Лекционно-практическое занятие

«Фрагменты в приложениях Android. Класс Fragment»

Фрагмент – часть интерфейса приложения, со своим жизненным циклом и, который можно использовать повторно. Фрагмент имеет собственную разметку и управляет ей. Фрагменты не могут жить сами по себе – они должны размещаться в Activity (каждая activity может иметь несколько фрагментов) или в другом фрагменте.

Зачем нужны фрагменты?

Организация приложения на основе нескольких activity не всегда может быть оптимальной. Мир ОС Android довольно сильно фрагментирован и состоит из множества устройств. И если для мобильных аппаратов с небольшими экранами взаимодействие между разными activity выглядит довольно неплохо, то на больших экранах - планшетах, телевизорах окна activity смотрелись бы не очень в силу большого размера экрана. Собственно, поэтому и появилась концепция фрагментов. На изображении ниже показано, как два модуля пользовательского интерфейса, определяемые фрагментами, могут быть объединены в одно действие для дизайна планшета, но разделены для дизайна телефона:

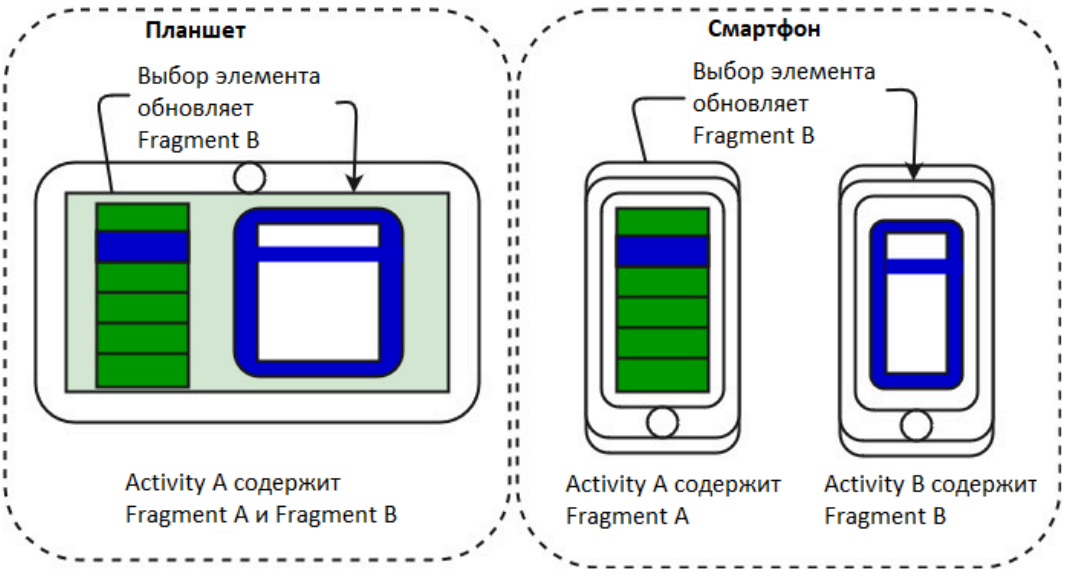


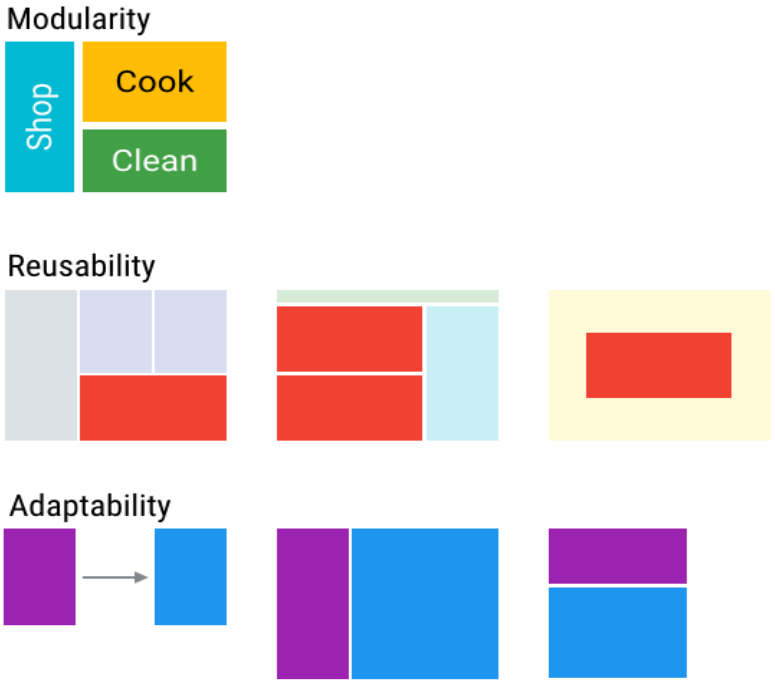
Рисунок 1 – Вид фрагментов на устройствах с разным размером экранов

Пример фрагментов для большего понимания:

Представьте на мгновение, что вы - суперзлодей. У вас много дел, поэтому вы можете нанять нескольких приспешников, чтобы они бегали по вашим поручениям и занимались стиркой и налогами в обмен на жилье и еду. Это похоже на взаимосвязь между действиями и фрагментами.

