|  |  |  |
| --- | --- | --- |
|  |  |  |
| 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 №4**

по дисциплине «Разработка мобильных приложений»

|  |  |
| --- | --- |
| **Выполнил:**  Студент группыИКБО-20-23 | Комисарик М.А. |
| **Проверил:**  Старший преподаватель кафедры МОСИТ | Шешуков Л.С. |

Москва 2025 г.СОДЕРЖАНИЕ

[1 ТЕОРЕТИЧЕСКОЕ ВВЕДЕНИЕ 2](#_Toc192942250)

[2 ПРАКТИЧЕСКОЕ ЗАДАНИЕ 2](#_Toc192942261)

# ТЕОРЕТИЧЕСКОЕ ВВЕДЕНИЕ

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

В прошлых темах 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 (Рисунок 1).

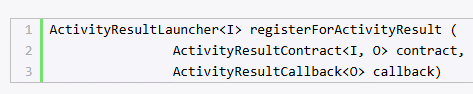


Рисунок 1 – Метод registerForActivityResult()

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. Пример представлен на рисунке Рисунок 2 – Пример метода registerForActivityResult().

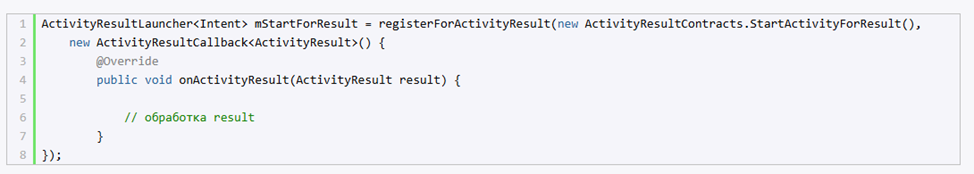


Рисунок 2 – Пример метода registerForActivityResult()

Класс ActivityResultContracts предоставляет ряд встроенных типов контрактов. Например, на рисунке 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-callback и возвращает объект ActivityResultLauncher. С помощью этого мы можем запустить activity. Для этого у объекта ActivityResultLauncher вызывается метод launch() (Рисунок 3).



Рисунок 3 – Вызов метода launch() объекта типа ActivityResultLauncher

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

Рассмотрим конкретный пример.

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 (Рисунок 4).

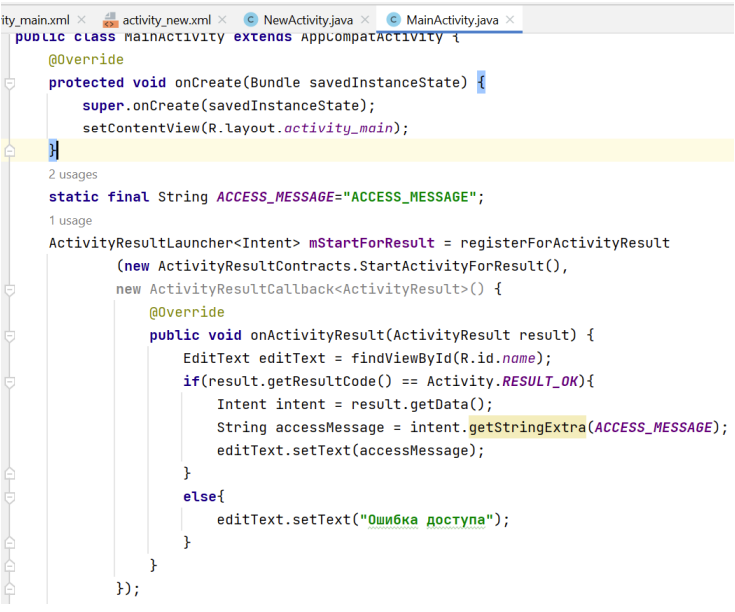


Рисунок 4 – Код класса MainActivity

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-callback 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 мы проверяем полученный код результата (Рисунок 5).



Рисунок 5 – Проверка кода 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 (Рисунок 6).



Рисунок 6 – Получение объекта Intent

Таким образом, мы определили объект ActivityResultLauncher.

Далее в обработчике нажатия onClick с помощью этого объекта запускаем вторую activity (Рисунок 7).



Рисунок 7 – Обработчик нажатий для перехода к NextActivity

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

После этого определим логику работы второй Activity (Рисунки Рисунок 8-Рисунок 9).

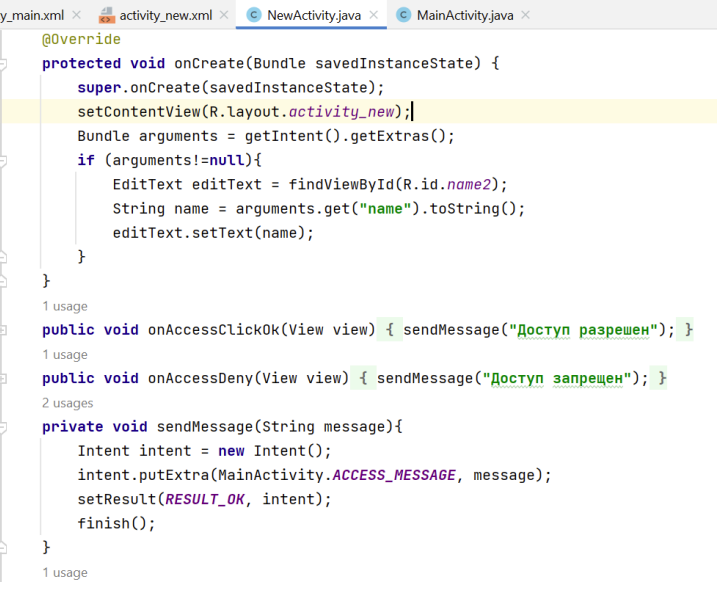


Рисунок 8 – Код класса NewActivity

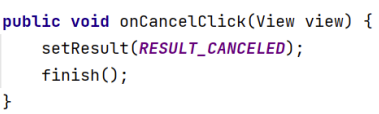


Рисунок 9 – Метод onCancelClick()

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

Для возврата результата необходимо вызвать метод setResult(), в который передается два параметра: числовой код результата и отправляемые данные.

После вызова метода setResult() нужно вызвать метод finish, который уничтожит текущую activity.

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

То есть условно говоря, мы получаем в NewActivity введенное в MainActivity имя и с помощью нажатия определенной кнопки возвращаем некоторый результат в виде сообщения. В зависимости от нажатой кнопки на NewActivity мы будем получать разные результаты в MainActivity.

Имя вводится в MainActivity (Рисунок 10).

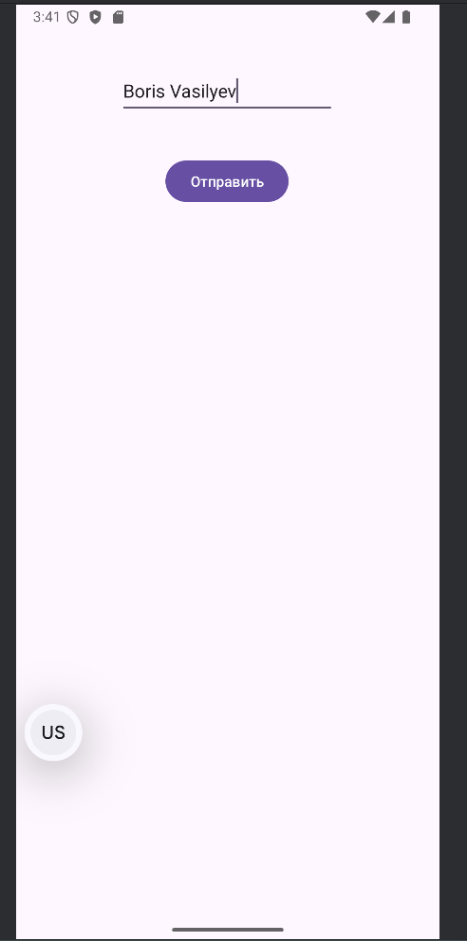


Рисунок 10 – Экран MainActivity

На экране NextActivity отображается имя переданное из MainActivity, а также представлены три кнопки (Рисунок 11).



Рисунок 11 – Экран NextActivity

Если пользователь нажимает на кнопку «Разрешить доступ» или на кнопку «Запретить доступ», то в NewActivity вызывается метод onAccessClickOk(View view), который создает Intent, добавляет в него строку "Доступ разрешен" или "Доступ запрещен", в зависимости от нажатия соответствующей кнопки, с ключом ACCESS\_MESSAGE, устанавливает результат RESULT\_OK и передает intent в setResult(), после чего завершает NewActivity с помощью finish(). Затем управление возвращается в MainActivity, где в onActivityResult проверяется код результата, и если он равен RESULT\_OK, то из полученного Intent извлекается строка ACCESS\_MESSAGE и устанавливается в EditText, иначе в поле ввода отображается текст "Ошибка доступа" (Рисунки Рисунок 12-Рисунок 13).



Рисунок 12 – Результат нажатия на кнопку «Разрешить доступ»

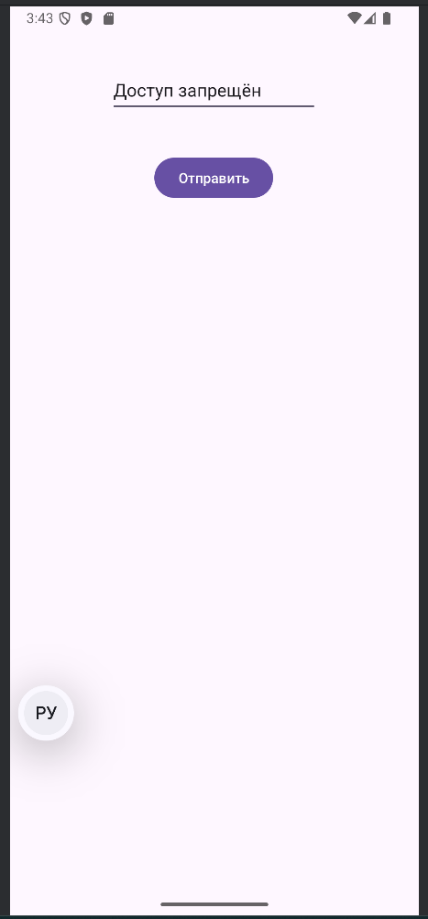


Рисунок 13 – Результат нажатия на кнопку «Запретить доступ»

Если же пользователь на экране NewActivity нажал на кнопку «Отмена», то вызывается метод onCancelClick(View view), который устанавливает результат RESULT\_CANCELED с помощью setResult(RESULT\_CANCELED) и завершает NewActivity, вызывая finish(). После этого управление возвращается в MainActivity, где метод onActivityResult получает результат. Так как код результата не равен RESULT\_OK, выполняется блок else, и в поле EditText устанавливается текст "Ошибка доступа" (Рисунок 14).

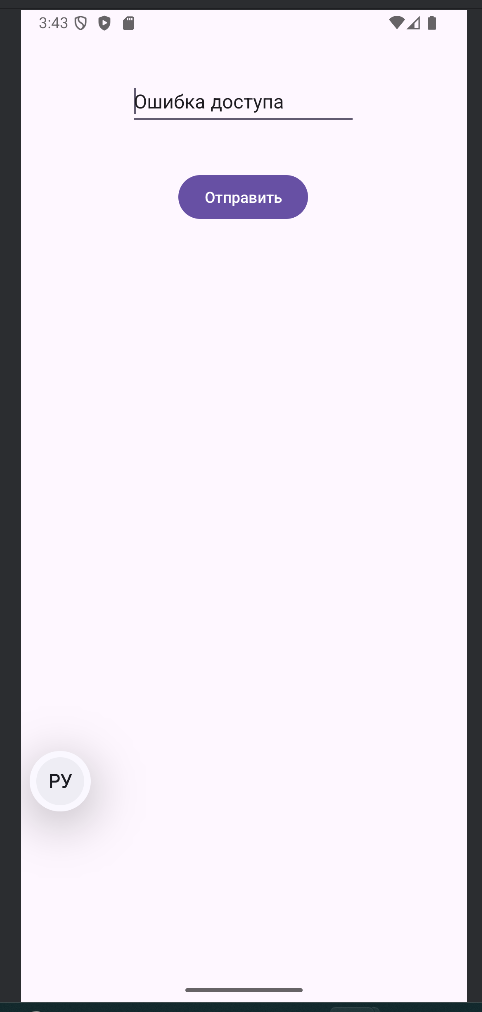


Рисунок 14 – Результат нажатия на кнопку «Отмена»

## Fragment

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 Fragment.

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

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

Среди основных характеристик Fragment можно выделить следующие:

* модульность: Fragment позволяют разделить активность 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 активностях,
* адаптивность: Fragment 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 Fragment имеет свой собственный жизненный цикл, но он тесно связан с жизненным циклом своей хост-activity. Это позволяет Fragment управлять своим состоянием и поведением в зависимости от состояния activity.

## Жизненный цикл Fragment

Жизненный цикл Fragment в Android представляет собой последовательность состояний, через которые проходит Fragment во время своего существования. Связанные с этими состояниями callback-методы позволяют управлять поведением Fragment на различных этапах его взаимодействия с пользователем и системой (Рисунок 15).

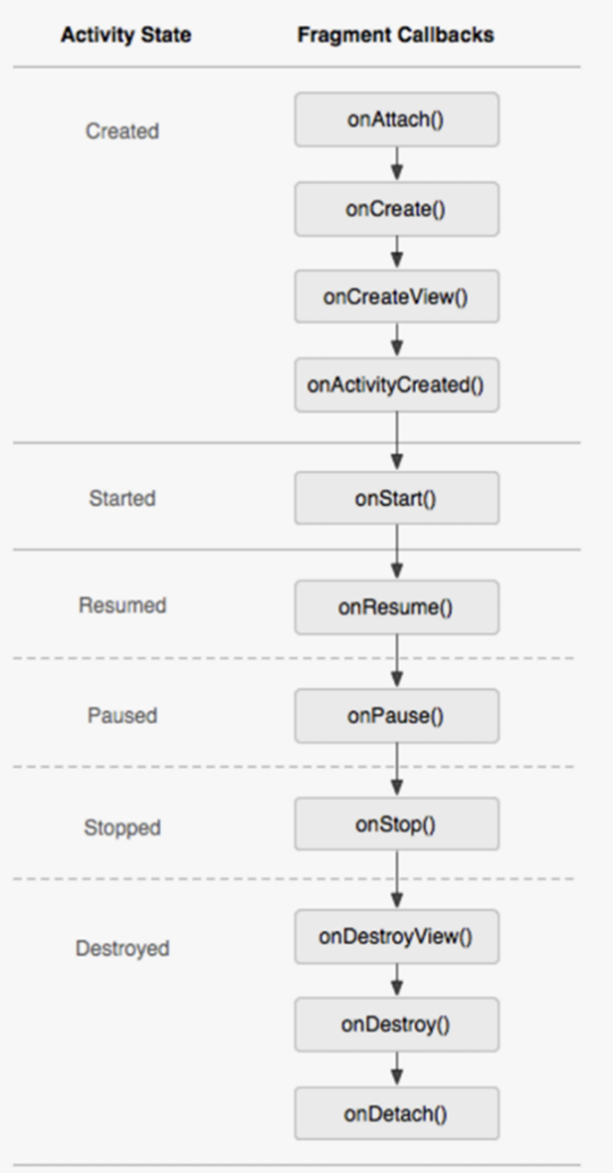


Рисунок 15 – Жизненный цикл Fragment

Привязка к активности: Когда Fragment привязывается к активности, вызывается метод onAttach(). Это означает, что Fragment теперь ассоциирован с активностью, и разработчик может взаимодействовать с ней.

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

Создание представления Fragment: Метод onCreateView() вызывается для создания представления Fragment (Рисунок 16). Здесь загружается макет, определяет пользовательский интерфейс Fragment. После создания представления оно возвращается системе для отображения.

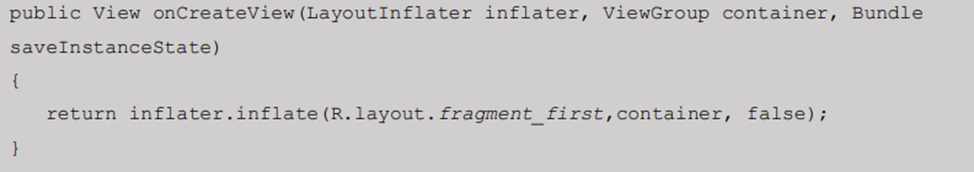


Рисунок 16 – Метод onCreateNew()

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

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

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

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

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

Запуск Fragment: Когда Fragment становится видимым для пользователя, вызывается метод onStart(). Здесь можно запускать анимации или выполнять задачи, которые должны быть видны пользователю.

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

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

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

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

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

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

## Создание и размещение Fragment

Можно добавить по отдельности Java класс, который представляет Fragment, и файл xml для хранения в нем разметки интерфейса, который будет использовать Fragment, однако Android Studio представляет готовый шаблон для добавления Fragment. Этим способом и будем пользоваться.

