|  |
| --- |
|  |

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«МИРЭА – Российский технологический университет»**

**РТУ МИРЭА**

|  |
| --- |
|  |

Институт информационных технологий (ИТ)

Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

**ОТЧЕТ**

**ПО ПРАКТИЧЕСКОЙ РАБОТЕ №3**

**Работа с Fragment**

по дисциплине

**«Разработка мобильных приложений»**

|  |  |
| --- | --- |
| Выполнил студент группы ИНБО-03-22 | Алтухов А.В. |
|  |  |
| Принял ассистент кафедры МОСИТ | Шешуков Л.С. |

|  |  |  |
| --- | --- | --- |
| Практическая  работа выполнена | «\_\_» \_\_\_\_\_\_\_\_\_2024 г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |  |  |
| «Зачтено» | «\_\_» \_\_\_\_\_\_\_\_\_2024 г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |  |  |

Москва 2024

СОДЕРЖАНИЕ

[1. ПОСТАНОВКА ЗАДАЧИ 3](#_Toc160734879)

[2. СОЗДАНИЕ ФРАГМЕНТА 4](#_Toc160734880)

[3. ОБРАБОТКА ПЕРЕХОДОВ СОСТОЯНИЙ 10](#_Toc160734881)

[4. ОРГАНИЗАЦИЯ ОБМЕНА ДАННЫХ 13](#_Toc160734882)

[6. ВЫВОДЫ 15](#_Toc160734883)

[7. ПРИЛОЖЕНИЕ 1 16](#_Toc160734884)

# ПОСТАНОВКА ЗАДАЧИ

1. Перенести разметки и логику работы экранов из практической работы №2, которые были представлены на Activity во Fragment. Разметка экранов должна остаться неизменной.
2. При помощи отображения Toast и сообщений в Log обработать переходы состояний фрагментов в соответствии с их жизненным циклом.
3. Согласно выбранной предметной области организовать обмен данными между несколькими Fragment’ами.

# СОЗДАНИЕ ФРАГМЕНТА

**Fragment** – часть приложения, представляющая повторно используемую часть пользовательского интерфейса. Фрагмент имеет собственную разметку и управляет ей, обладает собственным жизненным циклом. Фрагменты не могут жить сами по себе — они должны размещаться в Activity или другом фрагменте. Иерархия представлений фрагмента становится частью иерархии представлений хоста или присоединяется к ней. Фрагмент представляет собой отдельную часть пользовательского интерфейса внутри Activity. Фрагменты можно добавлять или удалять в зависимости от совершаемого действия.

Фрагменты привносят модульность и возможность повторного использования в пользовательском интерфейсе Activity, позволяя разделить пользовательский интерфейс на отдельные компоненты.

**Activity** — это идеальное место для размещения глобальных элементов пользовательского интерфейса приложения, таких как панель навигации. И наоборот, фрагменты лучше подходят для определения пользовательского интерфейса отдельного экрана или части экрана и управления им.

Чтобы создать фрагмент, необходимо создать класс-наследник от класса Fragment, находящегося внутри библиотеки AndroidX и переопределить его методы, чтобы реализовать логику приложения, аналогично тому, как при создании класса Activity. (Рисунок 2.1).

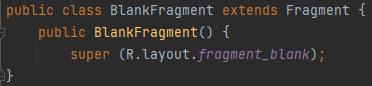


Рисунок 2.1 — Создание фрагмента

Здесь в родительский класс передается аргумент с разметкой, в результате чего будет создан фрагмент с разметкой fragment\_blank.

Библиотека фрагментов также предоставляет более специализированные базовые классы фрагментов:

**DialogFragment**

Отображает плавающий диалог.

**PreferenceFragmentCompat**

Отображает иерархию Preference объектов в виде списка.

Чтобы добавить фрагмент в разметку Activity используется элемент FragmentContainerView. Вот пример макета активности, содержащего один FragmentContainerVie (Рисунок 2.2).