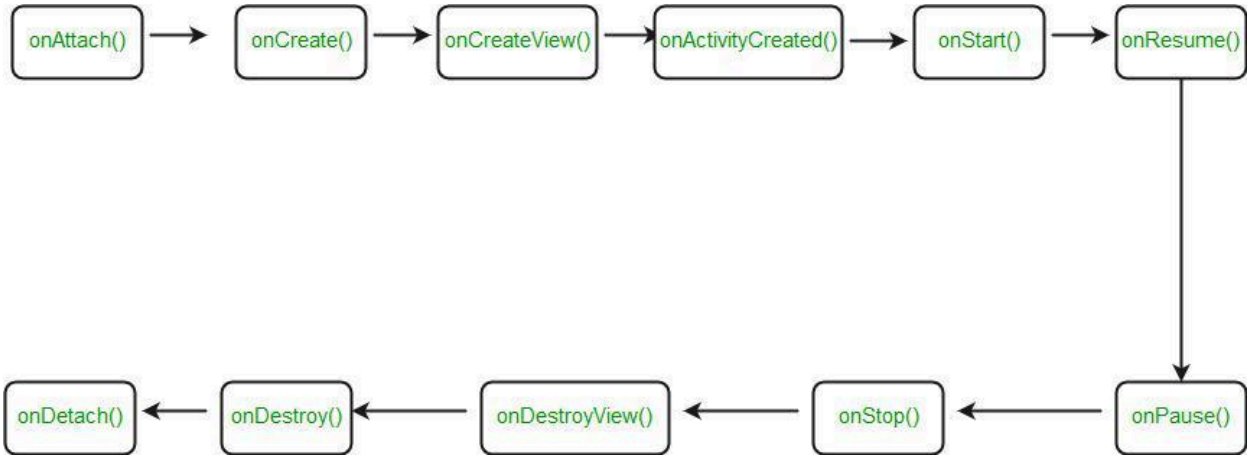
Точно так же, как вам не нужна армия маленьких помощников, чтобы выполнять ваши поручения, вам не обязательно использовать фрагменты. Однако, если вы будете правильно их использовать, они могут обеспечить:

- **Модульность**: разделение сложного кода на фрагменты для лучшей организации и обслуживания;
- **Возможность повторного использования**: размещение поведения или элементов пользовательского интерфейса во фрагментах, которыми могут совместно пользоваться несколько приложений;
- **Адаптивность**: представление разделов пользовательского интерфейса в виде отдельных фрагментов и использование разных макетов в зависимости от ориентации и размера экрана.



Жизненный фрагмент цикла:

Фрагменты Android имеют свой небольшой жизненный цикл, очень похожий на жизненный цикл активности Android.



Связь жизненных циклов активности и фрагмента.

Activity	Fragment	Событие
onCreate() создание активности	**onAttach(Activity)**	связывание фрагмента с активностью
	onCreate(Bundle)	создание фрагмента
	onCreateView(…)	фрагмент заполняет своё представления
	onActivityCreated(Bundle)	завершение создания активности
onStart()	**onStart()**	фрагмент становится **видимым**
onResume()	**onResume()**	фрагмент виден и активно работает
onPause()	**onPause()**	фрагмент перестает быть активным
onStop()	**onStop()**	фрагмент становится **невидимым**
onDestroy() уничтожение	**onDestroyView()**	фрагмент может освободить любые ресурсы, связанные с его представлением
	onDestroy()	фрагмент может освободить любые другие ресурсы, созданные им
	onDetach()	перед окончательным разрывом связи между фрагментом и активностью

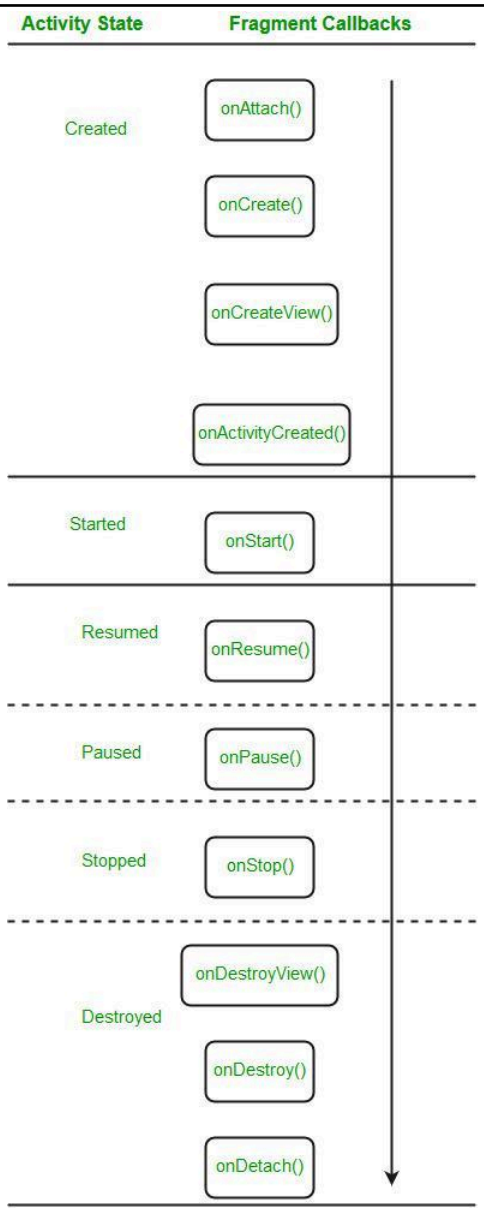
Управление жизненным циклом фрагмента

Фрагмент существует в трех состояниях:

- **Возобновлено**: Фрагмент виден в запущенной активности;
- **Приостановлено**: другая активность находится на переднем плане и имеет фокус, но активность, в которой находится этот фрагмент, все еще видна (активность переднего плана частично прозрачна или не покрывает весь экран);
- **Остановлено**: Фрагмент не отображается. Либо хост-активность была остановлена, либо фрагмент был удален из активности, но добавлен в

стек возврата. Остановленный фрагмент все еще активен (вся информация о состоянии и участниках сохраняется в системе). Однако он больше не отображается пользователю и будет уничтожен, если активность будет уничтожена.

Влияние жизненного цикла активности на жизненный цикл фрагмента:

Определение и использование фрагментов Чтобы определить новый фрагмент, мы либо расширяем класс android.app.Fragment , либо один из его подклассов.	

Типы фрагментов

- **Фрагменты одного кадра**: используются для портативных устройств, таких как мобильные телефоны. Здесь мы можем показать только один фрагмент в качестве вида;

- **Фрагменты списка**: фрагменты, имеющие специальный вид списка;
- **Транзакция фрагментов**: Используя транзакцию фрагментов, мы можем переместить один фрагмент в другой фрагмент.