Принцип создания новых Fragment схож с созданием activity. Нужно выбрать модуль "app", затем в верней панели выбрав "File" → "New" → "Fragment" и выбрать тип активности, например "Fragment (Blank)" либо нажать правой кнопкой мыши на "java"→ "New" → "Fragment", после чего выбрать название Fragment и как в случае с активностью будут созданы файлы класса и разметки (Рисунки Рисунок 17-Рисунок 18).

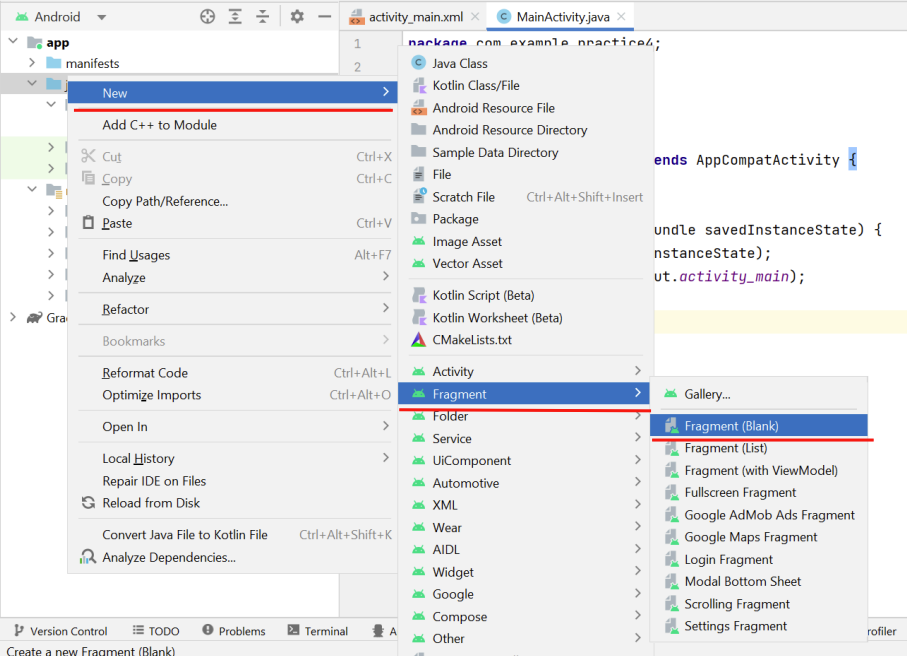


Рисунок 17 – Создание нового Fragment с помощью Android Studio

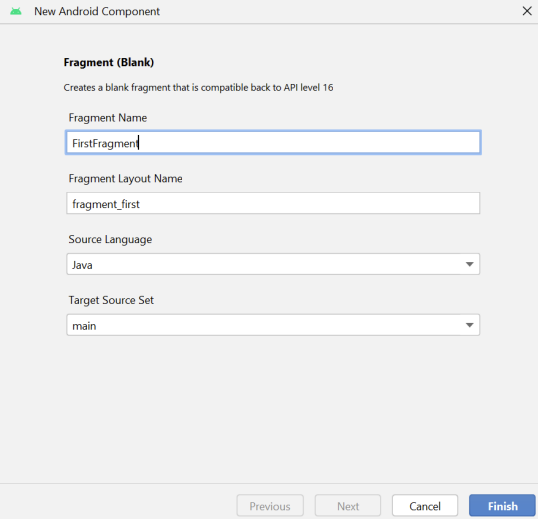


Рисунок 18 – Настройки при создании нового Fragment

После создания Fragment, класс для реализации Fragment и xml-файл для разметки будут созданы автоматически (Рисунок 19).

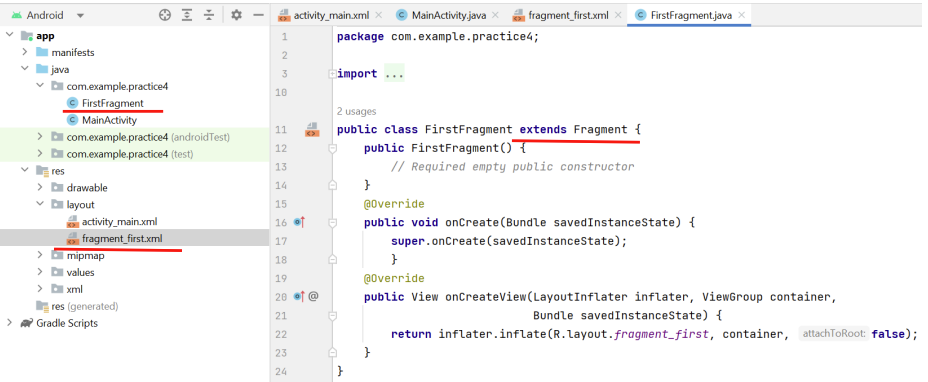


Рисунок 19 – Код созданного Fragment класса

Класс созданного Fragment должен наследоваться от класса Fragment. Чтобы указать, что Fragment будет использовать определенный xml-файл layout, идентификатор ресурса layout передается в вызов конструктора родительского класса (то есть класса Fragment).

Метод onCreateView предназначен для создания и возврата разметки Fragment, когда система запрашивает ее отображение. В данном случае он принимает три параметра: LayoutInflater inflater, который используется для раздувания (создания) представления из XML-файла, ViewGroup container, который является родительским контейнером для создаваемого представления, и Bundle savedInstanceState, который может содержать сохраненное состояние Fragment.

Метод inflater.inflate(int resource, ViewGroup root, boolean attachToRoot) используется для создания (раздувания) представления на основе XML-разметки.

Первый параметр resource принимает идентификатор ресурса разметки, который необходимо развернуть. Второй параметр root представляет родительский контейнер, к которому потенциально может быть присоединено созданное представление. Третий параметр attachToRoot определяет, нужно ли сразу прикреплять созданное представление к root.

Если attachToRoot установлен в true, созданное представление сразу добавляется в root, что полезно, если необходимо использовать параметры компоновки родительского контейнера. Если он установлен в false, разметка просто создается, но не добавляется к root, и это остается на усмотрение вызывающего кода.

Теперь после создания Fragment его необходимо разместить на активности. Сделать это можно тремя способами: статически, динамически и через элемент fragmentContainerView. Для размещения Fragment статическим способом необходимо добавить в XML файл активности элемент fragment (Рисунок 20).

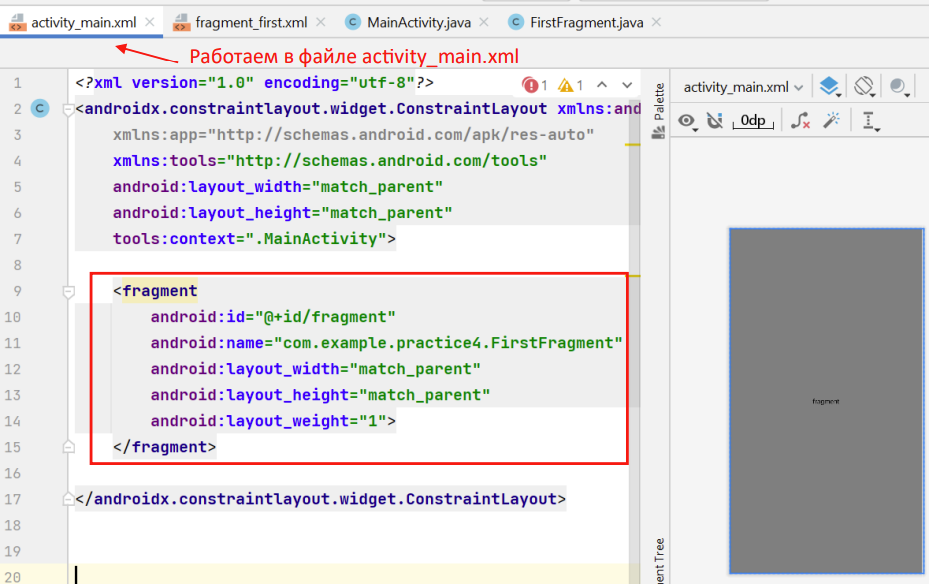


Рисунок 20 – Размещение Fragment статическим способом

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

В качестве примера во Fragment было определено текстовое поле с некоторым текстовым значением (Рисунок 21).

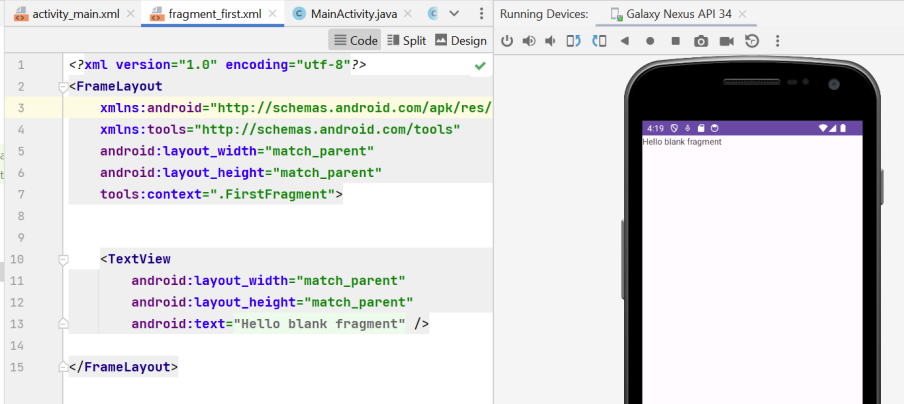


Рисунок 21 – Разметка Fragment

Теперь данный Fragment будет показываться при запуске приложения. Главным недостатком такого размещения Fragment является то, что статические Fragment тесно связаны с жизненным циклом активности, что может усложнить управление их состоянием. Например, сохранение и восстановление состояния Fragment при пересоздании активности может потребовать дополнительной логики.

Все преимущества Fragment раскрываются при их динамическом изменении в процессе работы приложения. Fragment, при динамическом размещении, управляются аналогично обычным View элементам. Для облегчения управления, каждый Fragment размещается в отдельном контейнере. Обычно для этой цели выбираются контейнеры типа FrameLayout (Рисунок 22).



Рисунок 22 – Контейнер FrameLayout для размещения Fragment

Далее в классе активности необходимо «разместить» данный Fragment, используя метод getSupportFragmentManager() класса FragmentActivity (Рисунок 23).

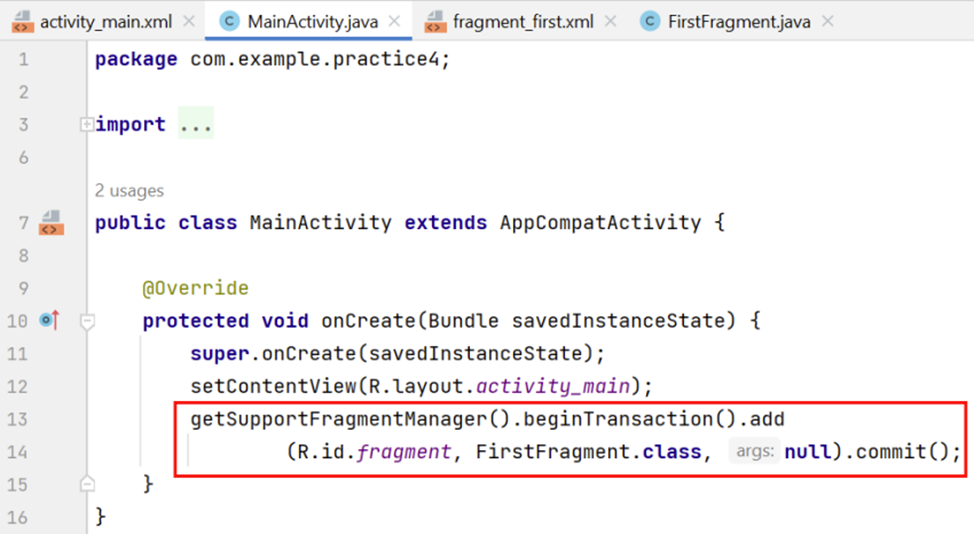


Рисунок 23 – Динамическое добавление Fragment в MainActivity с использованием FragmentManager

Метод getSupportFragmentManager() возвращает объект FragmentManager, который управляет Fragment. Объект FragmentManager с помощью метода beginTransaction() создает объект FragmentTransaction. FragmentTransaction выполняет два метода: add() и commit().

Метод add() добавляет указанный Fragment в контейнер макета, переданный в качестве первого аргумента. Этот метод не сразу отображает Fragment, а лишь подготавливает изменения, которые будут применены в рамках транзакции. Метод commit() завершает транзакцию и применяет все изменения, которые были добавлены в FragmentTransaction, что делает Fragment активной частью пользовательского интерфейса.

Транзакция Fragment представляет собой последовательность операций, таких как добавление, удаление или замена Fragment, которые выполняются как единое целое. Это обеспечивает атомарность изменений и корректное управление состоянием Fragment в пределах активности.

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

Для добавления фрамента применяется элемент FragmentContainerView. FragmentContainerView представляет объект View, который расширяет класс FrameLayout и предназначен специально для работы с Fragment.

Для его использования нужно разместить данный элемент в XML файле активности (Рисунок 24).

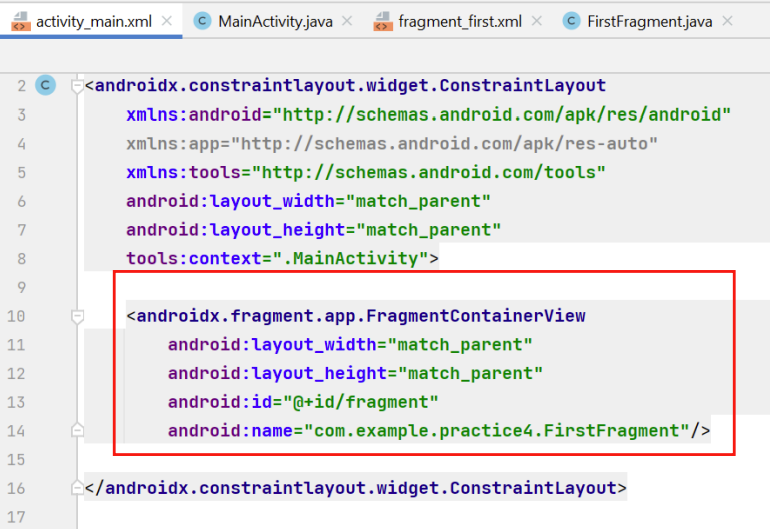


Рисунок 24 – Добавление Fragment в MainActivity с использованием FragmentContainerNew

Атрибут android:name в элементе FragmentContainerView указывает на класс Fragment, который должен быть загружен в этот контейнер при создании макета активности. При указании этого атрибута система автоматически создает экземпляр Fragment и добавляет его в контейнер без необходимости явного добавления через FragmentManager. Это позволяет задать Fragment в разметке XML, а не программно, что упрощает инициализацию и управление пользовательским интерфейсом.

Как и в случае с FrameLayout, FragmentContainerView позволяет управлять Fragment с помощью FragmentManager. Это дает возможность динамически добавлять, заменять и удалять Fragment, используя FragmentTransaction. Хотя через атрибут android:name можно задать Fragment, который загрузится автоматически, управление Fragment также может осуществляться программно, что делает FragmentContainerView удобным инструментом для работы с Fragment в Android.

## Навигация между Fragment

Навигация между Fragment осуществляется с помощью FragmentManager, который управляет операциями, такими как добавление, удаление или замена Fragment в контейнере. Ключевым понятием в навигации между Fragment является транзакция. Транзакция Fragment — это серия действий, которые выполняются вместе.

Для навигации между Fragment можно использовать методы replace и commit класса FragmentTransaction. Сначала необходимо создать экземпляр Fragment, в который планируется переход, затем передать его в метод replace, который заменит текущий Fragment на новый, и завершить операцию вызовом commit, чтобы зафиксировать изменения.

Метод beginTransaction() используется для создания новой Fragment транзакции, которая позволяет выполнять операции с Fragment, такие как добавление, удаление или замена. Он возвращает объект FragmentTransaction.

Метод replace(int containerViewId, Fragment fragment) выполняет замену текущего Fragment в указанном контейнере на новый Fragment. Первый параметр containerViewId представляет собой идентификатор контейнера (например, R.id.fragmentContainerView), в котором будет происходить замена. Второй параметр fragment — это новый экземпляр Fragment, который необходимо отобразить. После вызова replace() требуется выполнить commit(), чтобы зафиксировать транзакцию и применить изменения.

На рисунке Рисунок 25 представлена демонстрация использования методов getSupportFragmentManager(), beginTransaction(), replace() и commit(), которые позволяют заменить один Fragment другим внутри контейнера FragmentContainerView. Метод replaceFragment создает новую транзакцию, выполняет замену Fragment и применяет изменения.

