

# Навигация

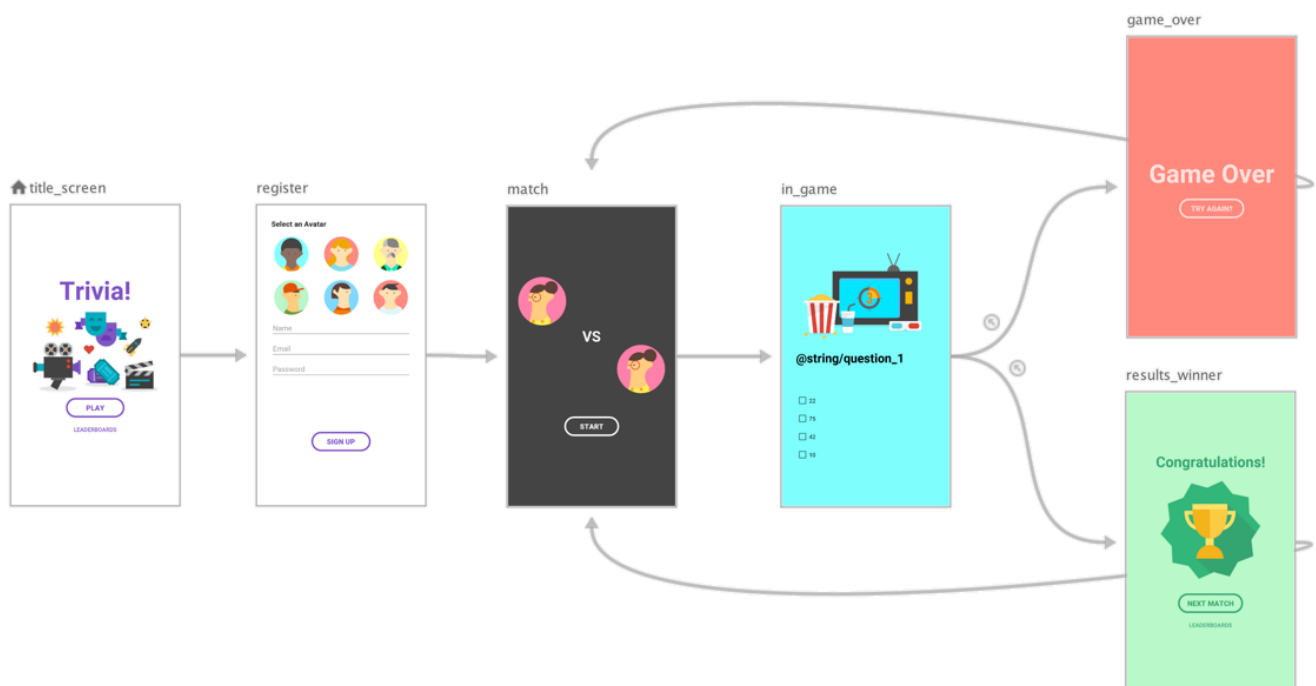
Навигация позволяет пользователям перемещаться по различным частям приложения. Компонент Android Jetpack помогает реализовать навигацию: от простых нажатий кнопок до панелей приложения и навигации.

Компонент «Навигация» состоит из трех ключевых частей:

- **Граф навигации:** XML-ресурс, содержащий всю информацию, связанную с навигацией. Сюда входят области контента, называемые *пунктами назначения* (англ. destinations), а также возможные пути, по которым пользователь может ходить по приложению. Пункты назначения связаны *действиями* (англ. actions).
- **NavHost:** контейнер, в котором отображаются пункты назначения из графа навигации. Обычно это `NavHostFragment`, который отображает фрагменты.
- **NavController:** объект, который управляет навигацией приложения внутри `NavHost`. Когда пользователь перемещается по приложению, `NavController` заменяет контент к `NavHost`.

