МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Практическое занятие № 2/4ч.

Разработка мобильных компонент анализа безопасности информационно-аналитических систем

(наименование дисциплины (модуля) в соответствии с учебным планом)

Уровень	бакалавриат
	(бакалавриат, магистратура, специалитет)
Форма обучения	очная
	(очная, очно-заочная, заочная)
Направление(-я) подготовки	10.05.04 «Информационно-аналитические системы безопасности»
	(код(-ы) и наименование(-я))
Институт	кибербезопасности и цифровых технологий
	(полное и краткое наименование)
Кафедра	КБ-2 «Информационно-аналитические системы кибербезопасности»
	(полное и краткое наименование кафедры, реализующей дисциплину (модуль))

Используются в данной редакции с учебного года 2024/25

(учебный год цифрами)

Проверено и согласовано «____» _____20___г. _____ _____

(подпись директора Института/Филиала с расшифровкой)

Москва 2025 г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.

Ключевым компонентом для создания визуального интерфейса в приложениях «*Android»* является «*activity»* (активность), как правило ассоциируемая с отдельным экраном или окном приложения, а переключение между окнами производится как перемещение от одной «*activity*» к другой. Учет размера экрана устройства является одним из важнейших факторов при разработке мобильного приложения. Зачастую отсутствует возможность размещения всех элементов приложения на одном экране. Очевидным решением данной проблемы является разделение интерфейса на функциональные части:

– использование различных сообщений (диалоговые окна, уведомления, всплывающие подсказки) – способ наиболее простой и не требует редактирования файла манифеста, однако таким образом возможно решить только часть задач;

– использование в одном приложении нескольких «*activity»* – способ является универсальным и подходит для любых приложений, но не самый оптимальный с точки зрения производительности и количества генерируемого кода;

– разместить компоненты на активности таким образом, что в нужный момент возможно будет легко переключиться на работу с другой частью интерфейса;

– использовать собственные экраны и отображать их на «*activity*» в зависимости от состояния приложения.

ГЛАВА 1 ИНТРУМЕНТЫ ОТЛАДКИ ПРИЛОЖЕНИЙ

Среда разработки позволяет отлаживать состояние приложений на подключенных устройствах. Имеется возможность просмотра системного журнала сообщений, установки точек остановы, проверки значений переменных и вычисления выражений во время работы, производить скриншоты и видеозаписи.

1.1 Режим отладки

Как правило, для запуска приложения требуется использовать вкладки меню «Run» (зелёный треугольник на панели инструментов среды разработки) или «Run *ИмяМодуля*». Дополнительным средством запуска приложения является режим отладки, для вызова которого требуется произвести нажатие кнопки «Debug» (рисунок 1.1).

Рисунок 1.1 – Панель инструментов среды разработки «*Android Studio*»

Перед вызовом режима отладки требуется отметить точки останова слева от кода, в которых будет приостановлено выполнение приложения. Режим отладки позволяет в нужной строчке кода выполнить проверку значения переменных, запустить выражение и продолжать выполнение кода строчка за строчкой, а также определить ошибки, которые не удаётся вычислить простым просмотром кода. На рисунке 1.2 представлен пример запущенного режима отладки.