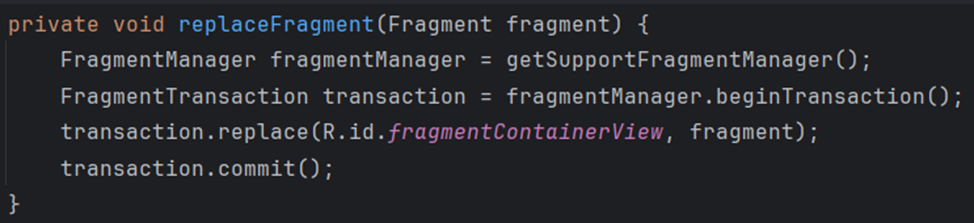


Рисунок 25 – Добавление Fragment в MainActivity с использованием FragmentContainerNew

На рисунке Рисунок 26 представлена часть кода, находящаяся в методе onCreate(). В нем устанавливаются обработчики нажатий на кнопки, при срабатывании которых вызывается метод replaceFragment(), заменяющий текущий Fragment на новый экземпляр Fragment\_1, Fragment\_2 или Fragment\_3. Также предусмотрена проверка if (savedInstanceState == null), которая гарантирует, что при первом запуске активности загружается Fragment\_1 по умолчанию.

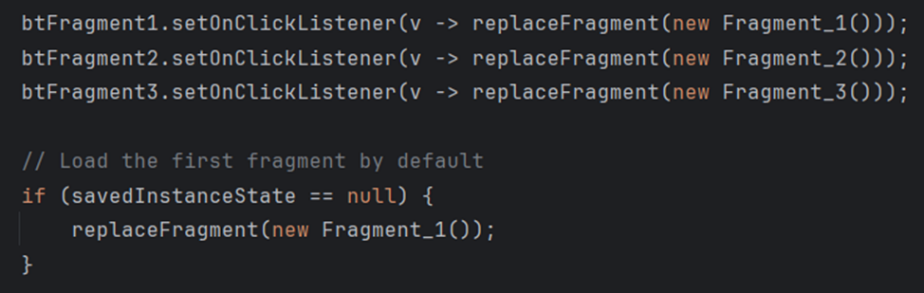


Рисунок 26 – Добавление Fragment в MainActivity с использованием FragmentContainerNew

На рисунках Рисунок 27-Рисунок 28 демонстрируется процесс смены Fragment в Android-приложении. При первом запуске автоматически загружается Fragment\_1 и отображается на экране. При нажатии на одну из кнопок происходит замена текущего Fragment на соответствующий выбранной кнопке, в данном случае – Fragment\_2.



Рисунок 27 – Начальный экран с загруженным Fragment\_1



Рисунок 28 – Интерфейс после переключение на Fragment\_2

# ПРАКТИЧЕСКОЕ ЗАДАНИЕ

## Работа с тремя Activity

### Создание Activity и файлов разметки

При создании проекта по шаблону Empty Views Activity, в проект автоматически добавляется Activity с классом MainActivity и файлом разметки activity\_main.xml (Рисунок 29).

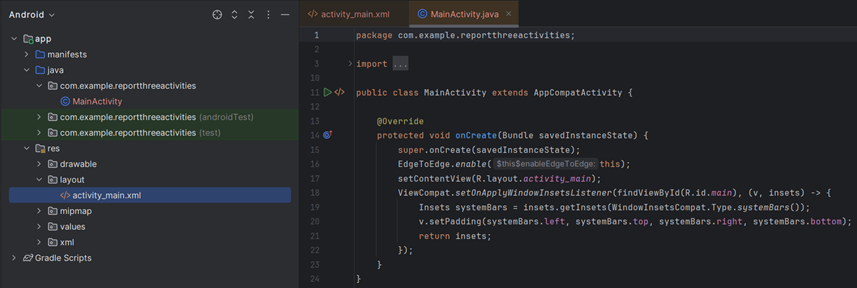


Рисунок 29 – Изначальный вид проекта

Добавим еще две Activity с названиями SecondActivity и ThirdActivity соответственно с помощью нажатия правой кнопкой мыши на "java"→ "New" → "Activity" → "Empty Views Activity". IDE Android Studio создаст как java классы, так и xml-файлы разметки (Рисунок 30).

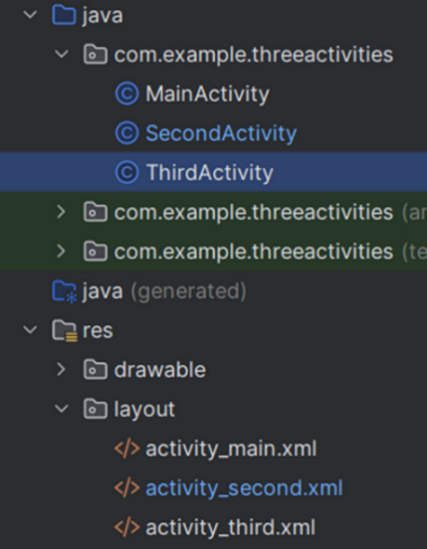


Рисунок 30 – Файлы классов и разметок трех Activity

В файле разметки activity\_main.xml расположим два поля для ввода имени и фамилии с id "edit\_main\_firstName" и "edit\_main\_lastName" соответственно, а также кнопку с надписью Next и id “button\_main\_next” (Рисунок 31).

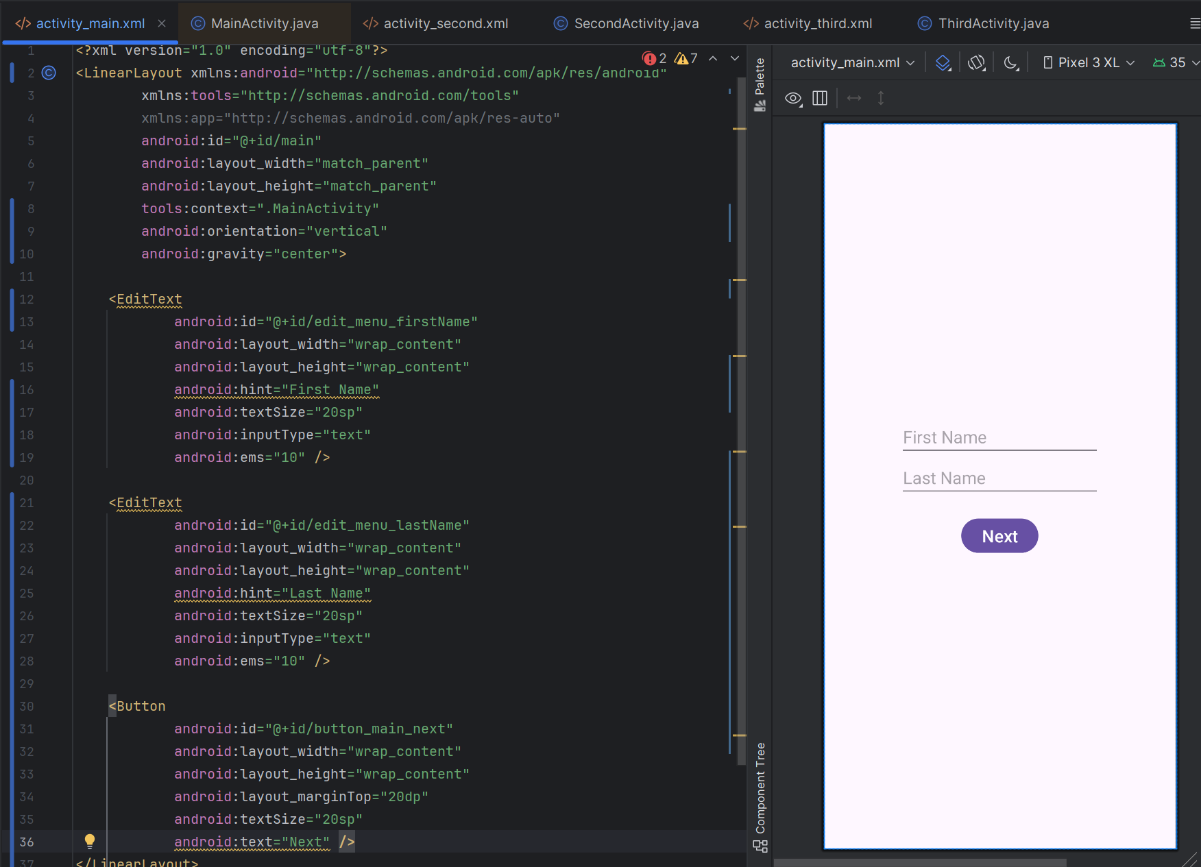


Рисунок 31 – Файл разметки activity\_main.xml

В файле разметки activity\_second.xml расположим два элемента TextView для отображения текста, изначально значения атрибута android:text установлено в значения "First name" и "Second name" соответственно, однако в дальнейшем текст будет заменен данными, переданными из MainActivity. Значения атрибутов android:id равны "text\_second\_firstName" и "text\_second\_lastName" соответственно (Рисунок 32).

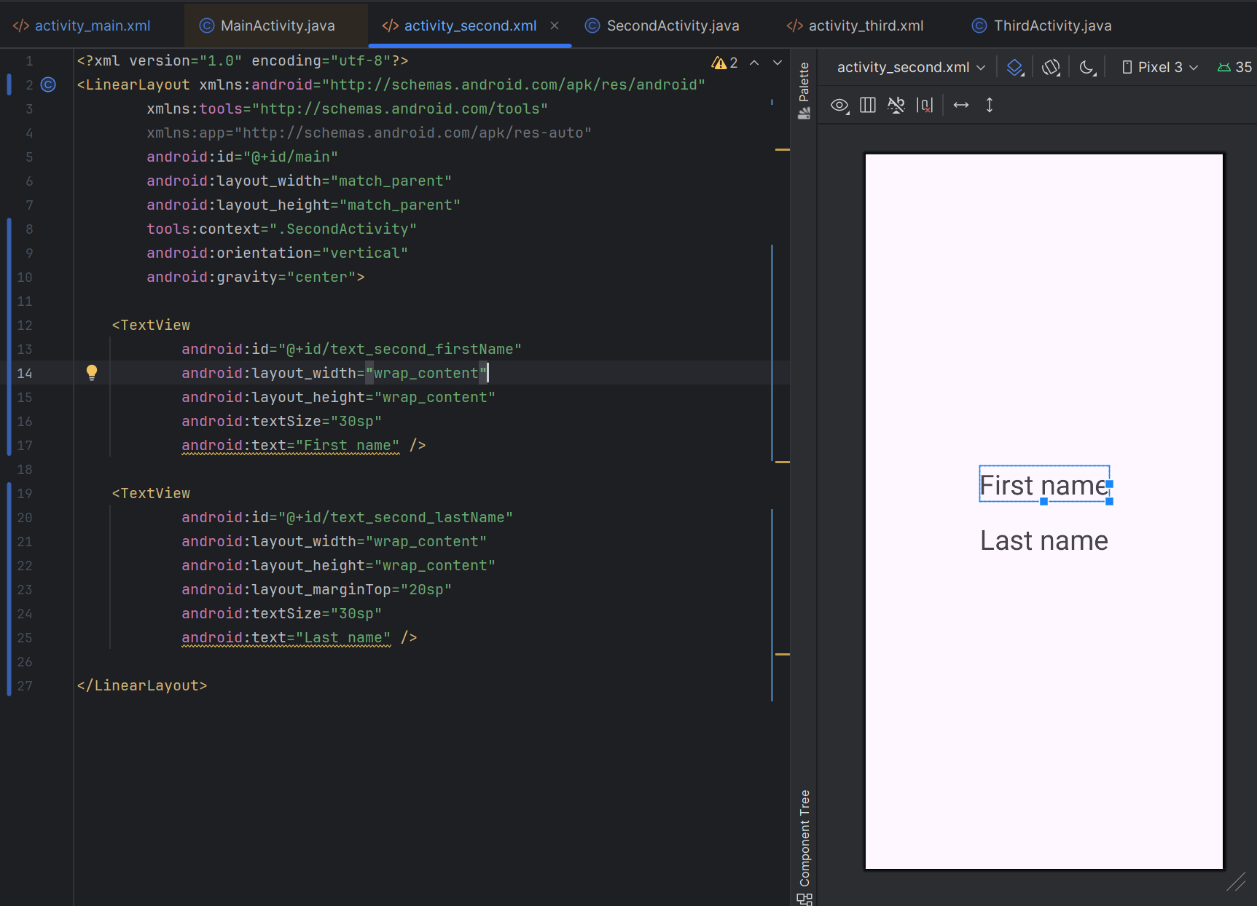


Рисунок 32 – Добавление элементов для отображения имени и фамилии в файле разметки activity\_second.xml

Далее добавим компонент EditText с id "edit\_second\_subject" для ввода предмета, компонент Button с id "button\_second\_submit", которая будет использована для запуска ThirdActivity, два компонента TextView с id "text\_second\_date" и "text\_second\_time" (Рисунок 33).

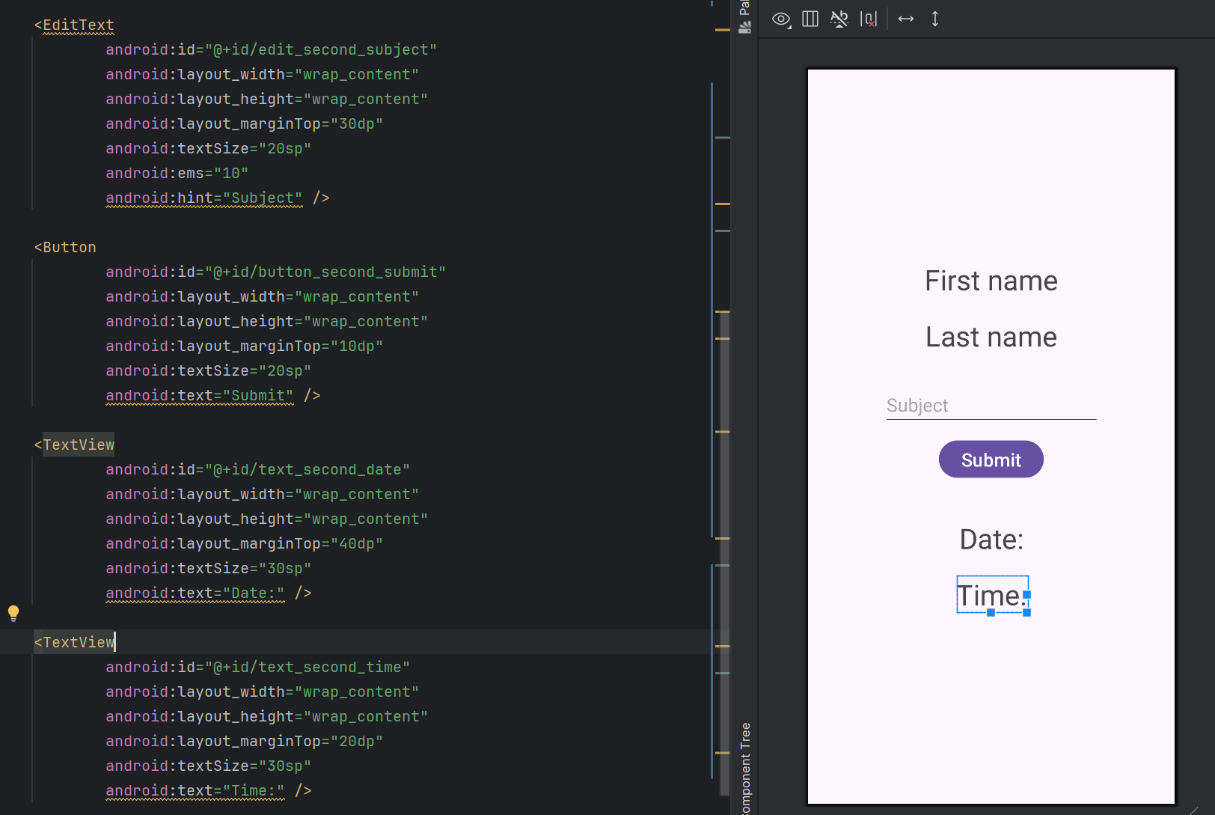


Рисунок 33 – Файл разметки activity\_second.xml

В файле activity\_third.xml расположим поле для ввода даты и поле для ввода времени с id="edit\_third\_date" и id="edit\_third\_time" соответственно. Для каждого из полей установлен атрибут android:inputType, который определяет тип данных, вводимых пользователем. В данном случае, inputType="date" задает формат ввода даты, а inputType="time" — формат времени. Выбор типа ввода настраивает отображаемую клавиатуру, допустимые символы и внешний вид поля ввода. Добавим также поле для ввода комментария с id="edit\_third\_comment", а также кнопку с id="button\_third\_ok". Для поля комментария установлен атрибут android:inputType="textMultiLine", позволяющий вводить многострочный текст. Кнопка OK будет использоваться для завершения работы ActivityThird и передачи введенных данных обратно в SecondActivity. (Рисунок 34).

