|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных технологий

**Отчет по практической работе №3**

по дисциплине «Разработка мобильных приложений»

|  |  |
| --- | --- |
| **Выполнил:**  Студент группыИКБО-21-23 | Муравьев А. О. |
| **Проверил:**  Старший преподаватель кафедры МОСИТ | Шешуков Л.С. |

Москва 2025 г.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc193537193)

[1 ТЕОРИТИЧЕСКАЯ ЧАСТЬ 5](#_Toc193537194)

[1.1 Передача сложных объектов между активностями 5](#_Toc193537195)

[1.2 BackStack 12](#_Toc193537196)

[1.3 Контейнеры 16](#_Toc193537197)

[1.3.1 LinearLayout 17](#_Toc193537198)

[1.3.2 RelativeLayout 18](#_Toc193537199)

[1.3.3 TableLayout 23](#_Toc193537200)

[1.3.4 FrameLayout 25](#_Toc193537201)

[1.3.5 GridLayout 26](#_Toc193537202)

[1.3.6 ConstraintLayout 27](#_Toc193537203)

[1.4 Ресурсы в Android 32](#_Toc193537204)

[2 ПРАКТИЧЕСКАЯ ЧАСТЬ 42](#_Toc193537205)

[2.1 Применение контейнеров Layout 42](#_Toc193537207)

[2.1.1 Контейнер LinerLayout 42](#_Toc193537208)

[2.1.2 Контейнер RelativeLayout 44](#_Toc193537209)

[2.1.3 Контейнер ConstraintLayout 47](#_Toc193537210)

[2.1.4 Контейнер FrameLayout 50](#_Toc193537211)

[2.2 Переход между активностями с передачей данных и возвратом на предыдущую страницу 53](#_Toc193537212)

[2.2.1 Создание класса для передачи данных 59](#_Toc193537213)

[2.2.2 Реализация передачи данных с помощью объекта 62](#_Toc193537214)

[2.2.3 Реализация возврата на предыдущую страницу 64](#_Toc193537215)

[2.2.4 Тестирование работы приложения 67](#_Toc193537216)

[2.3 Работа с ресурсами 69](#_Toc193537217)

[2.3.1 Добавление ресурсов 69](#_Toc193537218)

[2.3.2 Применение добавленных ресурсов 73](#_Toc193537219)

[2.3.3 Изменение настроек темы 75](#_Toc193537220)

[2.3.4 Добавление локализации 81](#_Toc193537221)

[ЗАКЛЮЧЕНИЕ 85](#_Toc193537222)

ВВЕДЕНИЕ

Цель данной работы — изучить ключевые аспекты разработки мобильных приложений на платформе Android, включая передачу данных между активностями (Activity) с использованием Activity Result API, а также работу с фрагментами (Fragment) для создания гибких и адаптивных пользовательских интерфейсов.

Выполнение работы включает в себя следующие задачи:

1. Создать три активности. На первой Activity расположить поле для ввода фамилии и имени и кнопку Next для запуска второго Activity. При нажатии кнопки выполнить передачу данных.

Во втором Activity создать два текстовых поля (TextView) для вывода переданной информации о пользователе (имя и фамилия), пустое по умолчанию текстовое поле (TextView) для ввода предмета (на который хотели бы дополнительно записаться), кнопку Ввести информацию.

В третьем Activity создать 3 редактируемых текстовых полей (EditText) для ввода информации о днях посещения (день, время, комментарии), кнопку OК для возврата во второе Activity. При нажатии кнопки ОК реализовать возврат во второе Activity с передачей в качестве результата информации о предпочтительном времени организации дополнительного занятия. Вывести всплывающее сообщение о том, что время занятия успешно передано. По желанию можно добавить еще поля, кнопки или другие элементы управления, изменить тему и т.д.

1. Создать новый проект. Разместить три фрагмента с разным наполнением тремя способами (статически, динамически и через элемент fragmentContainerView).
2. Сделать перемещение между созданными фрагментами.

# ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

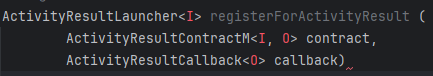
## Передача данных в вызвавший открытие activity с помощью Activity Result API

В предыдущей практике было рассмотрено как вызывать новую Activity, передавать ей некоторые данные и возвращаться обратно. Но мы можем не только передавать данные запускаемой Activity, но и ожидать от нее некоторого результата работы.

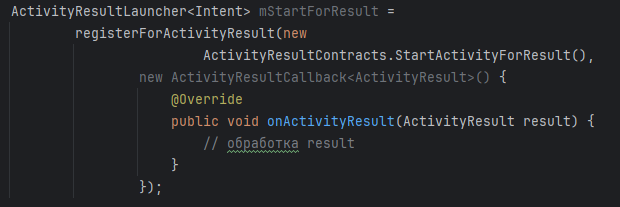
Ранее мы вызывали новую Activity с помощью метода startActivity(). Для получения же результата работы запускаемой Activity необходимо использовать Activity Result API.

Activity Result API предоставляет компоненты для регистрации, запуска и обработки результата другой Activity. Одним из преимуществ применения Activity Result API является то, что он отвязывает результат Activity от самой Activity. Это позволяет получить и обработать результат, даже если Activity, которая возвращает результат, в силу ограничений памяти или в силу других причин завершила свою работу.

Для регистрации функции, которая будет обрабатывать результат, Activity Result API предоставляет метод registerForActivityResult(). Этот метод в качестве параметров принимает объекты ActivityResultContract и ActivityResultCallback и возвращает объект ActivityResultLauncher, который применяется для запуска другой activity. Это показано на рисунке 1.

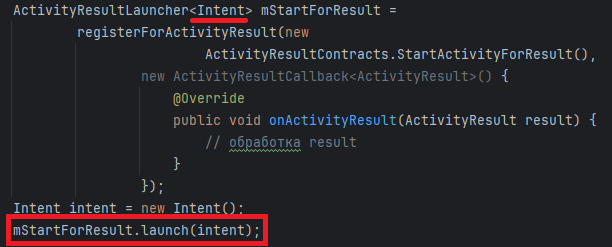
  
Рисунок 1 – Код метода registerForActivityResult()

ActivityResultContract определяет контракт: данные какого типа будут подаваться на вход и какой тип будет представлять результат. ActivityResultCallback представляет интерфейс с единственным методом onActivityResult(), который определяет обработку полученного результата. Когда вторая activity закончит работу и возвратит результат, то будет как раз вызываться этот метод. Результат передается в метод в качестве параметра. При этом тип параметра должен соответствовать типу результата, определенного в ActivityResultContract. Показано на рисунке 2.

  
Рисунок 2 – Шаблон для написания экземпляра класса ActivityResultLauncher

Класс ActivityResultContracts предоставляет ряд встроенных типов контрактов. Например, в листинге кода выше применяется встроенный тип ActivityResultContracts.StartActivityForResult, который в качестве входного объекта устанавливает объект Intent, а в качестве типа результата – тип ActivityResult.

Метод registerForActivityResult() регистрирует функцию-колбек и возвращает объект ActivityResultLauncher. С помощью этого мы можем запустить activity. Для этого у объекта ActivityResultLauncher вызывается метод launch(). Покажем на рисунке 3.