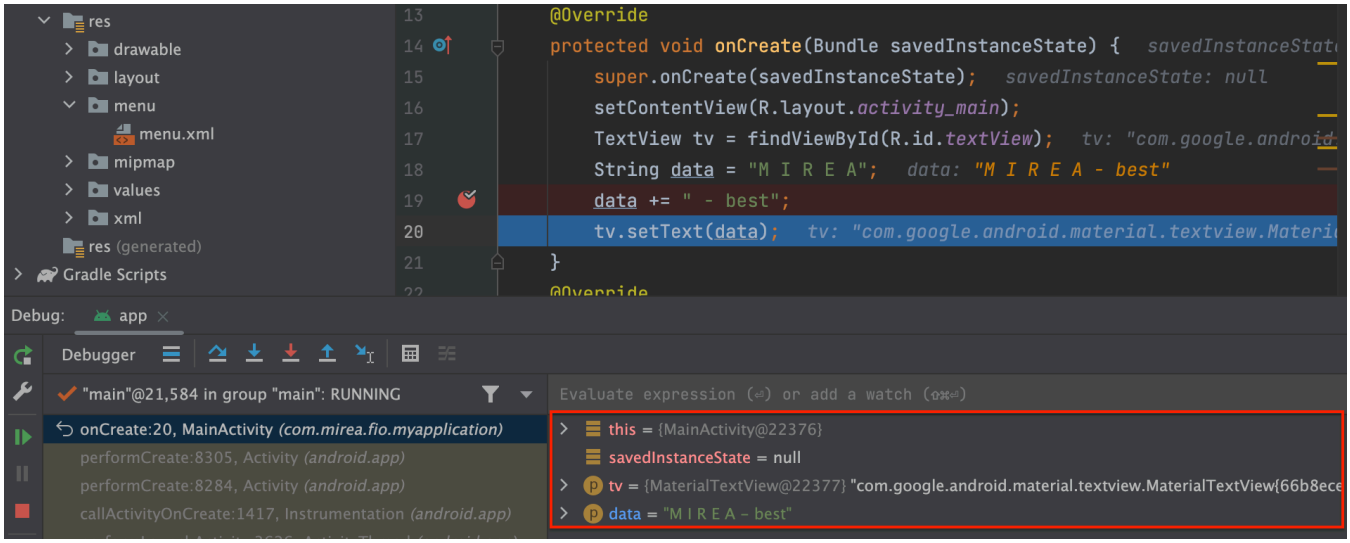


Рисунок 1.2 – Показатели приложения в режиме отладки в среде разработки «*Android Studio*»

При запущенном приложении в отладочном режиме выполнение кода происходит до установленной точки останова. На рисунке 1.2 представлено окно «*Debug*» в нижней части рисунка, отображающее потоки и переменные во вкладке «*Debugger*». Стоит отметить, что в случае запущенного приложения, возможно присоединиться к процессу в режиме отладки с помощью нажатия на кнопку «*Attach debugger to Android proccess*».

Также, существует инструмент «Android Profiler», позволяющий анализировать работу центрального процессора, памяти и сетевую активность. Для запуска инструмента требуется выбрать пункты меню «*View* | *Tool Windows* | *Profiler*» (рисунок 1.3). В режиме реального времени возможно отображать графики нагрузки на процессор, а также пользовательские действия (нажатия, «свайпы» и т.д.) выводятся в виде фиолетовых кружочков.

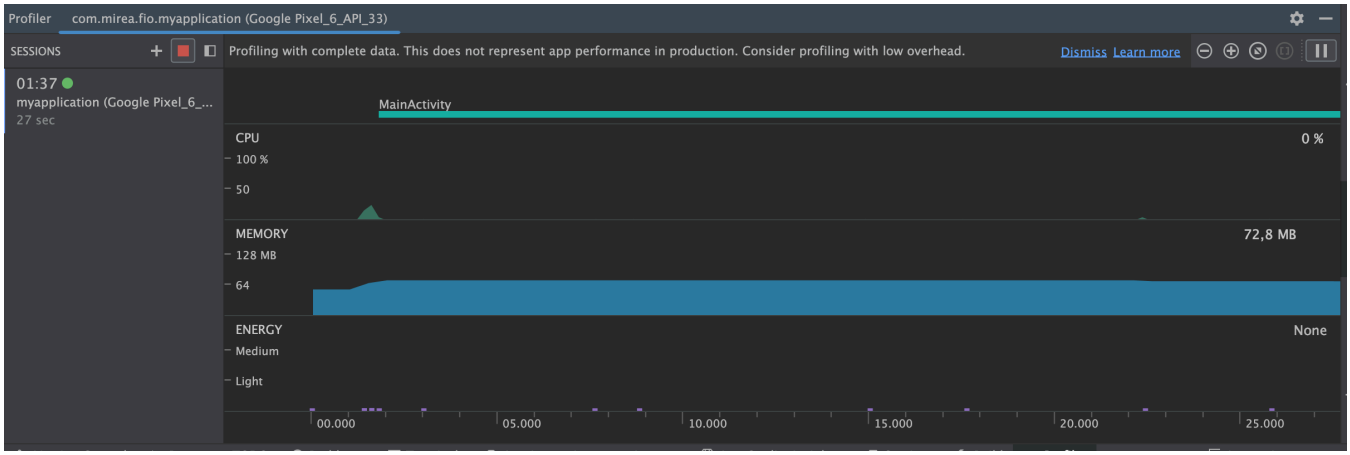


Рисунок 1.3 – Инструмент «Profiler» среды разработки «*Android Studio*»

1.2 Инструментарий просмотра отладочных сообщений

Инструмент «*LogCat*» входит в набор инструментов «*Android SDK*» и предназначен для отображения сообщений с устройства или эмулятора. Основное предназначение инструмента заключается в отображении промежуточных результатов дампа файла логирования. Класс «*android.util.Log*» позволяет классифицировать сообщения по категориям в зависимости от важности. Для разбивки по категориям используются специальные идентификаторы, указывающие на категорию:

- «*Log.e*» – ошибки («*error*»)
- «*Log.w*» – предупреждения («*warning*»)

- «*Log.i*» – информация («*info*»)
- «*Log.d*» – отладка («*debug*»)
- «*Log.v*» – подробности («*verbose*»)
- «*Log.wtf*» – серьезная ошибка, которая не должна была произойти («*What a Terrible Failure*», работает начиная с Android 2.2)

Создание сообщения требует вызова статического метода класса «*Log*». Далее приведён пример создания сообщения:

```
Log.d(TAG, "Мой код выполняется!");
```

Первым аргументом метода является строка, называемая тегом. Обычно принято объявлять строковую переменную класса «TAG» с указанием имени класса:

```
private String TAG = MainActivity.class.getSimpleName();
```

После запуска приложения и выполнения указанной строки созданное сообщение отобразиться во вкладке «*LogCat*» среды разработки (рисунок 1.4).