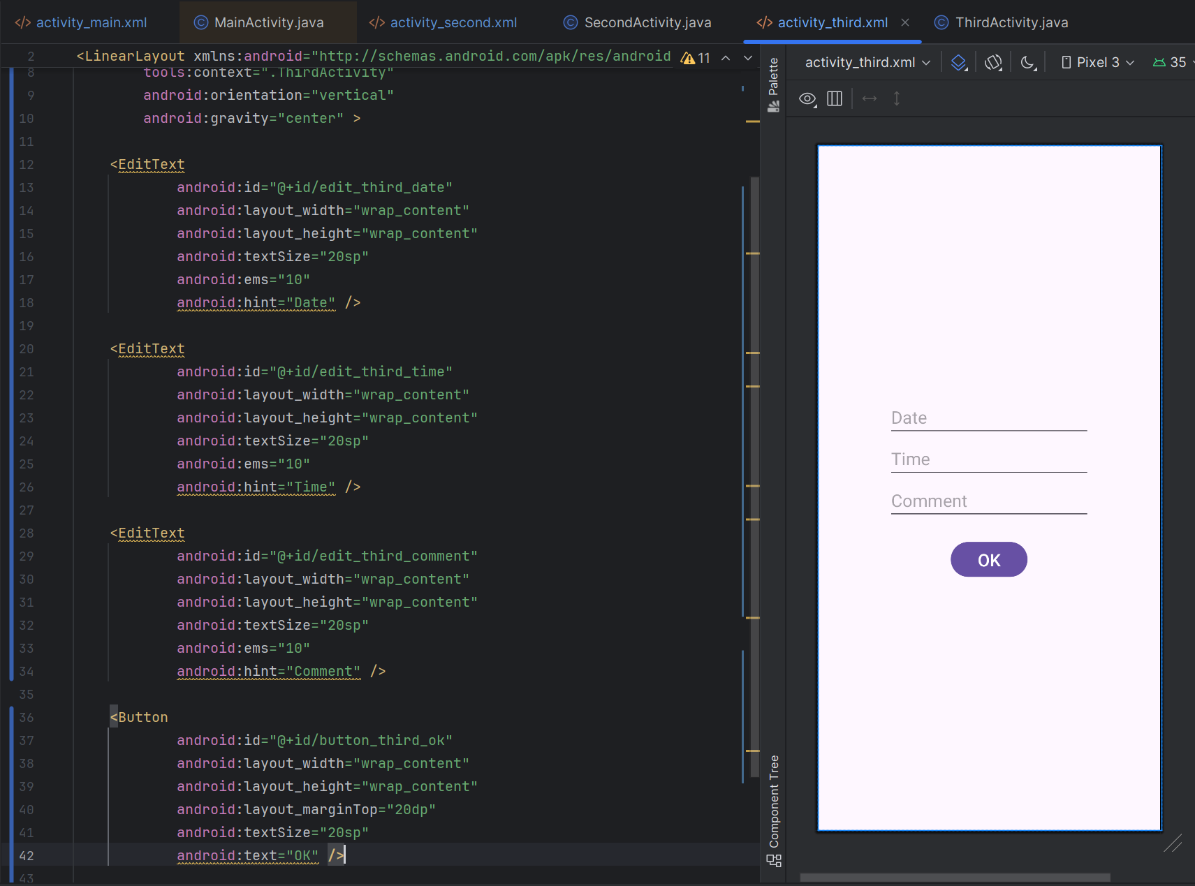


Рисунок 34 – Файл разметки activity\_third.xml

### Реализация перехода и передачи данных между активностями

В файле MainActivity.java в методе onCreate осуществляется обработка нажатия на кнопку editNext, при котором извлекаются данные, введенные пользователем в поля editFirstName и editLastName. Полученные значения преобразуются в строки с помощью getText().toString(), после чего формируется Intent для перехода на SecondActivity.

Передача данных осуществляется методом putExtra(), который добавляет в Intent пары "ключ-значение" для передачи строковых данных: имя (ARGUMENT\_FIRST\_NAME) и фамилия (ARGUMENT\_LAST\_NAME). Поскольку MainActivity не требует получения результата от SecondActivity, используется стандартный Intent с startActivity(intent), а не ActivityResult API, который применяется для обратного получения данных от целевой активности (Рисунок 35).

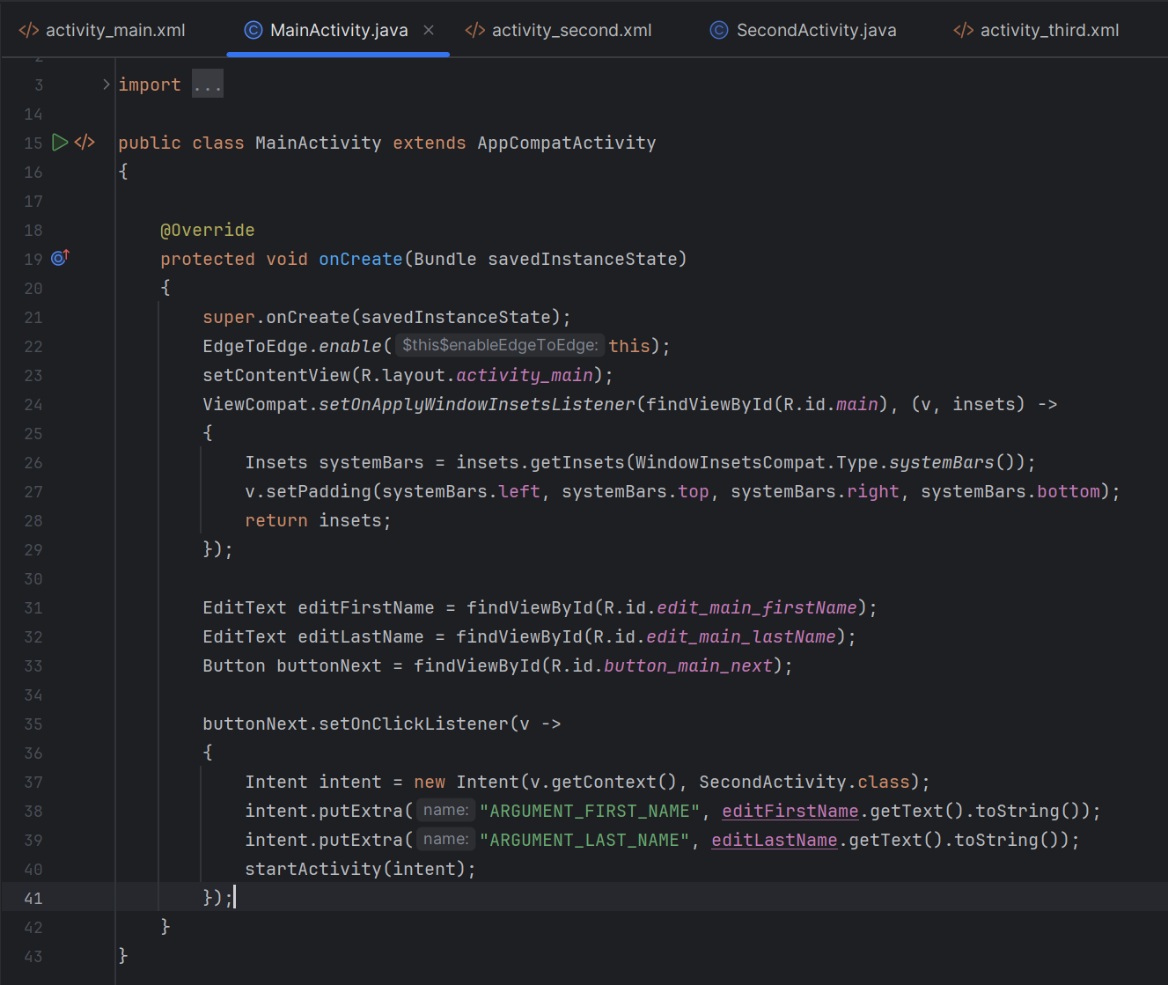


Рисунок 35 – Описание метода onCreate() класса MainActivity

В файле SecondActivity.java объявляется переменная thirdActivityLauncher, которая используется для запуска ThirdActivity и обработки возвращенных данных.

Для этого применяется ActivityResultLauncher<Intent>, который регистрируется через registerForActivityResult(). В анонимной функции-обработчике проверяется, что resultCode равен RESULT\_OK, и getData() не null. Далее извлекаются данные из Intent с помощью getStringExtra(), где "ARGUMENT\_DATE" и "ARGUMENT\_TIME" – это ключи переданных значений.

Затем с помощью Toast.makeText() выводится сообщение об успешной передаче данных. После этого найденные элементы TextView (textDate и textTime) заполняются полученными значениями, используя String.format() (Рисунок 36).

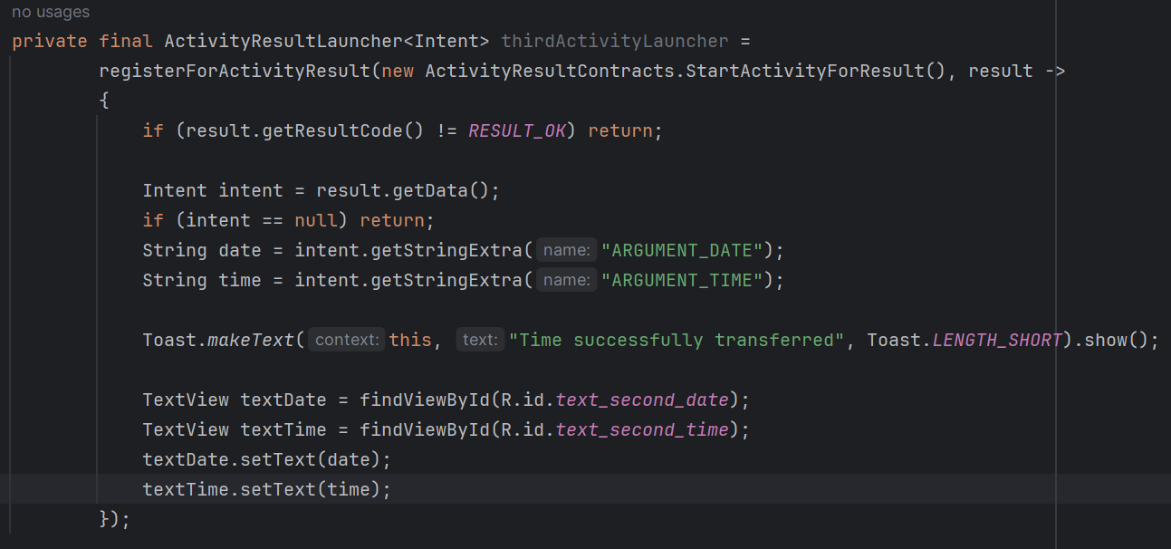


Рисунок 36 – Получение данных из ThirdActivity в SecondActivity с использованием ActivityResultLauncher

В файле SecondActivity.java в методе onCreate() происходит настройка интерфейса и обработка переданных данных. С помощью getIntent().getExtras() извлекается Bundle с переданными данными. Если extras не null, из него получаются строки "ARGUMENT\_FIRST\_NAME" и "ARGUMENT\_LAST\_NAME" с помощью getString(). Эти значения затем устанавливаются в TextView (textFirstName и textLastName) через setText().

Также в коде находится обработчик нажатия кнопки buttonSubmit, который создает Intent для перехода в ThirdActivity. Этот Intent передается в thirdActivityLauncher.launch(intent), что инициирует запуск ThirdActivity с возможностью получения результата обратно (Рисунок 37).

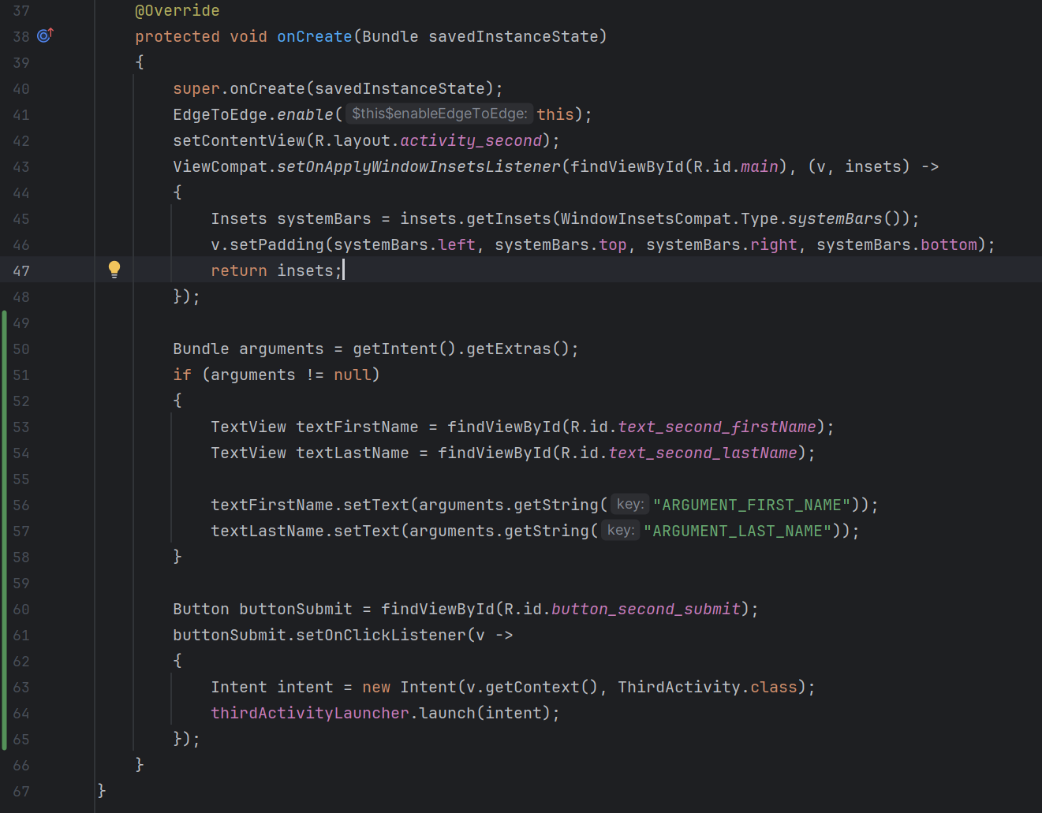


Рисунок 37 – Описание метода onCreate() класса SecondActivity

В классе ThirdActivity происходит обработка пользовательского ввода. Из разметки загружаются EditText для ввода времени (editTime) и даты (editDate), а также кнопка buttonOk. При нажатии на кнопку извлекаются введенные пользователем значения, которые затем добавляются в Intent в виде параметров с ключами "ARGUMENT\_TIME" и " ARGUMENT\_DATE". После этого вызывается setResult(RESULT\_OK, result), передавая результат обратно в вызывающую активность, и finish(), завершающий ThirdActivity (Рисунок 38).

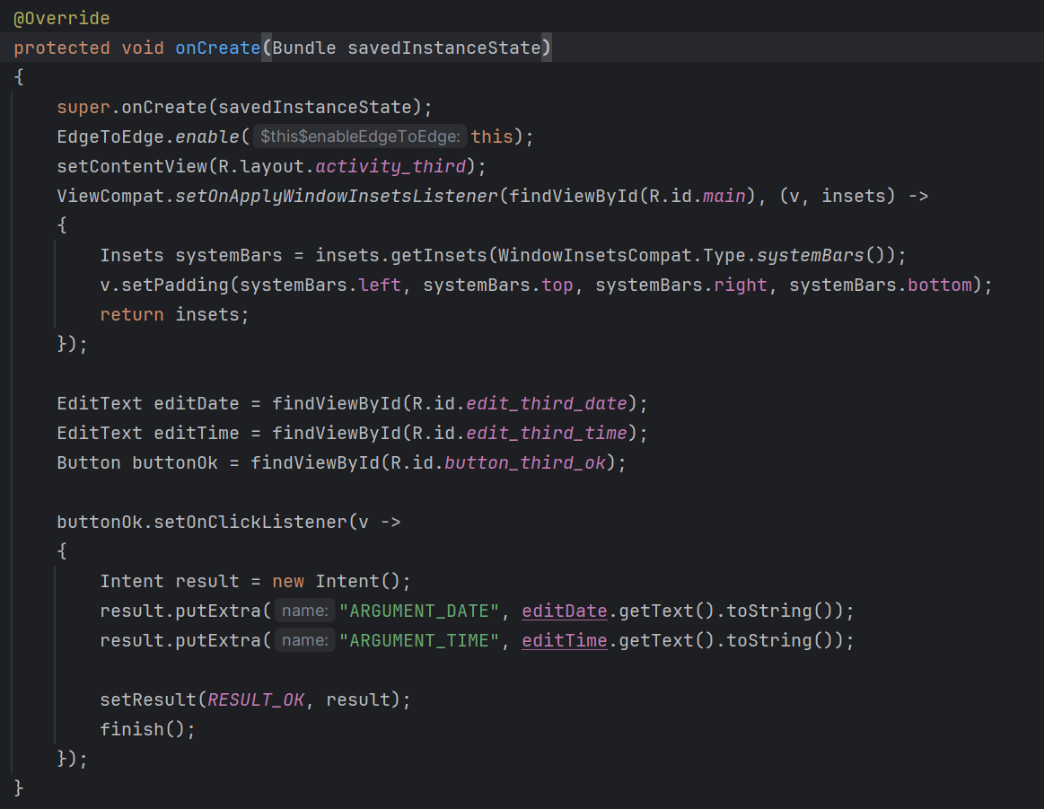


Рисунок 38 – Описание метода onCreate() класса ThirdActivity

### Тестирование

Запустим приложение. На экране MainActivity введем имя и фамилия в соответствующие поля EditText, а затем нажмем кнопку Next для перехода к SecondActivity (Рисунок 39).

