Практическая работа 2. Фрагменты

Для большинства приложений одного экрана недостаточно. До настоящего момента мы рассматривали приложения с одним экраном; для простых приложений этого хватает. А если в вашем приложении более сложные требования? В этой работе вы научитесь использовать фрагменты и компонент `Navigation` для построения приложений с несколькими экранами. Вы узнаете, что фрагменты похожи на субактивности, имеющие собственные методы. Вы научитесь проектировать эффективные графы навигации. В последней части работы представлен навигационный хост и навигационный контроллер; вы узнаете, как они используются для перемещения в приложениях.

У всех приложений, которые вы строили до сих пор, было кое-что общее: все они состояли из одного экрана. Каждое приложение было единственной активностью с соответствующим макетом, который определяет внешний вид приложения и его взаимодействие с пользователем. Однако большинство реальных приложений содержит более одного экрана. Например, почтовый клиент может использовать один экран для создания сообщений и другой экран для вывода списка полученных сообщений. Приложение-календарь может выводить список событий на одном экране и подробное описание отдельного события на другом.

Чтобы показать, как строятся многоэкранные приложения, мы создадим приложение `Secret Message`. Это приложение состоит из заставки, второго экрана, на котором пользователь вводит сообщение, и третьего экрана для вывода зашифрованной версии сообщения.

Каждый экран является фрагментом

Приложение `Secret Message` состоит из 2 разных экранов. Каждый экран будет построен в виде отдельного фрагмента. Фрагмент представляет собой своего рода вложенную активность, которая отображается внутри макета другой активности. Фрагмент содержит код Kotlin, который управляет его поведением, и макет, определяющий его внешний вид.

Два фрагмента, которые будут использоваться в нашем приложении:

WelcomeFragment Главный экран приложения, на котором должен размещаться краткий вводный текст и кнопка. Кнопка используется для перехода к следующему экрану — `MessageFragment`.

MessageFragment На этом экране пользователь вводит сообщение в текстовом поле.

Переходы между экранами с использованием компонента Navigation

Для перехода между фрагментами удобнее всего использовать компонент `Android Navigation`. Компонент `Navigation` входит в `Android Jetpack` и помогает реализовать стандартный механизм навигации в приложении.

Чтобы использовать компонент `Navigation` в приложении `Secret Message`, мы включили в него одну активность с именем `MainActivity`.

В ходе навигации по приложению активность последовательно выводит все фрагменты:

Создание нового проекта

Для приложения `Secret Message` нам потребуется новый проект. Выберите вариант Empty Activity, введите имя «SecretMessage» и имя пакета «com.hfad.secretmessage», подтвердите каталог для хранения проекта по умолчанию. Убедитесь в том, что выбран язык Kotlin и минимальный уровень SDK API 29, чтобы приложение работало на большинстве устройств Android.

Добавление строковых ресурсов

Прежде чем создавать какие-либо фрагменты, необходимо добавить в проект несколько строковых ресурсов. Они будут использоваться для вывода текста в макетах фрагментов: надписей на кнопках, пояснительного текста на первом экране. Чтобы добавить строковые ресурсы, откройте файл `strings.xml` в папке *SecretMessage/app/src/main/res/values*, а затем добавьте в него следующие ресурсы (выделены жирным шрифтом):

```
<resources>
 <string name="app_name">Secret Message</string>
 <string name="welcome_text">Welcome to the Secret Message app!
 Use this app to encrypt a secret message.
 Click on the Start button to begin.</string>
 <string name="start">Start</string>
 <string name="message_hint">Please enter your secret message</string>
 <string name="next">Next</string>
 <string name="encrypt_text">Here is your encrypted message:</string>
</resources>
```

Замечание: после выхода компонента `Navigation` команда Android рекомендует использовать фрагменты для построения многоэкранных приложений. Компонент `Navigation` предназначен прежде всего для работы с фрагментами, поэтому сейчас этот способ реализации навигации считается стандартным.

