|  |  |  |
| --- | --- | --- |
|  |  |  |
| 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](#_Toc192876081)

[1 ТЕОРИТИЧЕСКАЯ ЧАСТЬ 5](#_Toc192876082)

[1.1 Активность 5](#_Toc192876083)

[1.2 Логирование 7](#_Toc192876084)

[1.3 Взаимодействие с элементами пользовательского интерфейса 12](#_Toc192876085)

[1.4 Переход между экранами 16](#_Toc192876086)

[1.5 Передача данных между Activity 21](#_Toc192876087)

[2 ПРАКТИЧЕСКАЯ ЧАСТЬ 25](#_Toc192876088)

[2.1 Отслеживание работы жизненного цикла активности 25](#_Toc192876090)

[2.1.1 Этап onCreate 25](#_Toc192876091)

[2.1.2 Этап onStart 26](#_Toc192876092)

[2.1.3 Этап onResume 27](#_Toc192876093)

[2.1.4 Этап onPause 27](#_Toc192876094)

[2.1.5 Этап onStop 28](#_Toc192876095)

[2.1.6 Этап onDestroy 29](#_Toc192876096)

[2.2 Переход на активность с помощью кнопки 30](#_Toc192876097)

[2.2.1 Декларативный способ 30](#_Toc192876099)

[2.2.2 Программный способ 32](#_Toc192876100)

[2.3 Переход в активность с передачей данных 35](#_Toc192876101)

[ЗАКЛЮЧЕНИЕ 41](#_Toc192876102)

ВВЕДЕНИЕ

Цель данной работы — ознакомиться с работой контейнеров, переходов между активностями с передачей данных в определённые поля и различных ресурсов.

В ходе выполнения задания будет изменён проект из предыдущей практической работы. Выполнение работы включает в себя следующие задачи:

1. Реализовать несколько файлов разметки с применением следующих контейнеров: LinerLayout, RelativeLayout, Constraint Layout, FrameLayout (При желании можно расширить перечень другими контейнерами). Расположить в файлах разметки различные элементы управления (кнопки, текстовые поля и т.д.).
2. Реализовать переход между несколькими Activity с передачей данных (попробуйте передавать данные в определенные поля, а не сообщением, как в предыдущей практике) и возможностью возврата на предыдущую страницу.
3. Добавить в проект несколько строковых, размерных, цветовых и drawable ресурсов. Произвести изменение настроек темы. Добавить локализацию на другой язык.

# ТЕОРИТИЧЕСКАЯ ЧАСТЬ

## Передача сложных объектов между активностями

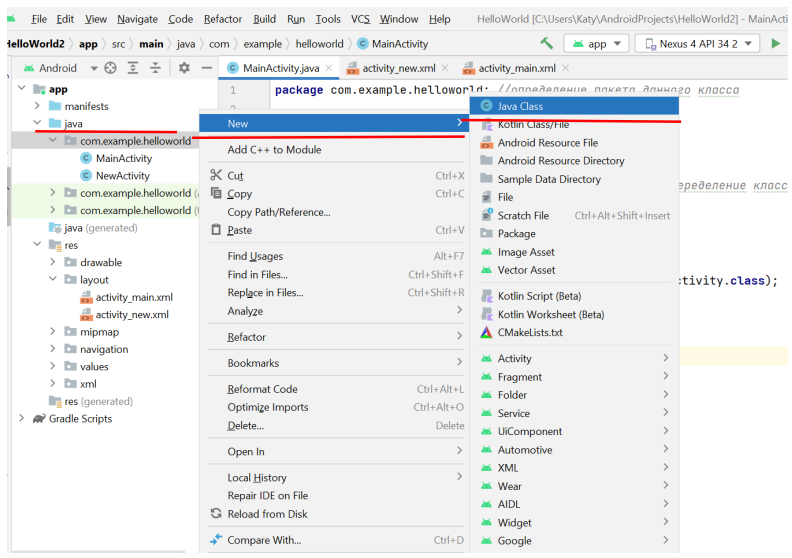
Передача данных между активностями является одной из ключевых функций при разработке приложений для Android. В прошлых практиках рассматривались простые примеры передачи таких данных, как строки или числовые значения. Однако в приложениях часто возникает необходимость передавать более сложные объекты, содержащие несколько полей различных типов.

Для реализации подобного сценария Android предоставляет специальный механизм, основанный на сериализации данных и использовании объекта Intent. Сериализация представляет собой процесс преобразования состояния объекта в поток байтов, который может храниться в памяти устройства или передаваться через сеть и затем снова преобразовываться в исходный объект. В Android существует два основных подхода к сериализации объектов: Serializable и Parcelable. Serializable прост и удобен в использовании, однако Parcelable предпочтительнее в Android-разработке за счёт большей производительности и меньших затрат ресурсов.

Передача объектов между активностями происходит в несколько этапов. Сначала следует создать объект того класса, который необходимо передать. Затем убедиться, что класс объекта реализует интерфейс Serializable или Parcelable, добавив необходимые методы для корректного преобразования данных объекта в байты и обратно. После чего необходимо упаковать объект в специальный объект Intent с помощью методов putExtra() (для Serializable) или putParcelableExtra() (для Parcelable).

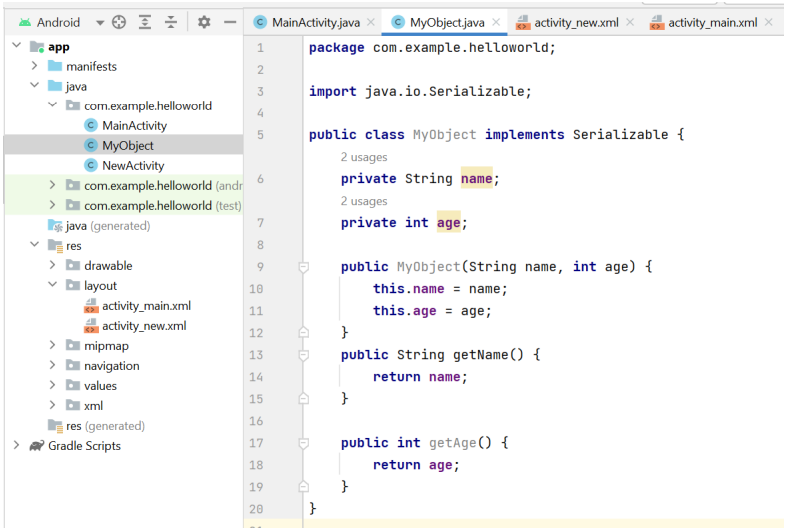
В качестве примера рассмотрим создание простого класса MyObject, который будет передаваться между активностями. Для создания нового класса необходимо перейти в среду разработки Android Studio, открыть контекстное меню для папки "java", затем выбрать создание нового класса Java. В появившемся диалоговом окне следует указать имя создаваемого класса.

На рисунке 1 представлен процесс создания нового класса в Android Studio.

  
Рисунок 1 – Создание нового Java Class

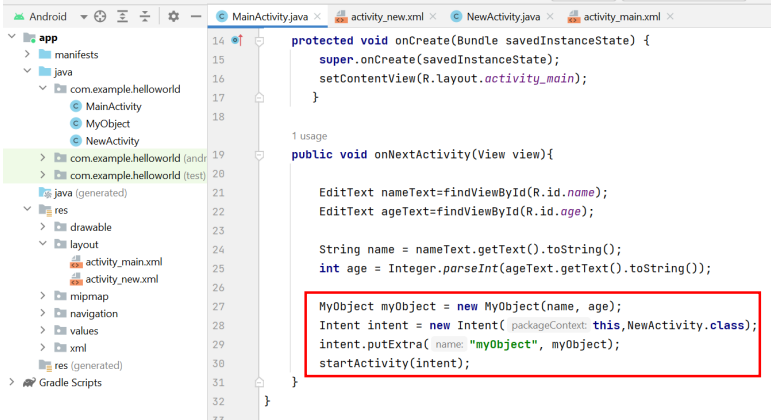
После того как новый класс создан, его необходимо описать. Для этого открываем созданный класс MyObject и объявляем необходимые переменные, а также методы, которые потребуются для корректной сериализации и десериализации объекта.

На рисунке 2 показан пример реализации класса MyObject, реализующего интерфейс Serializable.

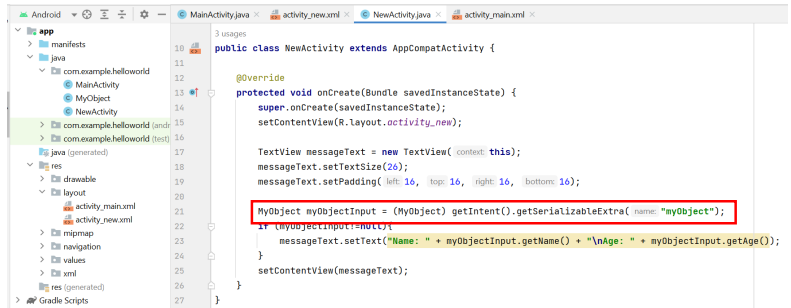
  
Рисунок 2 – Класс MyObject, реализующий интерфейс Serializable

Затем создадим объект этого класса в активности. Для наглядности используем пример из теоретического введения прошлой практической работы и будем передавать те же данные, которые вводили на первой Activity. В таком случае передаем конкретные переменные.