Транзакция фрагментов

Во время работы с приложением FragmentManager может добавлять, удалять, заменять и выполнять другие действия с фрагментами в ответ на взаимодействие с пользователем. Каждый набор изменений фрагментов, которые вы фиксируете, называется **транзакцией**, и вы можете указать, что делать внутри транзакции, используя API, предоставляемые FragmentTransaction классом. Вы можете объединить несколько действий в одну транзакцию - например, транзакция может добавлять или заменять несколько фрагментов. Эта группировка может быть полезна, когда на одном экране отображается несколько родственных фрагментов, например, при разделении представлений.

Задача приложения: Создать приложение, состоящее из трех кнопок: Обо мне, Перевод единиц и Случайное число. Каждая кнопка будет переводить на новое активити, которое состоит из двух фрагментов: первый фрагмент просит ввода некоторых данных, а затем эти данные посылает во второй фрагмент для отображения.

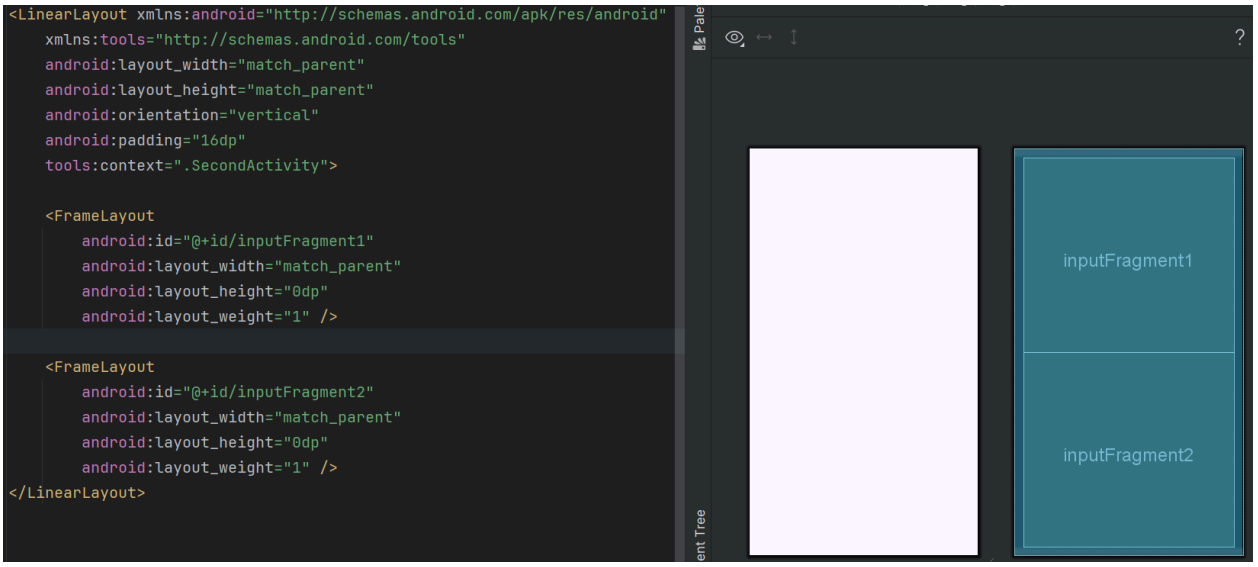
Этапы проектирования:

1) Создание активити
2) Создание и заполнение фрагментов
3) Добавление фрагментов в активити
4) Передача данных между фрагментами

1) Создание активити

В предыдущем проекте создавалось 7 экранов, но для этого проекта нужно создать всего 2: Главный экран, экран с вводом и выводом данных. Главный экран создан в проекте по умолчанию – MainActivity.kt, создайте вторую активити и назовите ее «**SecondActivity**».

В разметке **activity_second.xml** создайте 2 контейнера FrameLayout, которые позже будут являться фрагментами:



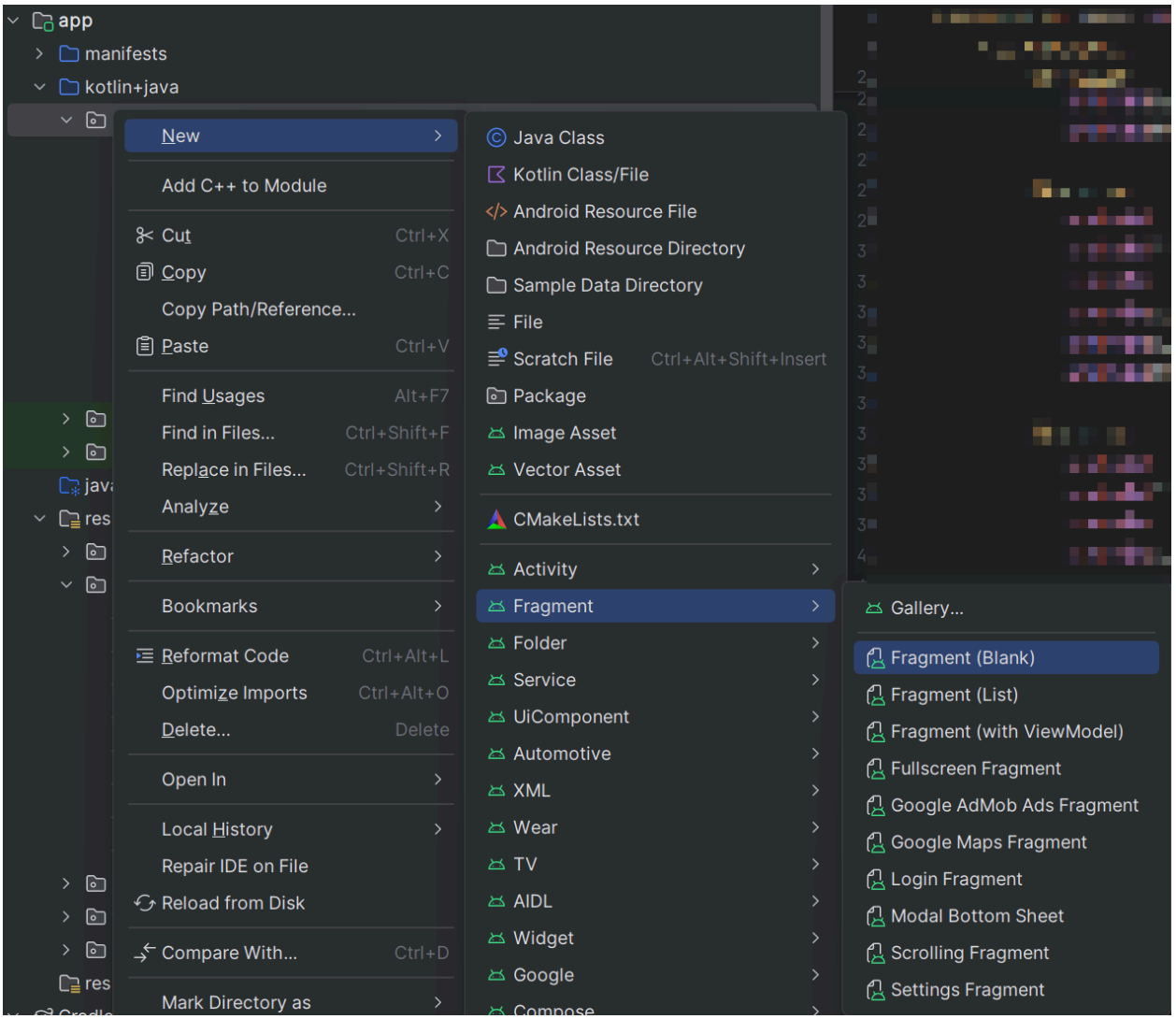
2) Создание и заполнение фрагментов

2.1. Создание фрагментов

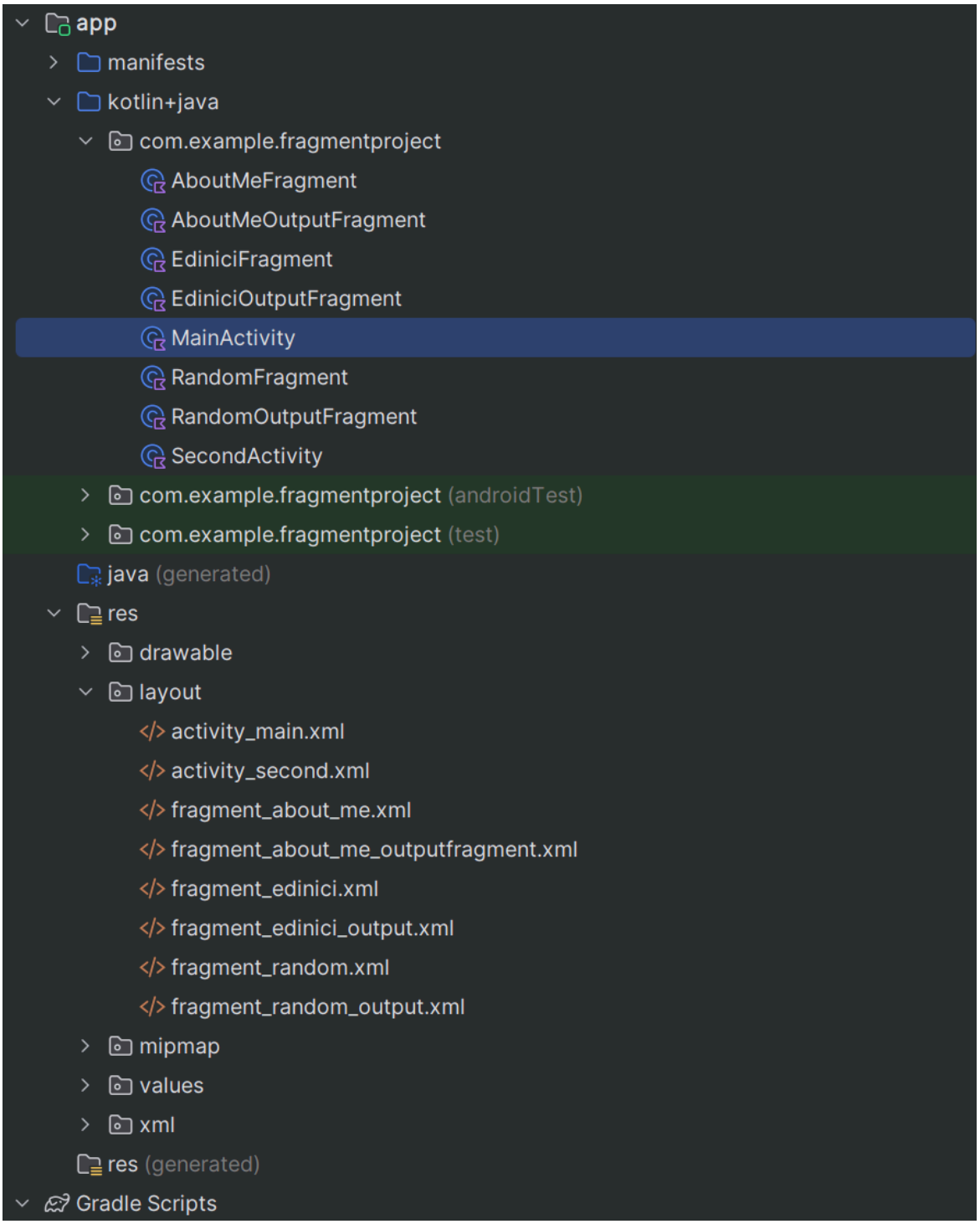
Для каждого экрана, кроме главного, нужно создать по 2 фрагмента, т.е. в проекте их будет 6.

Сам фрагмент всегда состоит из двух файлов (как активити): разметка (состоит из элементов, которые видит пользователь на экране (кнопки, текст и т.д.)) и класс (то как должны работать элементы на экране).