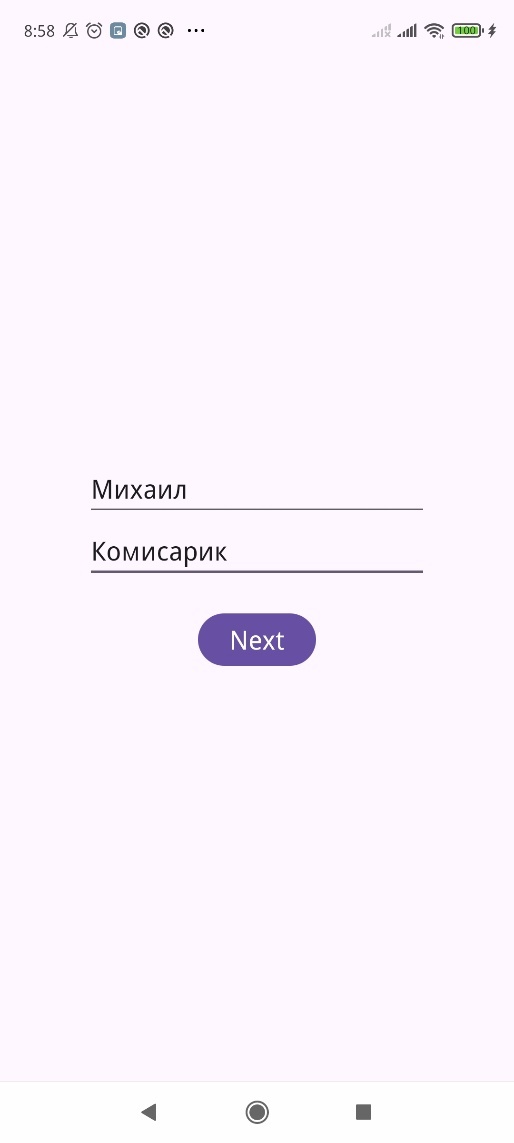


Рисунок 39 – Экран MainActivity

После перехода к SecondActivity увидим, что фамилия и имя были успешно переданы и отображены. Заполним поле для ввода предмета и нажмем кнопку Submit, чтобы запустить ThirdActivity с последующей передачей данных обратно в текущую Actvity, то есть в SecondActivity (Рисунок 40).

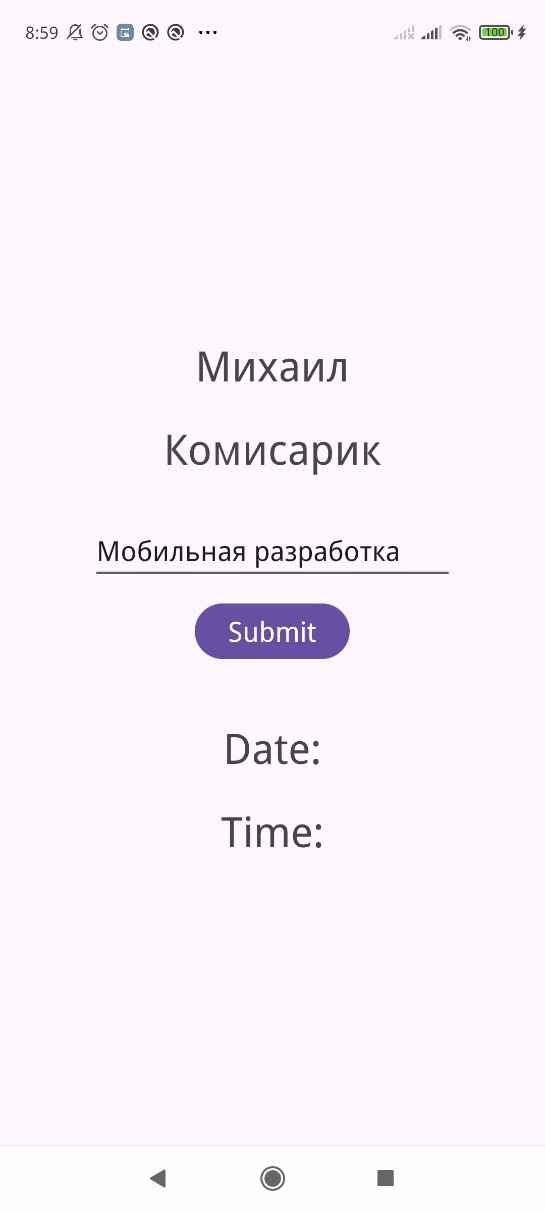


Рисунок 40 – Экран SecondActivity

На экране ThirdActivity заполним поля для ввода даты, времени и комментария, а затем нажмем кнопку OK, чтобы передать данные в SecondActivity и вернуться в нее (Рисунок 41).

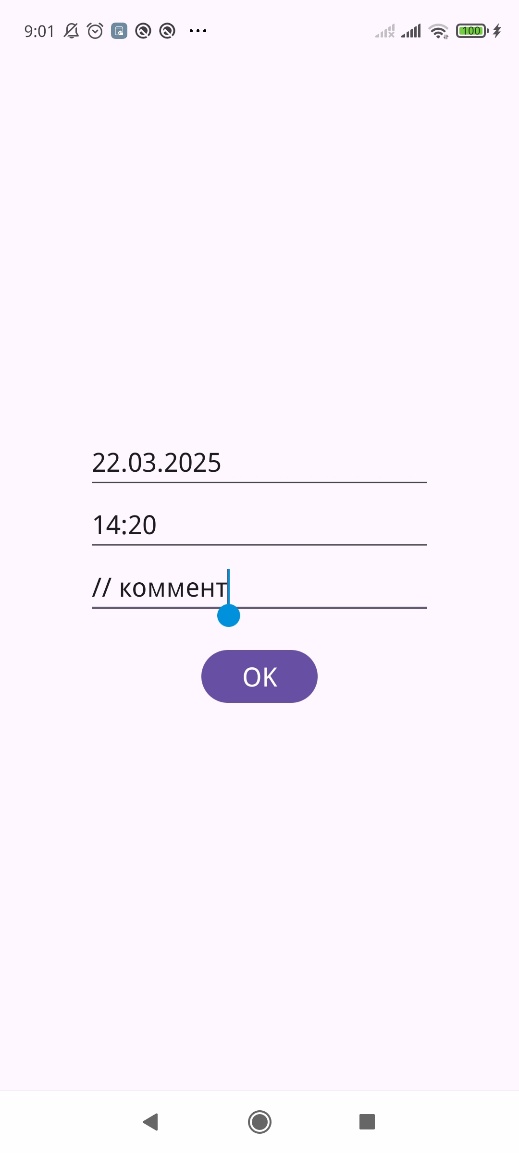


Рисунок 41 – Экран ThirdActivity

Вернувшись на экран SecondActivity, увидим, что данные о дате и времени были успешно переданы и отображены в соответствующих текстовых полях. Также заметим, что об успешной передаче данных нам сообщает всплывающее сообщение (Рисунок 42).

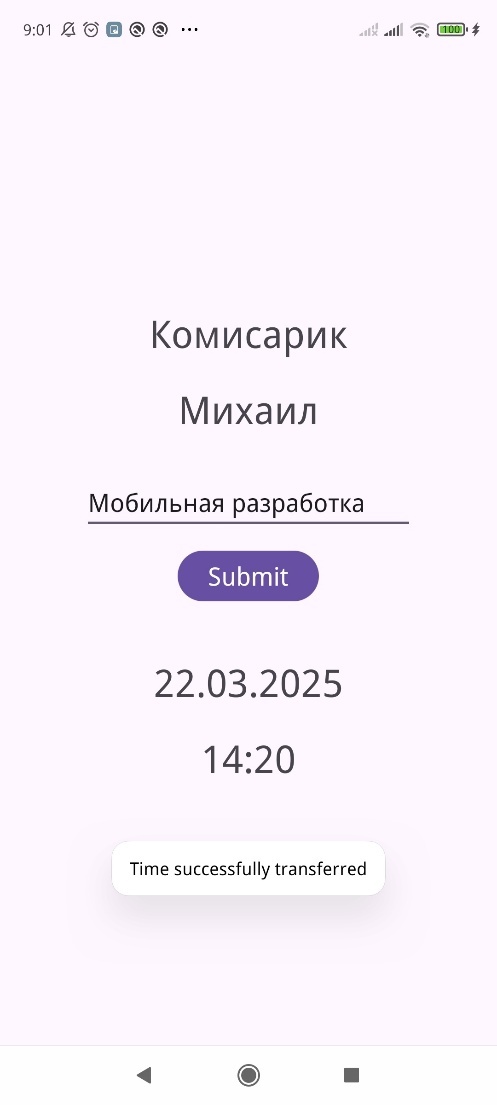


Рисунок 42 – Возвращение к экрану SecondActivity

## Работа с тремя Fragment

### Создание трех Fragment

Создаем новый проект в Android Studio по шаблону Empty Views Activity (Рисунок 43).

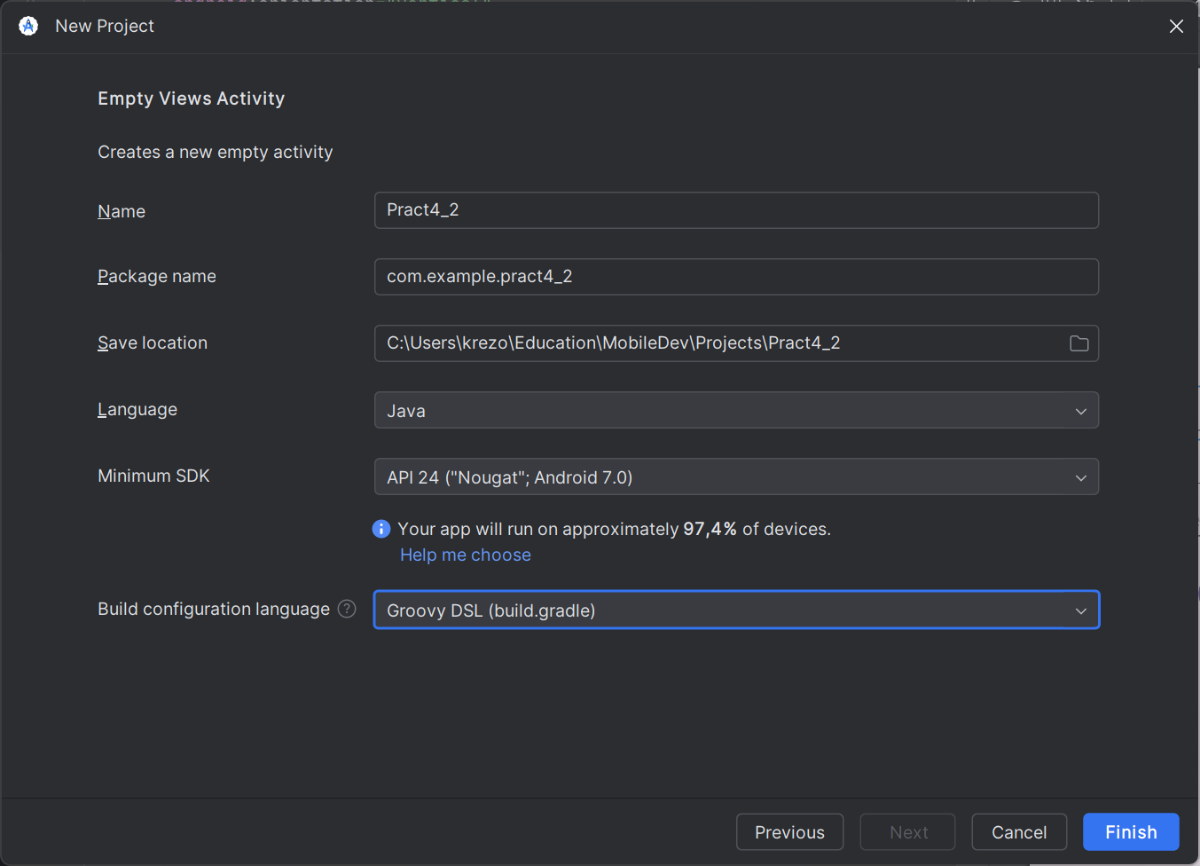


Рисунок 43 – Создание нового проекта

Android Studio позволяет удобным способом с помощью контекстных меню создать новый Fragment, для этого нажимаем правой кнопкой мыши на "java"→ "New" → "Fragment" → "Fragment (Blank)" (Рисунок 44). В диалоговом задаем его название (Рисунок 45).

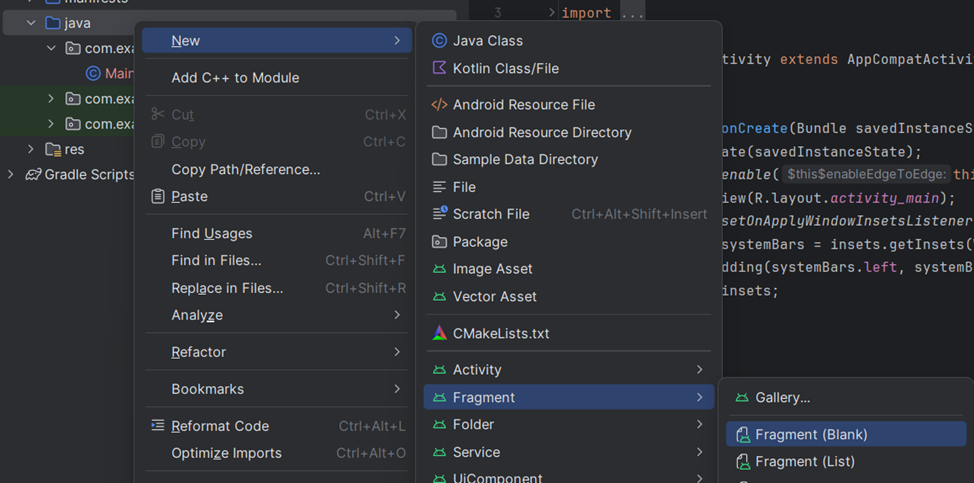


Рисунок 44 – Создание нового Fragment



Рисунок 45 – Настройка Fragment при создании

Добавим вышеописанным способом в наш проект три Fragment с названиями Fragment1, Fragment2 и Fragment3. Файлы их классов и разметок представлены на рисунке Рисунок 46.

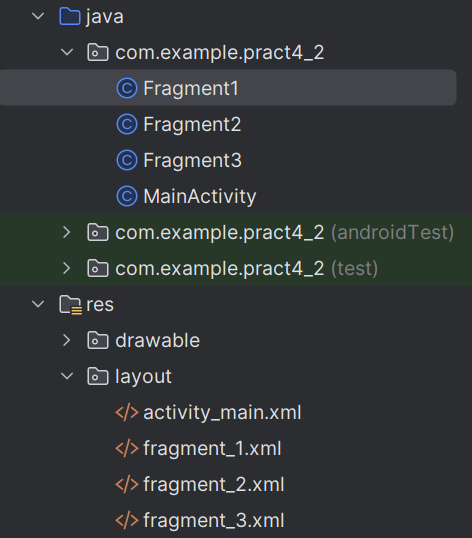


Рисунок 46 – Созданные файлы классов и разметок Fragment

Класс Fragment1 наследуется от Fragment, что позволяет ему представлять собой отдельный компонент пользовательского интерфейса внутри Activity. В этом классе переопределен метод onCreateView(), который отвечает за создание и возврат представления (View) для Fragment. Внутри метода используется LayoutInflater для подключения XML-файла разметки fragment\_1.xml, который передается в метод inflate(). Этот вызов обеспечивает раздувание макета и его привязку к переданному контейнеру container, при этом параметр attachToRoot установлен в false, что означает, что корневой элемент макета не будет автоматически добавляться в контейнер.