На рисунке 3 представлен код для запуска активности с передачей сложного объекта.

  
Рисунок 3 – Запуск активности с передачей сложного объекта

После чего необходимо извлечь объект из целевой активности. На рисунке 4 показан код извлечения объекта в целевой активности.

  
Рисунок 4 – Код запуска активности с извлечением сложного объекта

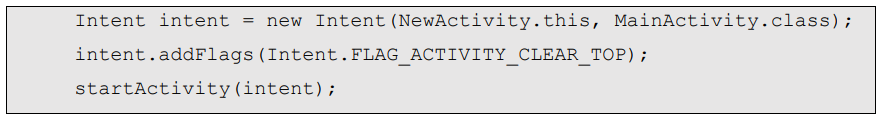
Таким образом, осуществляется передача сложных объектов между разными активностями.

## BackStack

BackStack в Android – это важный механизм, предназначенный для управления навигацией между экранами (активностями или фрагментами) внутри приложения. BackStack можно представить в виде стека, где каждое новое окно приложения добавляется на вершину, и при нажатии кнопки «Назад» пользователь возвращается на предыдущий экран, удаляя текущий с вершины стека.

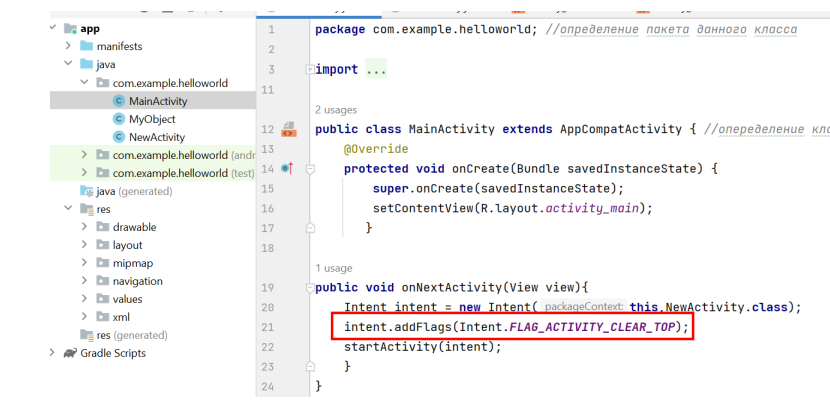
Android позволяет программно управлять BackStack, что дает возможность разработчику гибко настраивать поведение навигации, удаляя или очищая определённые активности из стека. Например, чтобы при возвращении к MainActivity удалить все активности выше неё из стека, можно использовать специальный флаг «Intent.FLAG\_ACTIVITY\_CLEAR\_TOP».

На рисунке 5 представлен пример кода, демонстрирующий использование данного флага при запуске новой активности.

  
Рисунок 5 – Использование флага «Intent.FLAG\_ACTIVITY\_CLEAR\_TOP» для очистки BackStack перед запуском активности

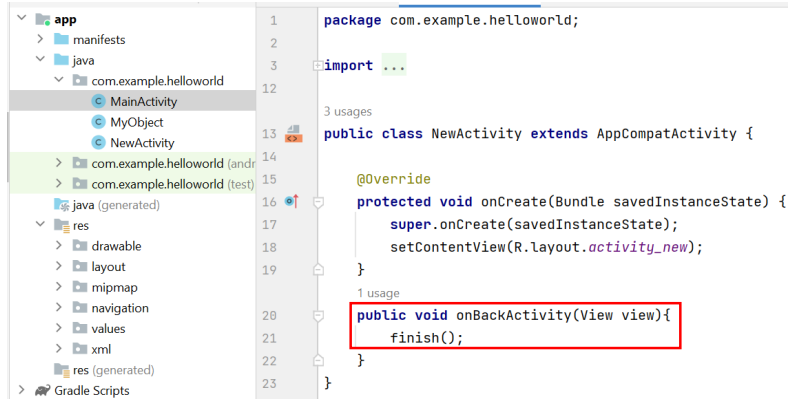
При добавлении флага «FLAG\_ACTIVITY\_CLEAR\_TOP» система удаляет все активности, расположенные выше указанной в Intent активности. Таким образом, когда пользователь возвращается в MainActivity, активность NewActivity будет удалена из стека.

На рисунке 6 показан метод перехода между активностями с использованием указанного выше кода.

  
Рисунок 6 – Метод перехода между активностями с использованием флага «Intent.FLAG\_ACTIVITY\_CLEAR\_TOP»

Еще одним способом управления навигацией является метод finish(), позволяющий завершать работу текущей активности. Этот метод полезен в случаях, когда активность уже не нужна, и её закрытие помогает экономить ресурсы системы.

На рисунке 7 представлен пример использования метода finish() для завершения активности.

  
Рисунок 7 – Пример использования метода finish() для завершения работы активности

## Контейнеры

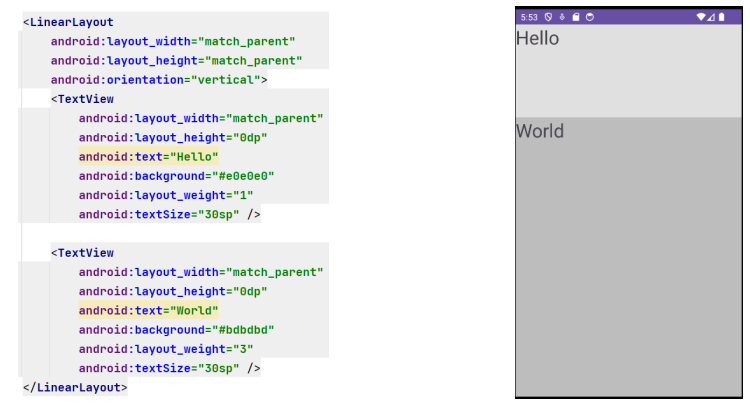
Контейнеры компоновки в Android представляют собой специализированные элементы, предназначенные для организации и управления расположением интерфейсных элементов на экране устройства. Они играют важную роль в адаптации пользовательского интерфейса к разнообразию размеров экранов и разрешений, характерных для устройств на платформе Android.

Контейнеры компоновки руководствуются определёнными правилами при размещении дочерних элементов, контролируя их положение и размеры. В Android существует несколько видов контейнеров, таких как LinearLayout, RelativeLayout, FrameLayout и другие.

### LinearLayout

LinearLayout упорядочивает элементы последовательно, горизонтально или вертикально, в зависимости от указанной ориентации. Одним из наиболее полезных свойств этого контейнера является атрибут android:layout\_weight, который позволяет управлять распределением свободного пространства между элементами. Например, два элемента с весами 1 и 3 займут соответственно 1/4 и 3/4 свободного пространства в контейнере.

На рисунке 8 представлен пример использования LinearLayout.

  
Рисунок 8 – Пример использования LinearLayout

Атрибут android:orientation используется для определения направления расположения вложенных элементов. Он может иметь иметь значения «vertical» или «horizontal».

### RelativeLayout

RelativeLayout позволяет гибко размещать элементы относительно друг друга или родительского контейнера, что предоставляет разработчику большие возможности в определении расположения и размеров элементов интерфейса.

Пример реализации такого подхода показан на рисунке 9.

  
Рисунок 9 – Пример использования RelativeLayout

### FrameLayout

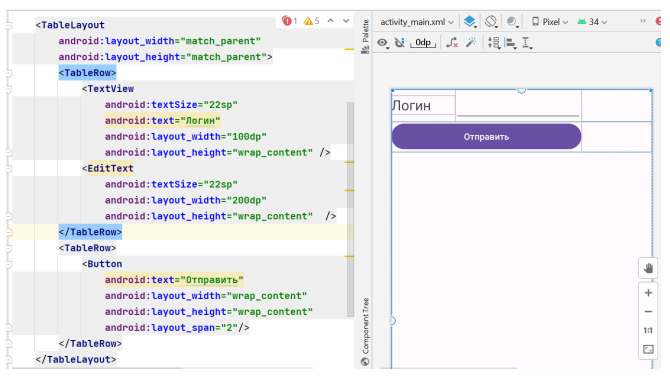
FrameLayout используется для наложения элементов друг на друга. Такой подход полезен при создании сложных интерфейсов, например, отображения загрузочных индикаторов или пользовательских компонентов с наложением элементов. Пример использования FrameLayout представлен на рисунке 10.