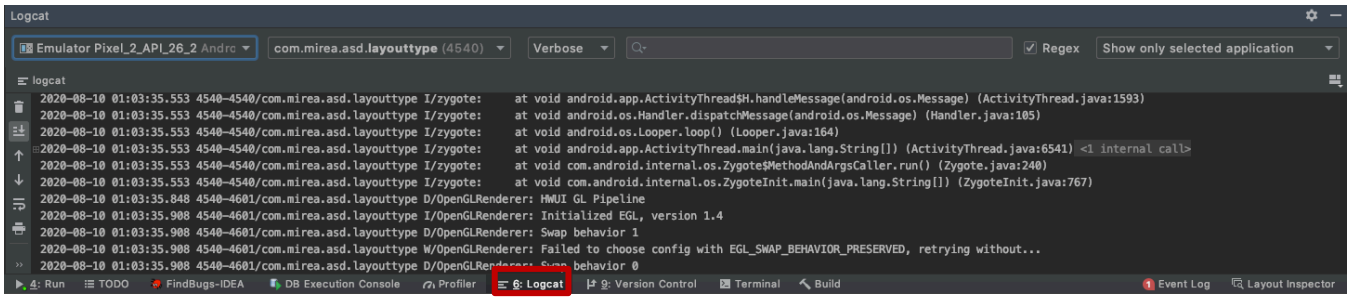


Рисунок 1.4 – Программа LogCat в Android Studio

В «*LogCat*» возможно производить фильтрацию сообщений по заданному тегу, процессу и устройству, чтобы видеть на экране сообщения только определённого типа. Для этого требуется выбрать нужный тип тега из выпадающего списка «*Log Level*».

Стоит отметить, что использование точек остановы не всегда может привести к полноценному определению поведения приложения. Наиболее используемыми случаями применения логирования являются работа с сетью, потоками и другими асинхронными реализациями.

ГЛАВА 2 ЖИЗНЕННЫЙ ЦИКЛ ACTIVITY

Создание графических интерфейсов вместе с дочерним классом «Activity» является не единственным способом, и в дальнейших занятиях будут изучены другие возможности.

Создание классического мобильного приложения требует наличия одного или нескольких «*activity*». Как правило, работа приложения начинается с класса «*MainActivity*»:

```
public class MainActivity extends AppCompatActivity {
…..
}
```

Все «*activity*» представляют собой объекты класса «*android.app.Activity*», который содержит базовую функциональность для всех «*activity*». В примере выше «*MainActivity*» является наследником класса «*AppCompatActivity*», который в свою очередь наследуется от базового класса «*Activity*».

Приложения для ОС «*Android*» имеют строго определенный системой жизненный цикл. При запуске пользователем приложения операционная система создает отдельный процесс, что позволяет определять различный приоритет. Благодаря этому имеется возможность при работе с различными приложениями не блокировать входящие звонки. После прекращения работы пользователя с приложением производится высвобождение связанных ресурсов и установка более низкого приоритета процесса. В связи с широкими функциональными возможностями мобильных устройств современные приложения могут выполнять различные операции, каждая из которых должна разрабатываться в соответствии с требуемыми ресурсами. Например, функциональность приложения электронной почты может содержать операцию для отображения списка новых сообщений, выбора и просмотра этого сообщения, создание ответа с возможностью прикрепления файлов, в том числе полученных с камеры устройства.

Функции одного приложения могут вызывать функции, реализованные в других приложениях на устройстве. Например, когда одно приложение инициирует отправку электронной почты, разработчик может создать в своем приложении

сообщение (в контексте ОС «*Android*» используется термин «*намерение*») с адресом электронной почты и текстом. После обнаружения намерения операционная система производит поиск соответствующих приложений с поддерживаемой функциональностью по отправке электронной почты. Далее выполняется отображение пользователю полученного списка или запуск приложения, выбранного по умолчанию. В рассматриваемом случае намерение заключается в том, чтобы отправить сообщение электронной почты, поэтому в почтовых приложениях запускается экран «составить сообщение». После отправки почтового сообщения приложение инициатор возобновляет работу, что позволяет не уводить пользователя в другое приложение. Несмотря на то, что функциональность относится к частям разных приложений, ОС «*Android»* поддерживает удобство работы пользователя, сохраняя обе операции в одной задаче. В данном контексте понятие «задача» подразумевает под собой коллекцию операций, с которыми взаимодействует пользователь при выполнении определенного задания. Операции упорядочены в виде стека (стека переходов назад), в том порядке, в котором открывались операции.

