|  |  |  |
| --- | --- | --- |
|  |  |  |
| 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 4](#_Toc192942250)

[2 ПРАКТИЧЕСКОЕ ЗАДАНИЕ 26](#_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 1.1.1).

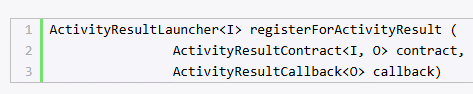


Рисунок 1 – Метод registerForActivityResult()

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 1.1.2.

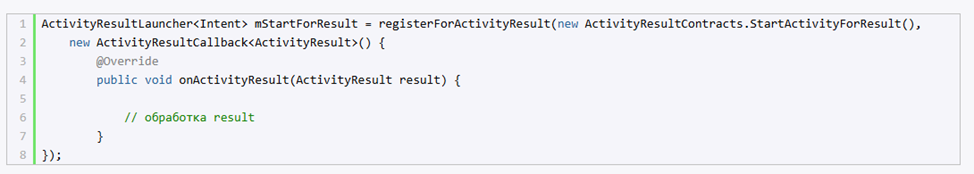


Рисунок 2 – Пример метода registerForActivityResult()

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-колбек и возвращает объект ActivityResultLauncher. С помощью этого мы можем запустить activity. Для этого у объекта ActivityResultLauncher вызывается метод launch() (рисунок 1.1.3).



Рисунок – 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 1.1.4).

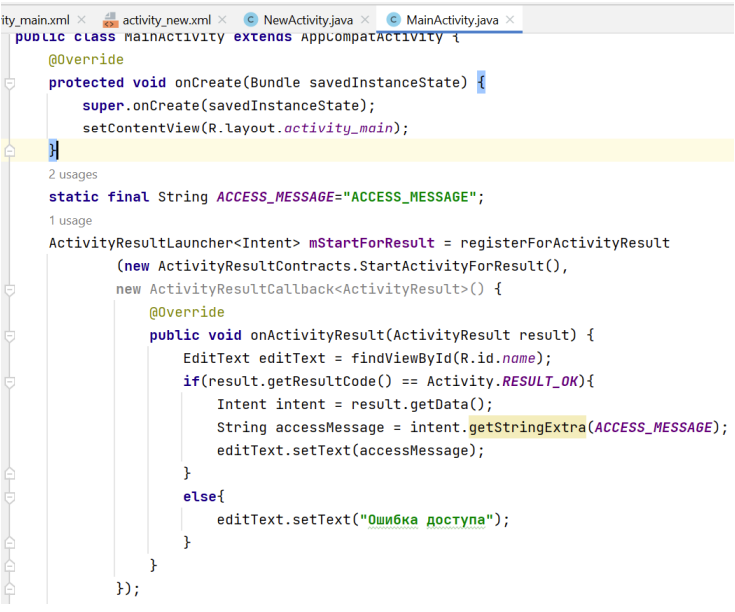


Рисунок 1.1.4 — Код класса MainActivity

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-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 мы проверяем полученный код результата (рисунок 1.1.5).



Рисунок 1.1.5 — 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 1.1.6).



Рисунок 1.1.6 — Получение объекта intent

Таким образом, мы определили объект ActivityResultLauncher.

Далее в обработчике нажатия onClick с помощью этого объекта запускаем вторую activity (рисунок 1.1.7).



Рисунок 1.1.7 — 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 1.1.8–1.1.9).