## Граф навигации

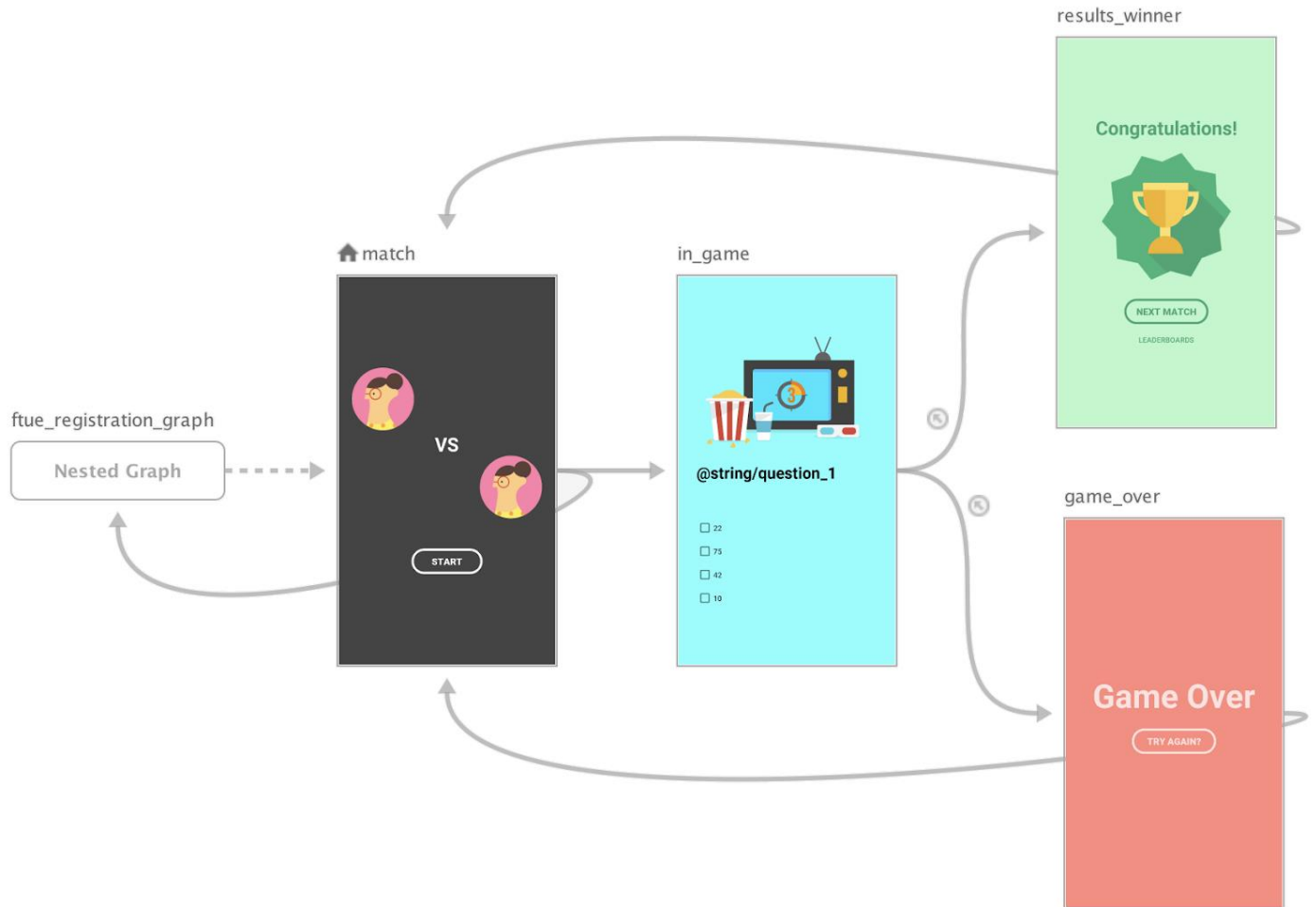
На рисунке показано визуальное представление графа навигации для приложения шестью пунктами назначения, которые связаны пятью действиями. Каждый пункт назначения представлен миниатюрой, а действия представлены стрелками, показывающими, как пользователи могут перемещаться от одного пункта назначения к другому:



Граф навигации *верхнего уровня* должен начинаться с начального пункта назначения, который пользователь видит при запуске приложения, и должен включать места назначения для перемещения по приложению.