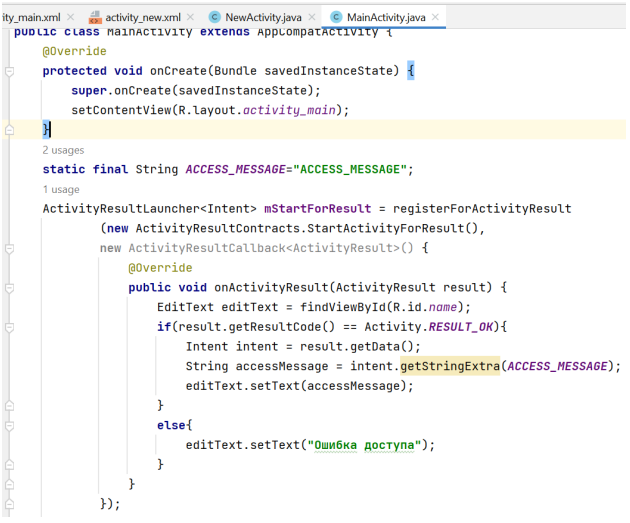
  
Рисунок 3 – Пример вызова метода launch

В метод lauch() передается объект того типа, который определен объектом ActivityResultContracts в качестве входного.

Приведём пример подобного использования ActivityResultContracts.

Допустим у нас есть поле для ввода имени. После ввода имени на второй activity, мы можем выбрать запретить доступ, разрешить доступ или отменить операцию вовсе.

Определим в классе MainActivity запуск второй activity на рисунке 4.

  
Рисунок 4 – Дополнение кода

Прежде всего, мы определяем объект ActivityResultLauncher, с помощью которого будем запускать вторую activity и передавать ей данные. Объект ActivityResultLauncher типизируется типом Intent, так как объект этого типа будет передаваться в метод launch() при запуске второй activity. Тип контракта определяется типом ActivityResultContracts.StartActivityForResult, который и определяет тип Intent в качестве входного типа и тип ActivityResult в качестве типа результата.

Второй аргумент метода registerForActivityResult() – объект ActivityResultCallback типизируется типом результата – типом ActivityResult и определяет функцию-колбек onActivityResult(), которая получает результат и обрабатывает его. В данном случае обработка состоит в том, что мы выводим в текстовое поле ответ от второй activity.

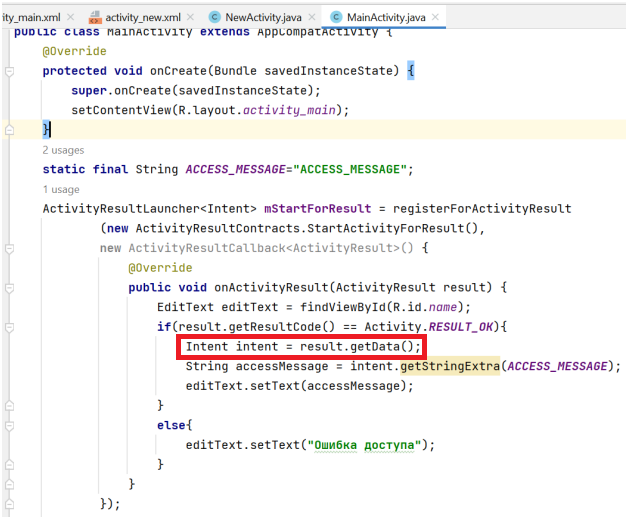
При обработке проверяем полученный код результата на рисунке 5.

  
Рисунок 5 - Проверка полученного результата

В качестве результата, как правило, применяются встроенные константы Activity.RESULT\_OK и Activity.RESULT\_CANCELED.

На уровне условностей Activity.RESULT\_OK означает, что activity успешно обработала запрос, а Activity.RESULT\_CANCELED – что activity отклонила обработку запроса.

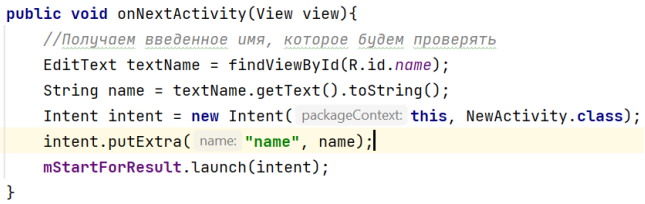
С помощью метода getData() результата получаем переданные из второй activity данные в виде объекта Intent. Покажем на рисунке 6.

Рисунок 6 – Получение данных из второй activity

Далее извлекаем из Intent строку, которая имеют ключ ACCESS\_MESSAGE, и выводим ее в текстовое поле.

Таким образом, мы определили объект ActivityResultLauncher.

Далее в обработчике нажатия onClick с помощью этого объекта запускаем вторую activity как на рисунке 7.

  
Рисунок 7 – Метод запуска второй activity

В обработчике нажатия кнопки onClick() получаем введенное в текстовое поле имя, добавляем его в объект Intent и запускаем SecondActivity с помощью метода launch().

После этого определим логику работы второй Activity на рисунке 8.

  
Рисунок 8 – Описание логики работы второй activity

Две кнопки вызывают метод sendMessage(), в который передают отправляемый ответ. Это и будет то сообщение, которое получить MainActivity в методе onActivityResult.

Для возврата результата необходимо вызвать метод setResult(), в который передается два параметра:

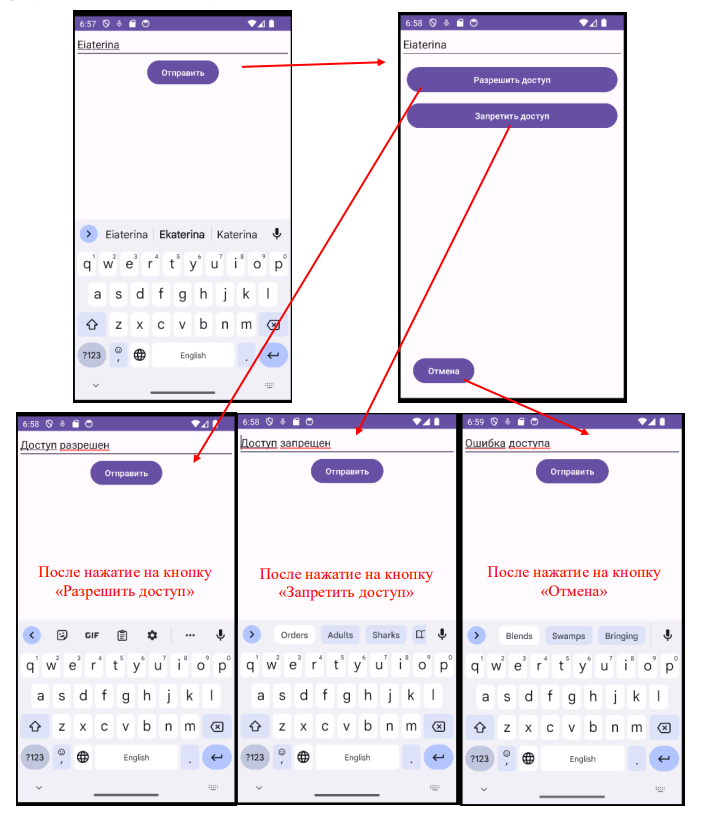
* числовой код результата;
* отправляемые данные.

После вызова метода setResult() нужно вызвать метод finish, который уничтожит текущую activity. finish – метод, при вызове которого система Android завершает работу текущей activity и удаляет её из Activity Stack (коллекция приостановленных системойactivities, которые можно легко возобновить).

Одна кнопка вызывает обработчик onCancelClick(), в котором передается в setResult только код результата - RESULT\_CANCELED.

То есть условно говоря, мы получаем в SecondActivity введенное в MainActivity имя и с помощью нажатия определенной кнопки возвращаем некоторый результат в виде сообщения.

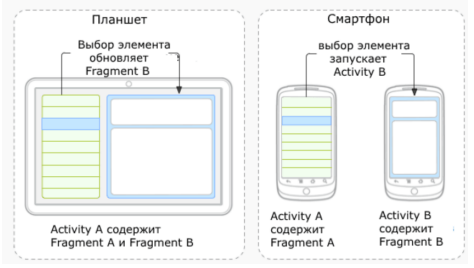
В зависимости от нажатой кнопки на SecondActivity мы будем получать разные результаты в MainActivity. Покажем это на рисунке 9.