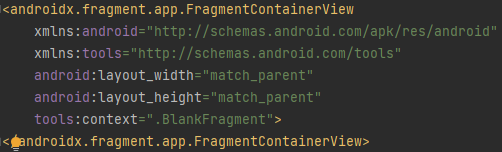


Рисунок 2.2 — Создание контейнера

Атрибут android:name указывает имя класса Fragment для создания экземпляра. Когда макет Activity используется приложением, создается конкретный фрагмент, вызывается метод onInflate() для вновь созданного фрагмента и FragmentTransaction создается для добавления фрагмента в FragmentManager.

Существует 2 способа задания фрагментов:

* Через XML файл;
* Программный.

Чтобы добавить фрагмент в разметку Activity используется элемент FragmentContainerView. В нем указывается атрибут android:name, который указывает имя класса Fragment для создания экземпляра.



Рисунок 2.3 — Добавление фрагмента через XML

Чтобы добавить фрагмент программно в макет активности, макет должен также включать в себя FragmentContainerView, который будет служить контейнером фрагментов.

В отличие от XML-подхода, атрибут “android:name” здесь не используется, поэтому фрагмент автоматически не создается. Вместо этого в макете указывается **android:id** (Рисунок 2.4).

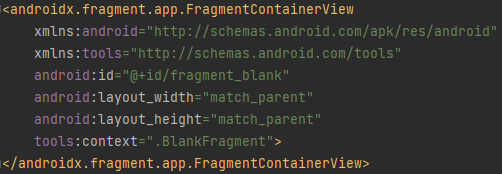


Рисунок 2.4 — Использование android:id

После этого используется FragmentTransaction (транзакция) во FragmentManager для создания экземпляра фрагмента и добавления его в макет действия.

Во время выполнения логики Activity можно совершать транзакции фрагментов, такие как добавление, удаление или замена фрагмента.

Add добавляет фрагмент, commit применяет изменения, сделанные в FragmentManager (Рисунок 2.5).

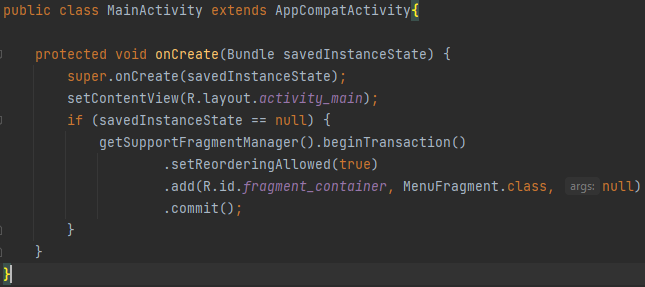


Рисунок 2.5 — Использование FragmentManager

Чтобы перейти от активити к фрагментам, создадим классы для фрагментов, а также файлы разметки с названием «fragment …» (Рисунок 2.6).

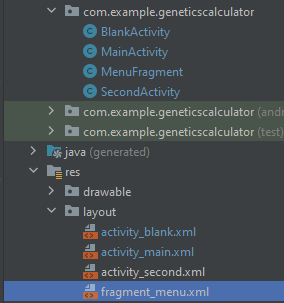


Рисунок 2.5 — Добавление/изменение классов фрагментов

Далее необходимо создать класс-наследник от класса Fragment, находящегося внутри библиотеки AndroidX и переопределить его методы, чтобы реализовать логику приложения, аналогично тому, как при создании класса Activity.

Во всех классах фрагмента инициализируем ViewBinding, а также переопределим методы onCreateView (вызывается до создания View) и onViewCreated (вызывается после создания View) (Рисунок 2.6).