Чтобы быстро создать все 6 фрагментов вместе с их классами и разметками нажмите ПКМ на папке, где лежат классы созданных активити, далее выберите **New – Fragment – Fragment (Blank)**:



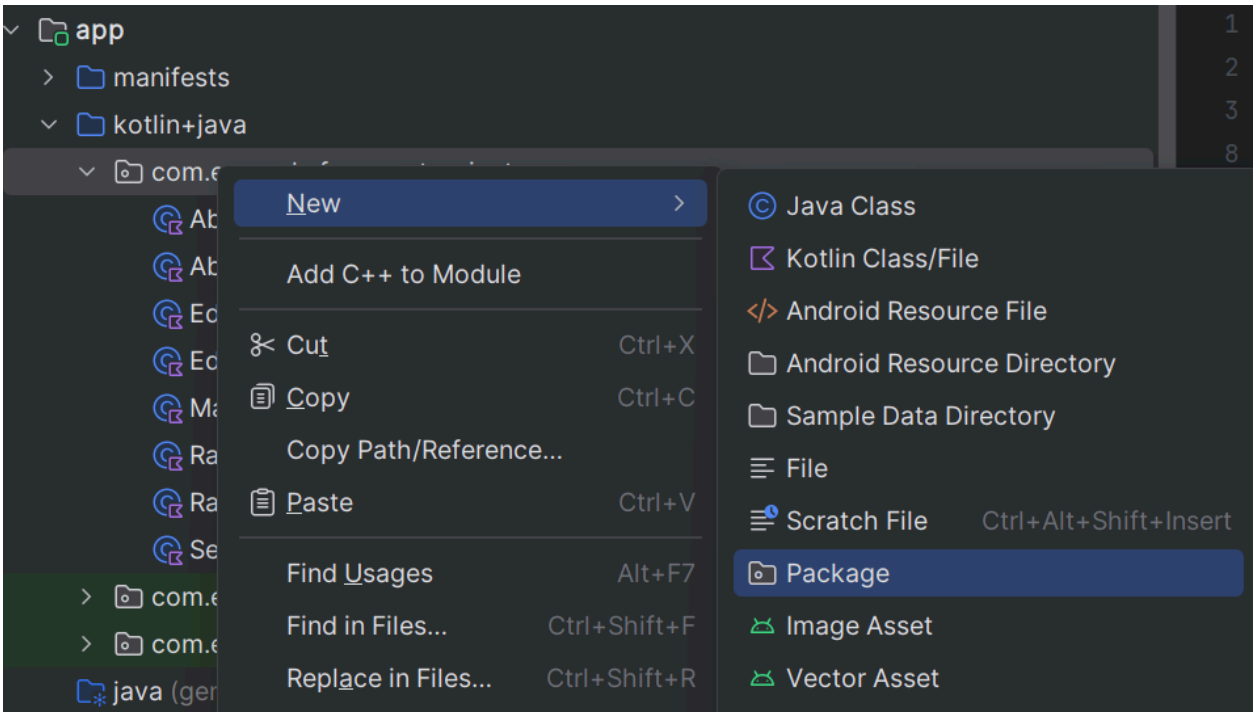
Затем дайте имя, которое будет соответствовать каждой кнопке, например, для кнопки «Обо мне», как говорилось выше, нужно создать 2 фрагмента: один для ввода данных (фрагмент с именем «AboutMeFragment»), другой для вывода данных (фрагмент с именем «AboutMeOutputFragment»). В итоге корневой вид проекта должен выглядеть следующим образом:



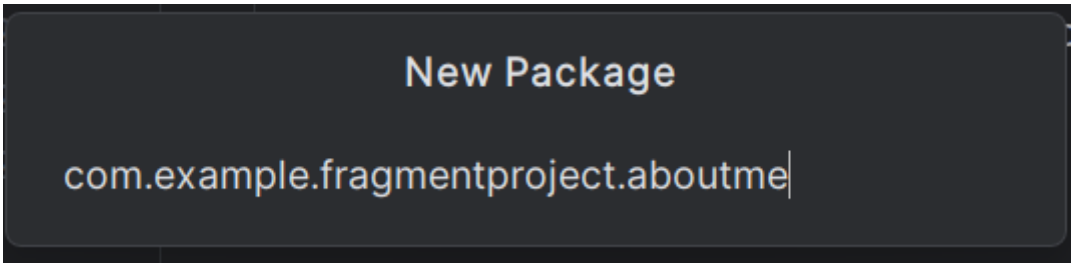
Как видно на рисунке выше очень много фрагментов и всего 2 активити. Но это усложняет поиск, поэтому можно:

- создать папку, в которой будут все фрагменты;
- создать папку для фрагментов каждой кнопки.

Выберем второй способ. Для этого щелкните ПКМ по папке со всеми классами проекта, затем выберите **New – Package**:



В появившемся окне введите в конце строки «aboutme» и нажмите Enter:



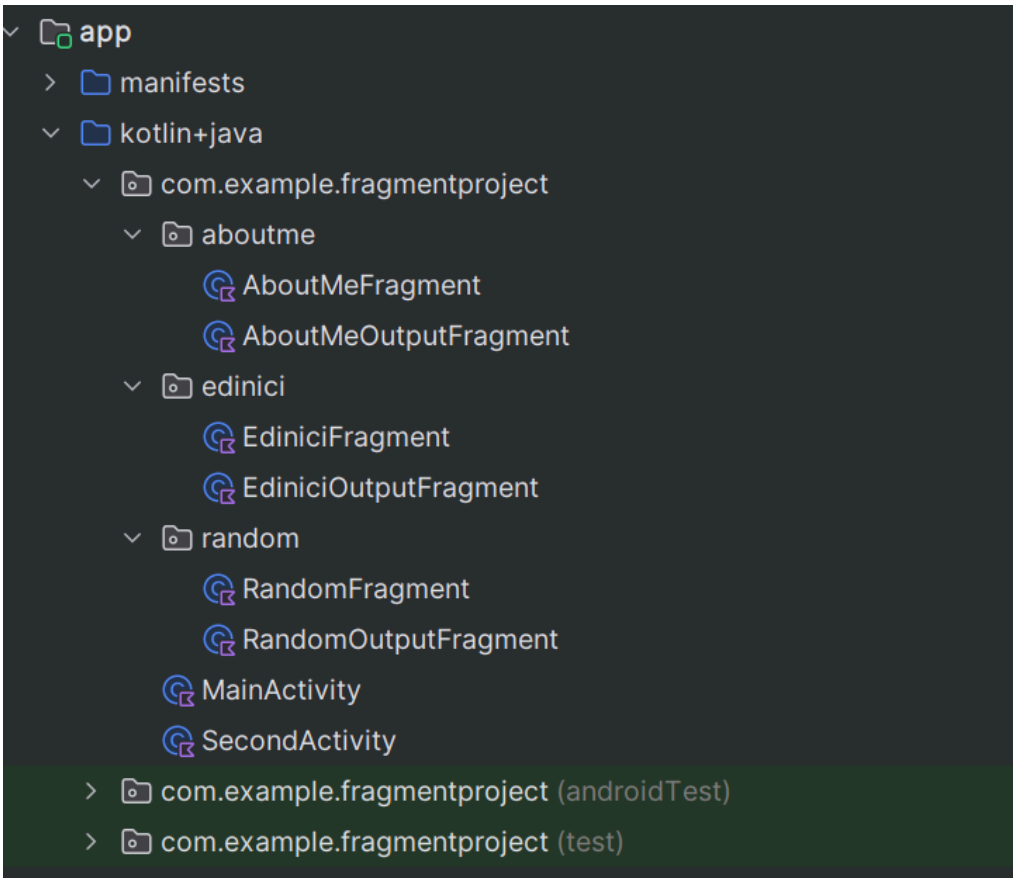
Сделайте еще 2 пакета.