  
Рисунок 9 – Отображение различных вариантов развития событий в MainActivity

## Фрагменты

Организация приложения на основе нескольких activity не всегда может быть оптимальной. Мир ОС Android довольно сильно фрагментирован и состоит из многих 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.

Фрагмент представляет кусочек визуального интерфейса приложения, который может использоваться повторно и многократно. У фрагмента может быть собственный файл layout, у фрагментов есть свой собственный жизненный цикл. Фрагмент существует в контексте activity и имеет свой жизненный цикл, вне activity обособлено он существовать не может. Каждая activity может иметь несколько фрагментов. Это показано на рисунке 10.

  
Рисунок 10 – Пример использования фрагментов

В Android Studio выделяют такие популярные виды фрагментов, как:

* UI-фрагменты (View Fragments): наиболее распространённый тип фрагментов, которые используются для отображения пользовательского интерфейса. Они имеют собственный жизненный цикл и могут управлять своим макетом (layout). Пример: фрагмент, отображающий список элементов, или фрагмент с деталями элемента;
* диалоговые фрагменты (Dialog Fragments): используются для отображения диалоговых окон. Позволяют создавать кастомные диалоги (небольшие окна, в которых пользователь может принять решение или ввести дополнительную информацию), которые могут быть повторно использованы в разных activities. Пример: диалог подтверждения действия или диалог с настройками;
* фрагменты-списки (List Fragments): предназначены для отображения списка элементов (например, с использованием ListView или RecyclerView). Упрощают работу со списками, так как уже содержат встроенные методы для управления ими.

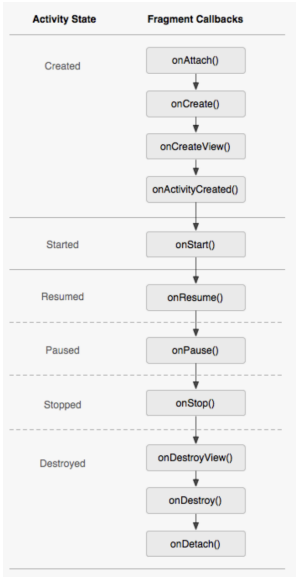
Фрагменты в Android представляют собой модульный сегмент пользовательского интерфейса в activity. Они используются для создания более гибких и адаптивных интерфейсов. Фрагменты могут быть добавлены, удалены, заменены или изменены в activity во время выполнения приложения. Это обеспечивает более динамичное и адаптивное взаимодействие с пользователем.

Среди основных характеристик фрагментов можно выделить:

* модульность: фрагменты позволяют разделить activity на несколько компонентов, каждый из которых имеет свой собственный жизненный цикл и обрабатывает свои собственные вводы. Это упрощает управление различными частями интерфейса пользователя и повторное использование компонентов в разных activities;
* адаптивность: фрагменты помогают создавать интерфейсы, которые легко адаптируются к различным размерам экрана и ориентациям, что особенно важно для устройств с разными размерами экранов, таких как телефоны и планшеты;
* управление жизненным циклом: каждый фрагмент имеет свой собственный жизненный цикл, но он тесно связан с жизненным циклом своей хост-activity. Это позволяет фрагментам управлять своим состоянием и поведением в зависимости от состояния activity.

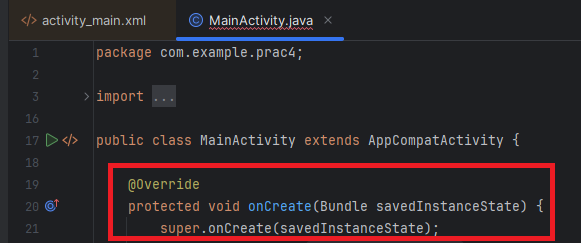
## Жизненный цикл фрагмента

Жизненный цикл фрагмента в Android представляет собой последовательность состояний, через которые проходит фрагмент во время своего существования. Эти состояния позволяют управлять поведением фрагмента на различных этапах его взаимодействия с пользователем и системой. Покажем это на рисунке 11.

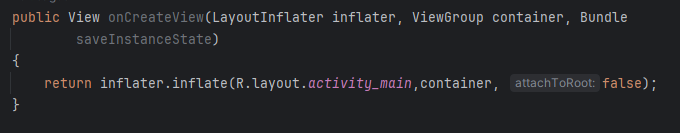
  
Рисунок 11 – Жизненный цикл фрагмента

Привязка к activity: когда фрагмент привязывается к activity, вызывается метод onAttach(). Это означает, что фрагмент теперь ассоциирован с activity, и разработчик может взаимодействовать с ней.

Создание фрагмента: Вызов метода onCreate() означает, что создается объект фрагмента. Здесь можно инициализировать компоненты, необходимые фрагменту для функционирования, но при этом не связанные с графическим интерфейсом. Покажем на рисунке 12.

  
Рисунок 12 – Определение метода OnCreate

Создание представления фрагмента: метод onCreateView() вызывается для создания представления фрагмента. Здесь загружается макет, определяет пользовательский интерфейс фрагмента. После создания представления оно возвращается системе для отображения. Покажем на рисунке 13.

  
Рисунок 13 – Определение метода onCreateView

Первый параметр – объект LayoutInflater позволяет получить содержимое ресурса layout и передать его во фрагмент.

Второй параметр – объект ViewGroup представляет контейнер, в которой будет загружаться фрагмент.

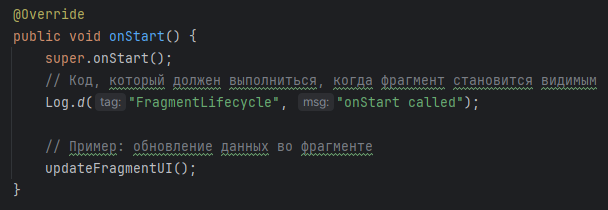
Третий параметр – объект Bundle представляет состояние фрагмента. (Если фрагмент загружается первый раз, то равен null).

На выходе метод возвращает созданное с помощью LayoutInflater представление в виде объекта View – собственно представление фрагмента.

Также идёт обращение к R.layout, где layout - это автоматически генерируемый класс, который содержит ссылки на все XML-файлы макетов (layout files), расположенные в папке res/layout проекта.

Activity фрагмента: после создания представления, метод onActivityCreated() вызывается, когда activity, к которой привязан фрагмент, полностью создана. Это хорошее место для выполнения финальных инициализаций, которые зависят от activity, например, настройка компонентов интерфейса или получение данных.

Запуск фрагмента: когда фрагмент становится видимым для пользователя, вызывается метод onStart(). Здесь можно запускать анимации или выполнять задачи, которые должны быть видны пользователю. Покажем на рисунке 14.

  
Рисунок 14 – Пример использования onStart

Возобновление фрагмента: в этом состоянии фрагмент полностью активен и взаимодействует с пользователем. Метод onResume() вызывается, когда фрагмент готов к пользовательскому взаимодействию.

Приостановка фрагмента: когда фрагмент перестает взаимодействовать с пользователем, система вызывает метод onPause(). Это происходит, например, при переключении на другой фрагмент или activity.

Остановка фрагмента: метод onStop() вызывается, когда фрагмент больше не виден пользователю. В этом состоянии можно освободить ресурсы, которые не нужны, пока фрагмент не активен.

Уничтожение представления фрагмента: Перед удалением фрагмента из activity или при его замене, метод onDestroyView() вызывается для очистки ресурсов, связанных с пользовательским интерфейсом фрагмента.