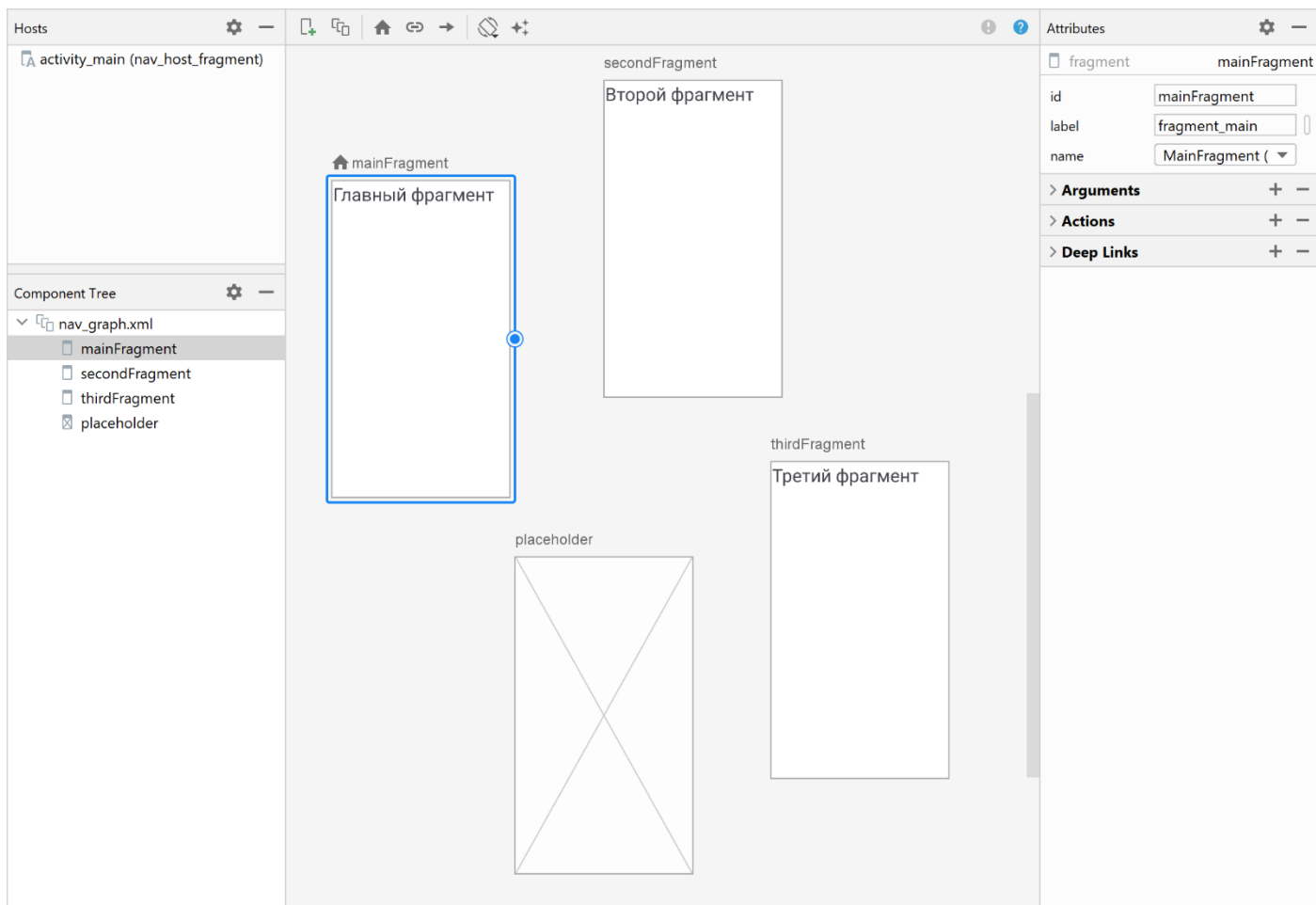
Иногда помимо графа верхнего уровня будет удобно реализовать вложенные графы. В примере выше регистрация требуется только при первом запуске приложения, поэтому

будет разумно сделать начальным пунктом назначения экран *match*, а приветственный экран и экран регистрации переместить во вложенный (англ. nested) граф:



При запуске приложение определяет зарегистрирован ли уже пользователь, и если нет, то перенаправляет его на экран регистрации.

Чтобы добавить граф навигации нужно щёлкнуть правой кнопкой мыши по папке *res* и выбрать *New* → *Android Resource File*. В пункте «Resource type» нужно выбрать тип «Navigation», и указать имя для файла, например, *nav\_graph.xml*. Android Studio предложит добавить в зависимости необходимые библиотеки (нужно согласиться), и после синхронизации навигационный граф будет открыт в редакторе:



В данном примере уже добавлены четыре фрагмента и [NavHostFragment](#), изначально редактор будет пустым. Редактор навигационного графа состоит из нескольких частей:

- в центральной части отображаются все фрагменты и связи между ними;
- панель *Component Tree* содержит список таких фрагментов, входящих в навигационный граф – если в приложении несколько графов, то в каждом может быть свой набор фрагментов;
- на панели *Attributes* можно посмотреть и отредактировать данные фрагментов;
- панель *Hosts* содержит перечень элементов типа [NavHost](#), в которых может осуществляться навигация.