Добавление фрагмента WelcomeFragment

Мы собираемся добавить в проект фрагмент с именем `WelcomeFragment`. Он станет первым экраном, который видит пользователь при запуске приложения. В нем будет выводиться краткое описание приложения и кнопка. Чтобы добавить фрагмент, выделите пакет com.hfad.secretmessage в папке *app/src/main/java* folder на панели проекта, откройте меню `File` и выберите команду `New→Fragment→Fragment(Blank)`. Вам будет предложено указать, как вы хотите настроить конфигурацию нового фрагмента. Введите имя фрагмента «WelcomeFragment» и имя макета «fragment_welcome». Убедитесь в том, что для проекта выбран язык Kotlin, и щелкните на кнопке Finish.

При создании нового фрагмента Android Studio добавляет в проект два файла: файл с кодом Kotlin, управляющий поведением фрагмента, и файл макета, описывающий его внешний вид. Начнем с рассмотрения кода Kotlin. Перейдите к пакету *com.hfad.secretmessage* в папке *app/src/main/java* и

откройте файл `WelcomeFragment.kt`. Затем замените код, сгенерированный Android Studio, следующим кодом:

```
package com.hfad.secretmessage
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
class WelcomeFragment : Fragment() {
    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
                              savedInstanceState: Bundle?): View? {
        // Заполнение макета для этого фрагмента
        return inflater.inflate(R.layout.fragment_welcome, container, false)
    }
}
```

Приведенный выше код определяет базовый фрагмент. Как видите, код фрагмента очень похож на код активности. Однако вместо расширения `AppCompatActivity` в нем расширяется `Fragment`.

Класс `androidx.fragment.app.Fragment` является частью Android Jetpack и используется для определения базового фрагмента. Он предоставляет новейшую функциональность фрагмента, при этом сохраняя обратную совместимость со старыми версиями Android. Фрагмент переопределяет метод `onCreateView()`, который вызывается в тот момент, когда Android потребуется макет фрагмента. Этот метод переопределяется почти всеми фрагментами, поэтому его стоит рассмотреть более подробно.

Метод onCreateView() фрагмента

Метод `onCreateView()` вызывается в тот момент, когда Android потребуется обратиться к макету фрагмента. Переопределять этот метод необязательно, но так как вам необходимо реализовать его каждый раз, когда вы определяете фрагмент с макетом, вы будете переопределять его практически для каждого создаваемого вами фрагмента.

Метод получает три параметра:

```
override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
                          savedInstanceState: Bundle?): View? {
}
```

Первый параметр, `LayoutInflater`, используется для заполнения макета фрагмента. Заполнение макета преобразует его представления XML в объекты.

Второй параметр, `ViewGroup?`, определяет представление `ViewGroup` в макете активности, используемое для отображения фрагмента.

Последний параметр, `Bundle?`, используется в том случае, если ранее вы сохранили состояние фрагмента, а теперь хотите восстановить его. Он работает по аналогии с параметром `Bundle?`,

передаваемым методу `onCreate()` активности.

Заполнение макета фрагмента и его возвращение

Метод `onCreateView()` возвращает `View?`— заполненную версию макета фрагмента. Для заполнения макета используется метод `inflate()` класса `LayoutInflater`:

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

Приведенный выше код эквивалентен вызову метода `setContentView()` активности, так как он используется для заполнения макета фрагмента и преобразования его в иерархию объектов View. Например, приведенный выше код заполняет макет фрагмента `WelcomeFragment` разметкой `fragment_welcome.xml`.

После того как макет фрагмента будет заполнен, иерархия View вставляется в макет активности и отображается на экране. Итак, мы рассмотрели код Kotlin `WelcomeFragment` и можем рассмотреть его макет.

Как говорилось выше, фрагменты используют файлы макетов для описания своего внешнего вида. Код макета фрагмента не отличается от кода макета активности, так что в коде макета фрагмента можно использовать любые представления и группы представлений, которые вам уже известны.