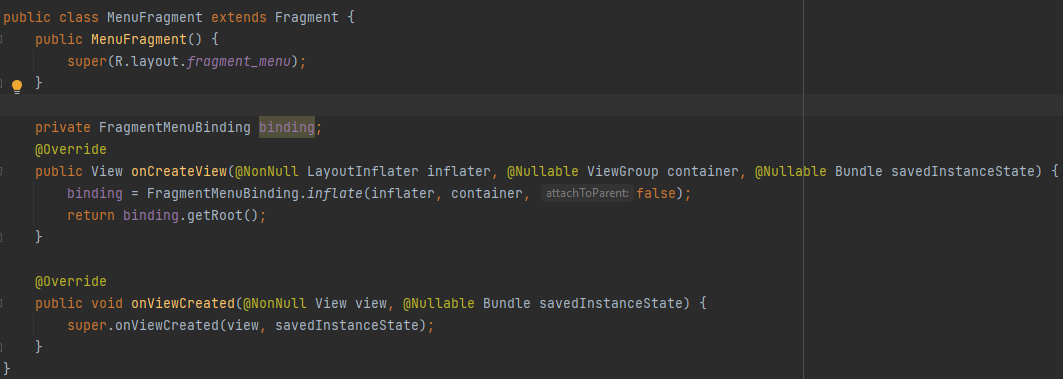


Рисунок 2.6 — Переопределение методов

Теперь после запуска приложения, MainActivity показывает фрагмент (Рисунок 2.7).