Если перейти в режим Code, то можно посмотреть текстовое представление навигационного графа, в котором перечислены все фрагменты и действия между ними:

```
<?xml version="1.0" encoding="utf-8"?>
<navigation
    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:id="@+id/nav_graph.xml"
    app:startDestination="@id/mainFragment">

    <fragment
        android:id="@+id/mainFragment"
        android:name="ru.vadimdorofeev.myapplication.MainFragment"
        android:label="fragment_main"
        tools:layout="@layout/fragment_main" />
```

...

</navigation>

Если нужно создать вложенный граф, то следует выделить требуемые фрагменты (с клавишей Shift), нажать на одной из них правой кнопкой мыши и выбрать Move to Nested Graph. В XML-коде такой вложенный граф будет представлять собой дочерний элемент `navigation`, который вложен в основной элемент `navigation`.

После создания графа навигации следует добавить хост на основе `NavHostFragment` (обычно используется `FragmentContainerView`) в ту активность, в которой будет осуществляться основная навигация:

```
<?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"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <androidx.fragment.app.FragmentContainerView
        android:id="@+id/nav_host_fragment"
        android:name="androidx.navigation.fragment.NavHostFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:defaultNavHost="true"
        app:navGraph="@navigation/nav_graph" />

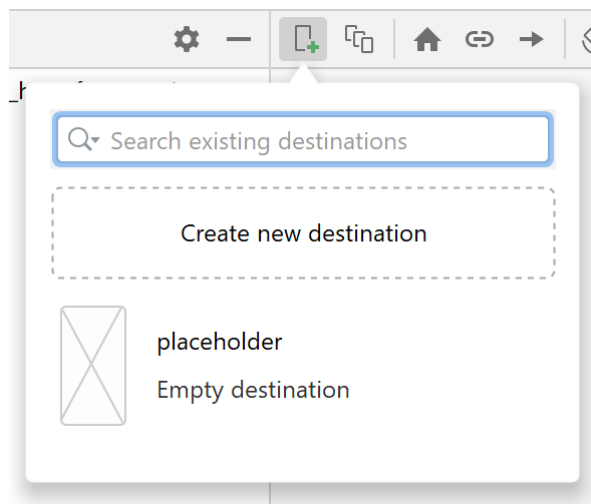
</LinearLayout>
```

Важные моменты:

- атрибут `name` содержит имя класса реализации `NavHost`
- атрибут `navGraph` связывает хост с используемым графом навигации
- атрибут `defaultNavHost` делает данный хост основным, это позволяет ему обрабатывать нажатие кнопки «Назад». Если в активности несколько хостов (например, двухпанельный интерфейс), то только один хост должен быть основным.

## Пункты назначения

Помимо хоста нужно создать и пункты назначения: можно добавить существующие фрагменты, или создать новые с помощью нажатия кнопки на панели в редакторе навигационного графа:

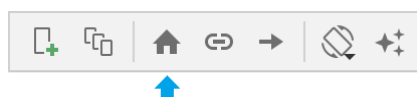


Вместо полноценного фрагмента можно создать временную *заглушку* (англ. *placeholder*), которую впоследствии заменить на настоящий фрагмент.

Если выделить фрагмент, то в панели Attributes можно посмотреть и настроить его параметры: идентификатор `id` для ссылки на пункт назначения в коде, имя класса `name`, название `label` (оно может быть отображено в пользовательском интерфейсе, поэтому рекомендуется делать его понятным пользователю). Для правильного отображения миниатюры рекомендуется также указать атрибут `layout`. Эти параметры также можно настроить в XML-коде:

```
<fragment
    android:id="@+id/secondFragment"
    android:name="ru.vadimdorofeev.myapplication.SecondFragment"
    android:label="fragment_second"
    tools:layout="@layout/fragment_second" />
```

Чтобы сделать какой-то фрагмент *начальным*, нужно выделить его и нажать кнопку с домиком на панели:

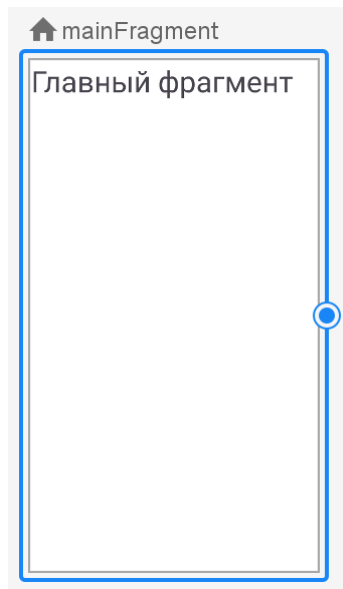


Или же можно указать этот фрагмент в элементе navigation в режиме просмотра XML:

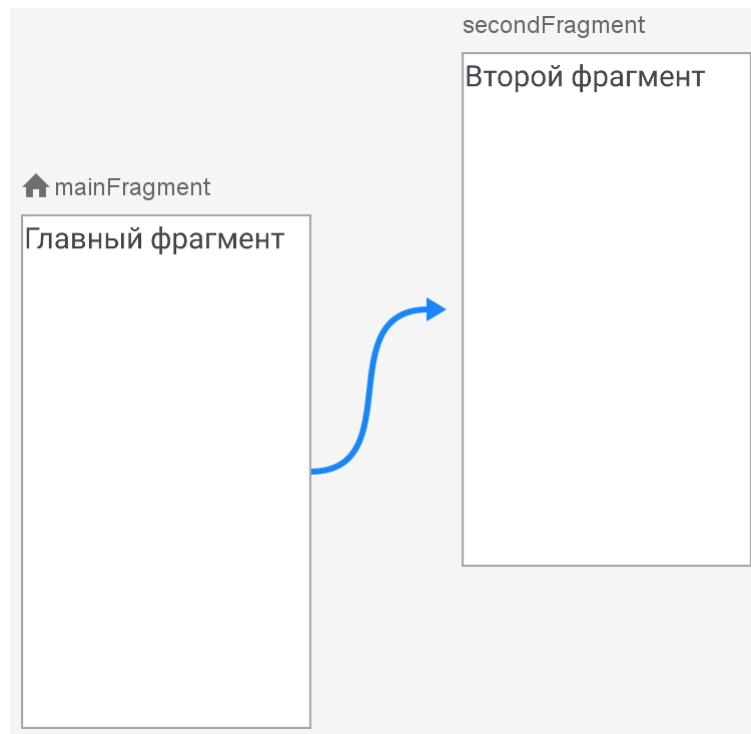
```
<navigation
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/nav_graph.xml"
    app:startDestination="@id/mainFragment">
```

## Действия

Для добавления действия нужно выделить фрагмент, справа появится кружок:



Нужно нажать мышкой на этот кружок и протянуть связь к тому фрагменту, к которому должен осуществляться переход:



Если выделить мышкой линию со стрелкой, то в панели Attributes можно посмотреть и настроить параметры действия: идентификатор действия (он генерируется автоматически) и пункт назначения. Как всегда эти параметры можно создать и в XML-коде:

```
<fragment
    android:id="@+id/mainFragment"
    android:name="ru.vadimdorofeev.myapplication.MainFragment"
    android:label="fragment_main"
    tools:layout="@layout/fragment_main" >
    <action
        android:id="@+id/action_mainFragment_to_secondFragment"
```

```
app:destination="@id/secondFragment" />
</fragment>
```

## Глубокие ссылки

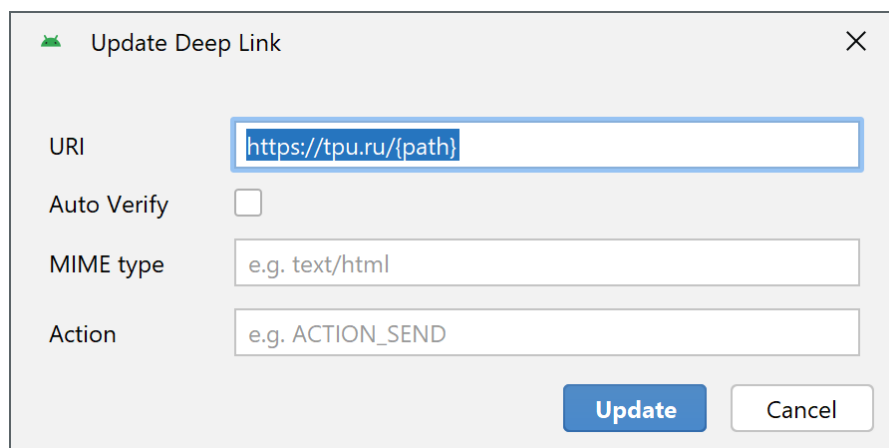
Глубокая ссылка (англ. deep link) ведёт непосредственно к пункту назначения, минуя все промежуточные. Глубокие ссылки могут быть *явным* и *неявным*.