Мы заменим код макета по умолчанию, сгенерированный за нас средой Android Studio, линейным макетом, который содержит текстовое представление с кратким описанием макета, и кнопку, которая будет использоваться для перехода к следующему фрагменту позднее в этой главе. Откройте файл `fragment_welcome.xml` из папки *app/src/main/res/layout* и замените его содержимое следующим кодом:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    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:orientation="vertical"
    android:gravity="center_horizontal"
    tools:context=".WelcomeFragment">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:layout_marginTop="20dp"
        android:textSize="20sp"
        android:text="@string/welcome_text" />
    <Button
        android:id="@+id/start"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:layout_marginTop="32dp"
        android:text="@string/start" />
</LinearLayout>
```

И это весь код, который понадобится для фрагмента `WelcomeFragment` (и его макета) в данный момент. Теперь разберемся, как отобразить его в приложении.

Отображение фрагмента в FragmentContainerView

Чтобы вывести фрагмент, необходимо включить его в макет активности. Например, в данном приложении мы собираемся отобразить фрагмент `WelcomeFragment`, добавив его в файл макета `MainActivity activity_main.xml`. Для добавления фрагмента в макет используется `FragmentContainerView`. Это разновидность `FrameLayout`, используемая для отображения фрагментов, а для ее добавления в файл макета используется код следующего вида:

```
<androidx.fragment.app.FragmentContainerView
 android:id="@+id/fragment_container_view"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:name="com.hfad.secretmessage.WelcomeFragment" />
```

Чтобы указать, какой фрагмент необходимо отобразить, присвойте атрибуту `android:name` представления `FragmentContainerView` полное имя фрагмента вместе с пакетом. В приложении `SecretMessage` должен отображаться фрагмент с именем WelcomeFragment из пакета `com.hfad.secretmessage`, поэтому значение атрибута `android:name` задается следующим образом:

```
android:name="com.hfad.secretmessage.WelcomeFragment"
```

Когда Android создает макет активности, `FragmentContainerView` заполняется объектом `View` , возвращаемым методом `onCreateView()` фрагмента. Этот объект `View` содержит пользовательский интерфейс фрагмента, так что вы можете рассматривать `FragmentContainerView` как зарезервированное место, в которое должен быть вставлен макет фрагмента:

Обновление кода activity_main.xml

Мы хотим, чтобы в активности `MainActivity` отображался фрагмент `WelcomeFragment`, а это означает, что в ее макет нужно добавить `FragmentContainerView`.

Ниже приведен полный код `activity_main.xml`: обновите код, чтобы он включал приведенные ниже изменения:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.fragment.app.FragmentContainerView
    xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/fragment_container_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:name="com.hfad.secretmessage.WelcomeFragment"
    tools:context=".MainActivity" />
```

Полный код MainActivity.kt

Чтобы в `MainActivity` отображался фрагмент, никакой дополнительный код Kotlin добавлять не нужно, потому что элемент `FragmentContainerView` макета сделает все за вас. Вам остается лишь проследить за тем, чтобы код в `MainActivity.kt` выглядел так:

```
package com.example.android.secretmessage

import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity

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

Создание MessageFragment

К настоящему моменту мы создали фрагмент с именем `WelcomeFragment`, который отображается в макете `MainActivity`. Затем мы создадим новый фрагмент с именем `MessageFragment`; приложение будет переходить к этому фрагменту, когда пользователь щелкает на кнопке Start в `WelcomeFragment`. Фрагмент `MessageFragment` добавляется точно так же, как добавлялся фрагмент `WelcomeFragment`. Выделите пакет `com.hfad.secretmessage` в папке *app/src/main/java* на панели `Project Explorer`, откройте меню `File` и выберите команду `New→Fragment→Fragment (Blank)`. Введите имя фрагмента «MessageFragment» и имя макета «fragment_message» и убедитесь в том, что выбран язык Kotlin. Затем щелкните на кнопке Finish, чтобы добавить фрагмент и его макет в проект

Обновление макета MessageFragment

При создании `MessageFragment` Android Studio добавляет в проект два новых файла: `MessageFragment.kt` (определяет поведение фрагмента) и `fragment_message.xml` (определяет его внешний вид). Мы обновим оба файла, начиная с макета. Фрагмент должен содержать текстовое поле, в котором пользователь будет вводить сообщение, и кнопку, которая позднее будет использоваться для навигации. Вы уже знаете код, используемый для добавления этих представлений, поэтому обновите разметку `fragment_message.xml` и приведите ее к следующему виду:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
 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:padding="16dp"
 android:orientation="vertical"
 android:gravity="center_horizontal"
 tools:context=".MessageFragment">
 <EditText
 android:id="@+id/message"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:textSize="20sp"
 android:hint="@string/message_hint"
 android:inputType="textMultiLine" />
 <Button
 android:id="@+id/next"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_marginTop="20dp"
 android:text="@string/next" />
</LinearLayout>
```