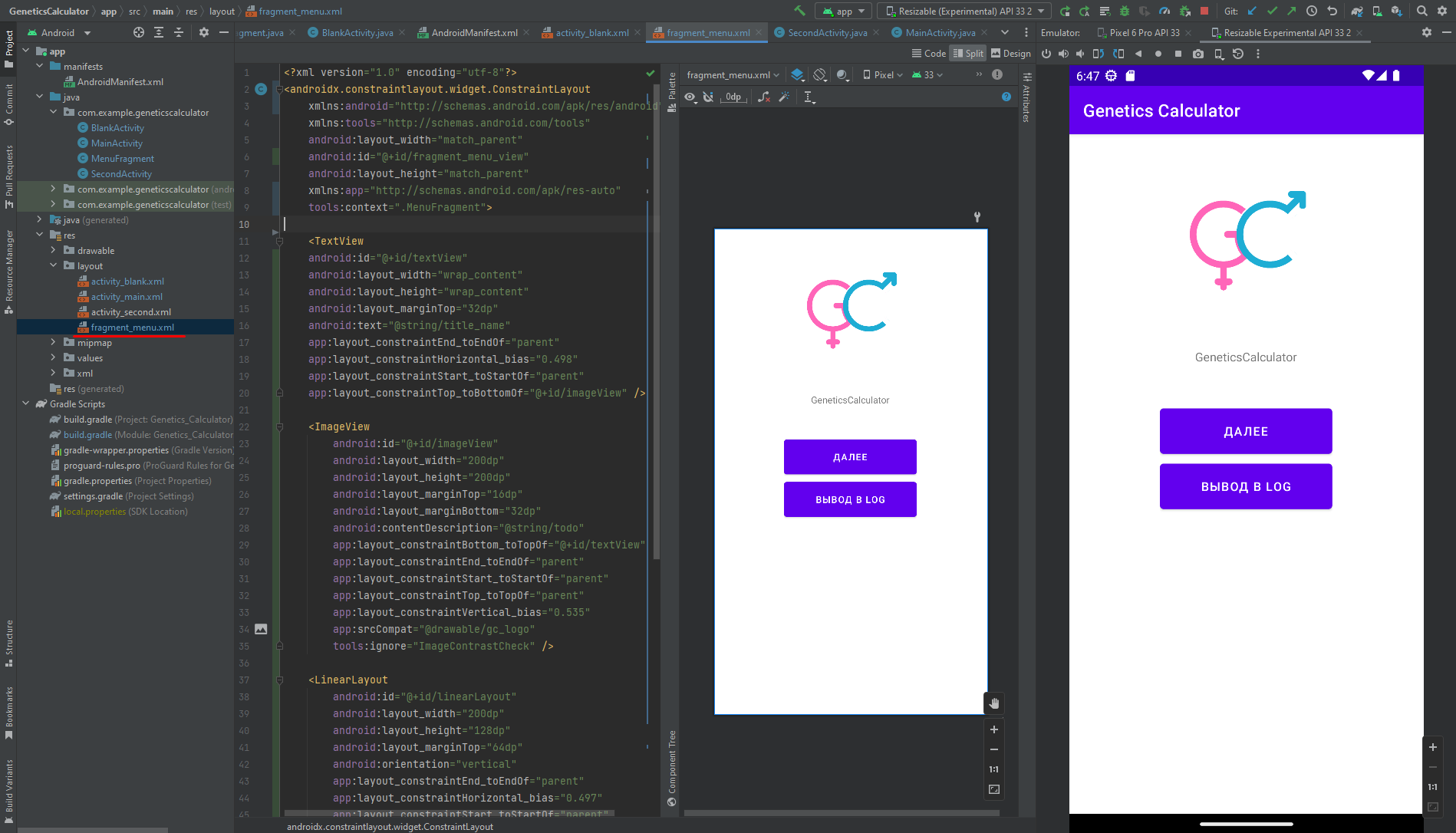


Рисунок 2.6 — Запуск приложения

Теперь перенесём разметку второго активити (Рисунок 2.7).