По умолчанию код в классе фрагмента выглядит сложно и не понятно. Удалите лишнее в каждом фрагменте, чтобы осталось как на изображении ниже:

```kotlin
package com.example.fragmentproject.aboutme

import ...

class AboutMeFragment : Fragment() {
    lateinit var binding: FragmentAboutMeBinding

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        binding = FragmentAboutMeBinding.inflate(layoutInflater)
        return binding.root
    }
}
```

Теперь перетащим все что относится к кнопке «Обо мне» в папку «aboutme». Откроется окно с рефакторингом, просто нажмите на кнопку «Refactor».

Теперь корневой вид проекта выглядит удобнее:

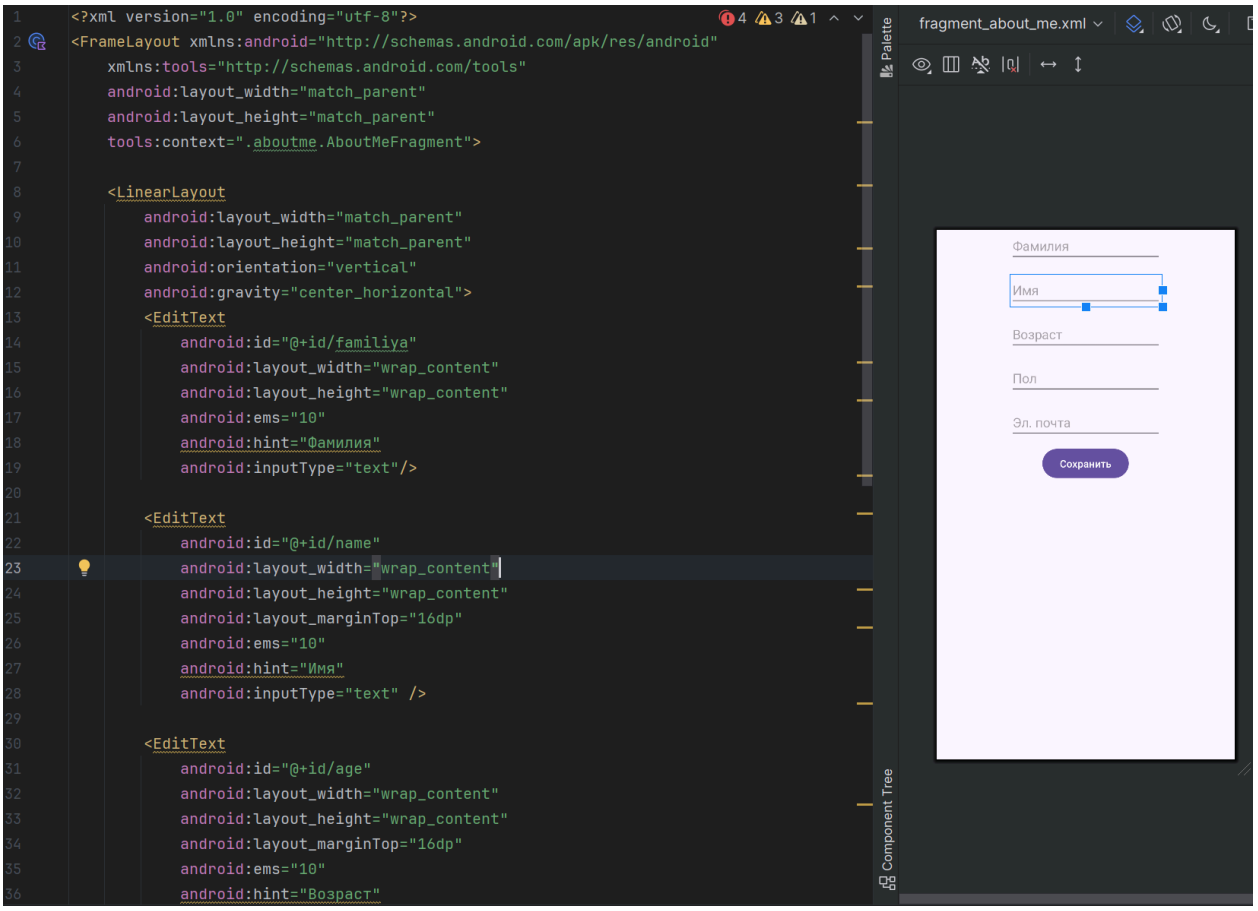


Делать тоже самое для разметок не будем!