*Явная глубокая ссылка* указывает конкретный фрагмент для открытия. Стек возврата при этом очищается, и если пользователь нажимает кнопку «Назад» то приложение будет закрыто и пользователь вернётся туда где сработала ссылка. Такой подход разумно применять для вызова приложения из виджета или уведомления.

Для явной глубокой ссылки используется `PendingIntent`, в котором указывается используемый навигационный граф, пункт назначения, и, при необходимости, передаваемые параметры:

```
val pendingIntent = NavDeepLinkBuilder(this)
    .setGraph(R.navigation.nav_graph)
    .setDestination(R.id.thirdFragment)
    .setArguments(args) // Bundle
    .createPendingIntent()
```

*Неявная глубокая ссылка* используется в тех случаях, когда приложение может обрабатывать какие-то действия по URL-ссылке или типы передаваемого содержимого, например, изображения. Для создания неявной ссылки нужно выделить фрагмент, который будет обрабатывать такую ссылку, и на панели Attributes в разделе Deep Links добавить ссылку:



Update Deep Link

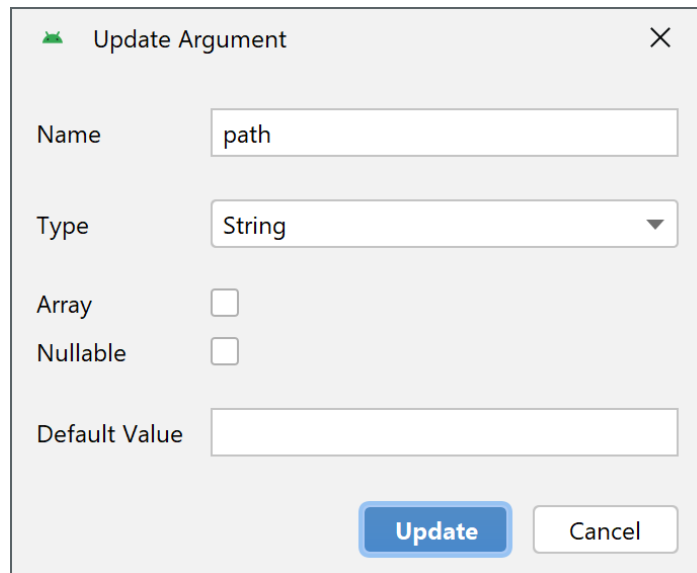
URI:

Auto Verify: ☐

MIME type:

Action:

В данном примере добавляется обработка ссылки на сайт `tpu.ru` (без `www!`), а путь `{path}`, который идёт после домена, будет передаваться в качестве параметра. Его нужно добавить в разделе Arguments:



Update Argument

Name: path

Type: String

Array: ☐

Nullable: ☐

Default Value:

Update Cancel

В XML-коде это будет выглядеть следующим образом:

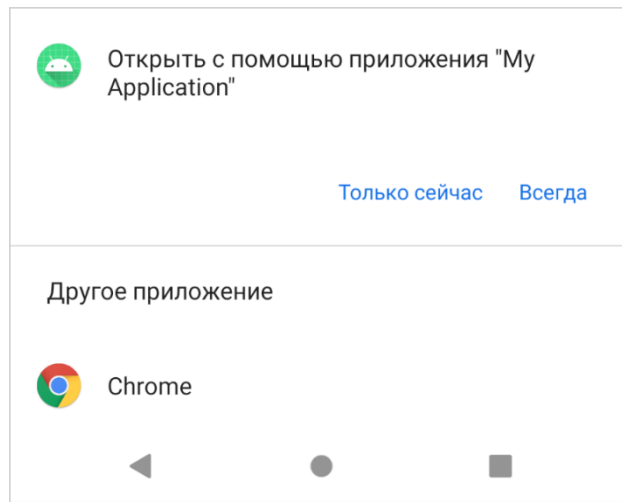
```
<fragment
    android:id="@+id/secondFragment"
    android:name="ru.vadimdorofeev.myapplication.SecondFragment"
    android:label="fragment_second"
    tools:layout="@layout/fragment_second">
    <deepLink
        android:id="@+id/deepLink"
        app:uri="https://tpu.ru/{path}" />
    <argument
        android:name="path"
        app:argType="string" />
</fragment>
```