Обновление MessageFragment.kt

Код `MessageFragment` на Kotlin определяет поведение фрагмента. Пока от нас требуется совсем немного — нужно убедиться в том, что среда Android Studio не добавила лишний код, из-за которого фрагмент будет работать не так, как требуется.

Откройте пакет `com.hfad.secretmessage` из папки *app/src/main/java* и откройте файл `MessageFragment.kt`. Затем замените код, сгенерированный Android Studio, следующим:

```
package com.hfad.secretmessage
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import com.example.android.secretmessage.R

class MessageFragment : Fragment() {
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        //Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_message, container, false)
```

```
        }
    }
```

Приведенный выше код — все, что необходимо `MessageFragment.kt` для определения базового фрагмента. Как и код, приведенный для `WelcomeFragment`, он расширяет класс `Fragment` и переопределяет метод `onCreateView()`. Этот метод заполняет макет фрагмента и возвращает его корневое представление. Он вызывается каждый раз, когда приложению потребуется отобразить фрагмент.

Мы завершили написание всей разметки и кода Kotlin, необходимого для `MessageFragment`. Далее нужно заставить `WelcomeFragment` переходить к этому фрагменту по нажатии кнопки. Как же это делается?

Применение компонента Navigation для перехода между фрагментами

Как говорилось ранее в этой главе, стандартный способ навигации между фрагментами использует компонент Android `Navigation`. Компонент `Navigation` является частью Android Jetpack — семейства библиотек, плагинов и инструментов, которые вы добавляете в свои проекты. Он в высшей степени гибок и упрощает многие сложности навигации между фрагментами (например, транзакции фрагментов и операции со стеком возврата), которые прежде реализовывались намного сложнее. Механизм навигации между фрагментами состоит из трех основных частей:

Граф навигации Граф навигации содержит всю информацию, относящуюся к навигации, которая требуется вашему приложению. Он описывает возможные пути, по которым может переходить пользователь в процессе навигации в приложении. Граф навигации представляет собой ресурс в формате XML, но обычно он редактируется в визуальном редакторе.

Хост навигации Хост навигации представляет собой пустой контейнер, используемый для отображения фрагмента, к которому вы переходите. Хост навигации добавляется в макет активности.

Контроллер навигации Контроллер навигации управляет тем, какой фрагмент отображается в хосте навигации при переходах пользователя в приложении. Для взаимодействия с контроллером навигации используется код Kotlin.

Все три части будут использоваться для реализации навигации в приложении `Secret Message`. Начнем с добавления библиотеки компонента `Navigation` в проект.

Добавление компонента Navigation в проект с использованием Gradle

Для включения в приложение дополнительных библиотек, инструментов и плагинов необходимо внести изменения в файлы `build.gradle`. Когда вы создаете новый проект, Android Studio автоматически включает два таких файла: для проекта и для приложения. Чтобы добавить компонент `Navigation`, нужно отредактировать обе версии `build.gradle`. Начнем с обновления версии проекта.

Добавление зависимости в файл build.gradle приложения

Затем зависимость для библиотеки необходимо добавить в версию файла `build.gradle` приложения. Откройте файл *SecretMessage/app/build.gradle* и добавьте следующую строку (выделенную жирным шрифтом) в разделе dependencies:

```
dependencies {

    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.appcompat:appcompat:1.6.1'
    implementation 'com.google.android.material:material:1.11.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    testImplementation 'junit:junit:4.13.2'

    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'

    implementation "androidx.navigation:navigation-fragment-ktx:2.3.5"

}
```

После внесения этих изменений щелкните на ссылке `Sync Now`, появившейся в верхней части редактора кода. Она синхронизирует внесенные изменения в проекте и добавляет библиотеку

Создание графа навигации

После добавления главной библиотеки компонента `Navigation` в проект `SecretMessage` можно переходить к реализации навигации.