Уничтожение фрагмента: На этом этапе вызывается метод onDestroy(), который сигнализирует о том, что объект фрагмента скоро будет уничтожен. Это последний шанс для освобождения оставшихся ресурсов.

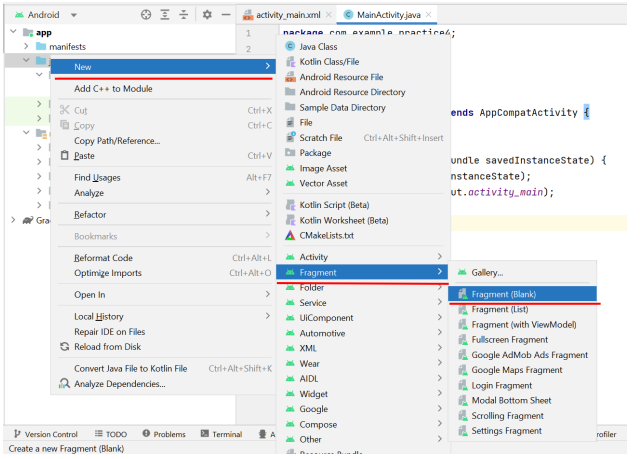
Открепление фрагмента от activity: Последний этап в жизненном цикле фрагмента — это его открепление от activity. Метод onDetach() вызывается, когда фрагмент отсоединяется от activity, что означает полное удаление фрагмента.

## Создание и размещение фрагментов

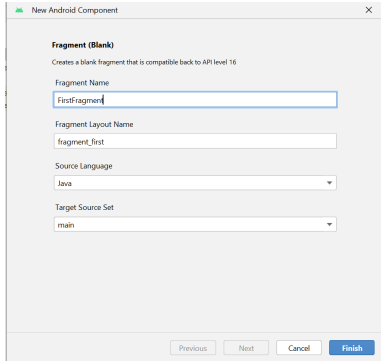
Можно добавить по отдельности класс Java, который представляет фрагмент, и файл xml для хранения в нем разметки интерфейса, который будет использовать фрагмент, однако Android Studio представляет готовый шаблон для добавления фрагмента. Этот способ и будет использован в данной работе.

Принцип создания новых фрагментов схож с созданием activity. Нужно выбрать модуль "app", затем в верней панели выбрав "File" → "New" → "Fragment" и выбрать тип activity, например "Fragment (Blank)" либо нажать правой кнопкой мыши на "java" → "New" → "Fragment", после чего выбрать название фрагмента и как в случае с activity будут созданы файлы класса и разметки.

На рисунке 15 показан процесс создания файла.

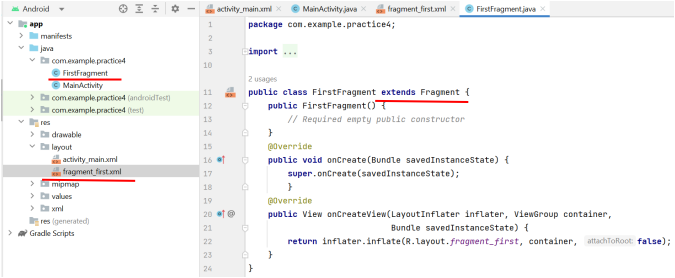
  
Рисунок 15 – Создание класса Java, представляющего фрагмент

После чего задаём имя классу как показано на рисунке 16.

  
Рисунок 16 – Описание класса-фрагмента

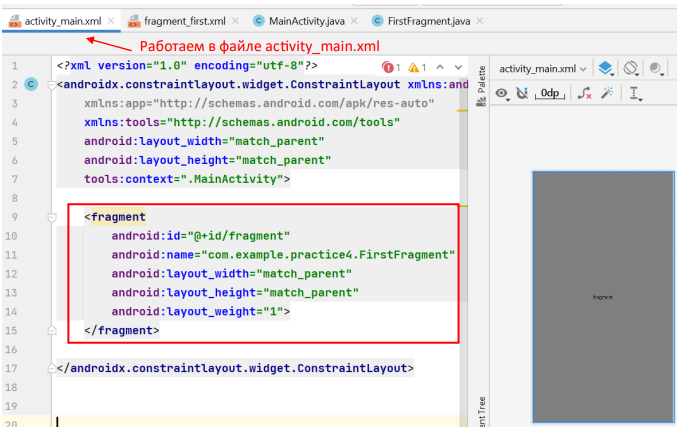
После создания фрагмента, класс для реализации фрагмента и xml-файл для разметки будут созданы автоматически.

Класс фрагмента должен наследоваться от класса Fragment. Чтобы указать, что фрагмент будет использовать определенный xml-файл layout, идентификатор ресурса layout передается в вызов конструктора родительского класса (то есть класса Fragment). Это показано на рисунке 17.

  
Рисунок 17 – Наследование класса-фрагмента

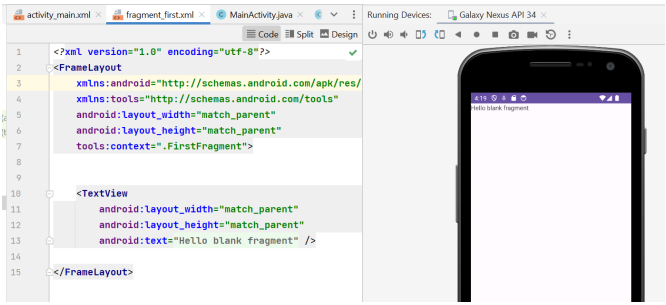
Теперь после создания фрагмента его необходимо разместить на activity. Сделать это можно тремя способами: статически, динамически и через элемент fragmentContainerView.

Для размещения фрагмента статическим способом необходимо добавить в XML файл activity элемент fragment. Это показано на рисунке 18.

  
Рисунок 18 – Размещение фрагмента статическим способом

Атрибут android:name указывает на полное имя класса фрагмента с учетом пакета, который будет использоваться.

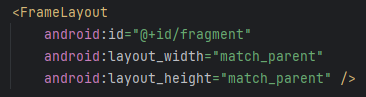
В качестве примера во фрагменте было определено текстовое поле с некоторым текстовым значением. Можем увидеть это на рисунке 19.

  
Рисунок 19 – Размещение фрагмента с текстовым полем

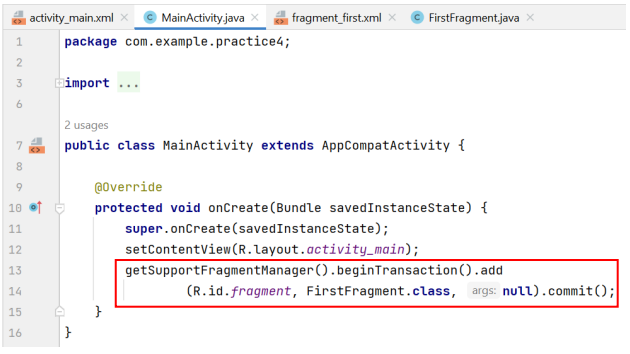
Теперь данный фрагмент будет показываться при запуске приложения.

Главным недостатком такого размещения фрагмента заключается в том, что статические фрагменты тесно связаны с жизненным циклом activity, что может усложнить управление их состоянием. Например, сохранение и восстановление работы в файле activity\_main.xml состояния фрагмента при пересоздании activity может потребовать дополнительной логики.

Все преимущества фрагментов раскрываются при их динамическом изменении в процессе работы приложения. Фрагменты, при динамическом размещении, управляются аналогично обычным View элементам. Для облегчения управления, каждый фрагмент размещается в отдельном контейнере. Обычно для этой цели выбираются контейнеры типа FrameLayout. Можем увидеть это на рисунке 20.