Поскольку система должна знать про возможность приложения обрабатывать такие ссылки, нужно разместить информацию о навигационном графе в манифесте. При сборке приложения ссылки из графа будут извлечены и из них будут сформированы фильтры намерений:

```
<manifest ... >
    <application ...>
        <activity
            android:name=".MainActivity" ...>
            <nav-graph android:value="@navigation/nav_graph" />
        </activity>
    </application>
</manifest>
```

Теперь если в каком-либо внешнем приложении есть ссылка, которую наше приложение может обработать, система предложит пользователю выбрать приложение:





Фрагмент получает параметры в переменной `arguments`, это объект типа `Bundle` из которого можно извлечь данные обычным образом:

```
path = arguments?.getString("path") ?: ""
```

## Контроллер

`NavController` — это центр навигации, он отслеживает места назначения, которые посетил пользователь, его методы позволяют пользователю перемещаться между пунктами назначения.

Каждому созданному `NavHost` соответствует свой собственный `NavController`, для его получения нужно обратиться к элементу `NavHostFragment`:

```
val navHostFragment = supportFragmentManager
    .findFragmentById(R.id.nav_host_fragment) as NavHostFragment
val navController = navHostFragment.navController
```

Для перехода по графу служит метод `navigate`, которому можно передать идентификатор пункта назначения или действия:

```
navController.navigate(R.id.secondFragment)
navController.navigate(R.id.action_mainFragment_to_secondFragment)
```

Если фрагменту пункта назначения требуются аргументы, то используется объект класса `Bundle`:

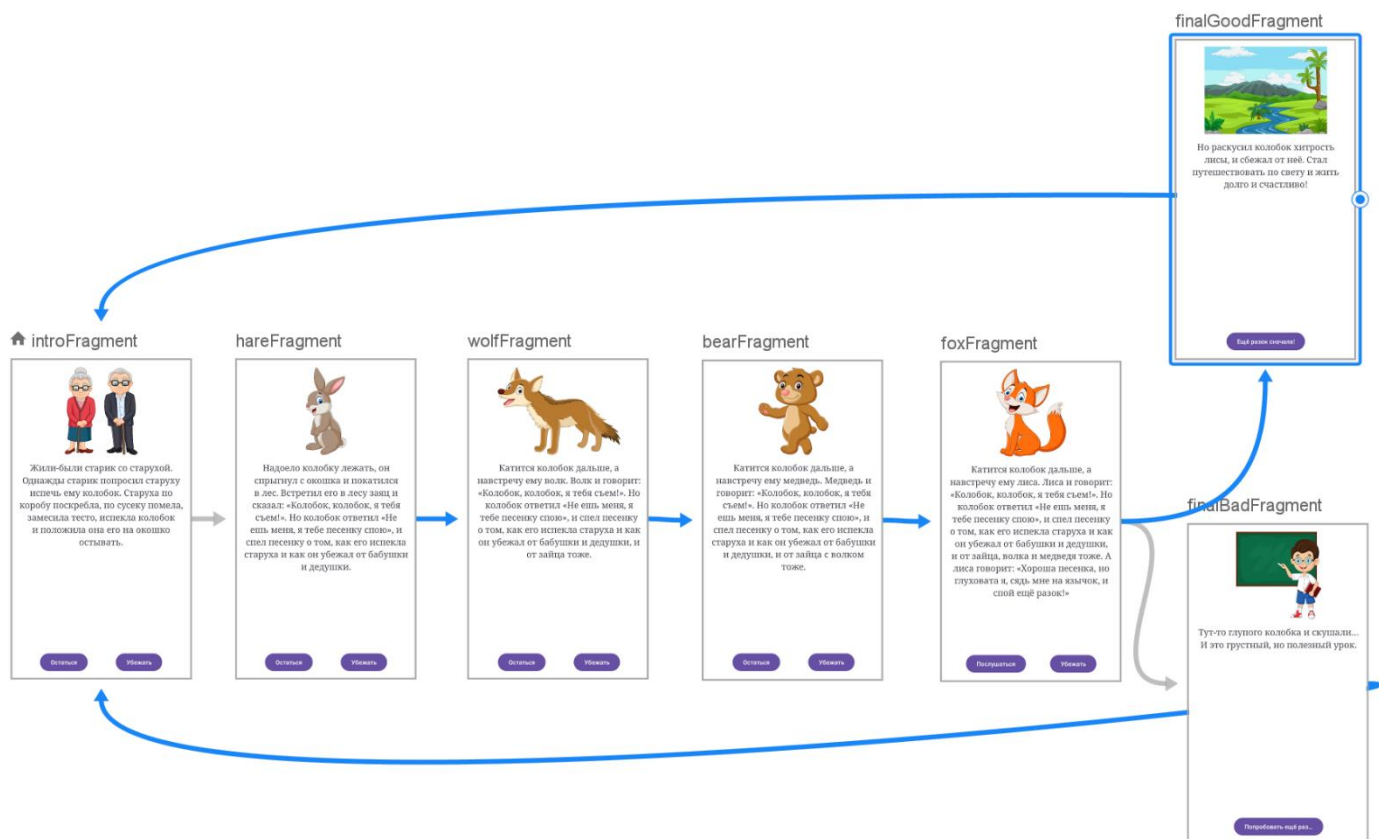
```
val bundle = Bundle()
bundle.putString("path", "/some/path/")
navController.navigate(R.id.secondFragment, bundle)
```