Начнем с добавления графа навигации в проект. Выберите папку *SecretMessage/app/src/main/res* на панели проекта, затем выберите команду `File→New→Android Resource File`. Введите имя файла «nav_graph», выберите тип ресурса «Navigation» и щелкните на кнопке OK. Кнопка создает файл с пустым графом навигации `nav_graph.xml` в папке *SecretMessage/app/src/main/res/navigation*. После того как граф навигации будет создан, откройте его (если он не был открыт ранее) двойным щелчком на файле `nav_graph.xml` на панели проекта. Файл открывается в визуальном редакторе графов навигации, который выглядит так:

Добавление фрагментов в граф навигации

Мы хотим, чтобы пользователь переходил от `WelcomeFragment` к `MessageFragment`, поэтому эти фрагменты нужно добавить в граф навигации как цели. Цель представляет собой экран в приложении — обычно фрагмент, — к которому может переходить пользователь. Начнем с добавления `WelcomeFragment`, так как это первый экран, который видит пользователь при запуске приложения. Щелкните на кнопке `New Destination` в верхней части визуального редактора, выберите вариант «fragment_welcome» (макет WelcomeFragment). При этом `WelcomeFragment` добавляется в граф навигации.

Затем добавьте `MessageFragment` в граф навигации. Щелкните на кнопке `New Destination button` и выберите вариант «fragment_message». При этом в граф навигации добавляется второй фрагмент.

Соединение фрагментов действиями

Затем необходимо указать, что пользователь может перейти от `WelcomeFragment` к `MessageFragment`; для этого используются действия. Действия соединяют цели на графе навигации и определяют возможные пути, по которым может перемещаться пользователь в процессе навигации в приложении. Мы добавим действие для перехода от `WelcomeFragment` к `MessageFragment`, так как именно в этом направлении пользователь должен перемещаться в приложении. Наведите указатель мыши на `WelcomeFragment` в визуальном редакторе, затем щелкните на круге, появившемся у правого края, и перетащите его на `MessageFragment`. При этом два фрагмента соединяются стрелкой — действием:

Каждому действию необходим уникальный идентификатор

Каждое действие должно иметь уникальный идентификатор. Android использует этот идентификатор для определения того, какая цель должна отображаться при переходе пользователя в приложении.

Каждый раз, когда вы создаете действие, Android Studio присваивает ему идентификатор по умолчанию. Этот идентификатор, как и все остальные свойства действия, можно отредактировать на панели `Attributes` справа от графа навигации.

Действию, которое вы только что создали, должен быть присвоен идентификатор «action_welcomeFragment_to_ messageFragment», используемый в коде этой главы. Чтобы убедиться в этом, выделите действие (стрелку) в визуальном редакторе и щелкните на значении его атрибута id на панели Attributes. Этот идентификатор будет использоваться через несколько страниц.

Графы навигации как ресурсы XML

Граф навигации, как и макет, в действительности состоит из кода разметки XML. Чтобы просмотреть его, щелкните на кнопке `Code` в верхней части визуального редактора. Вот как выглядит разметка XML для графа навигации приложения

Secret Message nav_graph.xml:

```
<?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"
    app:startDestination="@id/welcomeFragment">
    <fragment
        android:id="@+id/welcomeFragment"
        android:name="com.hfad.secretmessage.WelcomeFragment"
        android:label="fragment_welcome"
        tools:layout="@layout/fragment_welcome" >
        <action
            android:id="@+id/action_welcomeFragment_to_messageFragment"
            app:destination="@id/messageFragment" />
```

```
    </fragment>
    <fragment
        android:id="@+id/messageFragment"
        android:name="com.hfad.secretmessage.MessageFragment"
        android:label="fragment_message"
        tools:layout="@layout/fragment_message" />
</navigation>
```

Как видите, `nav_graph.xml` содержит корневой элемент `<navigation>` и два элемента `<fragment>`: для `WelcomeFragment` и для `MessageFragment`. Элемент `<fragment>` для `WelcomeFragment` включает дополнительный элемент `<action>`, который описывает только что добавленное действие.