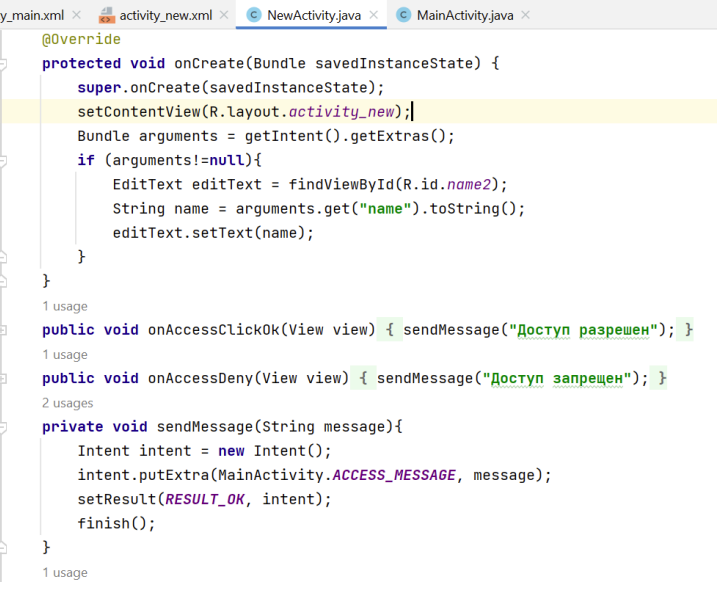


Рисунок 1.1.8 — код класса NewActivity

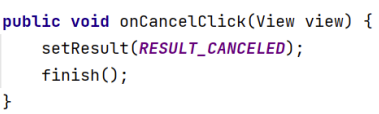


Рисунок 1.1.9 — Метод onCancelClick()

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 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 1.1.10).

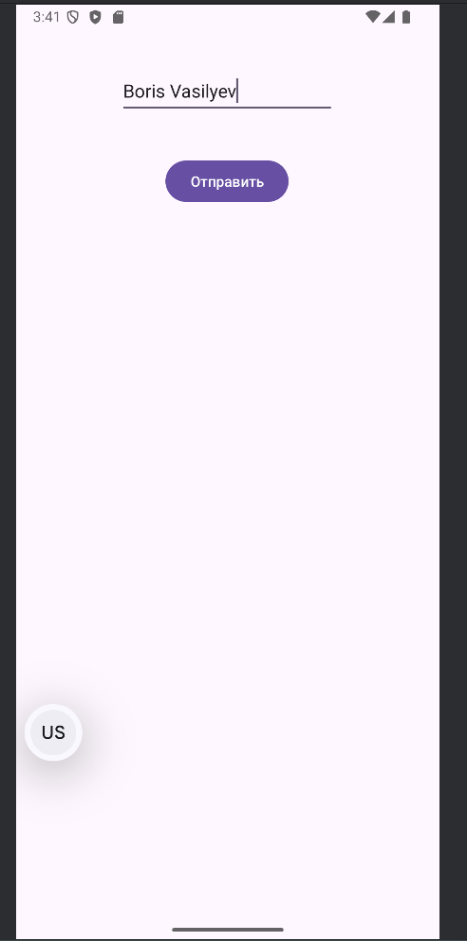


Рисунок 1.1.10 — Экран MainActivity

На экране SecondActivity отображается имя переданное из MainActivity, а также представлены три кнопки (рисунок 1.1.11).