Рисунок 10 – Пример использования RelativeLayout

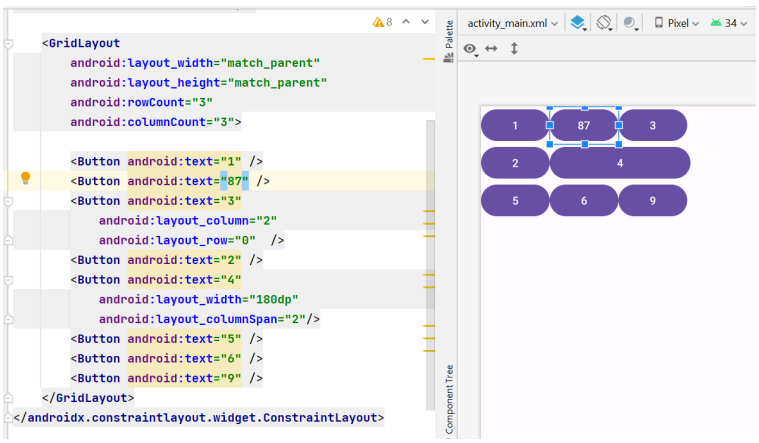
### TableLayout

TableLayout структурирует элементы управления по столбцам и строкам, формируя интерфейс в виде таблицы. Пример использования TableLayout показан на рисунке 11.

  
Рисунок 11 – Пример использования TableLayout

### GridLayout

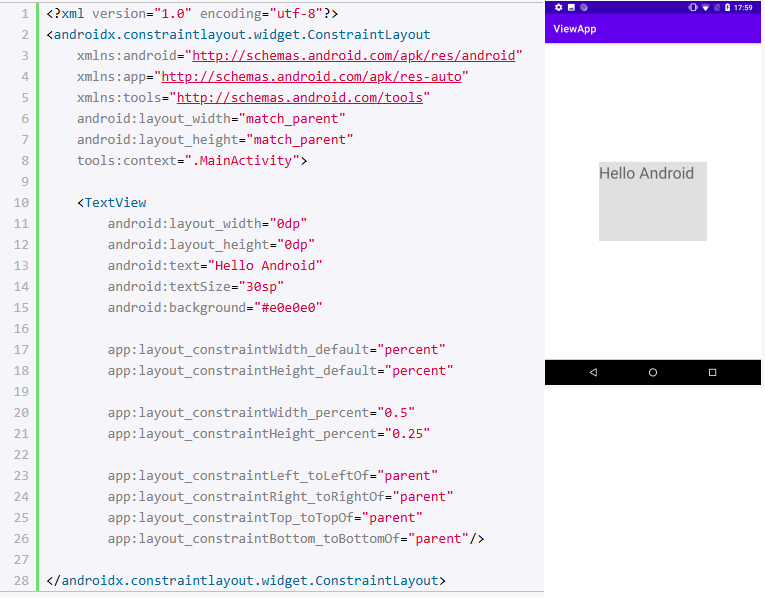
GridLayout – это контейнер, позволяющий создавать интерфейсы в виде таблицы с четким разделением на строки и ячейки. Пример использования GridLayout приведён на рисунке 12.

  
Рисунок 12 – Пример использования GridLayout

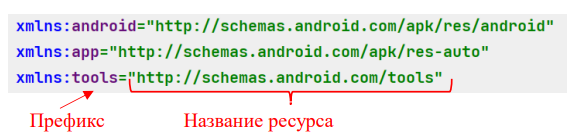
Атрибуты android:rowCount и android:columnCount используются для определения числа рядов и столбцов в сетке соответственно.

### ConstraintLayout

ConstraintLayout является относительно новым типом контейнера, который позволяет разрабатывать сложные и адаптивные интерфейсы с помощью набора ограничений для расположения элементов. По умолчанию при создании новой активности все элементы располагаются в этом контейнере. Пример ConstraintLayout показан на рисунке 13.

  
Рисунок 13 – Пример использования ConstraintLayout

Корневой элемент XML содержит определения используемых пространств имен, таких как android, app и tools. Они задаются с помощью атрибутов xmlns и определяют функциональность элементов и атрибутов внутри файла разметки. На рисунке 14 показан пример таких атрибутов.

  
Рисунок 14 – Примеры атрибутов xmlns

Каждое пространство имен задается следующим образом: xmlns:префикс="название\_ресурса". Через префикс можно ссылаться на функциональность этого пространства имен. Пространства имен определяют различную функциональность, необходимую для работы приложения:

* xmlns:android="http://schemas.android.com/apk/res/android": Содержит основные атрибуты, которые предоставляются платформой Android и определяют визуальные свойства элементов (например, размер, позиционирование);
* xmlns:app="http://schemas.android.com/apk/res-auto": Содержит атрибуты, которые определены самим приложением;
* xmlns:tools="http://schemas.android.com/tools": Используется для работы в режиме дизайнера в Android Studio.

Для организации элементов внутри контейнера используются параметры разметки, задаваемые в XML-файле с помощью атрибутов с префиксом layout\_. Основные атрибуты, такие как layout\_height и layout\_width, имеют следующие возможные значения:

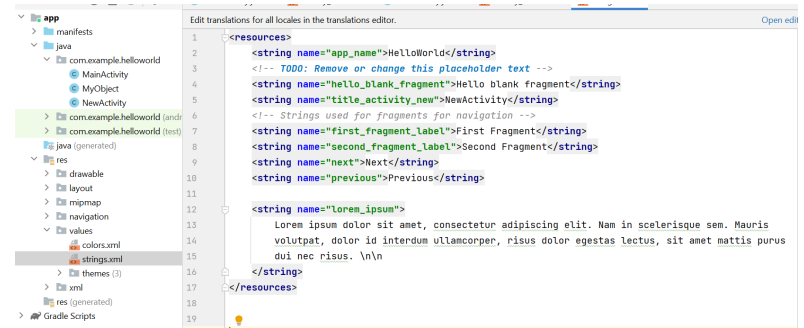
1. match\_parent: Элемент растягивается на всю ширину или высоту контейнера. Не рекомендуется для использования в ConstraintLayout, вместо этого используется 0dp;
2. wrap\_content: Элемент занимает столько места, сколько необходимо для его содержимого;
3. Точные значения (например, 50dp, 100px), задаваемые вручную.

## Ресурсы в Android

Ресурсы в Android — это специальные элементы, которые предназначены для хранения различных типов данных, отдельно от исходного кода приложения. Они включают строки, цвета, изображения, макеты и другие ресурсы. Важным преимуществом использования ресурсов является возможность их легкого управления и изменения без необходимости повторной компиляции всего приложения.

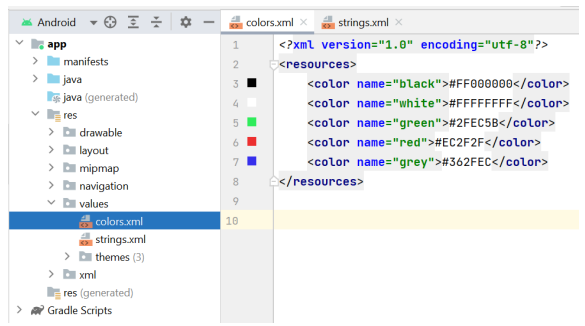
Одним из наиболее распространенных типов ресурсов являются строковые ресурсы. Они предназначены для хранения текстовой информации, такой как надписи на кнопках, сообщения пользователю и другие текстовые элементы. Использование строковых ресурсов значительно облегчает процесс локализации приложений, поскольку позволяет легко адаптировать текст для различных языков без необходимости изменения самого кода.

Пример файла строковых ресурсов представлен на рисунке 15.

  
Рисунок 15 – Пример файла strings.xml

Цветовые ресурсы используются для определения цветовой палитры приложения, включая цвета фона, текста, кнопок и других элементов интерфейса. Хранение цветов в отдельных ресурсах позволяет унифицировать и централизовать управление цветовой схемой, упрощая процесс дизайна. Также это облегчает изменение цветовой схемы для разных тем или режимов (например, дневного и ночного), позволяя изменять цвета во всем приложении без необходимости редактирования каждого элемента вручную.

Пример файла цветовых ресурсов представлен на рисунке 16.

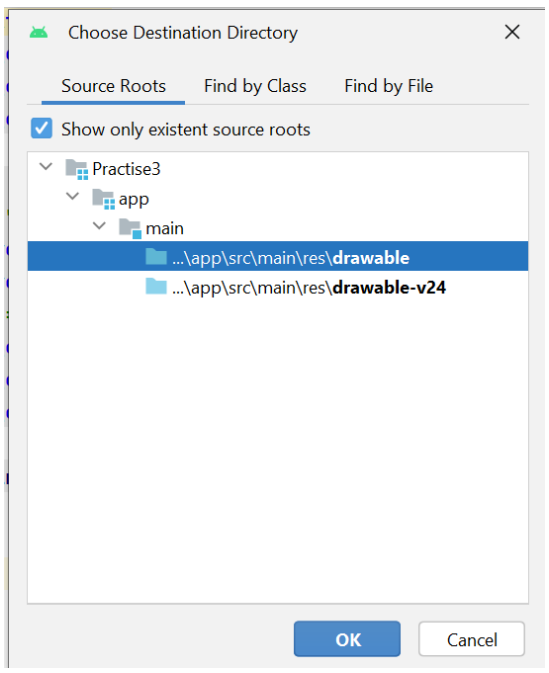
  
Рисунок 16 – Пример файла colors.xml

Оба ресурса располагаются в пакете values и создаются по единому шаблону: сначала указывается тип данных (например, color или string), затем в атрибуте name задаётся имя ресурса, после чего прописывается его значение.