  
Рисунок 20 – Описание контейнера типа FrameLayout

Далее в классе activity необходимо «разместить» данный фрагмент, используя метод класса "FragmentManager" — "getSupportFragmentManager", показанный на рисунке 21.

  
Рисунок 21 – Размещение фрагмента через код

Метод getSupportFragmentManager() возвращает объект FragmentManager, который управляет фрагментами.

Объект FragmentManager с помощью метода beginTransaction() создает объект FragmentTransaction.

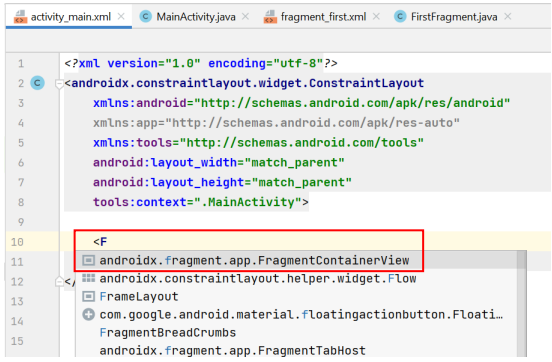
FragmentTransaction выполняет два метода: add() и commit(). Метод add() добавляет фрагмент.

И метод commit() подтверждает и завершает операцию добавления.

Последний способ размещения фрагментов является ключевым элементом современного подхода к управлению фрагментами в приложении. FragmentContainerView — это специализированный контейнер для фрагментов, который предлагает улучшенную замену FrameLayout при работе с фрагментами. Он более оптимизирован для работы с FragmentManager и предоставляет дополнительные возможности и преимущества.

Для добавления фрагмента применяется элемент FragmentContainerView. По сути, FragmentContainerView представляет объект View, который расширяет класс FrameLayout и предназначен специально для работы с фрагментами. Собственно, кроме фрагментов он больше ничего содержать не может.

Для его использования нужно разместить данный элемент в XML файле activity и в открывшимся окне выбрать нужный фрагмент, и он автоматически будет размещен внутри FragmentContainerView. Начало файла показано на рисунке 22.

  
Рисунок 22 – Начало размещения фрагмента в XML файле activity

А продолжение показано на рисунке 23.

  
Рисунок 23 – Конец размещения фрагмента в XML файле activity

## Навигация

Навигация между фрагментами осуществляется с помощью FragmentManager, который управляет операциями, такими как добавление, удаление или замена фрагментов в контейнере. Ключевым понятием в навигации между фрагментами является транзакция. Транзакция фрагмента — это серия действий, которые выполняются вместе. Эти действия могут включать добавление, удаление и замену фрагментов, а также добавление их в стек возврата, чтобы пользователь мог вернуться обратно, используя кнопку «Назад».

Для управления навигацией между фрагментами можно использовать программные методы, такие как вызовы "replace" и "commit" класса FragmentManager. Для этого нужно создать объект того фрагмента куда планируется осуществить перемещение и добавить его в метод replace и сохранить изменения через метод commit. Покажем это на рисунке 24.

  
Рисунок 24 - Реализация навигации между фрагментами

Где FirstFragment – это название java класса первого фрагмента, а container – идентификатор FrameLayout, который расположен в разметки activity\_main.

Тогда при нажатии на кнопку будут меняться используемые фрагменты. Покажем это на рисунке 25.

  
Рисунок 25 – Первая часть отображения используемых фрагментов

А также на рисунке 26 покажем продолжение.

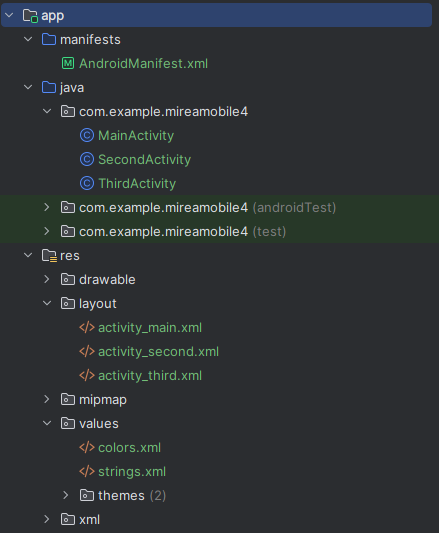
  
Рисунок 26 – Вторая часть отображения используемых фрагментов

# ПРАКТИЧЕСКАЯ ЧАСТЬ

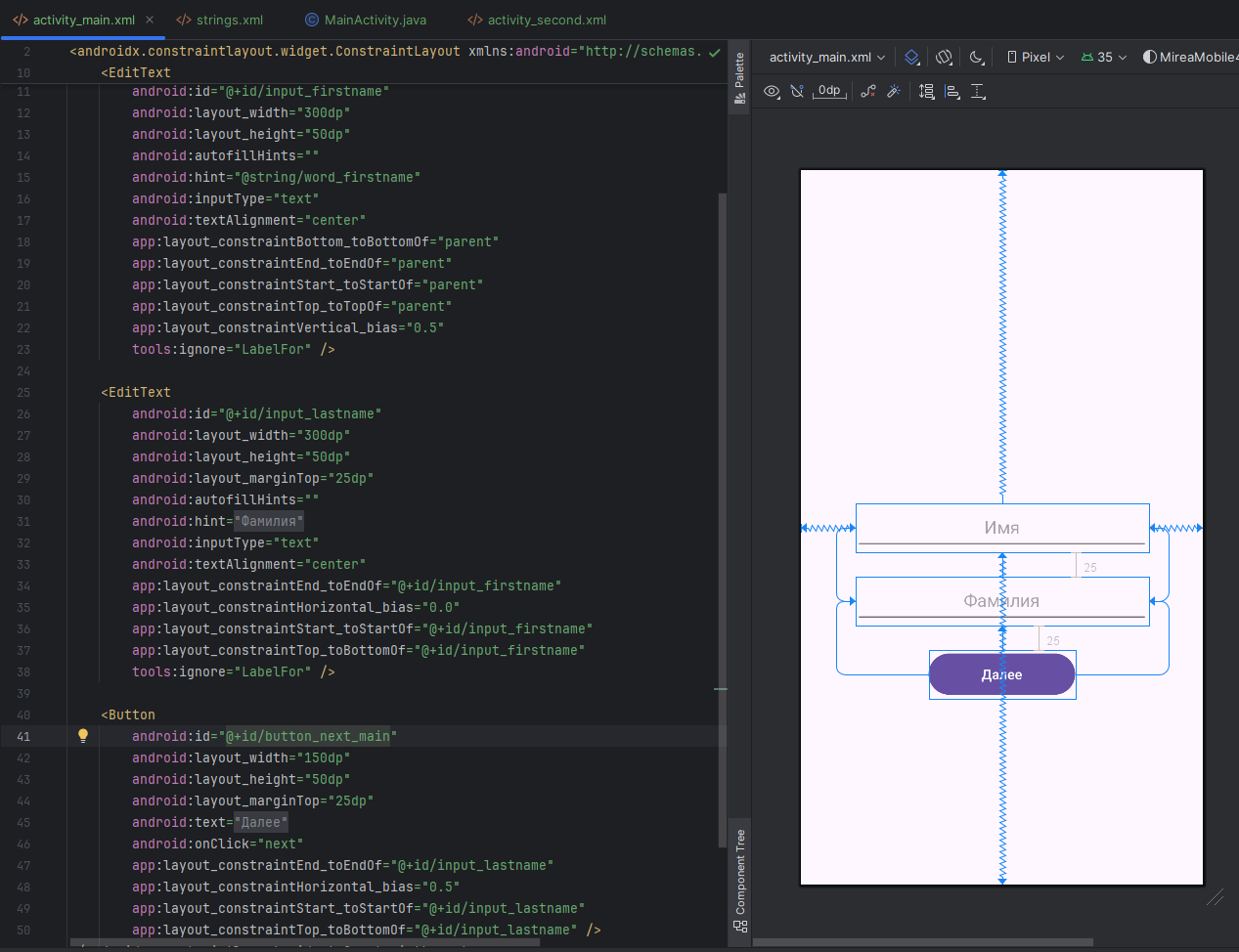
## Передача данных между тремя Activity

Для передачи данных между сообщениями использованы активности MainActivity (файл разметки – activity\_main.xml), SecondActivity (файл разметки – activity\_second.xml) и ThirdActivity (файл разметки – activity\_third.xml).