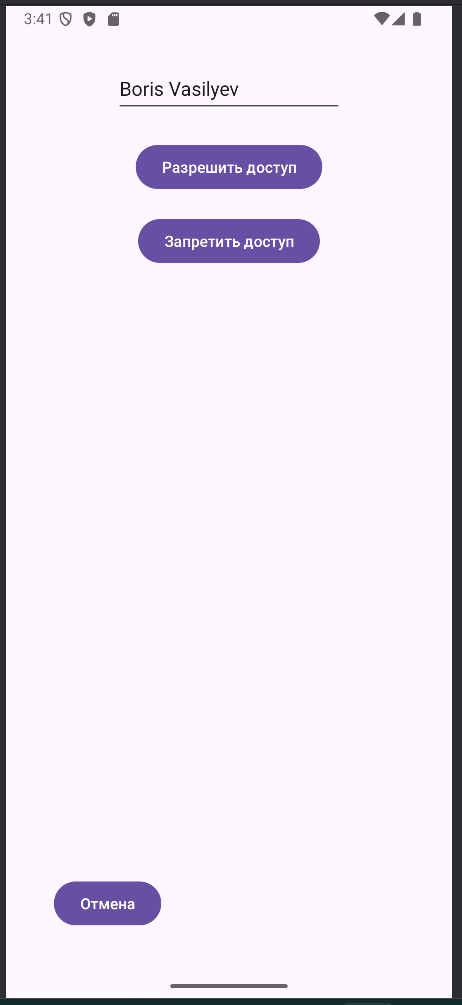


Рисунок 1.1.11 — Экран NextActivity

Если пользователь нажимает на кнопку «Разрешить доступ» или на кнопку «Запретить доступ», то в NewActivity вызывается метод onAccessClickOk(View view), который создаёт Intent, добавляет в него строку "Доступ разрешён" или "Доступ запрещён", в зависимости от нажатия соответствующей кнопки, с ключом ACCESS\_MESSAGE, устанавливает результат RESULT\_OK и передаёт 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 1.1.12–1.1.13).

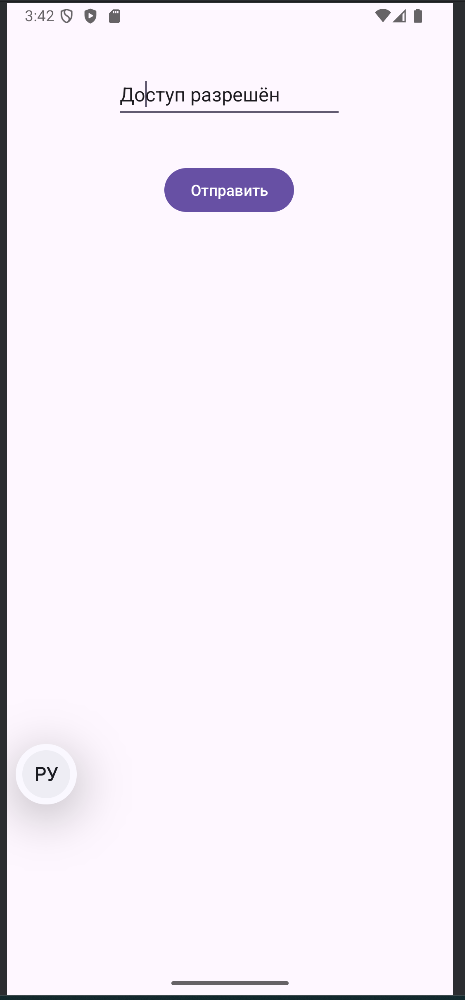


Рисунок 1.1.12 — Результат нажатия на кнопку «Разрешить доступ»

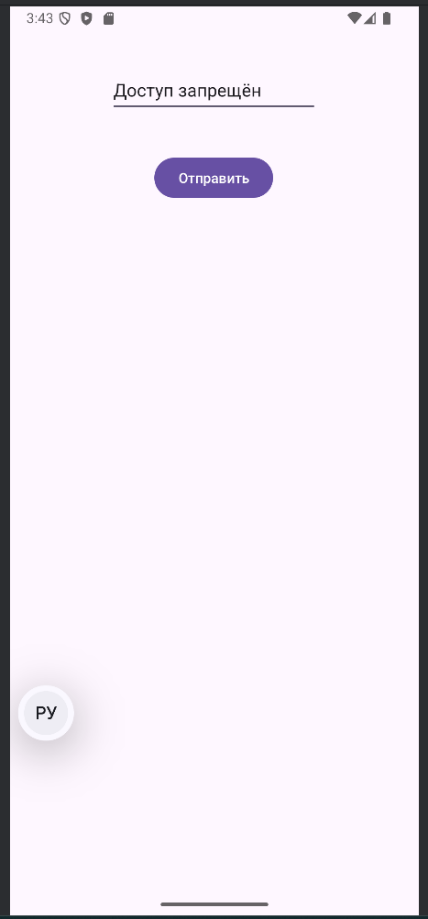


Рисунок 1.1.13 — Результат нажатия на кнопку «Запретить доступ»

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 1.1.14).