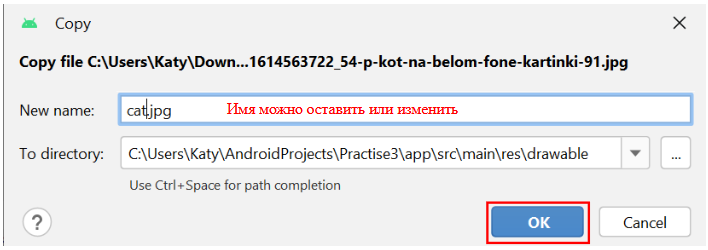
Изображения и графические элементы, как правило, хранятся в пакете drawable в форматах PNG и JPEG, так как использование других форматов может привести к ошибкам в Android Studio. Графические ресурсы играют важную роль в визуальном оформлении приложения, а их хранение в отдельной папке упрощает обновление и замену изображений без необходимости изменять код.

Чтобы добавить изображение в проект, нужно скопировать на жёстком диске файл в формате png или jpg и вставить его в папку res/drawable (копирование выполняется обычным способом — Copy**-**Paste).

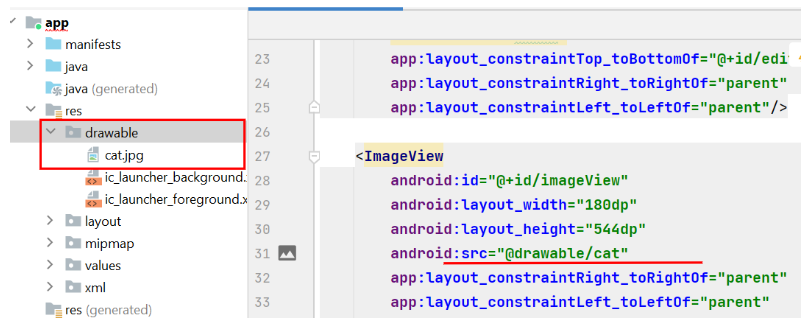
После вставки Android Studio предложит выбрать папку — drawable или drawable-24. Для добавления стандартных изображений следует выбрать drawable:



При копировании файла нам будет предложено установить для него новое имя.



После этого выбранное изображение будет добавлено в папку **drawable**. Чтобы отобразить его в **ImageView**, необходимо установить атрибут android:src для соответствующего элемента.

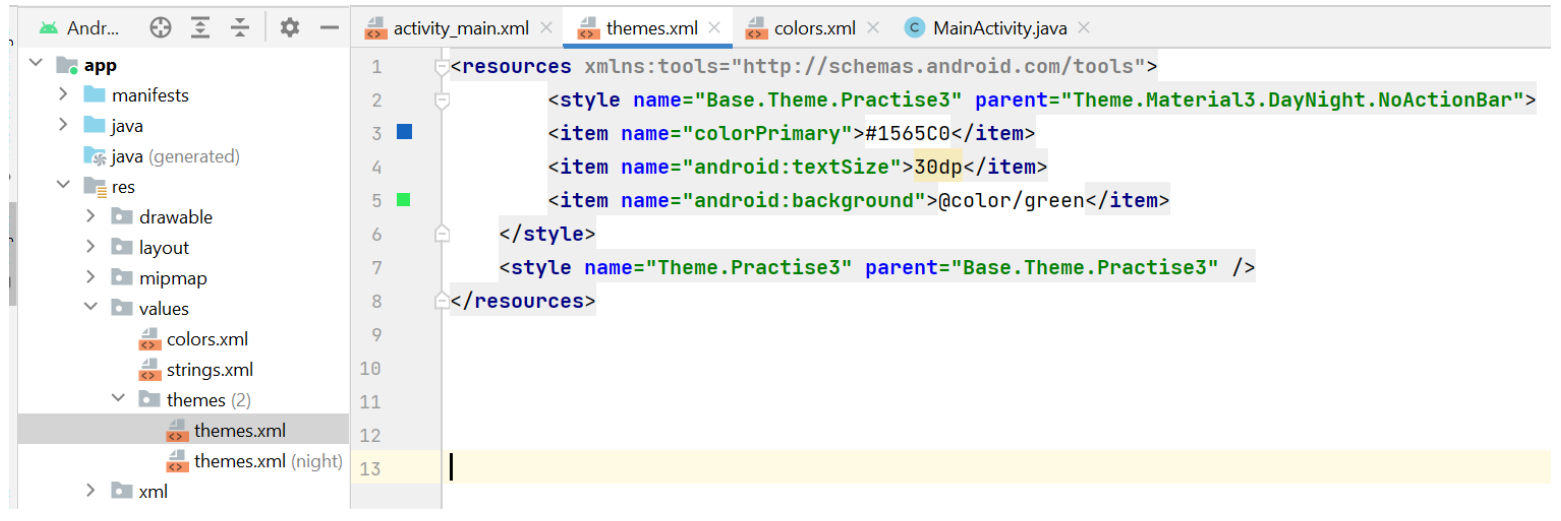


Макеты (layouts) описывают структуру пользовательского интерфейса приложения, определяя расположение элементов, таких как кнопки, текстовые поля и другие виджеты. Эти макеты хранятся в пакете layout.

Ресурсы в приложении обеспечивают более тонкую настройку и персонализацию, позволяя, например, изменить тему или локализовать приложение для конкретной страны или региона.

Для изменения темы нужно перейти в пакет values, а затем в подкаталог theme, где находятся файлы для светлой и темной тем. Эти файлы содержат стили, в которых задаются наборы атрибутов для настройки внешнего вида компонентов приложения.

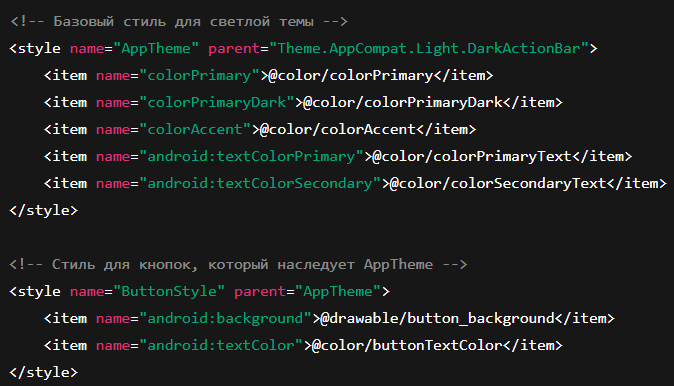
Основные атрибуты включают цвета, шрифты, размеры, отступы и другие параметры, определяющие визуальное представление элементов. Например, атрибут colorPrimary задает основной цвет темы, textSize — размер шрифта, а background — фон элемента.



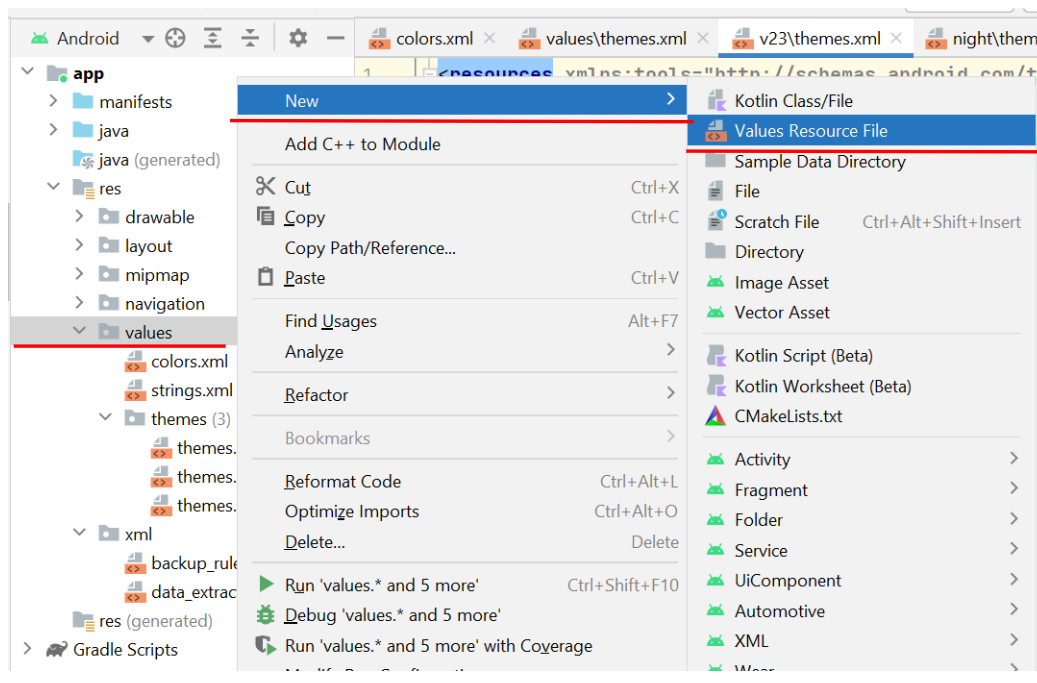
Результатом станет измененные параметры для всей темы.