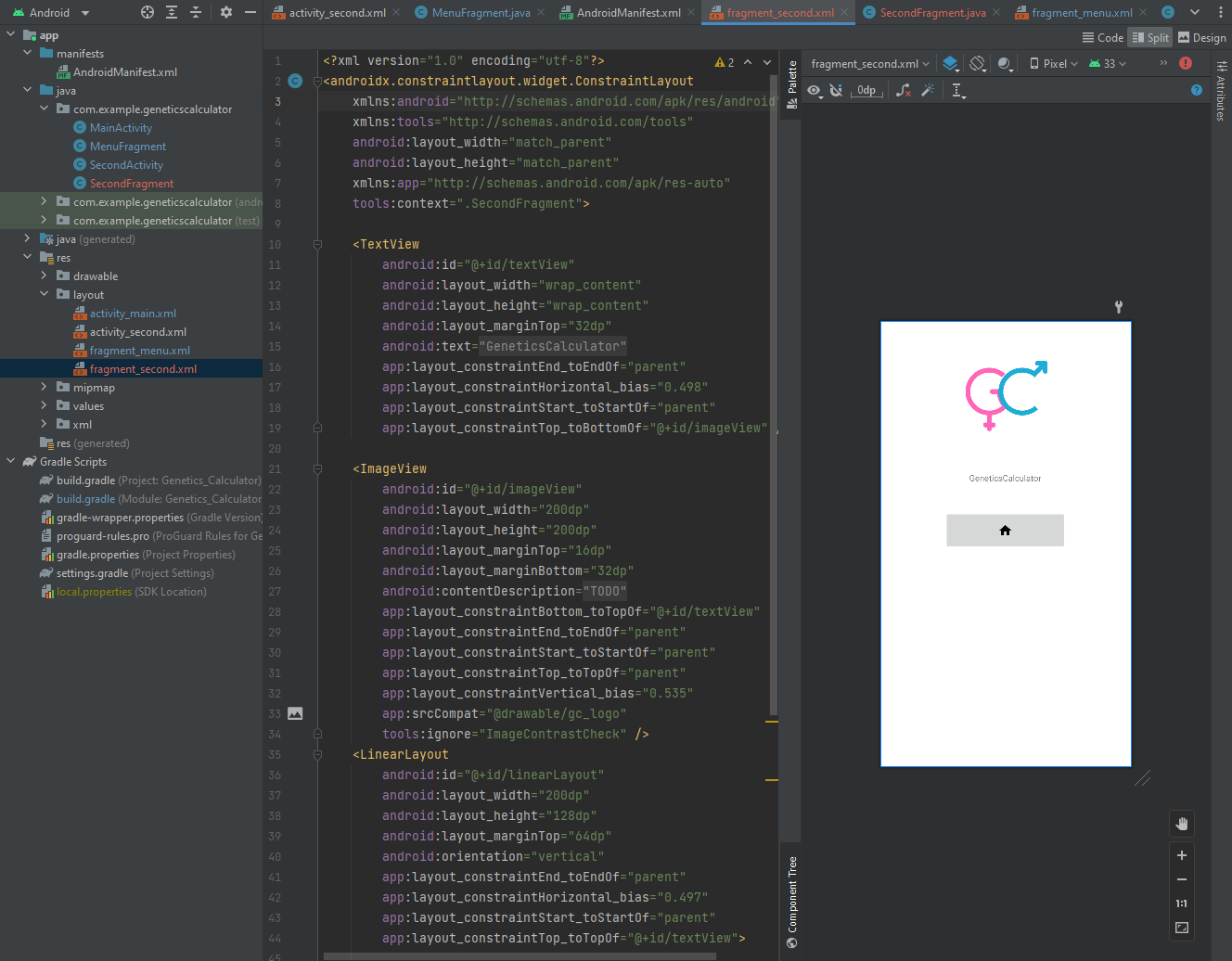


Рисунок 2.6 — Перенос разметки второго активити

Также добавим ещё один фрагмент, и изменим название прошлого (Рисунок 2.7).