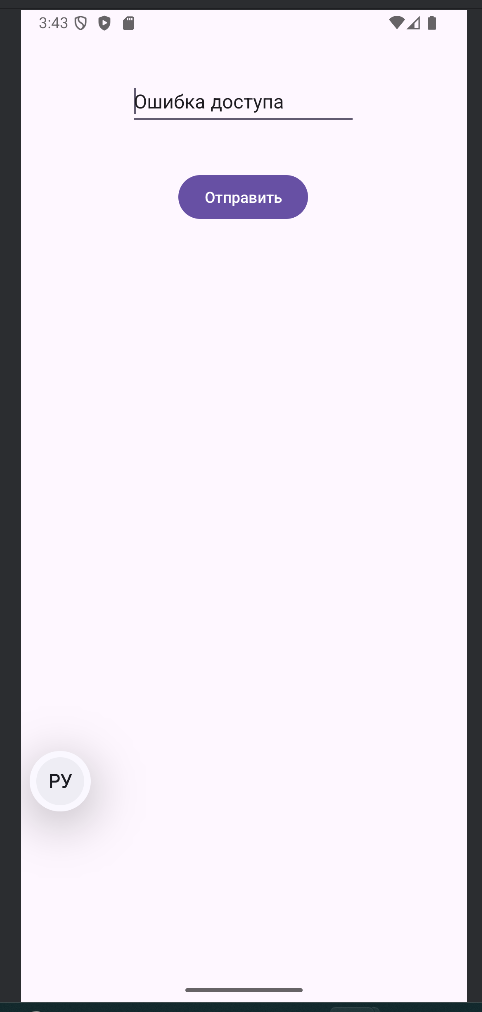


Рисунок 1.1.14 — Результат нажатия на кнопку «Отмена»

## 1.2 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.

Фрагменты в Android представляют собой модульный сегмент пользовательского интерфейса в активности. Они используются для создания более гибких и адаптивных интерфейсов. Фрагменты 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 фрагментов можно выделить следующие:

* модульность: Фрагменты 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: 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-активности. Это позволяет фрагментам управлять своим состоянием и поведением в зависимости от состояния активности.

## 1.3 Жизненный цикл фрагмента

Жизненный цикл фрагмента в Android представляет собой последовательность состояний, через которые проходит фрагмент во время своего существования. Связанные с этими состояниями callback-методы позволяют управлять поведением фрагмента на различных этапах его взаимодействия с пользователем и системой (рисунок 1.3.1).