Начальным местом большинства задач является главный экран устройства. Когда пользователь касается значка в среде запуска приложений (англ. «*launcher*») или ярлыка на главном экране, формируется задача в виде отображения приложения на переднем плане. Если для приложения нет задач (приложение не находится в «свернутых»), формируется новая задача и открывается «основная» операция для этого приложения, размещенная в вершине стека операций. В случаях запуска текущей операцией новой задачи, выполняется передача фокуса и перемещение в вершину стека. Предыдущая операция остается в стеке, но ее выполнение останавливается, и система сохраняет текущее состояние ее пользовательского интерфейса. Нажатие пользователем аппаратной или программной кнопки «*Назад*» удаляет из вершины стека операцию и возобновляется работа предыдущей (восстанавливается предыдущее состояние пользовательского интерфейса). Операции в стеке никогда не переупорядочиваются, только добавляются в стек и удаляются из него — добавляются в стек при запуске текущей операцией и

удаляются, когда пользователь выходит из нее с помощью кнопки «*Назад*». Соответственно, работа стека переходов назад соответствует принципу «*последним вошел — первым вышел*». На рисунке 2.1 приведено поведение на временной шкале: состояние операций и текущее состояние стека переходов назад показано в каждый момент времени. Все объекты «*activity*», которые есть в приложении, управляются системой в виде стека «*activity*», который называется «*Back Stack*». Запуск новой «*activity*» сопровождается её размещением поверх стека с последующим отображением на экране устройства, пока не появится новая «*activity*». Когда текущая «*activity*» заканчивает свою работу (например, пользователь уходит из приложения), производится удаление из стека, и возобновляет работу предыдущая «*activity*».

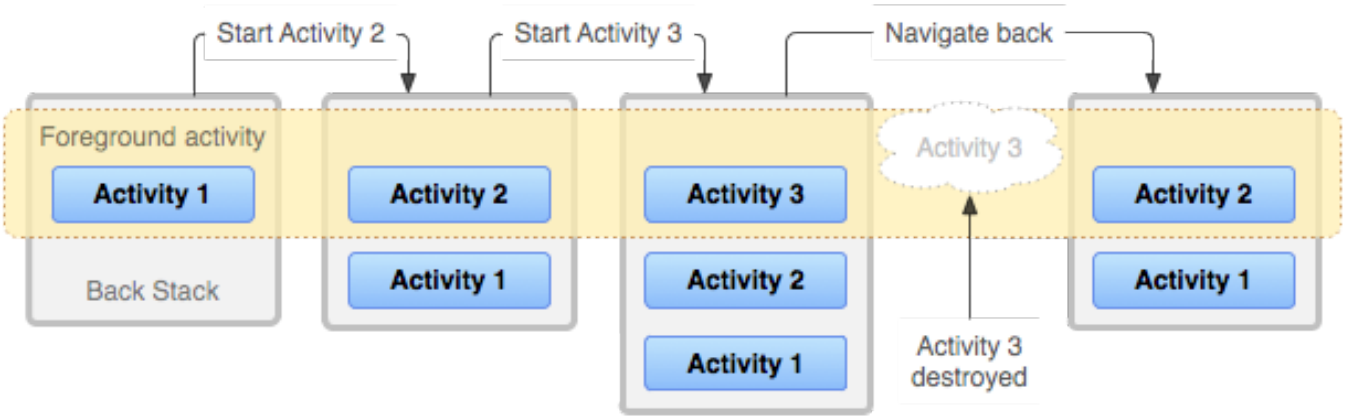


Рисунок 2.1 – Структура стека при вызове нового activity

Нажатие кнопки «*Назад*» приводит к поочередному удалению из стека операций до тех пор, пока пользователь не вернется на главный экран (или в операцию, которая была запущена в начале выполнения задачи). Когда все операции удалены из стека, задача прекращает существование.

2.1 Методы жизненного цикла «Activity»

Запуск мобильного приложения приводит, как правило, к вызову первого экрана – «*activity*». Запуск проходит через ряд событий, которые обрабатываются системой и для обработки которых существует ряд обратных вызовов:

```
protected void onCreate(Bundle saveInstanceState);
protected void onStart();
protected void onRestoreInstanceState(Bundle saveInstanceState);
protected void onRestart();
protected void onResume();
protected void onPause();
protected void onSaveInstanceState(Bundle saveInstanceState);
protected void onStop();
protected void onDestroy();
```

Схематичная взаимосвязь между всеми вызовами activity представлена на рисунке 2.2.