## Задание

Разработать приложение с интерактивной сказкой «Колобок». Приложение состоит из ряда фрагментов, отображающих иллюстрированные части сказки. Переходы между фрагментами осуществляются по нажатию кнопок в нижней части экрана. Иллюстрации можно взять из приложенного архива.

Вместо сказки «Колобок» по желанию можно использовать какую-либо другую историю, но механизм приложения должен остаться примерно таким же.

Навигационный граф для приложения может выглядеть примерно таким образом:



При желании допускается использовать один и тот же фрагмент несколько раз, передавая ему в качестве параметров информацию о том какая часть истории должна отображаться.

### Фрагмент intro

- Картинка: elders
- Текст: Жили-были старик со старухой. Однажды старик попросил старуху испечь ему колобок. Старуха по коробу поскребла, по сусеку помела, замесила тесто, испекла колобок и положила она его на окошко остывать.
- Кнопки выбора:
  - Убежать: переход на фрагмент hare
  - Остаться: переход на фрагмент final\_bad

### Фрагмент hare

- Картинка: hare
- Текст: Надоело колобку лежать, он спрыгнул с окошка и покатился в лес. Встретил его в лесу заяц и сказал: «Колобок, колобок, я тебя съем!». Но колобок ответил «Не ешь меня, я тебе песенку спою», и спел песенку о том, как его испекла старуха и как он убежал от бабушки и дедушки.
- Кнопки выбора:
  - Убежать: переход на фрагмент wolf
  - Остаться: переход на фрагмент final\_bad

### ***Фрагмент wolf***

- Картинка: wolf
- Текст: Катится колобок дальше, а навстречу ему волк. Волк и говорит: «Колобок, колобок, я тебя съем!». Но колобок ответил «Не ешь меня, я тебе песенку спою», и спел песенку о том, как его испекла старуха и как он убежал от бабушки и дедушки, и от зайца тоже.
- Кнопки выбора:
  - Убежать: переход на фрагмент bear
  - Остаться: переход на фрагмент final\_bad

### ***Фрагмент bear***

- Картинка: bear
- Текст: Катится колобок дальше, а навстречу ему медведь. Медведь и говорит: «Колобок, колобок, я тебя съем!». Но колобок ответил «Не ешь меня, я тебе песенку спою», и спел песенку о том, как его испекла старуха и как он убежал от бабушки и дедушки, и от зайца с волком тоже.
- Кнопки выбора:
  - Убежать: переход на фрагмент fox
  - Остаться: переход на фрагмент final\_bad

### ***Фрагмент fox***

- Картинка: fox
- Текст: Катится колобок дальше, а навстречу ему лиса. Лиса и говорит: «Колобок, колобок, я тебя съем!». Но колобок ответил «Не ешь меня, я тебе песенку спою», и спел песенку о том, как его испекла старуха и как он убежал от бабушки и дедушки, и от зайца, волка и медведя тоже. А лиса говорит: «Хороша песенка, но глуховата я, сядь мне на язычок, и спой ещё разок!»
- Кнопки выбора:
  - Послушаться: переход на фрагмент final\_bad
  - Убежать: переход на фрагмент final\_good

### ***Фрагмент final\_bad***

- Картинка: final\_bad
- Текст: Тут-то глупого колобка и скушали... И это грустный, но полезный урок.
- Кнопка:
  - Попробовать ещё раз...: переход на фрагмент intro

### ***Фрагмент final\_good***

- Картинка: final\_good
- Текст: Но раскусил колобок хитрость лисы, и сбежал от неё. Стал путешествовать по свету и жить долго и счастливо!
- Кнопка:
  - Ещё разок сначала!: переход на фрагмент intro