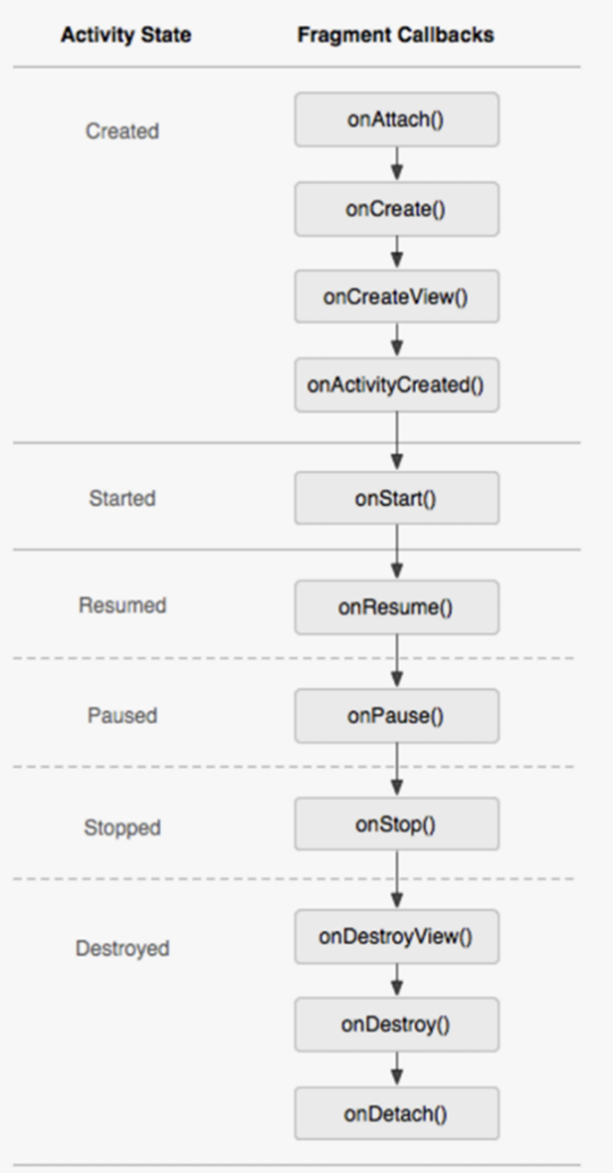


Рисунок 1.3.1 — Жизненный цикл фрагмента

Привязка к активности: Когда фрагмент привязывается к активности, вызывается метод onAttach(). Это означает, что фрагмент теперь ассоциирован с активностью, 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.

Создание представления фрагмента: Метод onCreateView() вызывается для создания представления фрагмента (рисунок 1.3.2). Здесь загружается макет, определяет пользовательский интерфейс фрагмента. После создания представления оно возвращается системе для отображения.

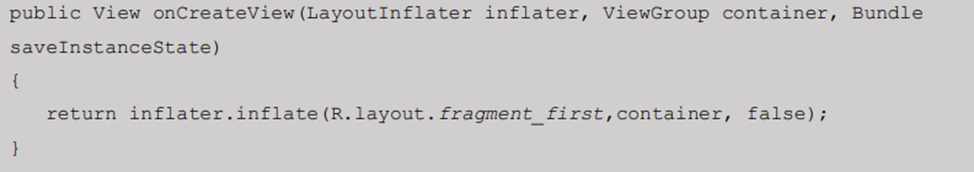


Рисунок 1.3.2 — метод onCreateView()

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.

Активность фрагмента: После создания представления, метод onActivityCreated() вызывается, когда активность, к которой привязан фрагмент, полностью создана. Это хорошее место для выполнения финальных инициализаций, которые зависят от активности, 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.

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 активность.

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

## 1.4 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. Этим способ и будем пользоваться.

Принцип создания новых фрагментов схож с созданием активностей. 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 фрагмента и как в случае с активностью будут созданы файлы класса и разметки (рисунки 1.4.1–1.4.2).