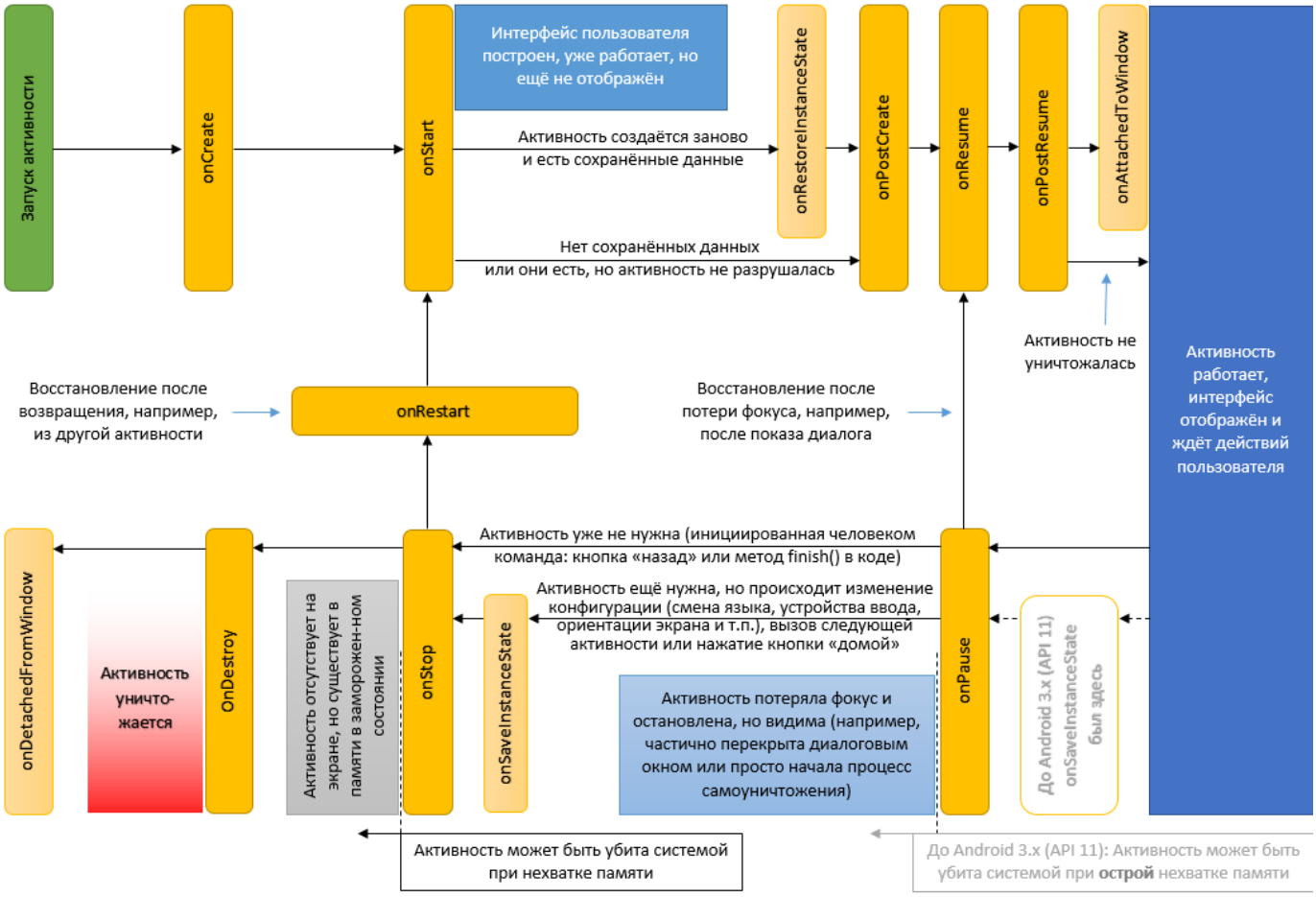


Рисунок 2.2 – Жизненный цикл activity

2.2 Создание активности

Метод «*onCreate*» является первым, с которого начинается выполнение «*activity*». Данный метод должен быть переопределен в классе «*activity*», предназначен для первоначальной настройки экрана посредством связывания кодовой реализации и графического интерфейса. Также, в данном методе производится получение объекта потокобезопасного класса «*Bundle*»,

предназначенного для хранения пар ключ-значение, который может содержать прежнее состояние «*activity*» в случае его сохранения. В случае создания нового «*activity»* данный объект имеет значение «*null*».

```
if ( savedInstanceState == null )   // приложение запущено впервые
{
   countStudents  = 0;    // инициализация студентов
   // другой код
}
else // приложение восстановлено из памяти
{
   // инициализация студентов из памяти суммой
   countStudents = savedInstanceState.getDouble(COUNT_STUDENTS);
}
```

Для построения интерфейса вызывается метод «*setContentView(int)»*, где «*int*» — идентификатор XML-разметки, определяющей интерфейс пользователя. Также в данном методе используют вызовы «*findViewById(@LayoutRes int/View)»* для получения доступа к визуальным элементам.