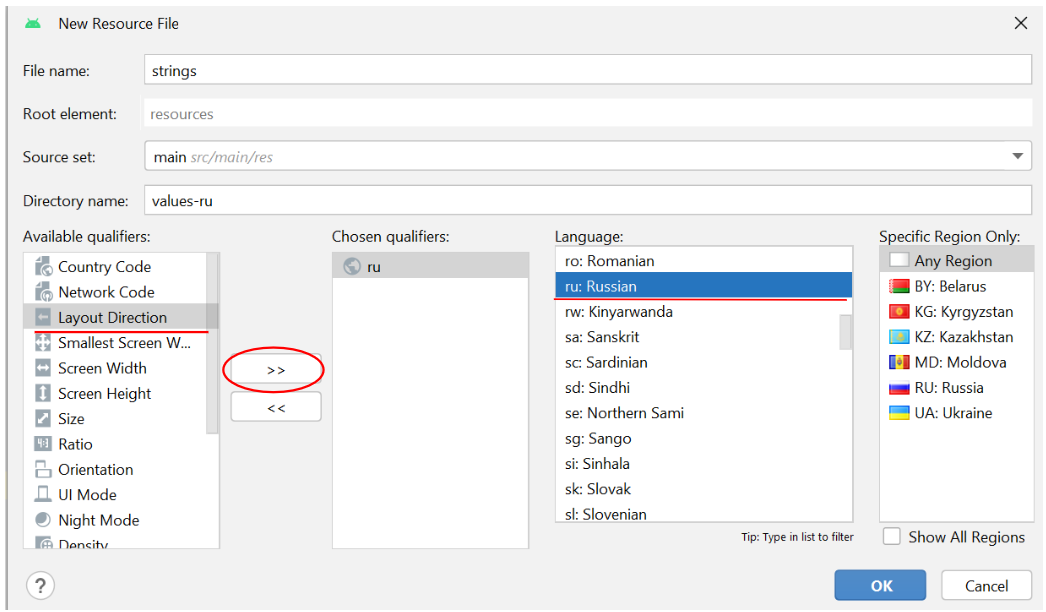
Не менее важным элементом в иерархии стилей является параметр parent, который позволяет указать родительский стиль для текущего. Это даёт возможность наследовать атрибуты родительского стиля и при необходимости изменять или расширять их.



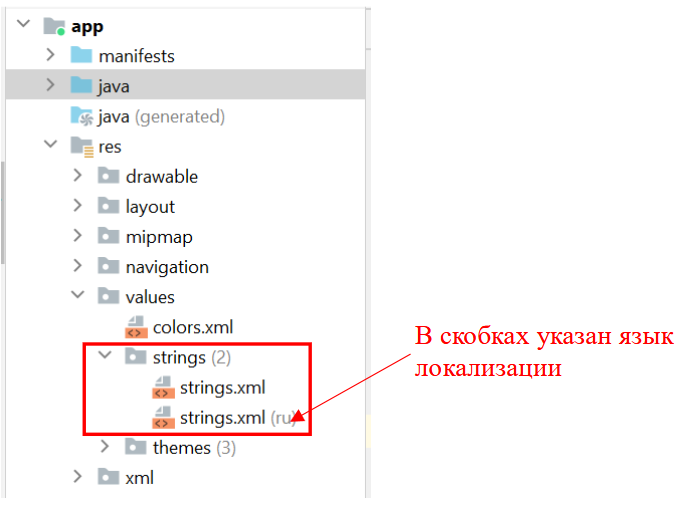
Локализация приложения осуществляется схожим образом, но с особенностью, что для каждого региона необходимо создавать отдельный файл со строками. Для этого в пакете values нужно кликнуть правой кнопкой мыши и выбрать в контекстном меню параметр Locale, а затем выбрать нужный регион, например, ru (название файла может немного отличаться в зависимости от региона).



В диалоговом окне, в левой части под названием Available qualifiers, выбираем пункт Locale и с помощью кнопки с двумя стрелками вправо переносим его в правую часть, в список Chosen qualifiers. В третьей колонке, которая появится после этого, выбираем нужный язык или регион, например, русский. После этого в поле Directory name автоматически отобразится соответствующее название папки для локализации.

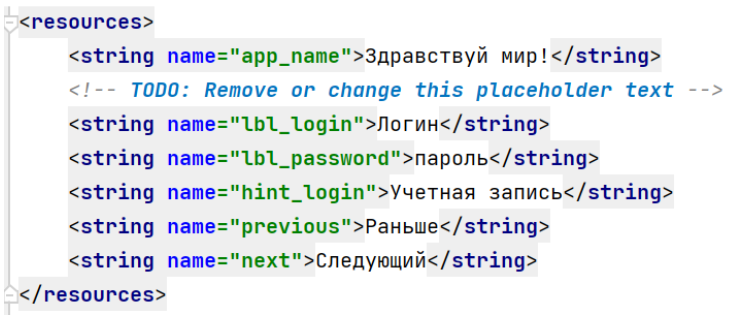


В результате создания ресурсных файлов в структуре проекта будут добавлены записи в каталог res/values/strings.xml. В этом файле будут храниться строки для локализации, которые можно использовать в приложении для различных языков или регионов.

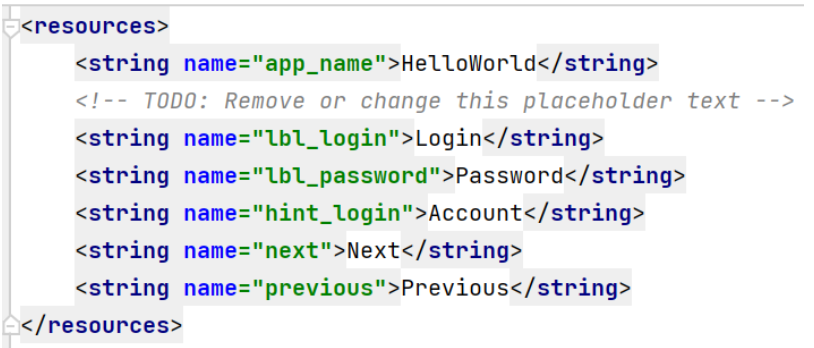


Для корректной работы перевода необходимо перенести ранее созданные строки в новый файл для локализации. Таким образом, в этом файле будут храниться тексты, адаптированные для российского региона, например, в res/values-ru/strings.xml. Это обеспечит правильное отображение переведённых строк при использовании русского языка в приложении.

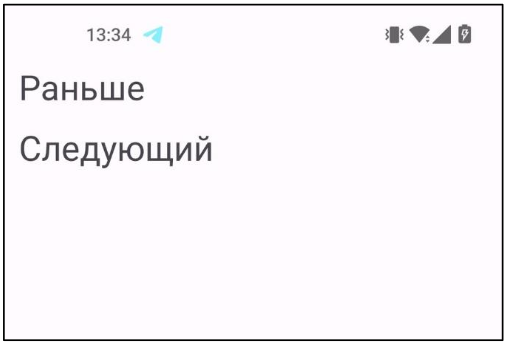
Листинг файла values-ru/string.xml



Листинг файла values/string.xml



Если запустить приложение, то локализация будет работать.



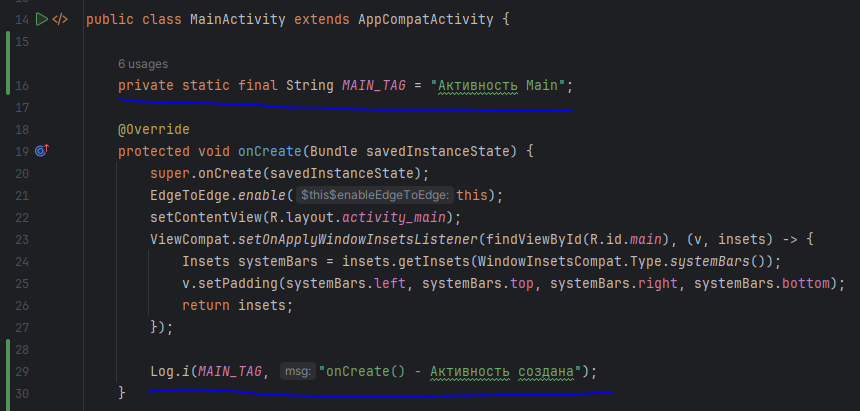
# ПРАКТИЧЕСКАЯ ЧАСТЬ



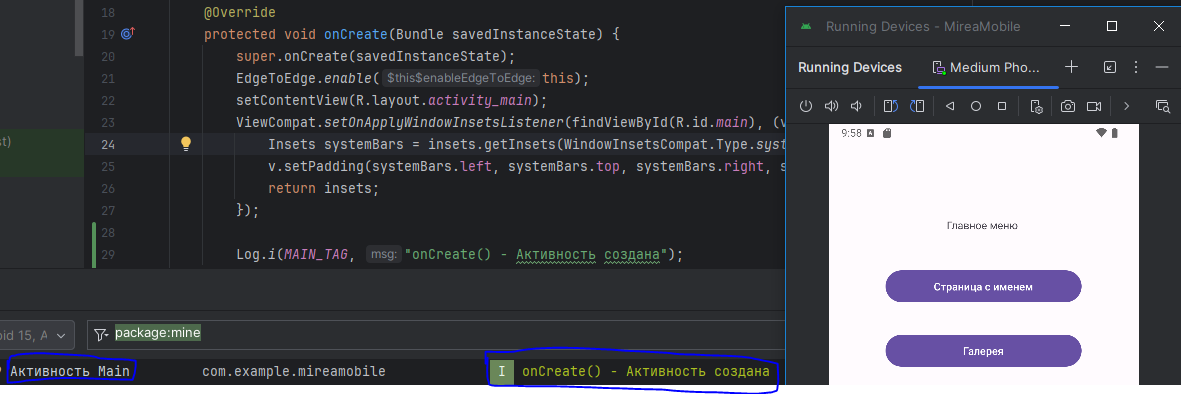
## Отслеживание работы жизненного цикла активности

### Этап onCreate

Для логирования создания активности создадим тег «MAIN\_TAG» с значением «Активность Main», обозначающий работу в MainActivity и добавим логирование в методе onCreate. Полученный код продемонстрирован на рисунке 24.