Дополнительные методы в данном классе не требуются, так как он не содержит специфической логики и использует стандартный жизненный цикл Fragment. Конструктор объявлять также не нужно, поскольку по умолчанию создается неявный публичный конструктор. Отсутствие обработки аргументов исключает необходимость работы с Bundle, а если нет необходимости реагировать на изменения в жизненном цикле, то переопределение таких методов, как onCreate() или onDestroyView(), также не требуется (Рисунок 47).

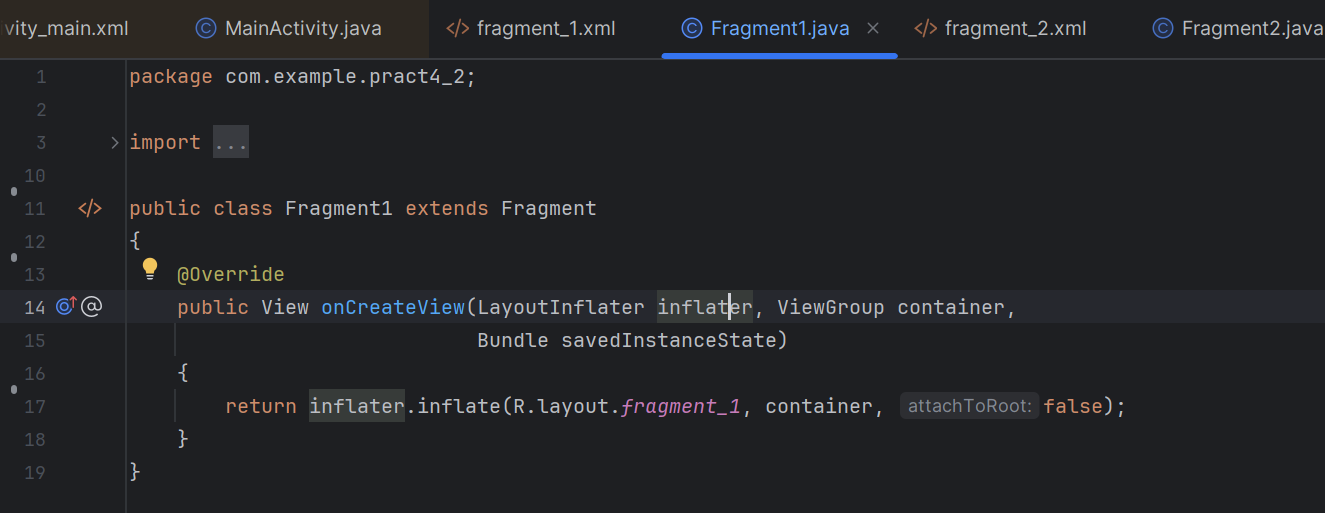


Рисунок 47 – Класс Fragment1

Классы Fragment2 и Fragment3 практически идентичны классу Fragment1, за исключением названия самих классов и подключаемых файлов разметки. В Fragment2 загружается fragment\_2.xml, а в Fragment3 — fragment\_3.xml. Остальная структура остается неизменной: оба класса наследуются от Fragment, переопределяют метод onCreateView(), используют LayoutInflater для создания представления и передают false в параметр attachToRoot, чтобы избежать автоматического присоединения к контейнеру, так как этим будет управлять FragmentManager (Рисунок 48).

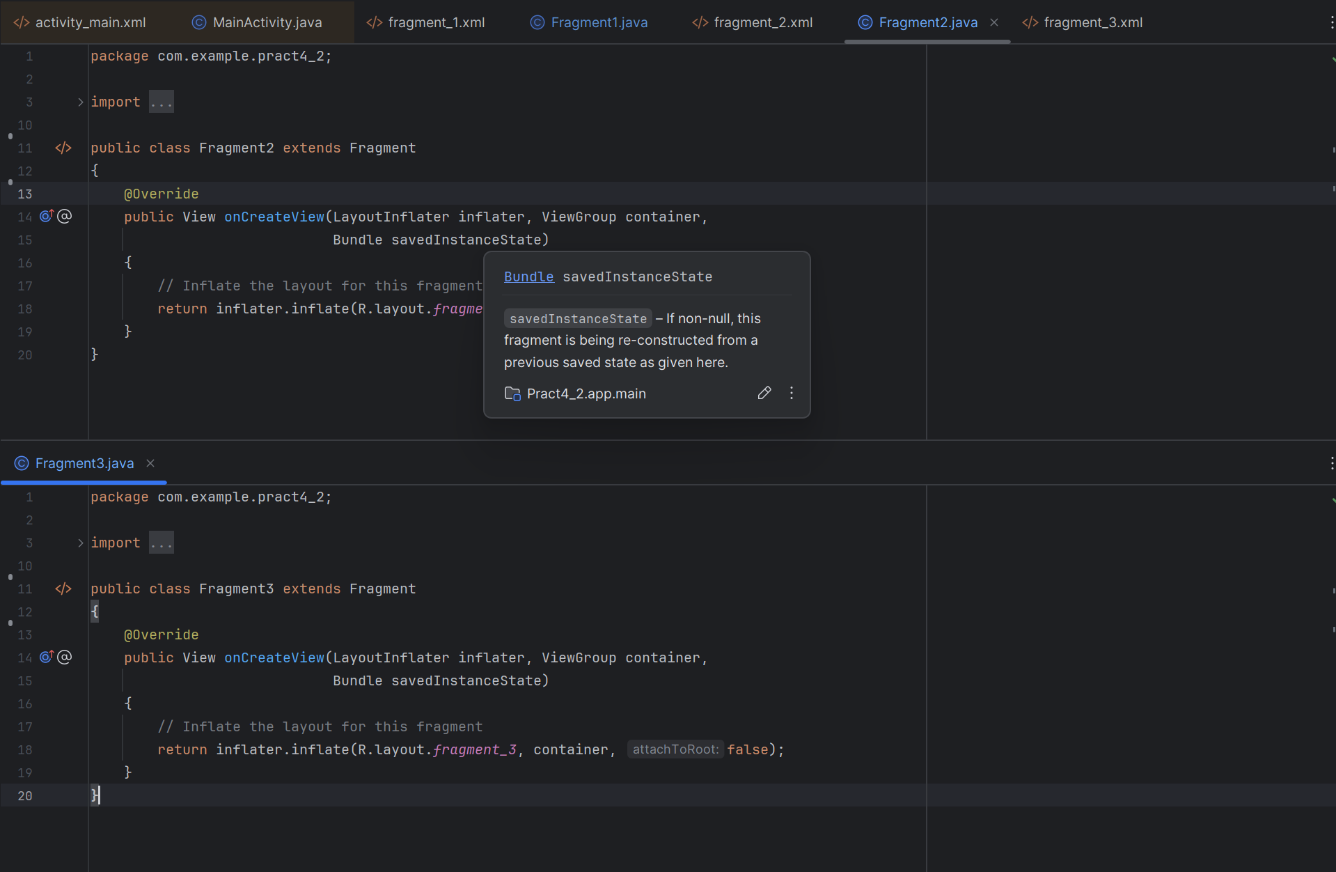


Рисунок 48 – Классы Fragment1 и Fragment3

Определим простую разметку для первого Fragment в файле fragment\_1.xml, используя LinearLayout в качестве корневого элемента. Зададим для него фоновый цвет, который будет отличаться от других Fragment, чтобы визуально их различать. Внутри разместим TextView с текстом "Fragment 1", где цифра в надписи будет изменяться в других Fragment, и элемент Button, расположенный ниже, который неактивен. Подобная структура будет использоваться и в других Fragment, но с изменением цвета фона, текста и элементов, что поможет явно их различать (Рисунок 49).

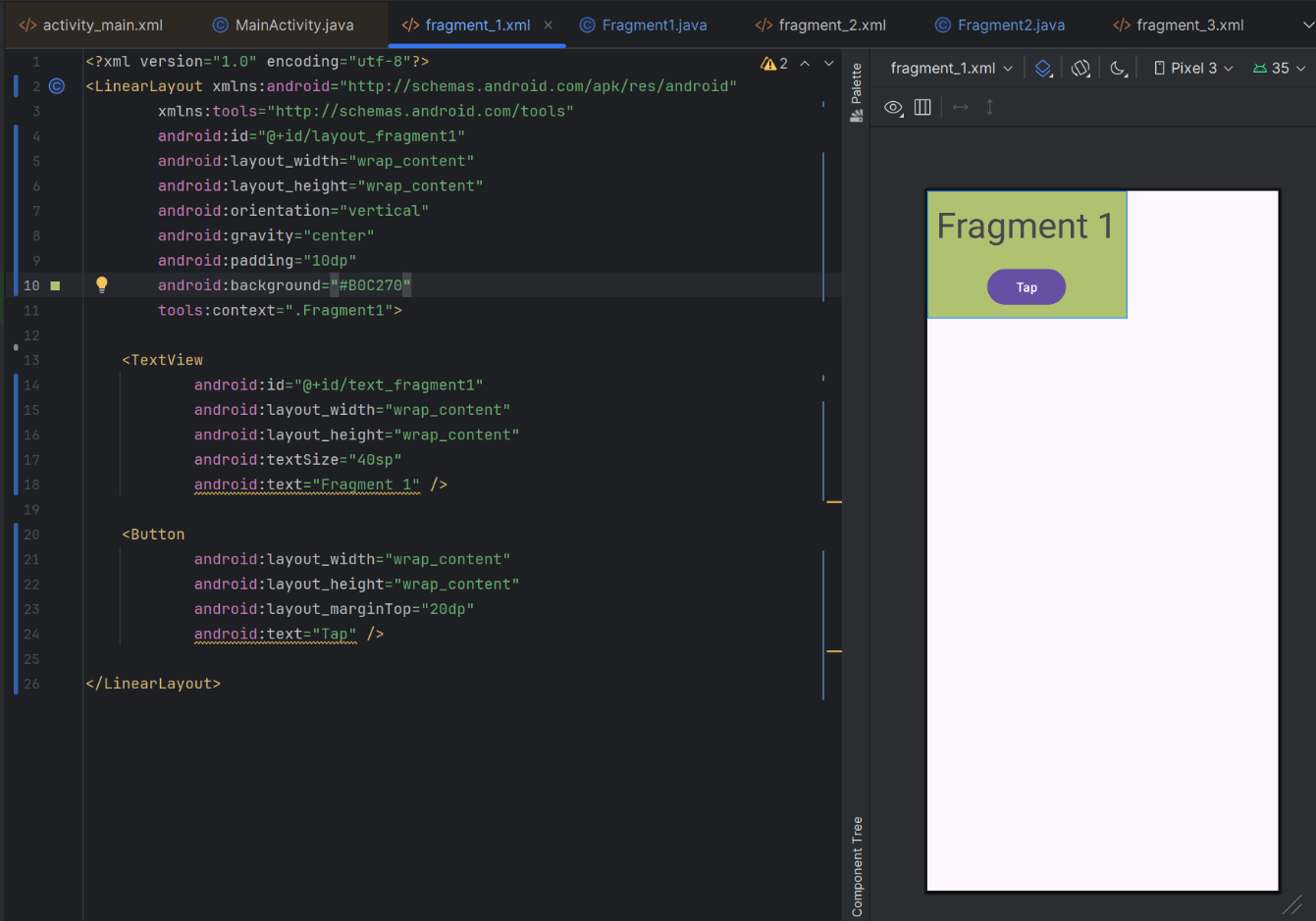


Рисунок 49 – Файл разметки fragment\_1.xml

По аналогии с разметкой первого Fragment, для второго и третьего Fragment также определяется простая разметка с корневым элементом LinearLayout, в которой изменяются цвет фона, номер Fragment в TextView и тип дополнительного элемента. Во втором Fragment используется ImageView с изображением звезды, а в третьем — SwitchCompat, представляющий собой переключатель. Эти различия позволяют явно отличать Fragment друг от друга, сохраняя при этом общий стиль их оформления (Рисунки Рисунок 50-Рисунок 51).

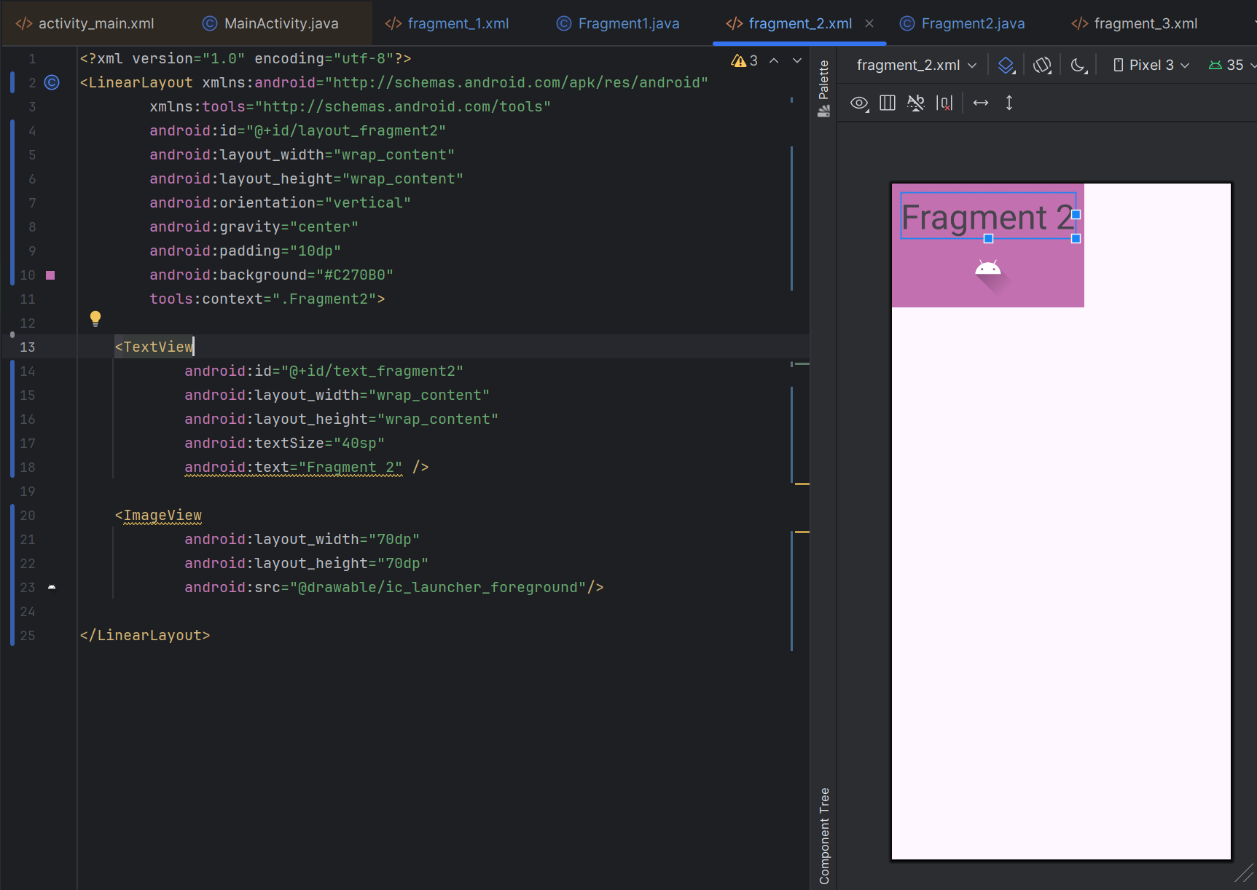


Рисунок 50 – Файл разметки fragment\_2.xml

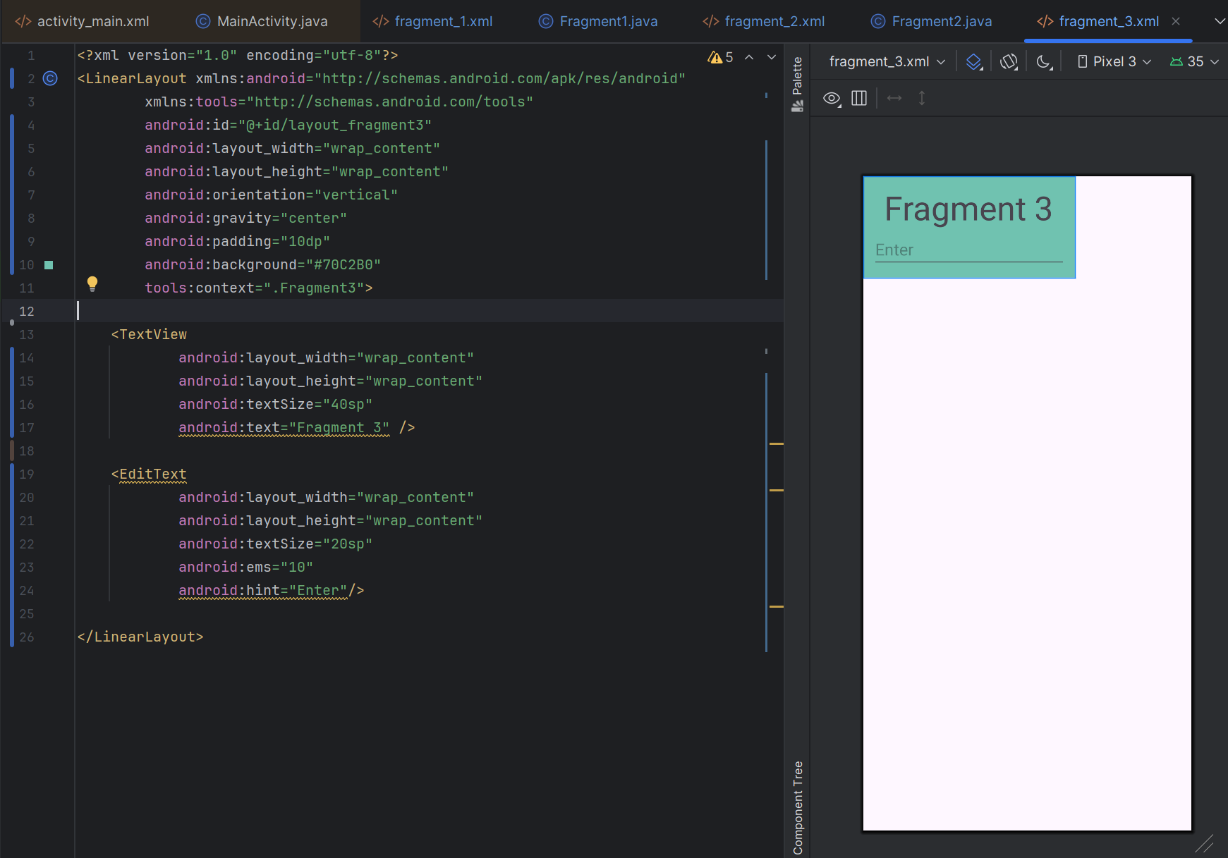


Рисунок 51 – Файл разметки fragment\_3.xml

### Размещение трех Fragment

Первый Fragment будет размещен статическим образом. В файле activity\_main.xml корневым элементом разметки выступает LinearLayout с вертикальной ориентацией. Для наглядности добавлен TextView, в котором явно указано, что Fragment добавляется статически. Само размещение первого Fragment осуществляется с помощью тега <fragment>, где в атрибуте android:name указан класс Fragment1 в соответствии с его пакетом. Такой способ позволяет включить Fragment в разметку на этапе компиляции, без необходимости управлять им программно (Рисунок 52).