2.3 Основные методы подготовки отображения активности

Перед отображением пользователю графического интерфейса производится вызов следующих методов:

– метод «*onStart*» выполняет подготовку к выводу «*activity*» на экран устройства. Данный метод не требует переопределения, а всю основную работу производит код родительского класса. Графический интерфейс реализован, однако еще не отображается на экране устройства;

– метод «*onRestoreInstanceState*» предназначен для восстановления сохраненного состояния из объекта «*Bundle*», передаваемого в качестве входного аргумента. Следует учитывать, что данный метод вызывается в случаях, когда «*Bundle*» не равен «*null*» и содержит ранее сохраненное состояние, т.е. при первом запуске приложения метод вызываться не будет;

– метод «*onPostCreate*» предназначен для проведения окончательной инициализации и реализуется в системных классах. Разработчиками переопределение данного метода, как правило, не используется;

– метод «*onResume*» переводит «*activity*» в состояние «*Resumed*» и предоставляет возможность пользователю взаимодействия с ней. «*Activity*» остается в данном состоянии пока не потеряет фокус. Одним из примеров события потери фокуса является переключение на другое «*activity*» или выключение экрана устройства. Также, в данном методе возможно инициализировать компоненты приложения, выполнять регистрацию широковещательных приемников или других процессов, которые были освобождены/приостановлены в «*onPause()*» и т.д. В данном методе требуется выполнять легковесные операции, чтобы пользовательский интерфейс оставался отзывчивым, а основную инициализацию графических компонентов перенести в методы «*onCreate()*» и «*onRestoreInstanceState*». Например, в данном методе возможно производить инициализацию камеры:

```
@Override
public void onResume() {
    super.onResume();
// Производится инициализации камеры после получения фокуса экрана
    if (mCamera == null) {
        initializeCamera(); //Метод работы с камерой
    }
}
```

– метод «*onPostResume*» предназначен для проведения окончательной инициализации после выполнения кода метода «*onResume*» и реализован на уровне системных классов. Разработчиками переопределение данного метода, как правило, не используется;

– метод «*onAttachedToWindow*» вызывается, когда главное окно активности подключается к оконному менеджеру;

– метод «*onPause*» вызывается в случаях, когда пользователь переходит в другую «*activity*». С целью снижения требуемых ресурсов в методе возможно высвобождать используемые ресурсы и приостанавливать процессы (например, воспроизведение аудио, анимации, останавливать работу камеры и т.д.). Однако, требуется учитывать, что продолжительность работы данного метода мала, поэтому не стоит выполнять длительные операции.

```
@Override
public void onPause() {
     super.onPause();
// Высвобождение ресурсов камеры, потому что во время паузы
// не используется
     if (mCamera != null){
          mCamera.release()
          mCamera = null;
     }
}
```

После выполнения данного метода «*activity*» становится невидимой и не отображается на экране, но все еще активна. В случае, если пользователь решит вернуться к данной «*activity*», система вызовет метод «*onResume*», что отобразит ее на экране;

– метод «*onSaveInstanceState*» вызывается после метода «*onPause*», но до вызова «*onStop*». В «*onSaveInstanceState*» производится сохранение состояния приложения с помощью заполнения объекта «*Bundle*». В объект «*Bundle*» возможно передать параметры, динамическое состояние активности как пары «ключ-значение». Когда активность будет снова вызвана, объект «*Bundle*» передаётся системой в качестве параметра в методы «*onCreate*» и «*onRestoreInstanceState*» для восстановления предыдущего состояния активности. Прежде чем передавать изменённый параметр «*Bundle*» в обработчик родительского класса, производится сохранение значения с помощью методов «*putXXX*» и восстановление с помощью «*getXXX*», где «*XXX*» – является типом возвращаемых данных (напр. string, int, float и т.д.). В качестве примера использования «*onSaveInstanceState*» возможно привести состояния флажков, текущего выделенного элемента или введенных, но не сохранённых данных), чтобы объект «*activity*» при следующем входе в активное состояние мог вывести на экран то же состояние. В отличие от базовых методов, методы «*onSaveInstanceState*» и «*onRestoreInstanceState*» не относятся к методам жизненного цикла активности. Например, ОС «*Android*» вызывает «*onSaveInstanceState*» прежде, чем активность становится уязвимой к уничтожению системой, но не вызывает его, когда экземпляр активности разрушается пользовательским действием (при нажатии клавиши «*Back*»). Метод «*onSaveInstanceState*» вызывается системой в случае изменения конфигурации

устройства в процессе выполнения приложения (например, при вращении устройства пользователем или выдвижении физической клавиатуры устройства).

```
@Override
protected void onSaveInstanceState(Bundle outState) {
   super.onSaveInstanceState(outState);
   outState.putInt(COUNT_STUDENTS, 2);
}
```

