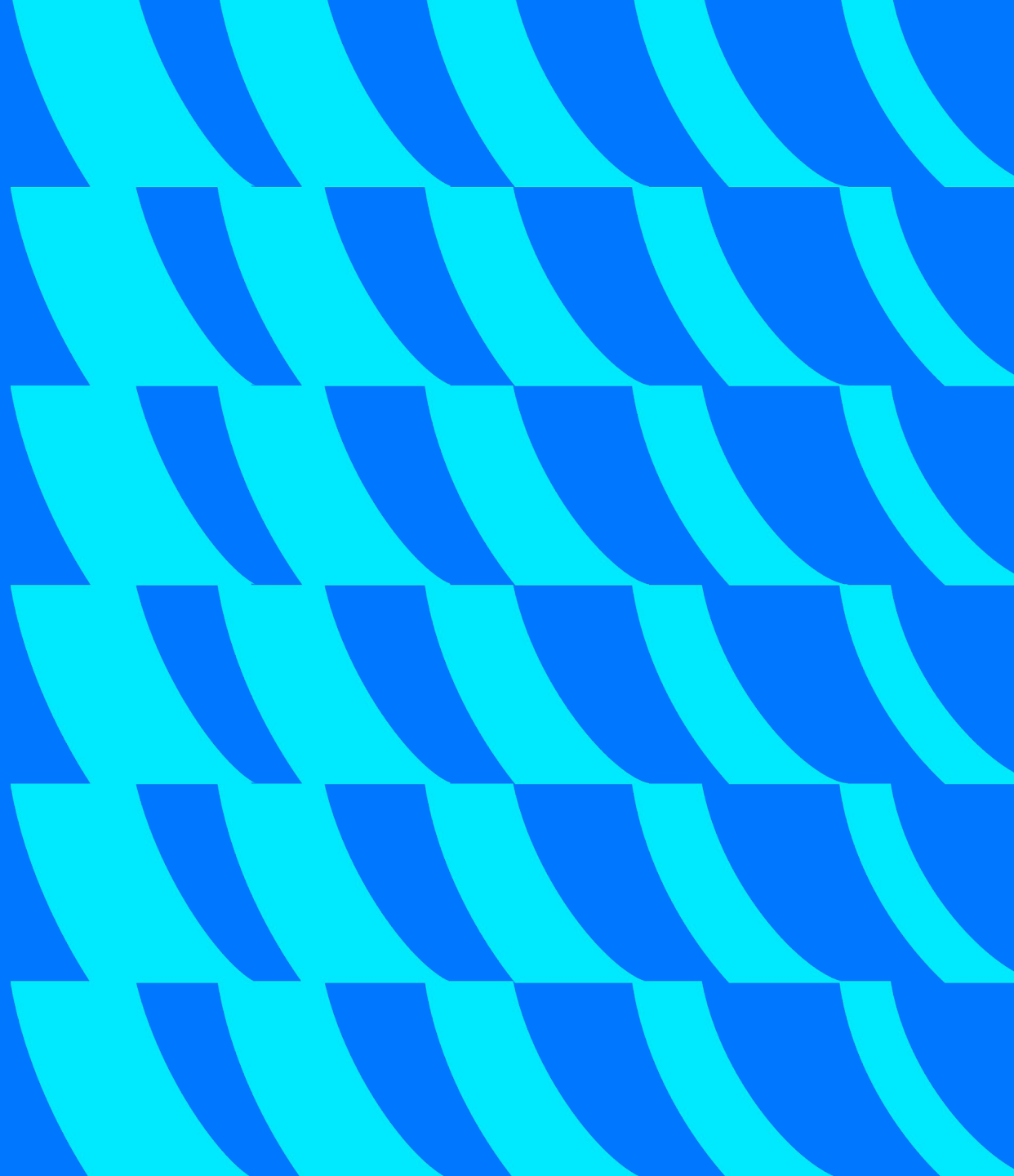
**onDestroy** – Вызывается перед тем, как фрагмент будет уничтожен системой.

**onDetach** – Вызывается перед тем, как фрагмент будет отсоединен от активити.



Вопросы?

Пишем код





Оставьте  
отзыв!



Спасибо  
за внимание!

Червяков Алексей