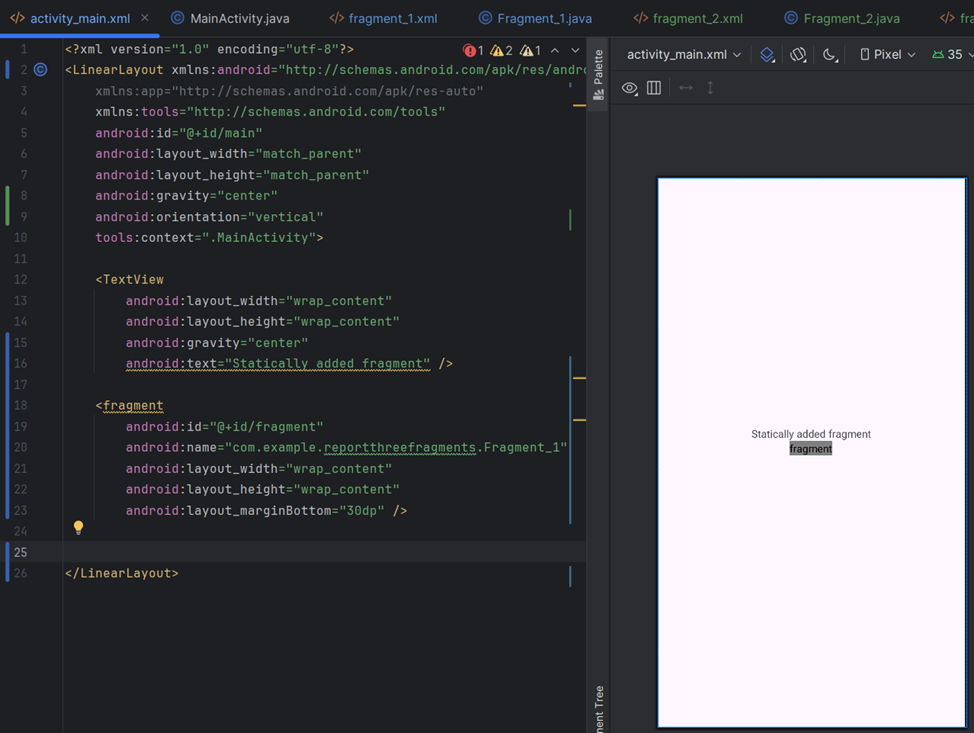


Рисунок 52 – Статическое добавление Fragment в activity\_main.xml

Второй Fragment будет добавляться динамически. В разметке, под первым Fragment, расположена поясняющая надпись в TextView, которая указывает на этот способ размещения. Для самого Fragment предусмотрен контейнер FrameLayout c id "frame\_main\_fragment2". Этот контейнер будет использоваться для программного добавления Fragment во время выполнения кода, что позволяет более гибко управлять его отображением (Рисунок 53).

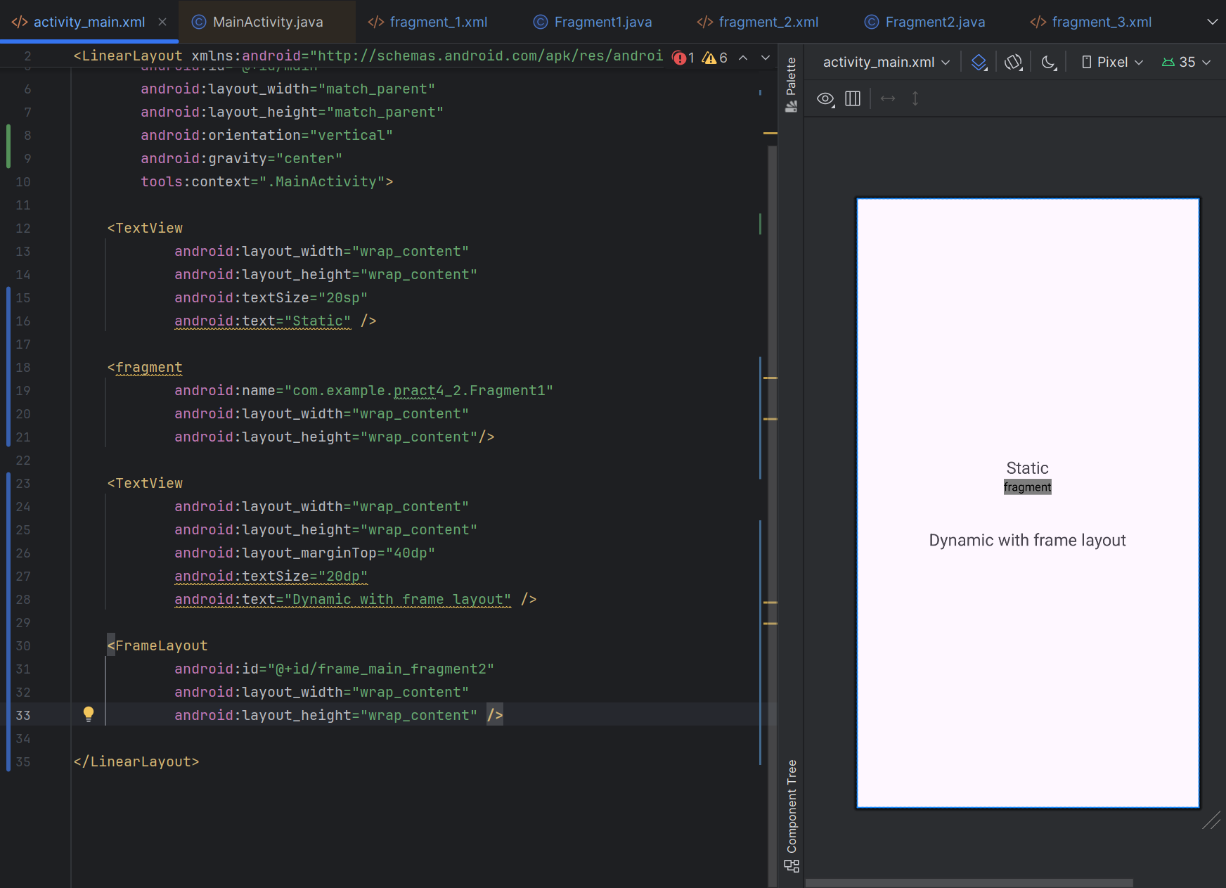


Рисунок 53 – Добавление контейнера FrameLayout, в который будет динамически добавлен Fragment2

В методе onCreate() MainActivity происходит динамическое добавление второго Fragment в контейнер FrameLayout, используя FragmentManager.

Сначала getSupportFragmentManager() возвращает экземпляр FragmentManager, который управляет Fragment. Затем метод beginTransaction() создает новую транзакцию, в рамках которой будут выполняться действия с Fragment. Вызов add(R.id.frame\_main\_fragment2, Fragment2.class, null) добавляет экземпляр Fragment2 в контейнер с идентификатором "frame\_main\_fragment2". Здесь "frame\_main\_fragment2" указывает на FrameLayout, определенный в разметке и предназначенный для динамического размещения Fragment, Fragment2.class определяет, какой именно Fragment будет добавлен, а аргумент null означает, что Fragment создается без передачи дополнительных данных. Наконец, метод commit() фиксирует изменения и применяет транзакцию, что приводит к отображению Fragment в контейнере (Рисунок 54).

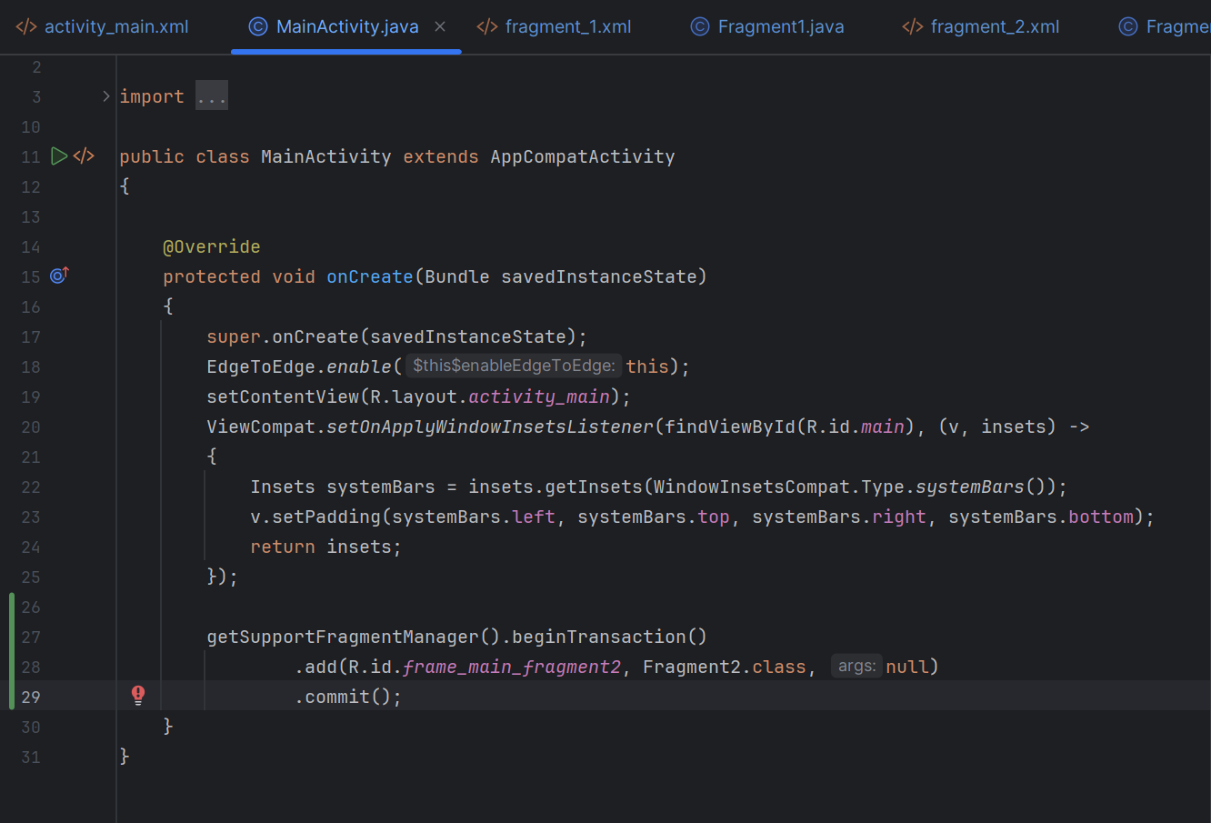


Рисунок 54 – Динамическое добавление второго Fragment в контейнер FrameLayout в методе onCreate() класса MainActivity

Третий Fragment добавляется в разметку с использованием элемента FragmentContainerView. Этот контейнер является усовершенствованной версией <fragment> и предоставляет поддержку как для статического размещения Fragment с помощью атрибута android:name, так и для динамического добавления с использованием FragmentManager. Однако в данном случае Fragment Fragment3 устанавливается первым способом, указывая его в android:name. В результате Fragment автоматически создается и встраивается в FragmentContainerView, который определен в разметке с id "fragmentContainer\_main\_fragment3". Это позволяет использовать преимущества FragmentContainerView, такие как корректное управление состоянием Fragment при изменении конфигурации экрана, сохраняя при этом простоту статического размещения (Рисунок 55).

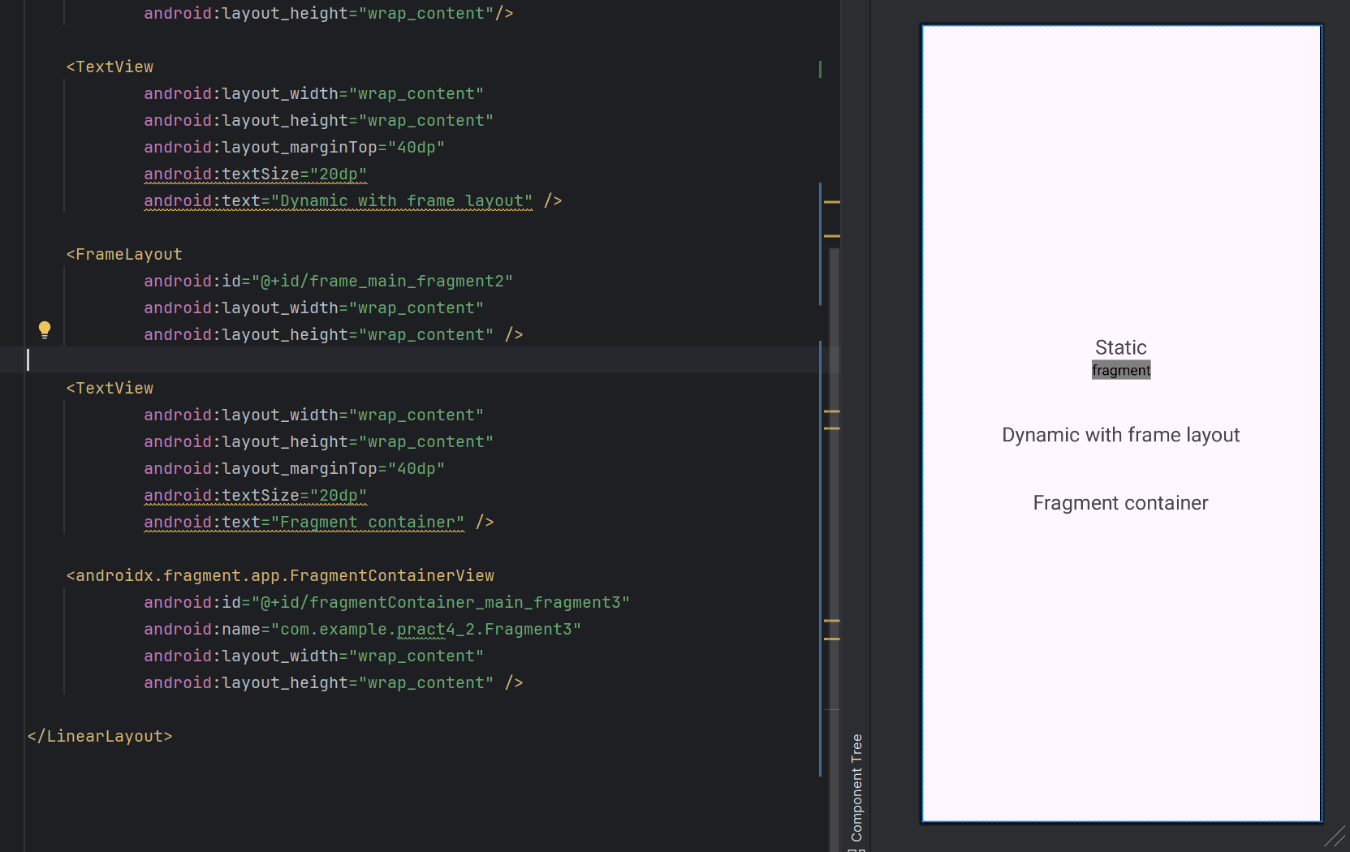


Рисунок 55 – Добавление третьего Fragment с помощью FragmentContainerView

Запустим наше приложение и убедимся, что все три Fragment корректно загружаются на экране. Первый, добавленный статически через XML-разметку с использованием <fragment>, располагается в верхней части. Второй, который был динамически вставлен в FrameLayout через FragmentManager, появляется ниже. Третий, размещенный с помощью FragmentContainerView и заданный через android:name, также успешно выводится в интерфейсе (Рисунок 56).