– метод «*onStop*» переводит активность в состояние «*Stopped*». В методе «*onStop*» следует освобождать используемые ресурсы, которые не требуются пользователю. Здесь также возможно сохранять данные, например, в базу данных. При этом во время состояния «*Stopped*» «*activity*» остается в памяти устройства и сохраняется состояние всех элементов интерфейса. К примеру, если в текстовое поле «*EditText*» был введен текст, то после возобновления работы activity и перехода ее в состояние «*Resumed*» пользователю отобразится ранее введенный текст. Если после вызова метода «*onStop*» произойдет возврат к прежней «*activity*», тогда система вызовет метод «*onRestart*»;

– метод «*onDestroy*» вызывается в случаях уничтожения «activity» системой, либо при вызове метода «*finish*». Также следует отметить, что при изменении ориентации экрана производится завершение жизненного цикла «*activity*» и затем выполняется вызов метода «*onCreate*»;

– метод «*onDetachedFromWindow*» вызывается, когда главное окно активности отключается от оконного менеджера;

– метод «*onRestart*» выполняется, когда окно возвращается в приоритетный режим после вызова «*onStop*», т.е. после того, как активность была остановлена и запущена снова пользователем. Метод предшествует вызовам метода «*onStart*» за исключением первого раза запуска приложения. Используется для специальных действий, которые должны выполняться только при повторном запуске активности в рамках «полноценного» состояния.

Механизм организации жизненного цикла «*Activity*» в ОС «*Android*» очень схож по реализации с навигацией в браузере. Пользователь может находится в одной вкладке («*Task*») и открывать страницы («*Activity*») переходя по ссылкам

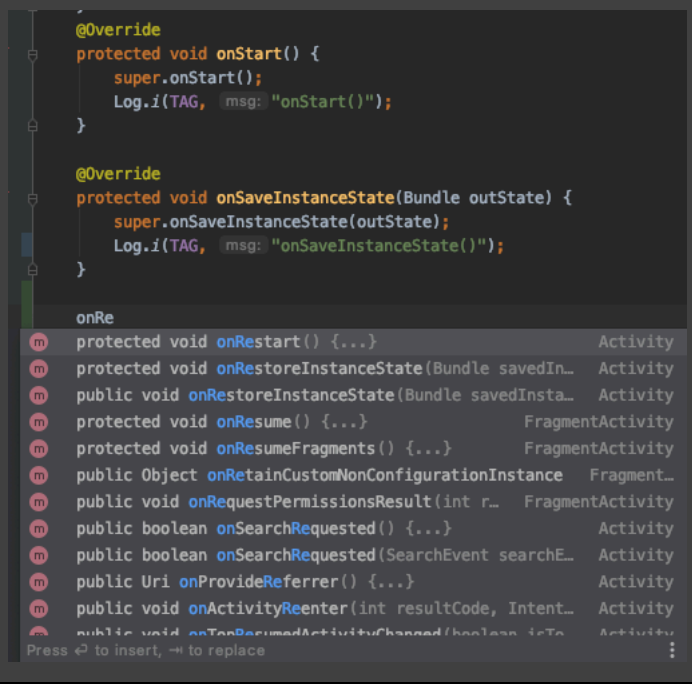
(«*Intent*»). В любой момент возможно вернуться на предыдущую страницу, нажав кнопку «Назад». Также, кнопка «Вперед» отсутствует, т.к. страница, на которой была нажата кнопка «Назад», стирается из памяти. Для отображения новых экранов создается новая вкладка и далее в ней открываются страницы. В результате создано несколько вкладок и большинство из них размещены на заднем фоне. Соответствие аналога браузера и ОС «*Android*» заключается в следующем:

- браузер – ОС «*Android*»;
- вкладка с историей посещений – «*Task*»;
- страница – «*Activity*»;
- ссылка – «*Intent*».

2.4 Контрольное задание

Требуется создать новый проект «*ru.mirea.«фамилия».Lesson2*».

Создайте модуль «*ActivityLifecycle*». В созданном классе изначально присутствует реализация только метода «*onCreate*». Требуется переопределить основные методы жизненного цикла родительского класса, рассмотренные выше. Для создания метода возможно произвести ввод названия метода, среда разработки отобразит всплывающее меню с доступными методами.



Требуется добавить в разметку «*activity_main.xml*» поле текстового ввода «*EditText*» и реализовать отображение состояния «*activity*» используя класс «*Log*»**.** Каждый метод жизненного цикла должен содержать сообщение, содержащее имя метода жизненного цикла. Стоит обратить внимание, что методы жизненного цикла выделяются аннотацией «***@Override***»:

```
@Override
protected void onStart() {
    super.onStart();
    Log.i(TAG, "onStart()");
}
```

Переопределение метода (англ. «*Method overriding*») – позволяет заменять реализацию метода родительского класса в дочернем.

Далее требуется осуществить запуск проекта и изучить сообщения в окне «*logcat*». Дополнительным способом изучения методов жизненного цикла является отслеживание изменения состояния графических элементов на примере созданного «*EditText*». Требуется внести изменения в поле ввода текста, а затем перейти на «Главный экран» (кнопка «*Home*»). Далее снова запустить приложение.