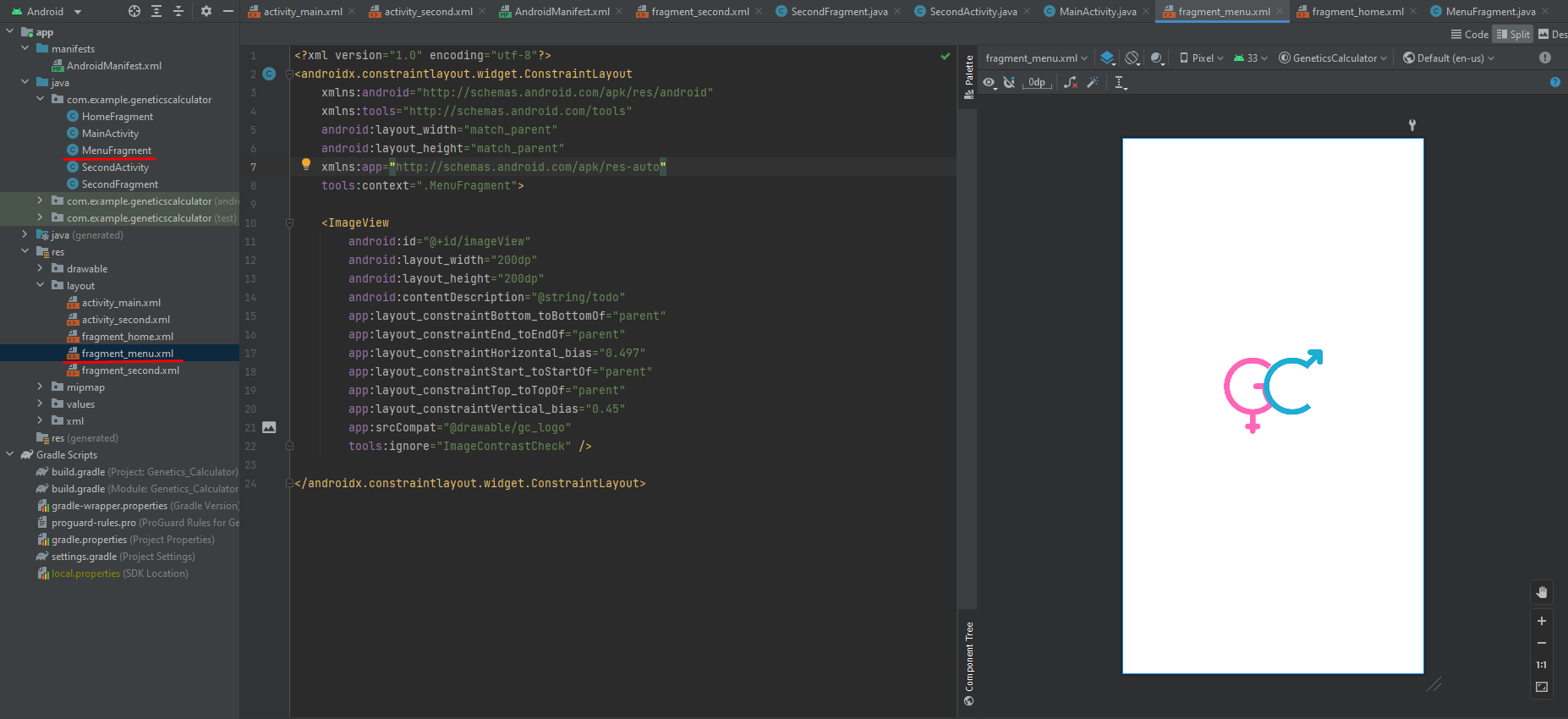


Рисунок 2.7 — Создание ещё одного фрагмента

# 3. ОБРАБОТКА ПЕРЕХОДОВ СОСТОЯНИЙ

Каждый экземпляр Fragment имеет свой жизненный цикл. Когда пользователь перемещается по приложению и взаимодействует с ним, фрагменты проходят через различные состояния в своем жизненном цикле:

* **onAttach** – фрагмент прикрепляется к Activity, с этого метода доступен контекст Acitivty;
* **onCreate** – фрагмент еще не видно, в этом методе можно получить аргументы или восстановить состояние экрана;
* **onCreateView** – создает view, связанное с фрагментом;
* **onViewCreated** – вызывается после того, как View создана, с этого момента можно работать с view элементами фрагмента;
* **onViewDestroy** – вызывается после того, как View уничтожена, с этого момента нельзя работать с view элементами фрагмента;
* **onStart** – фрагмент виден, но еще не в фокусе (например, нельзя нажимать на кнопки);
* **onResume** – фрагмент виден и в фокусе;
* **onPause** – фрагмент виден, но не в фокусе;
* **onStop** – фрагмент не виден;
* **onDestroyView** – уничтожает view, связанное с фрагментом;
* **onDestroy** – фрагмент уничтожается;
* **onDetach** – фрагмент открепляется от Activity, доступ к контексту Activity теряется.

Управления жизненным циклом Fragment реализует класс LifecycleOwner, предоставляя объект Lifecycle, к которому можено получить доступ через getLifecycle() метод. Каждое возможное состояние Lifecycle представлено в перечислении(enum) Lifecycle.State:

* INITIALIZED;
* CREATED;
* STARTED;
* RESUMED;
* DESTROYED.

Для отображения всех методов жизненного цикла в переопределенных методах используем логи и всплывающие сообщения (Рисунок 3.1).