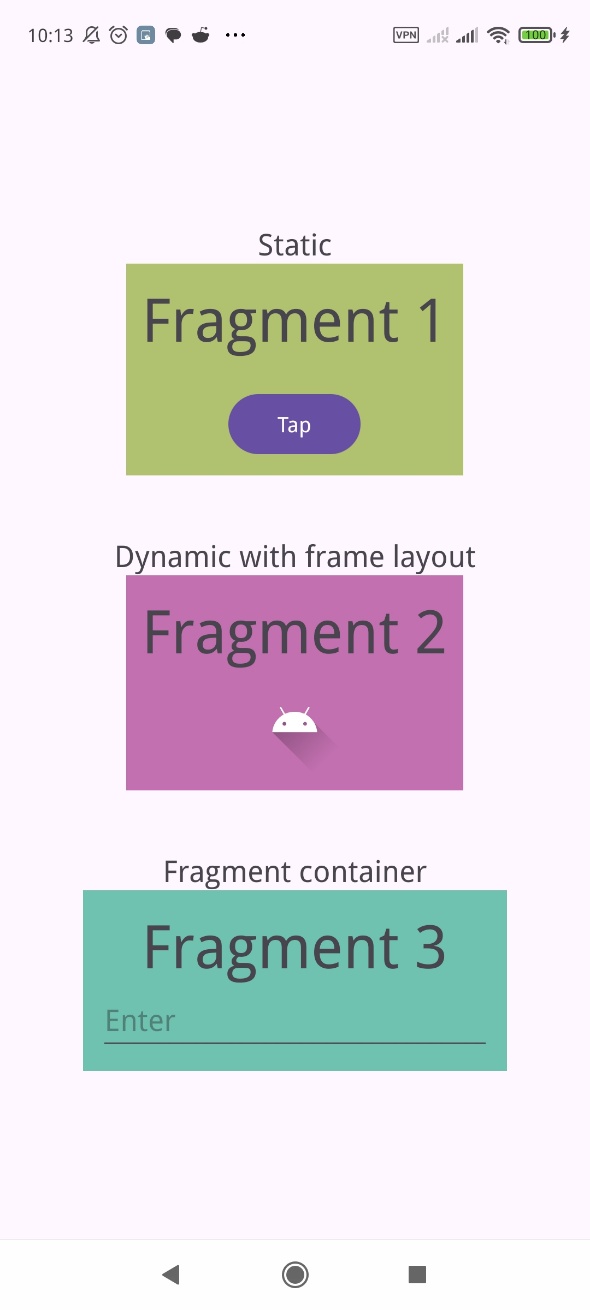


Рисунок 56 – Тестирование отображения всех Fragment

### Реализация перемещения между Fragment

Реализуем перемещение между созданными ранее Fragment. Изменения будут касаться только MainActivity. В файле разметки activity\_main.xml добавим LinearLayout с вертикальной ориентацией и layout\_weight=1, в котором расположим три кнопки, которые будут использоваться для отображения соответствующих Fragment. Каждая кнопка имеет уникальный идентификатор и текст, указывающий на Fragment, который она будет вызывать (Рисунок 57).

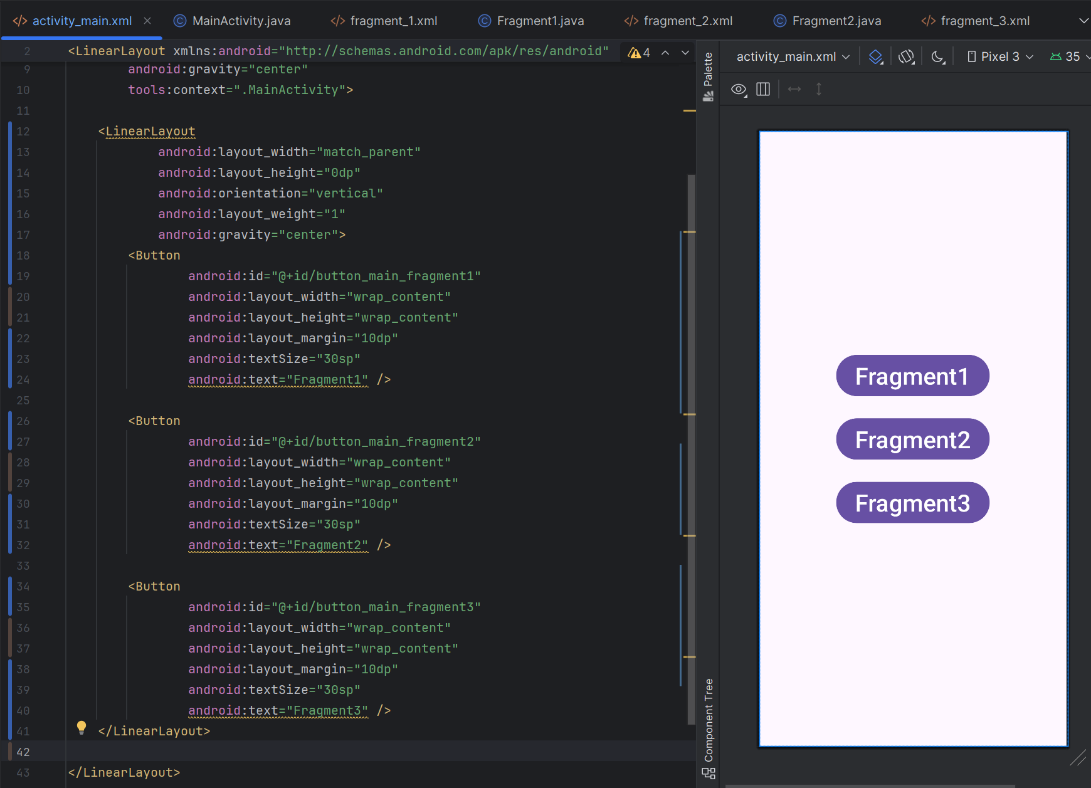


Рисунок 57 – Добавление кнопок в файл разметки activity\_main.xml

Добавим FragmentContainerView, который будет использоваться для отображения Fragment. Добавим атрибут layout\_weight=1, чтобы он занимал половину экрана. Какой именно Fragment в нем будет размещаться, полностью определяется программно в классе MainActivity, поэтому атрибут android:name не указывается (Рисунок 58).

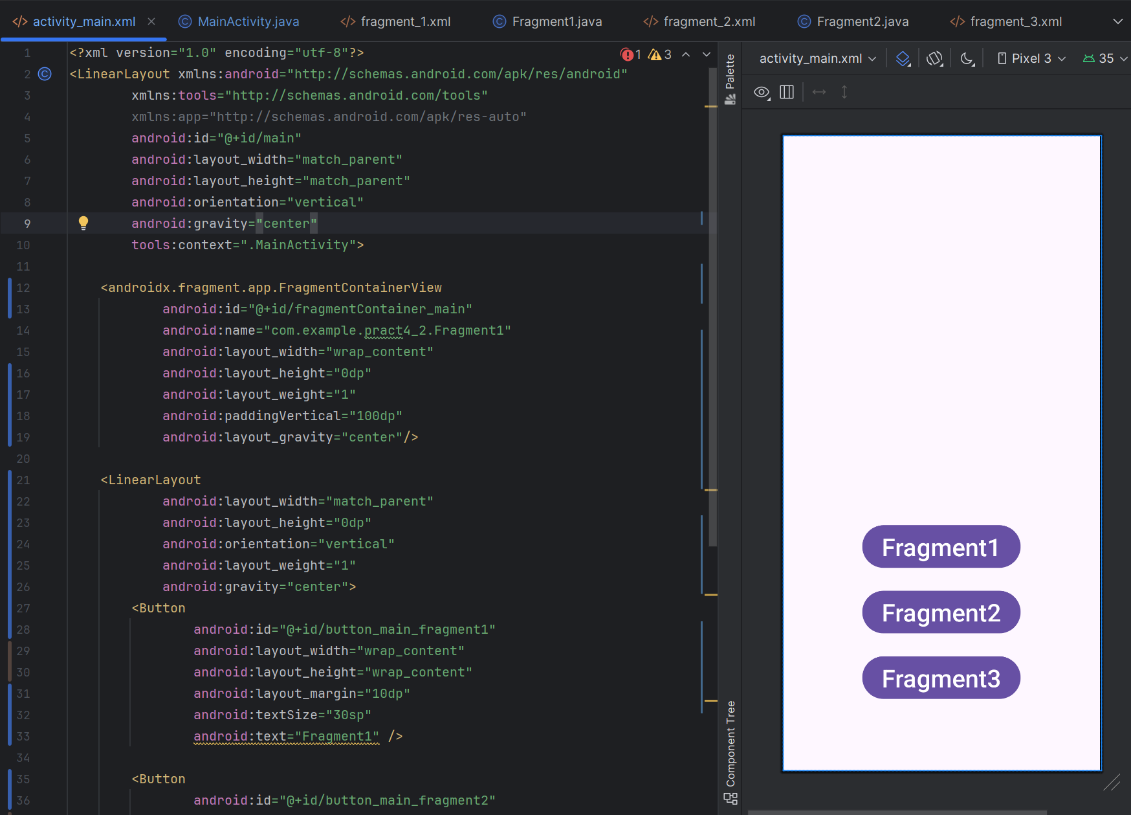


Рисунок 58 – Добавление элемента FragmentContainerView

В классе MainActivity мы создадим метод replaceFragment, который позволит заменять Fragment внутри контейнера fragmentContainerView. Этот метод принимает объект типа Fragment в качестве параметра и выполняет его установку в контейнер. Вначале мы получаем экземпляр FragmentManager с помощью getSupportFragmentManager(), который отвечает за управление Fragment в активности. Затем последовательно вызовем методы beginTransaction(), для создания транзакции, replace(), для замены текущего Fragment и commit(), чтобы зафиксировать изменения и применить их. Такой подход позволяет динамически изменять Fragment в приложении в зависимости от действий пользователя (Рисунок 59).

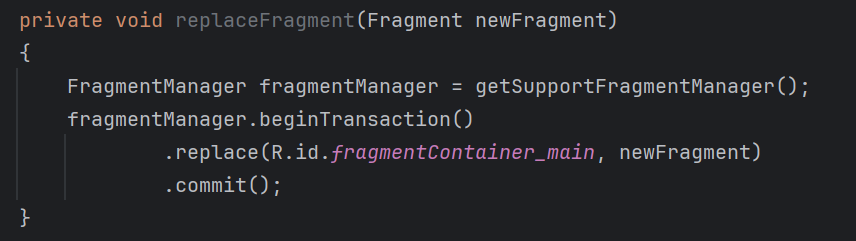


Рисунок 59 – Описание метода replaceFragment()

В методе onCreate класса MainActivity создаются ссылки на три кнопки, каждая из которых предназначена для переключения Fragment. Используя метод findViewById(), мы находим кнопки buttonFragment1, buttonFragment2 и buttonFragment3 по их идентификаторам, после чего устанавливаем для них обработчики нажатия с помощью setOnClickListener(). При нажатии на кнопку вызывается метод replaceFragment(), которому передается новый экземпляр соответствующего Fragment, что приводит к его отображению в контейнере fragmentContainerView. Таким образом, нажатие на кнопки загружает соответствующий фрагмент. В конце кода выполняется проверка, был ли сохранен экземпляр активности, и если savedInstanceState равен null, то есть активность создается впервые, вызывается replaceFragment() с Fragment1, чтобы при запуске приложения в контейнере сразу отображался первый Fragment (Рисунок 60).

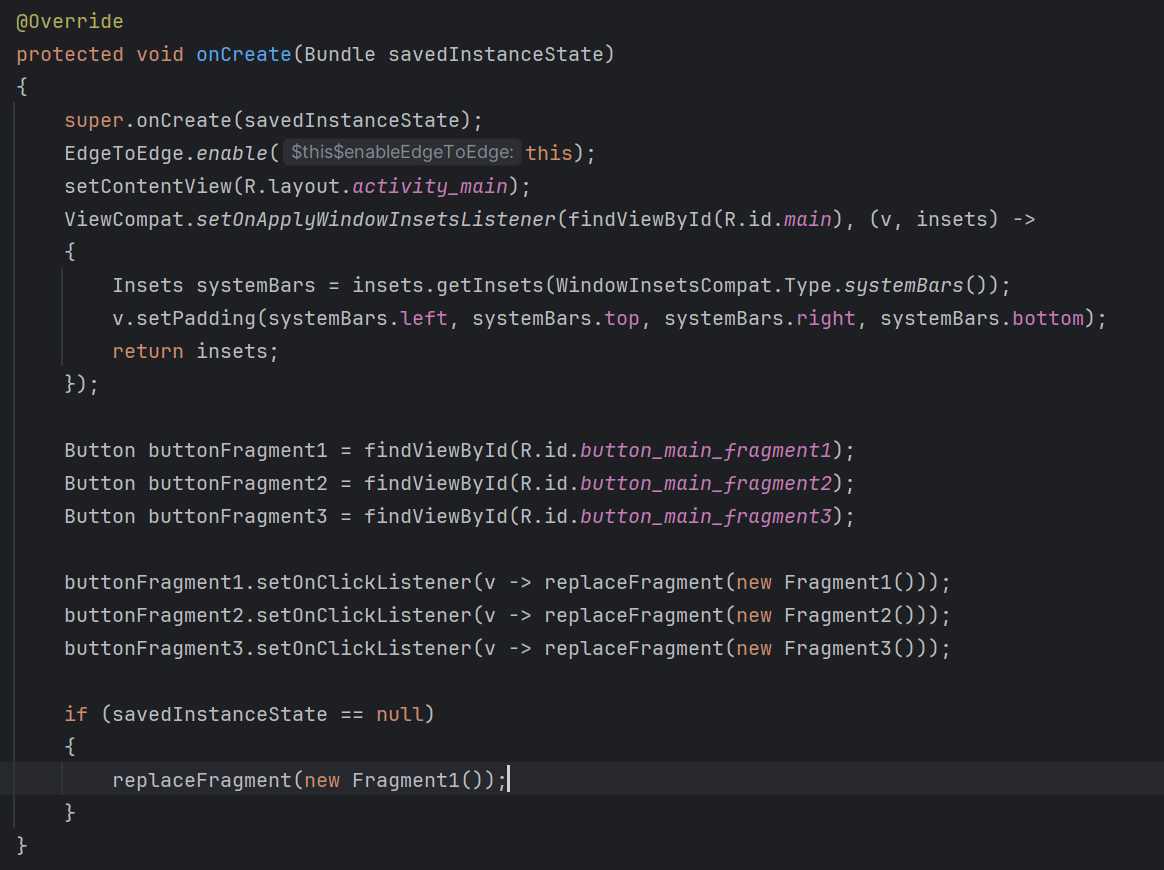


Рисунок 60 – Реализация переключение Fragment в методе onCreate() класса MainActivity с использованием кнопок

### Тестирование перемещения между Fragment

Проведем тестирование перемещения между Fragment. Запустим приложение и убедимся, что при запуске на экране отображается первый Fragment (Рисунок 61).



Рисунок 61 – Отображение Fragment1 при запуске приложения

Нажатие кнопки Fragment2 приводит к корректной смене отображаемого Fragment на второй Fragment (Рисунок 62).



Рисунок 62 – Отображение Fragment2 при нажатии соответствующей кнопки

Нажатие кнопки Fragment 3 приводит к корректной смене отображаемого Fragment на третий Fragment (Рисунок 63).



Рисунок 63 – Отображение Fragment3 при нажатии соответствующей кнопки

Нажатие кнопки Fragment 1 приводит к корректной смене отображаемого Fragment на первый Fragment, таким образом, можно заключить, что приложение работает корректно (Рисунок 64).



Рисунок 64 – Отображение Fragment1 при нажатии соответствующей кнопки

# ЗАКЛЮЧЕНИЕ

В ходе выполнения работы было изучено использование Activity Result API для передачи данных между активностями. Также было рассмотрено управление фрагментами, включая их создание, размещение и навигацию между ними с помощью FragmentManager.