2.2. Заполнение фрагментов

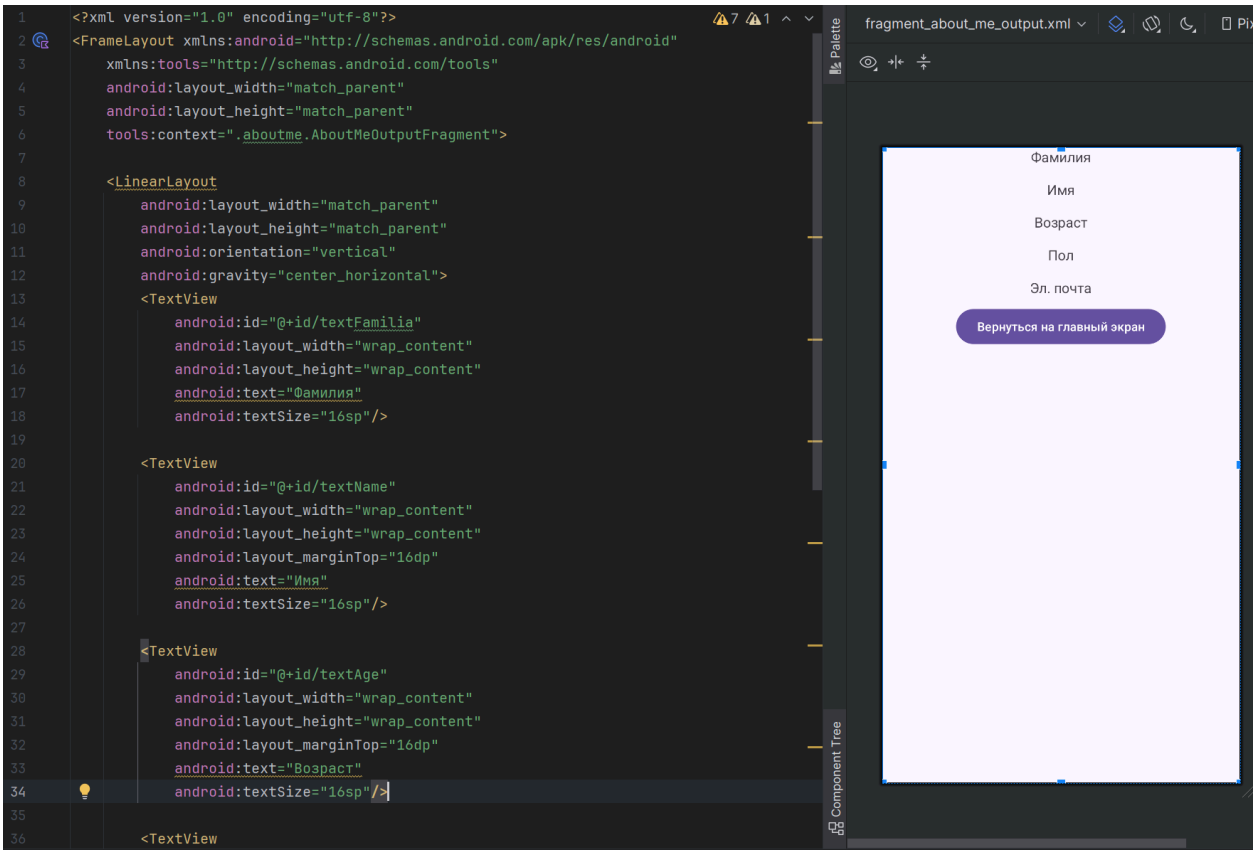
Откройте предыдущий проект и скопируйте готовые элементы каждой кнопки. Для более удобного расположения измените родительский контейнер на LinearLayout.

Пример разметки первого фрагмента «fragment_about_me.xml» для кнопки «Обо мне»:




```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout 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"
    tools:context=".aboutme.AboutMeFragment">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:gravity="center_horizontal">
        <EditText
            android:id="@+id/familiya"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:ems="10"
            android:hint="Фамилия"
            android:inputType="text"/>

        <EditText
            android:id="@+id/name"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="16dp"
            android:ems="10"
            android:hint="Имя"
            android:inputType="text" />

        <EditText
            android:id="@+id/age"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="16dp"
            android:ems="10"
            android:hint="Возраст"
```

Пример разметки второго фрагмента «fragment_about_me_output.xml» для кнопки «Обо мне»:



3) Добавление фрагментов в активити

3.1. Обработка MainActivity

В главной активити нужно добавить слушатель на каждую кнопку. В каждом слушателе будем выполнять переход на вторую активити «SecondActivity», передавая сообщение о том, какой тип фрагмента будем ожидать.

Пример кода:

```
binding.aboutMe.setOnClickListener {
    val intent = Intent( packageContext: this, SecondActivity::class.java)
    intent.putExtra( name: "fragmentType",  value: "aboutMe")
    startActivity(intent)
}
```

3.2. Обработка фрагментов в SecondActivity

Нужно обработать 2 фрагмента ссответсвующим образом, для этого получим сообщение из MainActivity и сохраним в переменную fragmentType.

Затем вызовем supportFragmentManager, начнем транзакцию сделав подстановку первого фрагмента, затем тоже самое для второго фрагмента:

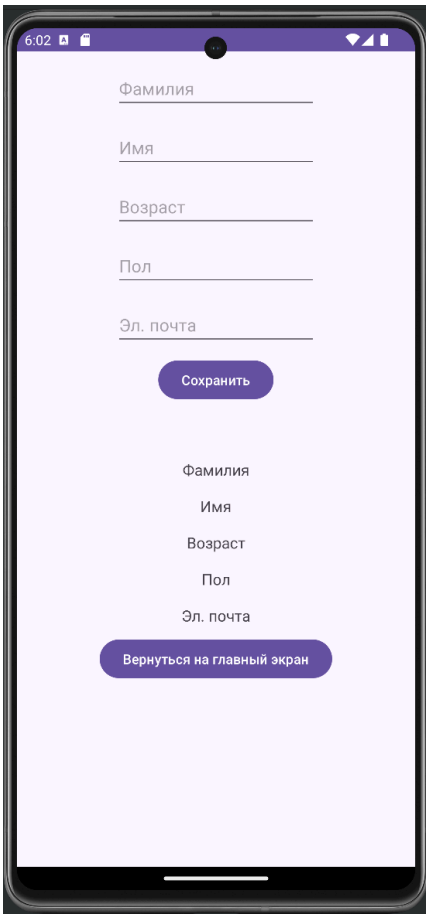
```
class SecondActivity : AppCompatActivity() {
    private lateinit var binding: ActivitySecondBinding
    private lateinit var fragmentType: String

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivitySecondBinding.inflate(layoutInflater)
        setContentView(binding.root)

        fragmentType = intent.getStringExtra( name: "fragmentType") ?: ""

        supportFragmentManager.beginTransaction()
            .replace(R.id.inputFragment1, when(fragmentType){
                "aboutMe" -> AboutMeFragment()
                "unitConverter" -> EdiniciFragment()
                "randomNumber" -> RandomFragment()
                else -> Fragment()
            }).commit()

            supportFragmentManager.beginTransaction()
                .replace(R.id.inputFragment2, when(fragmentType){
                    "aboutMe" -> AboutMeOutputFragment()
                    "unitConverter" -> EdiniciOutputFragment()
                    "randomNumber" -> RandomOutputFragment()
                    else -> Fragment()
                }).commit()
    }
}
```