Итак, граф навигации создан, и мы можем перейти к следующей части компонента `Navigation`.

Добавление хоста навигации в макет при помощи FragmentContainerView

Как упоминалось ранее, компонент `Navigation` состоит из трех основных частей: графа навигации, определяющего возможные пути навигации; хоста навигации, отображающего цели; и контроллера навигации, управляющего тем, какая цель должна отображаться. Мы только что создали граф навигации, пора сделать следующий шаг — добавить хост навигации. Чтобы добавить хост навигации, следует включить его в макет активности. К счастью, компонент `Navigation` включает встроенный хост с именем `NavHostFragment`, так что вам не придется писать его самостоятельно. Он является субклассом `Fragment`, который реализует интерфейс `NavHost`. Так как `NavHostFragment` является разновидностью фрагмента, для его добавления в файл макета используется `FragmentContainerView`. Код выглядит примерно так:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.fragment.app.FragmentContainerView
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
android:id="@+id/nav_host_fragment"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:name="androidx.navigation.fragment.NavHostFragment"
app:navGraph="@navigation/nav_graph"
app:defaultNavHost="true" />
```

Приведенный выше код напоминает код `FragmentContainerView`, который вы уже видели, но он включает два дополнительных атрибута: `app:navGraph` и `app:defaultNavHost`. Атрибут `app:navGraph` сообщает хосту навигации, какой граф навигации тот должен использовать, — в данном случае `nav_graph`. xml. Граф навигации определяет фрагмент, который должен отображаться первым (начальную цель), и обеспечивает возможность перемещения пользователям между целями. Атрибут `app:defaultNavHost` обеспечивает взаимодействие хоста навигации с кнопкой возврата на устройстве: эта тема рассматривается в следующей главе.

Добавление NavHostFragment в activity_main.xml

Мы добавим хост навигации в макет MainActivity, использующий созданный нами граф навигации. Обновите код разметки `activity_main.xml`.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.fragment.app.FragmentContainerView
    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_host_fragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"

    android:name="androidx.navigation.fragment.NavHostFragment"
    app:navGraph="@navigation/nav_graph"
    app:defaultNavHost="true"
    tools:context=".MainActivity" />
```

Приложение должно переходить между фрагментами

Мы создали граф навигации и связали его с хостом навигации, который содержится в элементе `FragmentContainerView` в макете `MainActivity`. При запуске приложения будет отображаться `WelcomeFragment` — начальная цель в графе навигации. Последнее, что осталось сделать в этой главе, — реализовать переход от `WelcomeFragment` к `MessageFragment`, когда пользователь щелкал на кнопке `Start` в макете `WelcomeFragment`. Давайте разберемся, как это делается.

Добавление OnClickListener для кнопки

Чтобы переходить от `WelcomeFragment` к `MessageFragment`, сначала необходимо позаботиться о том, чтобы кнопка `Start` фрагмента `WelcomeFragment` реагировала на щелчки. Для этого мы добавим к ней слушатель `OnClickListener`.

Ранее мы добавили `OnClickListener` для кнопки активности. Для этого мы сначала получали ссылку на кнопку вызовом `findViewById()`, а затем вызывали ее метод `setOnClickListener`. Этот код содержался в методе `onCreate()` активности, так как именно он первым получает доступ к представлениям в своем макете.

Но когда требуется добавить `OnClickListener` к кнопке фрагмента, ситуация слегка меняется.

Код OnClickListener для фрагмента несколько отличается

Первое отличие заключается в том, что слушатель `OnClickListener` добавляется к кнопке фрагмента в методе `onCreateView()` фрагмента, а не в `onCreate()`. Дело в том, что фрагмент впервые получает доступ к своим представлениям в `onCreateView()`, поэтому этот метод лучше подходит для назначения `OnClickListener`.