После создания всех пустых активностей получаем следующую структуру проекта (рисунок 27).

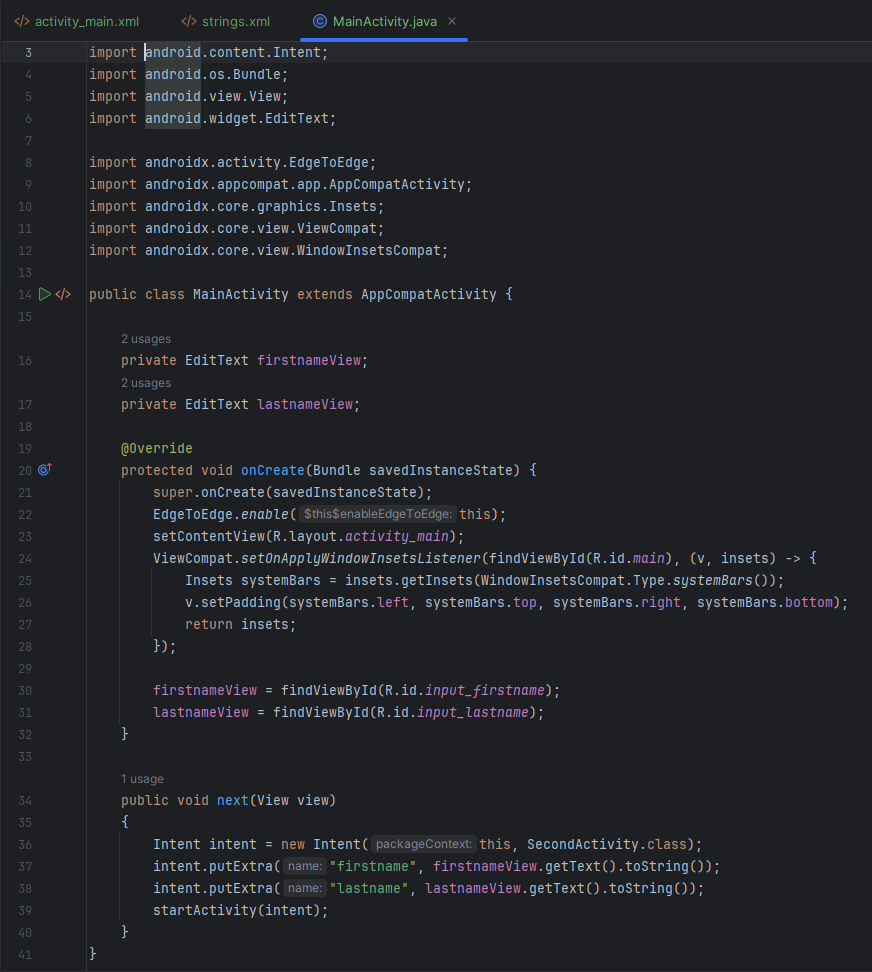
  
Рисунок 27 – Структура проекта после создания пустых активностей

В файле разметки activity\_main.xml нужно создать элементы ввода и подтверждения ввода.