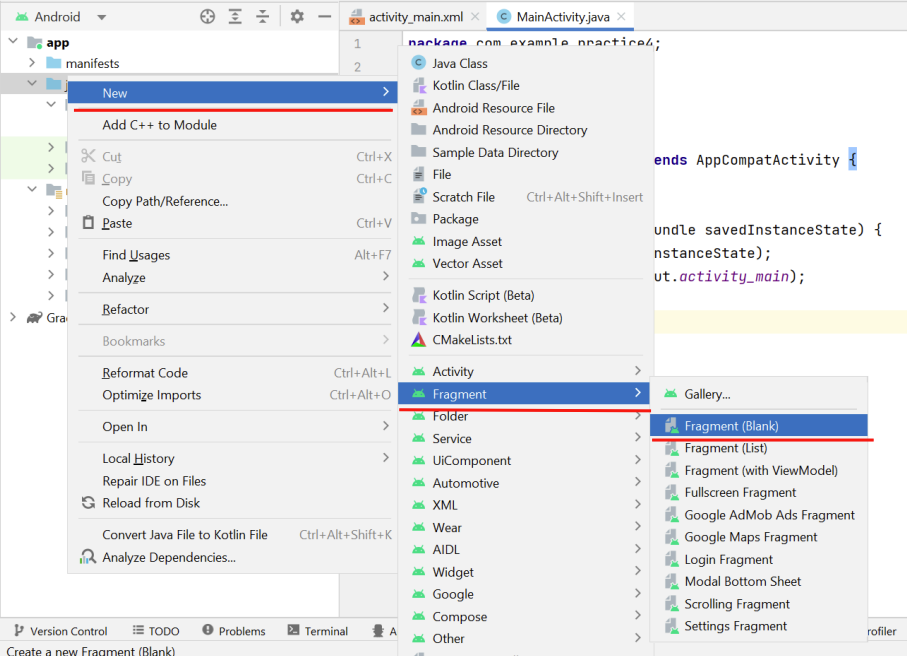


Рисунок 1.4.1 — Создание нового фрагмента с помощью Android Studio

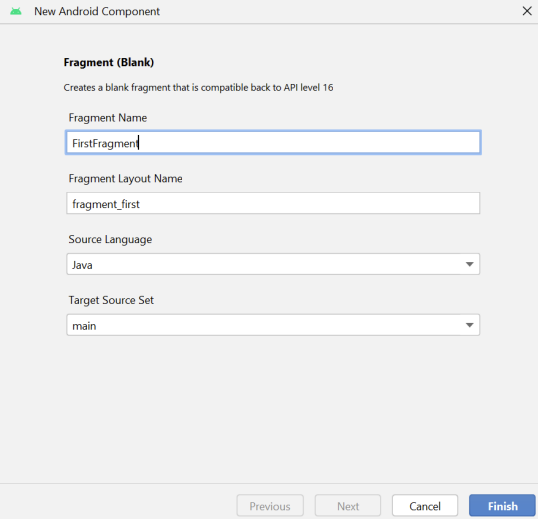


Рисунок 1.4.2 — Настройки при создании нового фрагмента

После создания фрагмента, класс для реализации фрагмента и xml-файл для разметки будут созданы автоматически (рисунок 1.4.3).

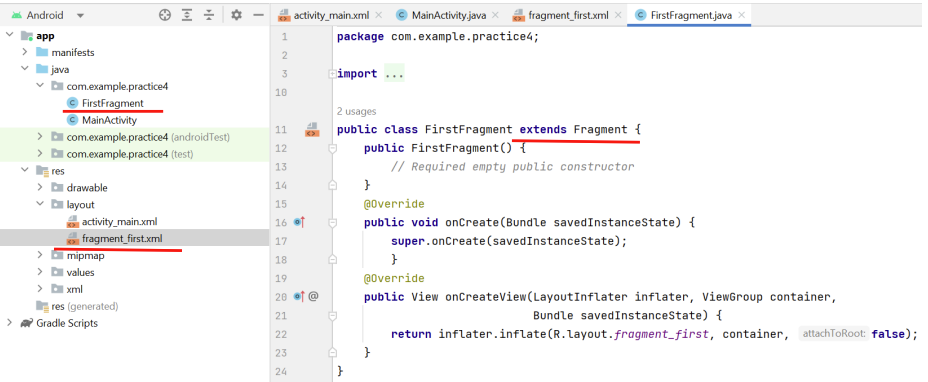


Рисунок 1.4.3 — Класс созданного фрагмента

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).

Метод onCreateView предназначен для создания и возврата разметки фрагмента, 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 фрагмента.

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 фрагмента его необходимо разместить на активности. Сделать это можно тремя способами: статически, динамически и через элемент fragmentContainerView. Для размещения фрагмента статическим способом необходимо добавить в XML файл активности элемент fragment (рисунок 1.4.4).