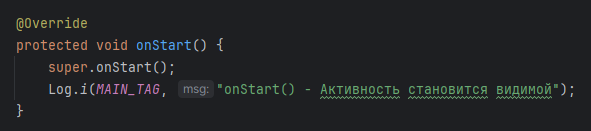
  
Рисунок 24 – Код для логирования метода onCreate, выделены тег и вызов логирования

Теперь при создании активности в Logcat будет отображаться информация, отображённая на рисунке 25.

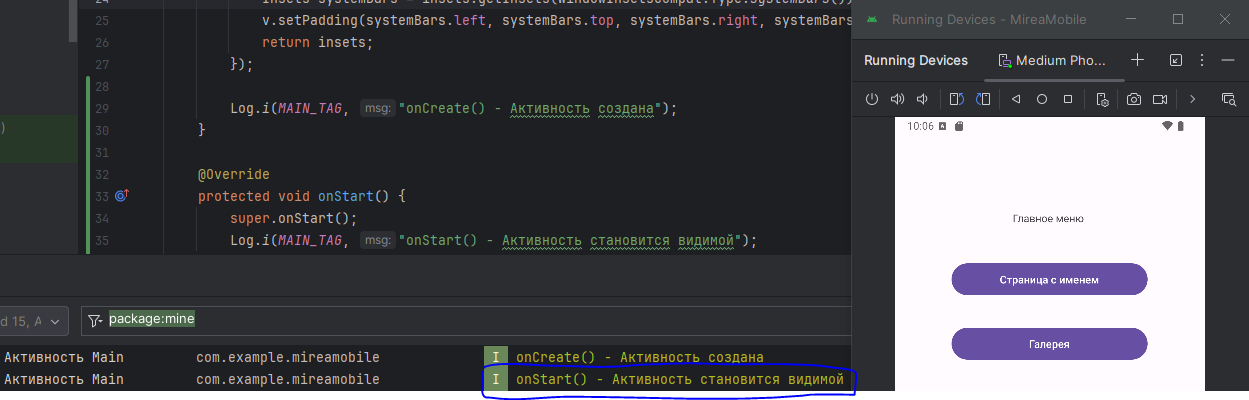
  
Рисунок 25 – Логирование этапа onCreate, выделены тег и текст лога

### Этап onStart

Для логирования этапа onStart нужно перезаписать метод onStart. Новый фрагмент кода показан на рисунке 26.

  
Рисунок 26 – Метод логирования метода onStart

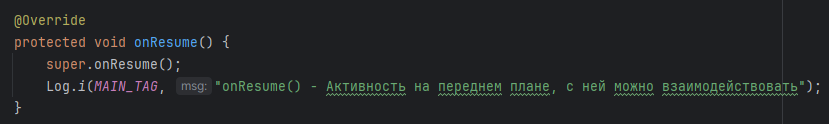
Теперь при открытии активности в Logcat будет отображаться информация, отображённая на рисунке 27.

  
Рисунок 27 – Логирование метода onStart

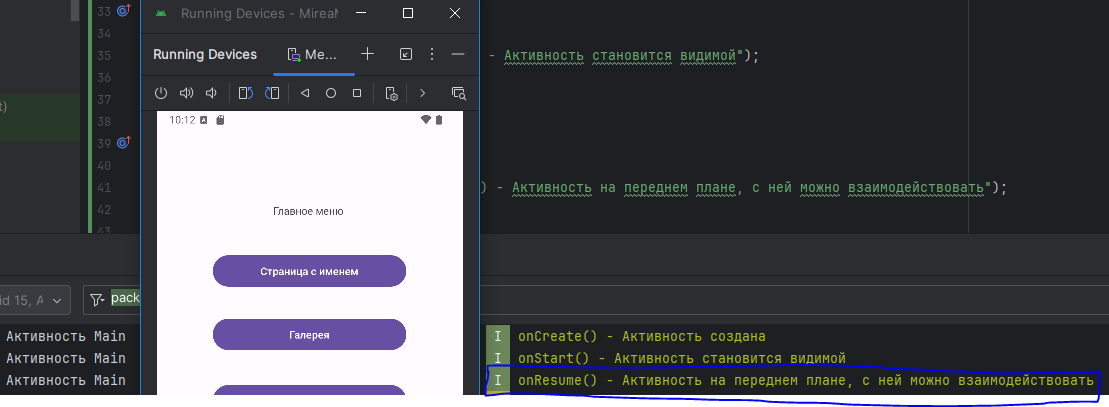
Также этот лог будет появляться каждый раз, когда активность становится видима пользователю.

### Этап onResume

Для логирования этапа onResume нужно перезаписать метод onResume. Новый фрагмент кода показан на рисунке 28.

  
Рисунок 28 – Код для логирования этапа onResume

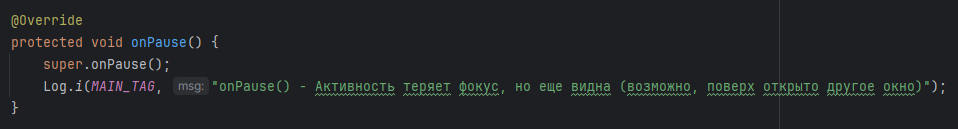
Теперь при возобновлении работы активности в Logcat будет отображаться информация, отображённая на рисунке 29.

  
Рисунок 29 – Логирование этапа onResume

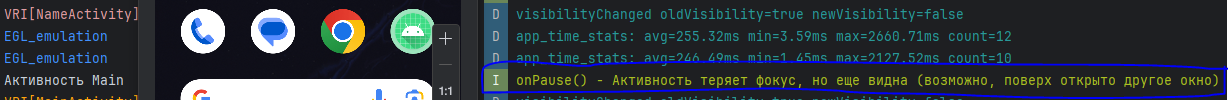
Этот лог будет появляться каждый раз, когда активность оказывается на первом плане и с ней можно взаимодействовать.

### Этап onPause

Для логирования этапа onPause нужно перезаписать метод onPause. Новый фрагмент кода показан на рисунке 30.

  
Рисунок 30 – Метод логирования метода onPause

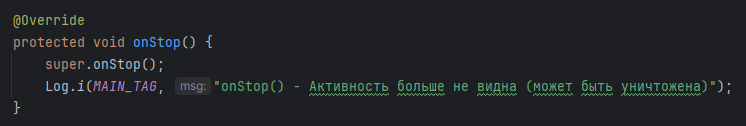
Теперь при смещении фокуса с активности (например, при закрытии приложения) в Logcat будет отображаться информация, отображённая на рисунке 31.

  
Рисунок 31 – Логирование метода onPause

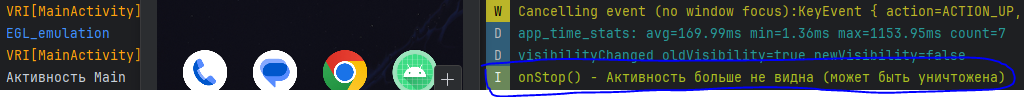
Также этот лог будет появляться каждый раз, когда активность теряет фокус, но всё ещё видима пользователю.

### Этап onStop

Для логирования этапа onStop нужно перезаписать метод onStop. Новый фрагмент кода показан на рисунке 32.

  
Рисунок 32 – Код для логирования этапа onStop

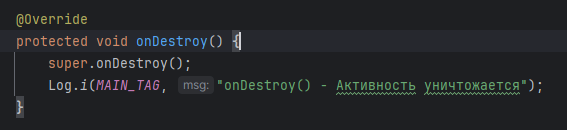
Теперь при скрытии активности (например, при закрытии приложения) в Logcat будет отображаться информация, отображённая на рисунке 33.