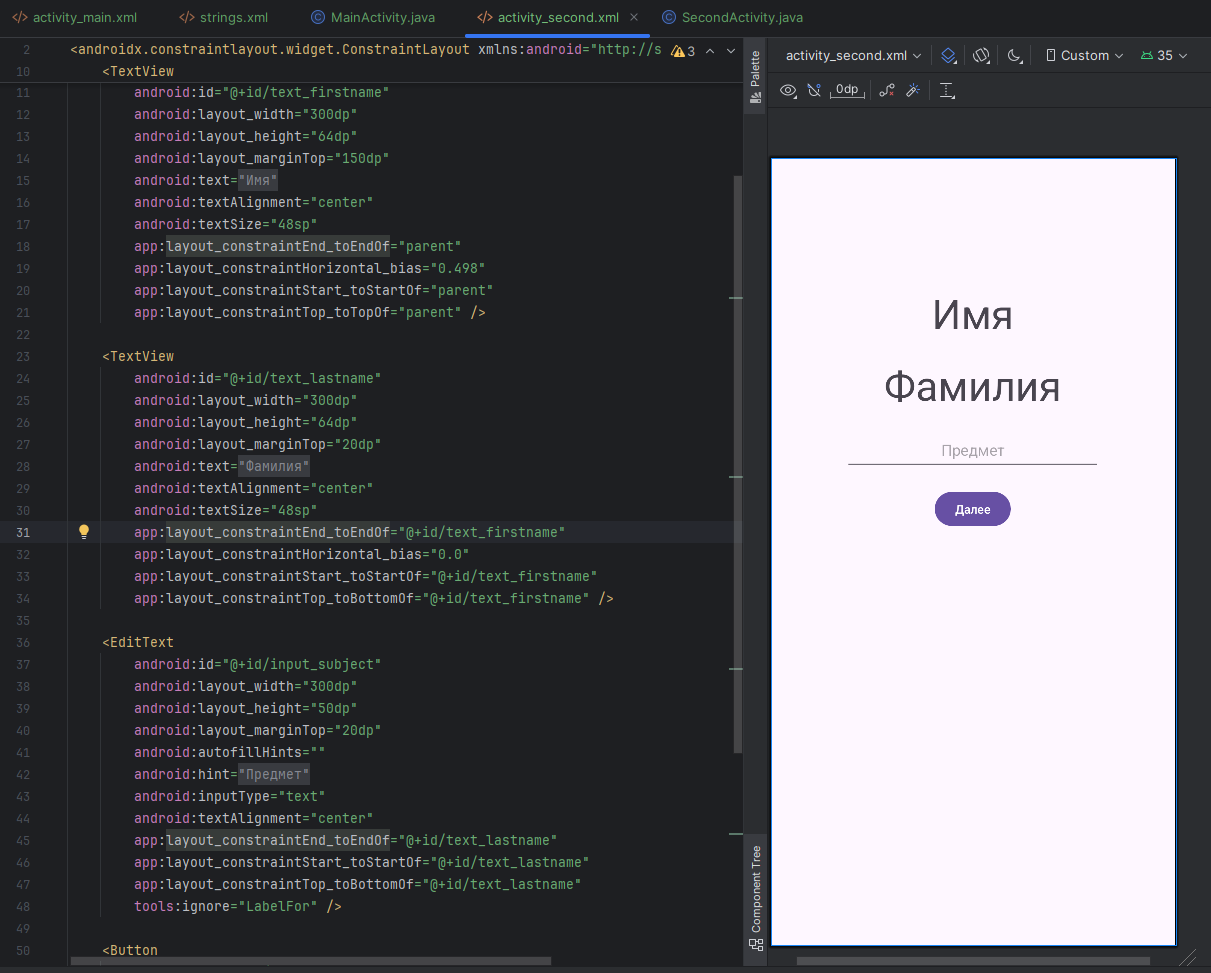
  
Рисунок 28 – Активность MainActivity

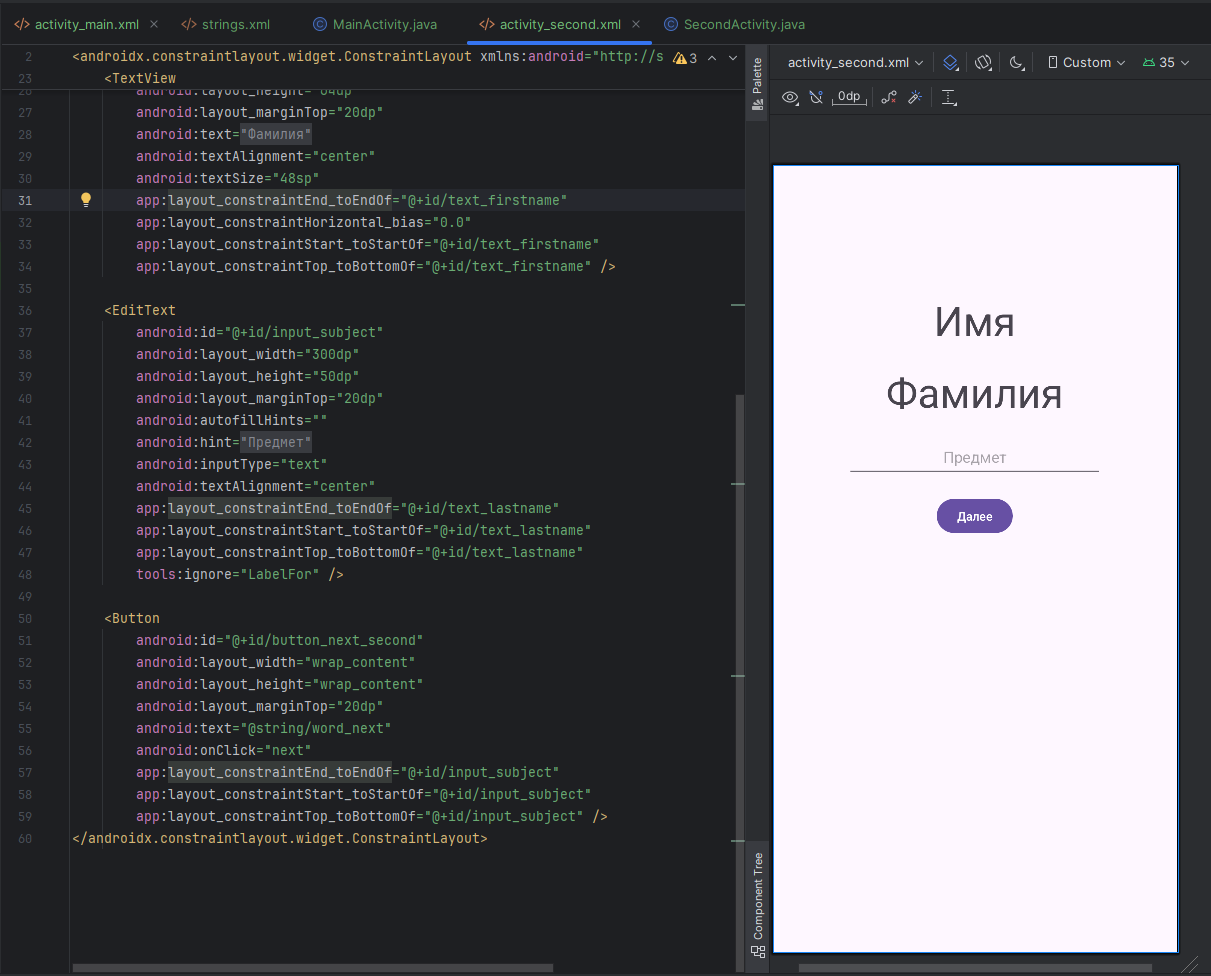
Декларативно объявляем вызов метода next при нажатии кнопки.

Затем нужно написать обработчик ввода и перевода на новую активность. Код класса MainActivity представлен на рисунке 29.

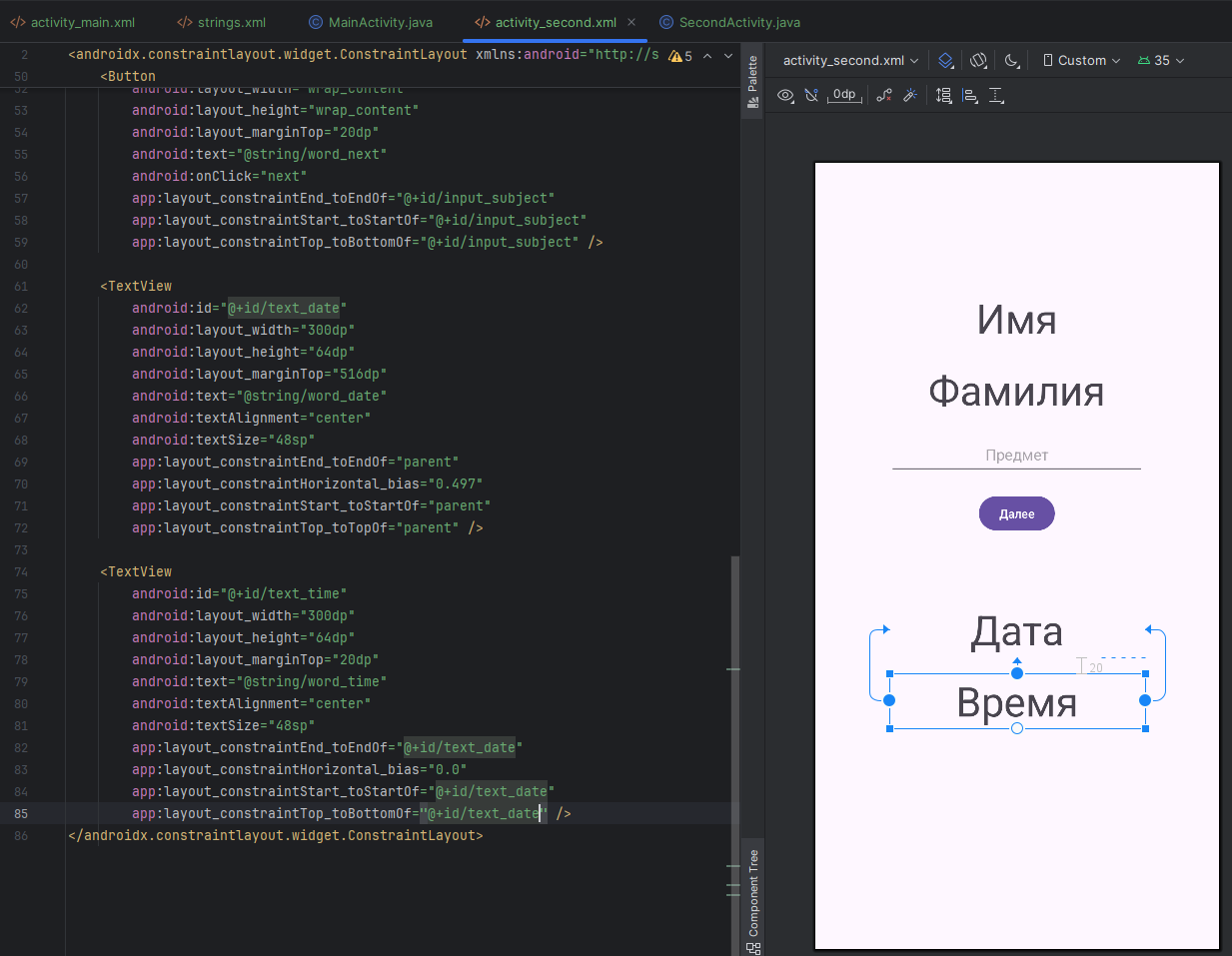
  
Рисунок 29 – Код класса MainActivity

В файле разметки activity\_second.xml располагаются два элемента TextView для отображения имени и фамилии, один элемент EditText для ввода предмета и кнопка для подтверждения (также декларативно установим вызов метода). Код файлов разметки представлен на рисунках 30-31.