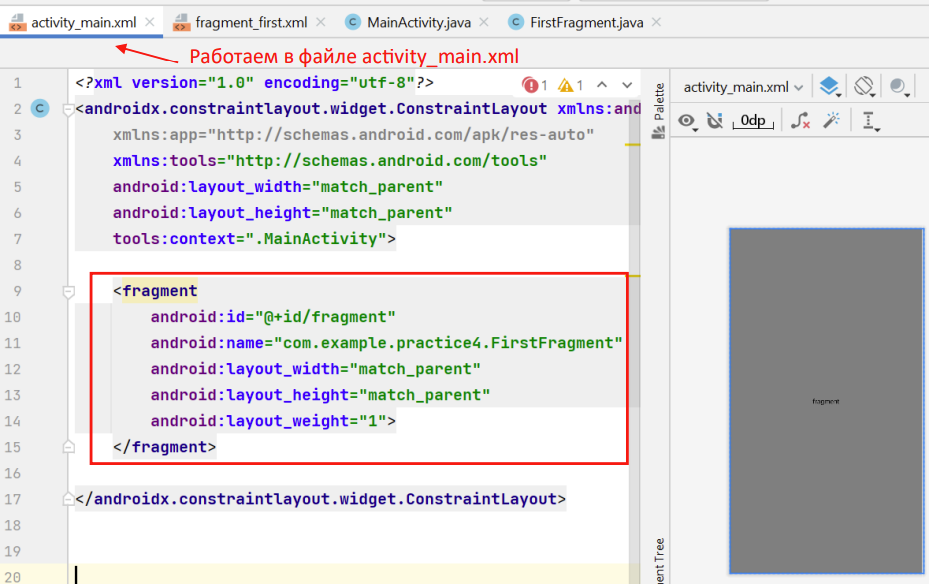


Рисунок 1.4.4 — Размещение фрагмента статическим способом

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 (рисунок 1.4.5).

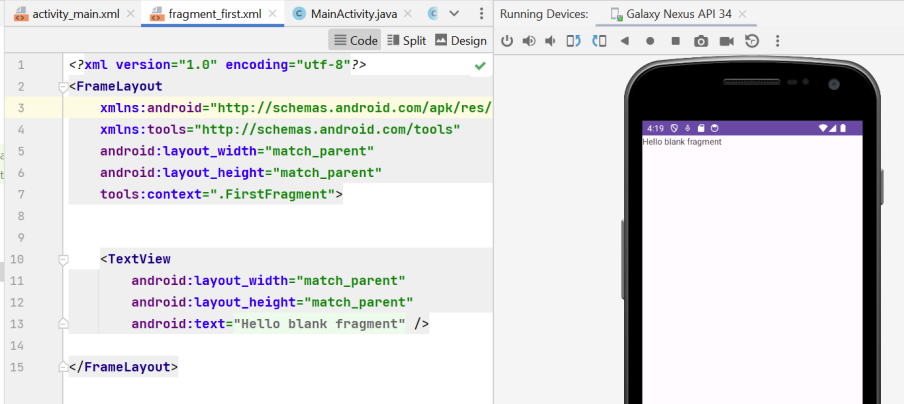


Рисунок 1.4.5 — Разметка фрагмента

Теперь данный фрагмент будет показываться при запуске приложения. Главным недостатком такого размещения фрагмента является то, что статические фрагменты тесно связаны с жизненным циклом активности, что может усложнить управление их состоянием. Например, сохранение и восстановление состояния фрагмента при пересоздании активности 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 (рисунок 1.4.6).



Рисунок 1.4.6 — Контейнер FrameLayout для размещения фрагмента

Далее в классе активности необходимо «разместить» данный фрагмент, используя метод класса "FragmentManager" — "getSupportFragmentManager" (рисунок 1.4.7).

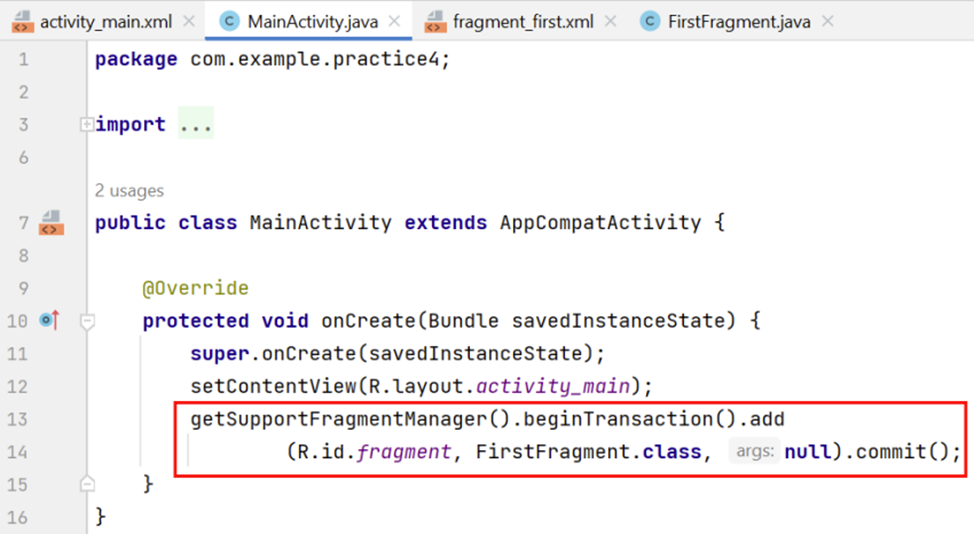


Рисунок 1.4.7 — Динамическое добавление фрагмента в MainActivity с использованием FragmentManager