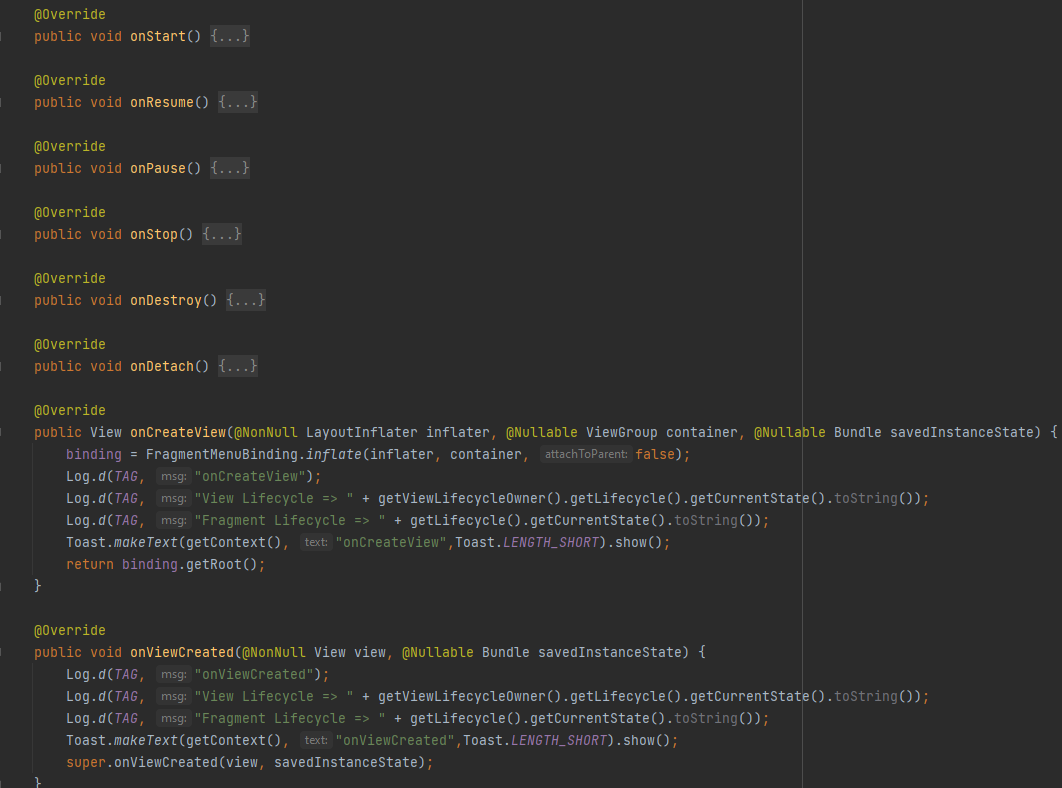


Рисунок 3.1 — Отображение методов жизненного цикла

Теперь проверим работоспособность, посмотрев логи (Рисунок 3.2).

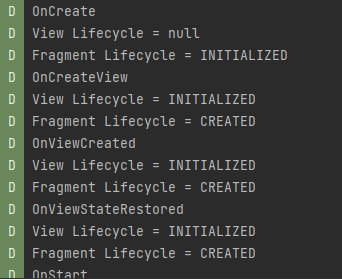


Рисунок 3.2 — Проверка работоспособности

1. **ОРГАНИЗАЦИЯ ОБМЕНА ДАННЫХ**

Для передачи данных между фрагментами будем использовать метод setArguments, который в качестве параметра принимает Bundle. В сам Bundle передаются различные ключи (Рисунок 4.1).

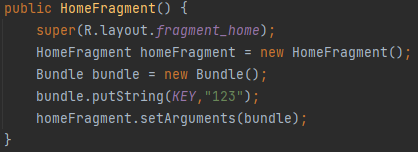


Рисунок 4.1 — Передача данных

Для получения переданных данных также используется Bundle, который инициализируется методом getArguments у фрагмента (Рисунок 4.2).

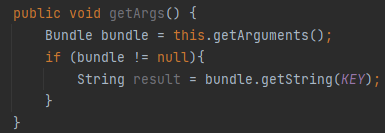


Рисунок 4.2 — Получение результатов

Для получения данных из закрывающегося фрагмента нужно обратиться к FragmentManager и установить слушатель результирующих данных из фрагмента (setFragmentResultListener).

В метод слушателя необходимо передать ключ, по которому можно будет вытащить данные из нужного нам фрагмента, жизненный цикл view, а также создать анонимный класс слушателя результатов из фрагмента FragmentResultListener и переопределить его метод onFragmentResult, который на вход получает ключ, по которому находятся результаты и Bundle с результатами (Рисунок 4.3).

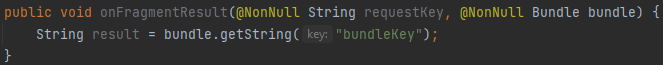


Рисунок 4.3 — Передача результата из дочернего в родительский

В фрагменте, из которого нужно передать данные, нужно создать Bundle, вызывать FragmentManager и вызвать у него метод setFragmentResult, в который передается ключ, по которому можно получить данные, и экземпляр класса Bundle (Рисунок 4.4).

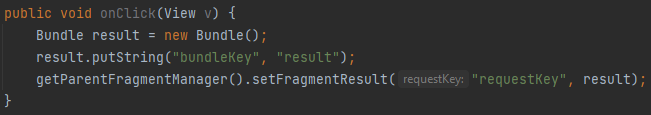


Рисунок 4.4 — Получение результата в родительском фрагменте

# 6. ВЫВОДЫ

В данной работе была перенесена разметка и логика экранов из Активити во Фрагменты, при помощи отображения Тостов и Логов были обработаны переходы состояний фрагментов в соответствии с их жизненным циклом, а также был организован обмен данными между несколькими фрагментами.

# 7. ПРИЛОЖЕНИЕ 1

Ссылка на репозиторий с проектом на GitHub:

<https://github.com/MLPeekk/RMP_2_3>