  
Рисунок 33 – Логирование этапа onStop

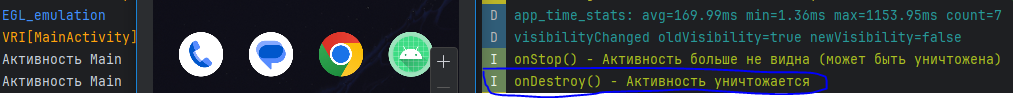
Этот лог будет появляться каждый раз, когда активность перестаёт быть видимой. Этот этап подготавливает активность к удалению её из памяти.

### Этап onDestroy

Для логирования этапа onDestroy нужно перезаписать метод onDestroy. Новый фрагмент кода показан на рисунке 34.

  
Рисунок 34 – Метод логирования метода onDestroy

Теперь при удалении активности из памяти (например, при закрытии приложения) в Logcat будет отображаться информация, отображённая на рисунке 35.

  
Рисунок 35 – Логирование метода onDestroy

Также этот лог будет появляться каждый раз, когда активность удаляется из памяти. Обычно перед этим этапом вызывается onPause или onStop.

Однако система может убить процесс приложения без вызовов onPause, onStop или onDestroy (например, из-за нехватки памяти).

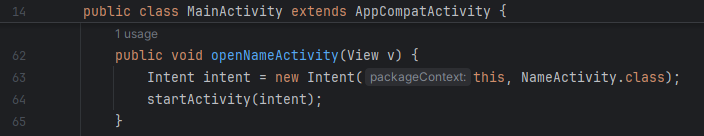
## Переход на активность с помощью кнопки



### Декларативный способ

Для реализации перехода от одной активности к другой, воспользуемся кнопкой и активностями, реализованными в рамках прошлой практической работы. Кнопка будет вести из главного меню (MainActivity) в активность с ФИО студента (NameActivity).

Для работы перехода необходимо написать обработчик события, в котором будет создан Intent. Код обработчика представлен на рисунке 36.

  
Рисунок 36 – Код обработчика события нажатия на кнопку

Далее в атрибут кнопки onClick нужно записать метод, который будет обработчиком события нажатия на кнопку (openNameActivity). Реализация в XML-коде показана на рисунке 37.

  
Рисунок 37 – XML-код кнопки, ведущей на активность с ФИО студента, выделен атрибут onClick

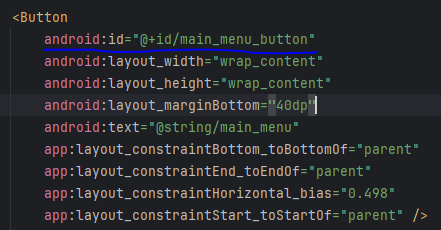
Теперь при нажатии на кнопку «Страница с именем» пользователя перенесёт на активность с ФИО студента. Открывшаяся после нажатия кнопки активность продемонстрирована на рисунке 38.

  
Рисунок 38 – Активность, на которую переносит пользователя при нажатии на кнопку «Страница с именем»

### Программный способ

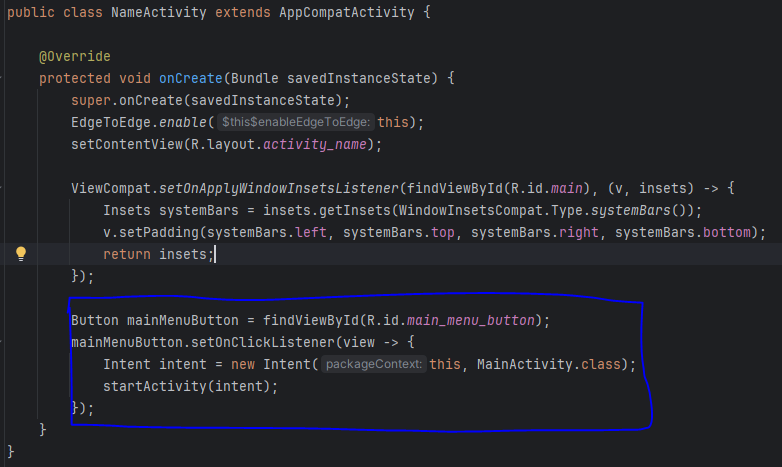
Кнопку «Главное меню», ведущую из активности с ФИО студента обратно в главное меню, реализуется уже программным способом.

Сперва реализуется верстка кнопки и указывается идентификатор кнопки («main\_menu\_button»). Полученный XML-код отображен на рисунке 39.

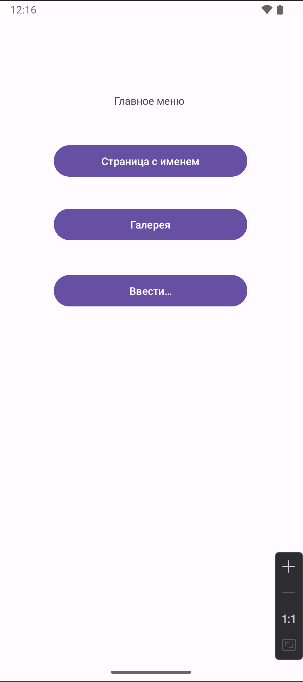
  
Рисунок 39 – XML-код кнопки, ведущей обратно в главное меню, выделен атрибут идентификатора кнопки

Для работы кнопки нужно добавить слушатель к этой кнопке. Его можно добавить в методе onCreate. Для этого необходимо найти нужную кнопку через «R.id.main\_menu\_button» и добавить к нему слушатель, который будет создавать новый Intent, перенося пользователя обратно в активность MainActivity.

Реализованный код активности показан на рисунке 40.

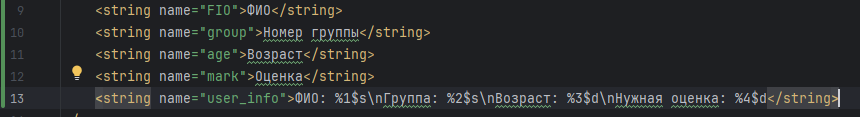
  
Рисунок 40 – Java-код активности NameActivity, выделен фрагмент кода, отвечающий за создание слушателя нажатия на кнопку

В результате при нажатии кнопки «Главное меню» пользователя вернёт в главное меню. Открывшаяся после нажатия кнопки активность продемонстрирована на рисунке 41.

  
Рисунок 41 - Активность, на которую переносит пользователя при нажатии на кнопку «Главное меню»

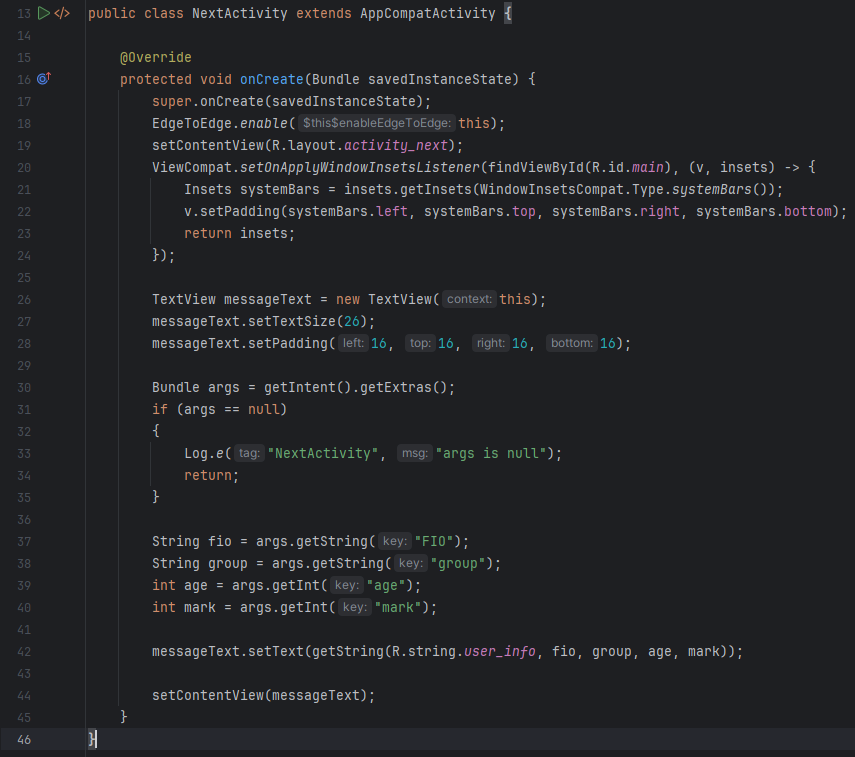
## Переход в активность с передачей данных

Сперва для удобства редактирования нужно добавить заготовленные строки в файл strings.xml, которые понадобятся во время разработки. Новые строки представлены на рисунке 42.