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 указанный фрагмент в контейнер макета, переданный в качестве первого аргумента. Этот метод не сразу отображает фрагмент, 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 фрагмент активной частью пользовательского интерфейса.

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

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

Для его использования нужно разместить данный элемент в XML файле активности (рисунок 1.4.8).



Рисунок 1.4.8 — Добавление фрагмента в MainActivity с использованием FragmentContainerView

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

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

## 1.5 Навигация между фрагментами

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.

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

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

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

На рисунке 1.5.1 представлена демонстрация использования методов getSupportFragmentManager(), beginTransaction(), replace() и commit(), которые позволяют заменить один фрагмент другим внутри контейнера FragmentContainerView. Метод replaceFragment создает новую транзакцию, выполняет замену фрагмента и применяет изменения.

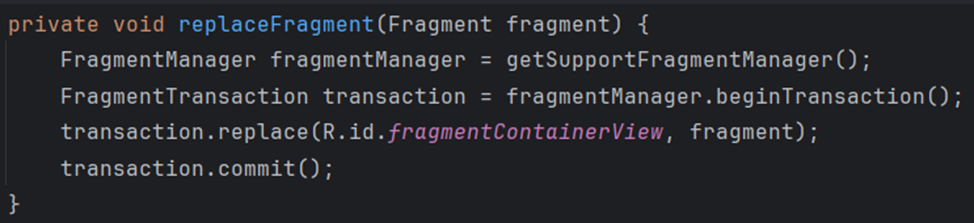


Рисунок 1.5.1 — Добавление фрагмента в MainActivity с использованием FragmentContainerView

На рисунке 1.5.2 представлен фрагмент кода, находящийся в методе onCreate() активности. В нем устанавливаются обработчики нажатий на кнопки, при срабатывании которых вызывается метод replaceFragment(), заменяющий текущий фрагмент 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.

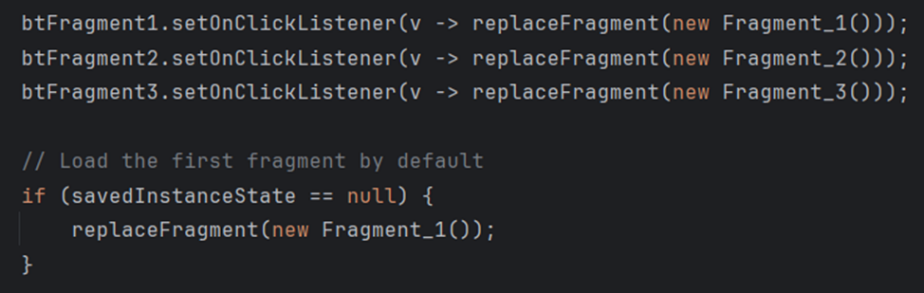


Рисунок 1.5.2 — Добавление фрагмента в MainActivity с использованием FragmentContainerView

На рисунках 1.5.3–1.5.4 демонстрируется процесс смены фрагментов в Android-приложении. При первом запуске автоматически загружается Fragment\_1 и отображается на экране. При нажатии на одну из кнопок происходит замена текущего фрагмента на соответствующий выбранной кнопке, в данном случае — Fragment\_2.



Рисунок 1.5.3 — Начальный экран с загруженным Fragment\_1



Рисунок 1.5.4 — Интерфейс после переключения на Fragment\_2

# ПРАКТИЧЕСКОЕ ЗАДАНИЕ

# ЗАКЛЮЧЕНИЕ