Второе отличие заключается в том, что класс `Fragment` не содержит метод `findViewById()`, поэтому вы не сможете воспользоваться этим методом для получения ссылок на любые представления. Впрочем,

вместо этого можно вызвать `findViewById()` для корневого представления фрагмента.

А вот как выглядит код добавления `OnClickListener` для представления в коде фрагмента: мы добавим его в `WelcomeFragment` через пару страниц:

```
class WelcomeFragment : Fragment() {
    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
                              savedInstanceState: Bundle?): View? {
        val view = inflater.inflate(R.layout.fragment_welcome, container, false)
        val startButton = view.findViewById<Button>(R.id.start)
        startButton.setOnClickListener {
            //Код, выполняемый по щелчку на кнопке
        }
        return view
    }
}
```

Получение контроллера навигации

Каждый раз, когда вы хотите перейти к новому фрагменту, сначала необходимо получить ссылку на контроллер навигации. Для этого следует вызвать метод `findNavController()` для корневого объекта `View`. Например, следующий код получает ссылку на контроллер навигации, связанный с корневым объектом представления с именем view:

```
val navController = view.findNavController()
```

Выбор цели перехода при помощи действия

Получив контроллер навигации, вы отдаете ему команду перейти к новой цели, вызывая его метод `navigate()`. Этот метод получает один параметр: идентификатор действия навигации. Как вы, возможно, помните, при создании графа навигации мы включили действие для перехода от `WelcomeFragment` к `MessageFragment`. Этому действию был присвоен идентификатор “action_welcomeFragment_to_messageFragment”. Если передать этот идентификатор методу `navigate()` контроллера навигации, контроллер увидит, что действие переходит от `WelcomeFragment` к `MessageFragment` и использует его для перехода к новому фрагменту. Код выглядит так:

```
view.findNavController().navigate(R.id.action_welcomeFragment_to_messageFragment)
```

Когда пользователь щелкает на кнопке `Start` фрагмента `WelcomeFragment`, приложение должно переходить к `MessageFragment`. Соответственно, мы включаем следующий код в слушатель `OnClickListener` кнопки `Start`:

```
val view = inflater.inflate(R.layout.fragment_welcome, container, false)
val startButton = view.findViewById<Button>(R.id.start)
startButton.setOnClickListener {
    view.findNavController()
        .navigate(R.id.action_welcomeFragment_to_messageFragment)
}
```

Полный код WelcomeFragment.kt

Ниже приведен полный код `WelcomeFragment`; обновите файл `WelcomeFragment.kt` (изменения выделены жирным шрифтом):

```
package com.example.android.secretmessage
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Button
import androidx.navigation.findNavController
import com.example.android.secretmessage.R


class WelcomeFragment : Fragment() {
    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
                              savedInstanceState: Bundle?): View? {
        val view = inflater.inflate(R.layout.fragment_welcome, container, false)
        val startButton = view?.findViewById<Button>(R.id.start)
        startButton?.setOnClickListener {

view?.findNavController()?.navigate(R.id.action_welcomeFragment_to_messageFragment
)
        }
        return view
    }
}
```

Резюме:

- Фрагмент содержит код Kotlin и макет.
- Каждый фрагмент расширяет класс Fragment или один из его субклассов.
- Для добавления фрагментов в макет активности используется представление FragmentContainerView.
- Метод onCreateView() вызывается каждый раз, когда Android требуется макет фрагмента.
- Класс class не расширяет Activity.

- Фрагменты не содержат метод findViewById().
- Компонент Navigation входит в семейство библиотек, плагинов и модулей, которые добавляются в проект при помощи Gradle.
- Граф навигации описывает возможные точки перехода и пути навигации. Для описания этих путей используются действия.
- Хост навигации представляет собой пустой контейнер, используемый для отображения фрагмента, к которому вы переходите. Компонент Navigation включает хост навигации по умолчанию с именем NavHostFragment, который расширяет класс Fragment и реализует интерфейс NavHost.
- Контроллер навигации использует действия для управления тем, какой фрагмент должен отображаться в хосте навигации.

Отчет

- файл readme с разметкой и кодом Activity
- скриншот