  
Рисунок 42 – Добавленные заготовленные строки

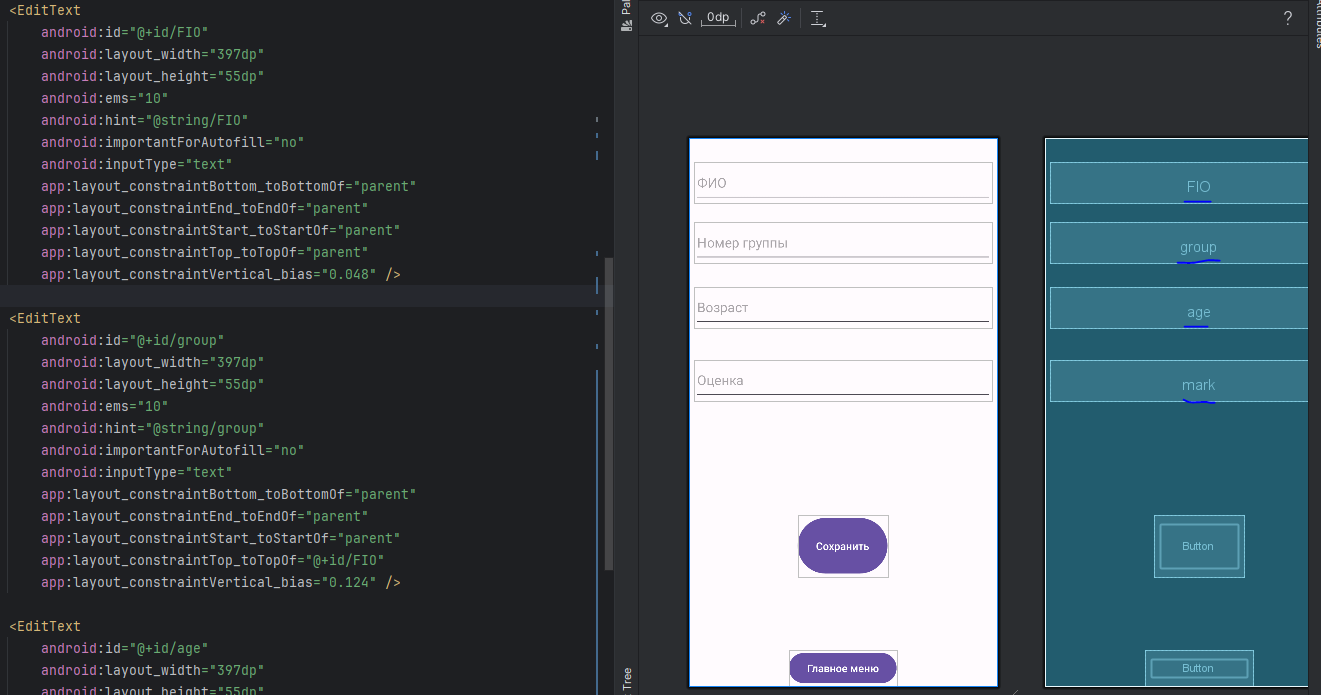
Нужно создать пустую активность NextActivity. Она будет принимать аргументы «FIO», «group», «age», «mark» при запуске и показывать их пользователю.

На рисунке 43 продемонстрирован Java-код активности NextActivity.

  
Рисунок 43 – Java-код активнсти NextActivity

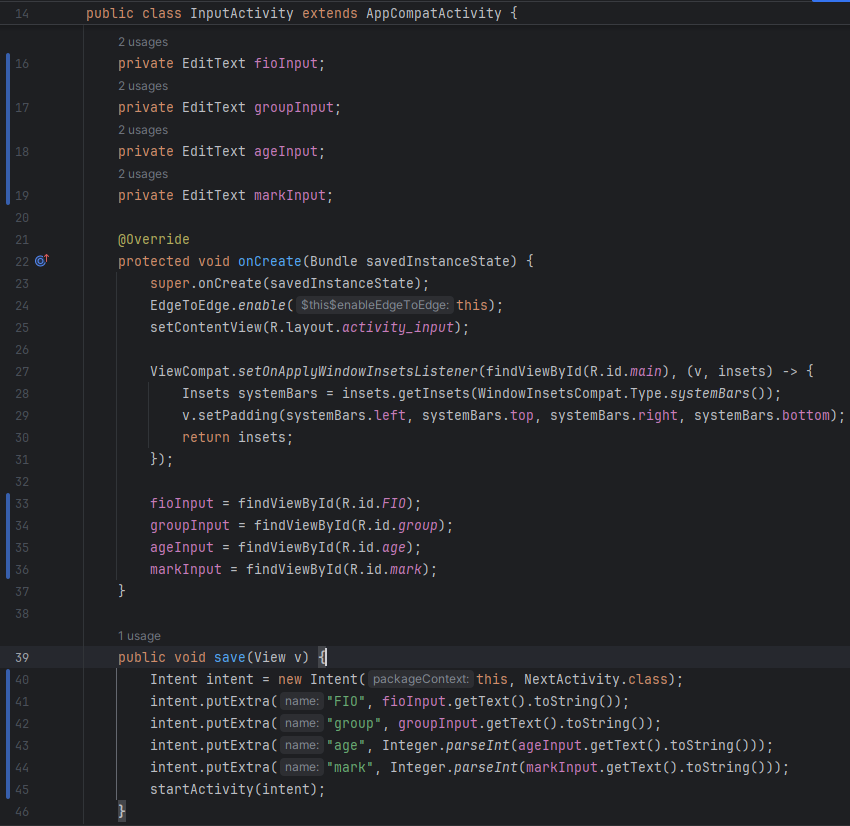
В этом коде при создании активности создаётся элемент TextView, в который вносятся данные, введённые пользователем.

Чтобы протестировать работу новой активности, были изменены поля у активности ввода. Были добавлены новые поля для ввода ФИО, номера группы, возраста и оценки. Модель изменённой активности ввода показана на рисунке 44.

  
Рисунок 44 – Изменённая модель активности ввода, выделены идентификаторы новых полей

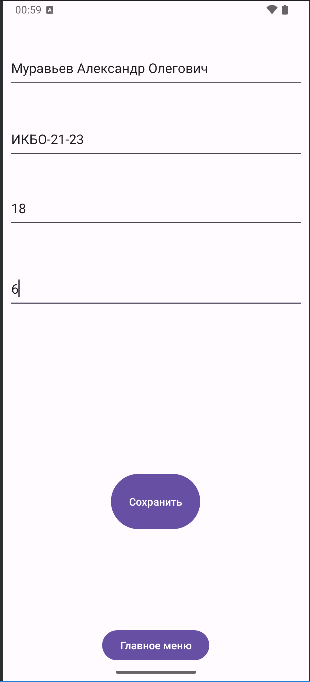
Для создания новой активности с введёнными данными нужно при нажатии кнопки «Сохранить» считать данные полей и создать Intent с добавлением этих полей.

Java-код активности InputActivity представлен на рисунке 45.

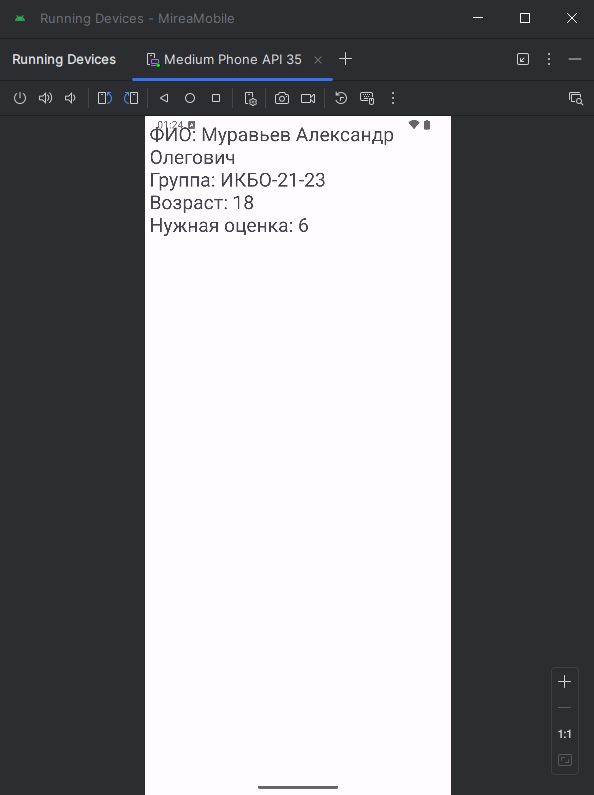
  
Рисунок 45 – Java-код активности InputActivity

Метод save декларативно закреплён за кнопкой «Сохранить».

Для проверки нужно ввести данные студента в поля в приложении. Заполненная форма показана на рисунке 46.

  
Рисунок 46 – Заполненная форма с информацией о студенте

После нажатия кнопки «Сохранить» создаётся активность «NextActivity» с введёнными данными, что продемонстрировано на рисунке 47.

  
Рисунок 47 – Создавшаяся активность с данными студента

ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы были получены навыки работы с логированием и активностями в Android-приложениях, а также их взаимодействием с помощью механизма Intent. Были освоены методы обработки событий пользовательского интерфейса, включая программное назначение обработчиков нажатий через setOnClickListener.

Кроме того, была изучена передача данных между активностями и их последующее получение, что позволяет реализовывать динамическое взаимодействие между экранами приложения.

Таким образом, в рамках данной работы были закреплены базовые знания по навигации в Android-приложениях и взаимодействию между компонентами.

В результате практической работы было модифицировано мобильное приложение, которое получило новый функционал логирования и перехода между активностями в том числе с передачей данных между активностями.