Пояснения кода:

- **supportFragmentManager**: Извлекает FragmentManager данные, связанные с текущей активностью. FragmentManager управляет фрагментами в рамках активности;
- **beginTransaction()**: Инициирует новую транзакцию фрагмента.
- **.replace(R.id.inputFragment1, ...)**: Это основа операции замены фрагмента.

- **R.id.inputFragment1**: Это идентификатор FrameLayout в XML-файле макета SecondActivity. Этот FrameLayout выступает в качестве контейнера, в котором будет отображаться фрагмент. Фрагмент будет помещен в этот FrameLayout.
- **when(fragmentType)**: Это выражение проверяет значение строковой переменной fragmentType, вызывая соответствующий фрагмент.

3.3. Запустите и протестируйте проект:



4) Передача данных между фрагментами

4.1. Зависимости

Для работы с данными между фрагментами нужно подключить следующие зависимости:

```
implementation("androidx.fragment:fragment-ktx:1.6.2")
implementation("androidx.lifecycle:lifecycle-viewmodel-ktx:2.6.2")
implementation("androidx.lifecycle:lifecycle-livedata-ktx:2.6.2")
```

4.2. ViewModel

Существует несколько способов передачи данных между двумя фрагментами в рамках одного приложения в Kotlin для Android. Но мы разберем популярный. Суть этого способа заключается в том, чтобы использовать общую модель представления. Создайте ViewModel в каждой папке с фрагментами кнопок. Пример **AboutMeViewModel** (остальные такие же и отличаются только наименованием класса):

```
package com.example.fragmentproject.aboutme

import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel

class AboutMeViewModel : ViewModel() {
    val dataFromFragment = MutableLiveData<AboutMeData?>( value: null)
}
```

4.3. Данные из фрагмента «Обо мне»

Чтобы корректно передать данные из одного фрагмента во второй нужно создать список, а точнее класс с хранимыми данными. Для этого в папке «**aboutme**» создайте **Data class** и назовите его «**AboutMeData**». Код класса будет такой:

```
package com.example.fragmentproject.aboutme

data class AboutMeData(
    val familiya: String,
    val name: String,
    val age: String,
    val pol: String,
    val adress: String
)
```

Откройте фрагмент из которого будем вытаскивать данные (AboutMeFragment) и добавьте метод **onViewCreated()** – в нем будем получать данные из разметки фрагмента «fragment_about_me.xml» и отправлять в AboutMeViewModel при помощи data-класса:

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

После объявления глобальной переменной binding добавьте еще одну глобальную переменную **sharedViewModel** типа данных AboutMeViewModel:

```
private val sharedViewModel: AboutMeViewModel by activityViewModels()
```

Затем в методе **onViewCreated()** добавьте слушатель кнопки и в нем получите данные из разметки фрагмента binding.

Осталось исключить получения пустых данных из полей ввода. Сделать это можно при помощи метода ***isNotBlank*()**. Далее после проверки задействуем запись **в data-класс** и отправку данных через глобальную переменную **sharedViewModel**:

```
if (familiya.isNotBlank() && name.isNotBlank() &&
    age.isNotBlank() && pol.isNotBlank() && adress.isNotBlank()){
    val aboutMeData = AboutMeData(familiya, name, age, pol, adress)
    sharedViewModel.dataFromFragment.value = aboutMeData
} else {
    Toast.makeText(requireContext(), text: "Fill all fields!", Toast.LENGTH_SHORT).show()
}
```

4.4. Данные, полученные из пункта 4.3.

Откройте фрагмент где будем получать данные (AboutMeOutputFragment) и добавьте метод **onViewCreated()**:

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

    sharedViewModel.dataFromFragment.observe(viewLifecycleOwner) { aboutMeData ->
        aboutMeData?.let {data ->
                binding.textFamilia.text = data.familiya
                binding.textName.text = data.name
                binding.textAge.text = data.age
                binding.textPol.text = data.pol
                binding.textAdress.text = data.adress
            }
        }
}
```

Пояснения кода:

- **sharedViewModel.dataFromFragment**: Это доступ к объекту LiveData из ViewModel. **LiveData** является наблюдаемым хранилищем данных. При изменении значения dataFromFragment все зарегистрированные наблюдатели получат уведомление.
- **.observe(viewLifecycleOwner)**: Регистрирует наблюдателя в LiveData.viewLifecycleOwner - это LifecycleOwner объект, привязанный к жизненному циклу фрагмента. Это очень важно: наблюдатель автоматически удаляется при уничтожении фрагмента, что предотвращает утечку памяти.
- **{ aboutMeData -> ... }**: Это лямбда-выражение, которое определяет действие, выполняемое при изменении значения LiveData. Параметр aboutMeData содержит новое значение LiveData. Обратите внимание, что aboutMeData может быть равно нулю.
- **aboutMeData?.let { data -> ... }**: Это безопасный вызов с блокировкой **let**. Он проверяет, не является ли aboutMeData значением null. Если оно не равно null, выполняется код внутри let блока, где **data** ненулевым значением является aboutMeData. Если aboutMeData равно null, код внутри let блока пропускается.
- Далее записываются данные в TextView используя соответствующие данные из AboutMeData объекта.

САМОСТОЯТЕЛЬНАЯ ЧАСТЬ

Задание: Сделайте подобное для остальных фрагментов (с пункта 4.2 по 4.4. включительно). Реализуйте все кнопки.