  
Рисунок 30 – Код файла разметки activity\_second.xml, часть 1

  
Рисунок 31 – Код файла разметки activity\_second.xml, часть 2

Добавим также два компонента TextView, значение текста в которых будет установлено в соответствии с данными, которые будут возвращены из ThirdActivity (рисунок 32).

  
Рисунок 32 – Добавленные компоненты для отображения даты и времени

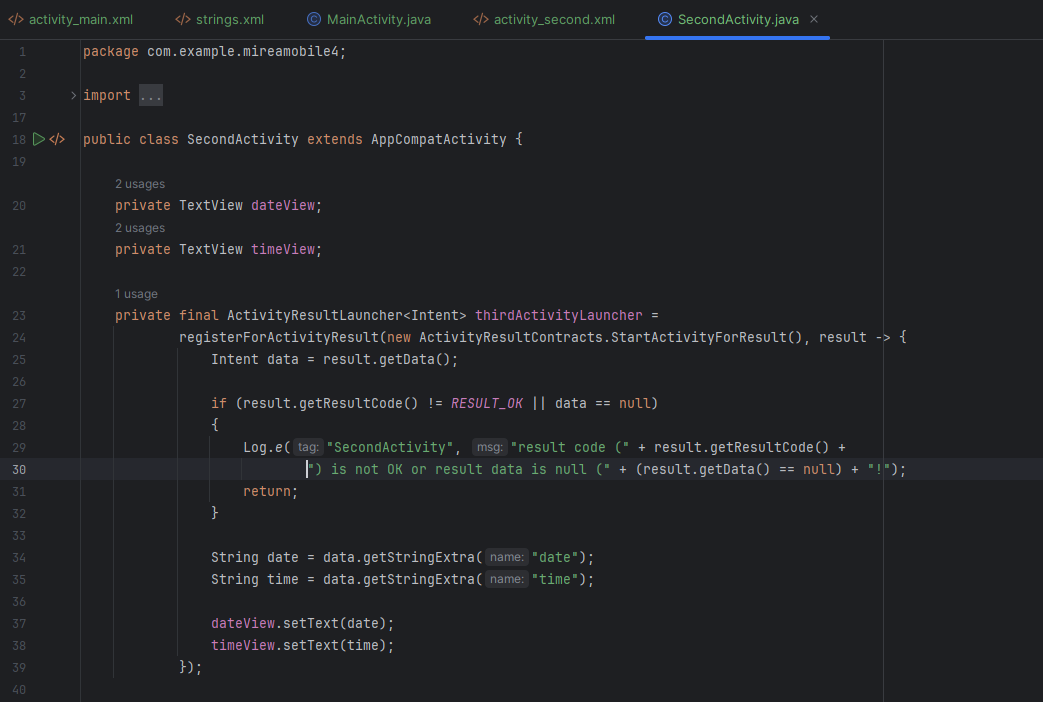
В классе SecondActivity будет проводиться запись данных в соответствующие поля.

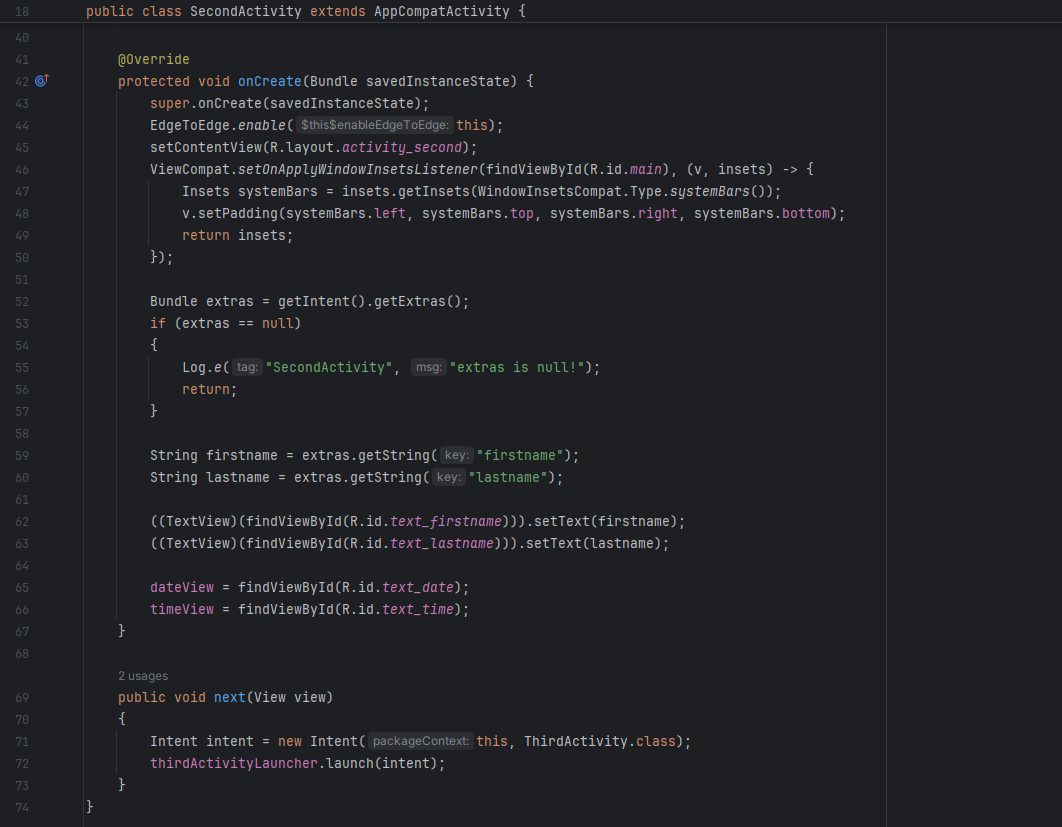
Кнопка «Далее» будет вести на ThirdActvivity.

Имя и фамилия будут приниматься через getExtras при создании активности под ключами «firstname» и «lastname» соответственно.

Дата и время же будут передаваться через ActivityResultLauncher, чтобы получить данные обратно после открытия ThirdActivity. В колбэке обрабатывается результат, проверяется, что resultCode равен RESULT\_OK, и что getData() не null. Далее извлекаются данные из Intent с помощью getStringExtra(), где «date» и «time» – это ключи переданных значений.

Код класса SecondActivity представлен на рисунках 33-34.

  
Рисунок 33 – Код класса SecondActvity, часть 1

  
Рисунок 34 – Код класса SecondActivity, часть 2

## Работа с фрагментами

Акакаа

## Переключение между фрагментами

Быбыбы

ЗАКЛЮЧЕНИЕ

В ходе выполнения данного проекта были изучены различные виды контейнеров разметки, таких как LinearLayout, RelativeLayout, ConstraintLayout и FrameLayout, что позволило лучше понять принципы построения интерфейсов в Android. Созданы несколько XML-файлов, в которых реализовано размещение элементов управления, включая кнопки, текстовые поля и другие виджеты, что продемонстрировало возможности гибкого позиционирования и адаптивного дизайна.

Кроме того, была реализована навигация между несколькими Activity с передачей данных посредством Intent, что обеспечило возможность отображения конкретной информации в соответствующих полях и последующего возврата к предыдущему экрану. В проект интегрированы различные ресурсы – строковые, размерные, цветовые и графические, а также выполнена настройка темы приложения, что позволило создать единый стиль интерфейса и обеспечить локализацию на другой язык. Эти доработки значительно расширили функциональные возможности приложения и улучшили его удобство для пользователей.