Вопросы:

1. Будет ли вызван метод «*onCreate*» после нажатия на кнопку «Home» и возврата в приложение?
2. Изменится ли значение поля «*EditText*» после нажатия на кнопку «*Home*» и возврата в приложение?
3. Изменится ли значение поля «*EditText*» после нажатия на кнопку «*Back*» и возврата в приложение?

ГЛАВА 3 СОЗДАНИЕ И ВЫЗОВ ACTIVITY.

На предыдущих практических работах были созданы приложения, содержащие только один экран. В качестве примера будет рассмотрено типовое почтовое приложение, содержащее следующие экраны: список аккаунтов, список писем, просмотр письма, создание письма, настройки и т.д. В данной части практического занятия будут рассмотрены вопросы создания многоэкранного приложения.

3.1 Намерение

Связующим компонентом многооконных приложений является *«Intent»* – асинхронные сообщения, позволяющие компонентам приложения запрашивать функциональность от других компонентов ОС «*Android*». «*Intent*» позволяют взаимодействовать как с другими компонентами того же приложения, так и с компонентами других приложений. Например, один «*Activity*» может вызвать внешний «*Activity*», чтобы произвести фотосъёмку (рисунок 3.1).

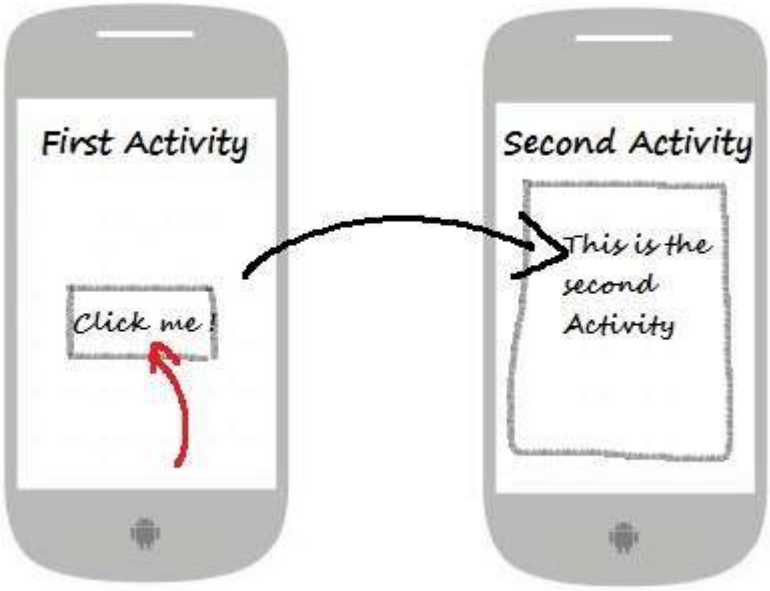


Рисунок 3.1 – Алгоритм вызова нового activity

«*Intent*» это объект класса «*android.content.Intent*», в котором возможно указать, какое «*Activity*» необходимо вызвать. Далее экземпляр объекта «*Intent*» передаётся методу «*startActivity*», который находит соответствующее «*Activity*» и отображает его. Для создания объекта возможно использовать несколько

конструкторов, но наиболее распространенный имеет два аргумента «*Intent (Context packageContext, Class class)*». Пример использования конструкции приведен ниже:

```
Intent intent = new Intent(this, SecondActivity.class);
startActivity(intent);
```

Первым параметром является объект класса «*Context*». «*Activity*» является дочерним классом «*Context*», поэтому используется ключевое слово «*this*». «*Context*» предоставляет доступ к базовым функциям приложения: ресурсы устройства, файловая система, вызов «*Activiy*» и т.д. Вторым параметром является имя вызываемого «*Activity*», который указывается в манифест-файле. В результате, после определения в коде имени вызываемой активности ОС «Android» производит поиск по манифест-файлам установленных приложений, обнаруживает соответствие и отображает соответствующую activity. Если информация об activity не добавлена в манифест-файл, то система выдаст ошибку следующего характера: «*android.content.ActivityNotFoundException: Unable to find explicit activity class*».

Вызов «*Activity*» посредством «*Intent*» с указанием имени относится к явному вызову. Т.е. с помощью класса явно указывается какую Activity требуется отобразить. Схематичный пример использования явного вызова активностей представлен на рисунке 3.2 и обычно используется в рамках одного приложения. В данном примере создаётся «*Intent*» с указанием имени класса «*Class_B*», который передается в метод «*startActivity*» в качестве аргумента. Далее производится проверка «*AndroidManifest*» на наличие «*Activity*» связанной с классом «*Class_B*» и в случае успешного обнаружения отображает его. Данный алгоритм действий происходит в пределах одного приложения.

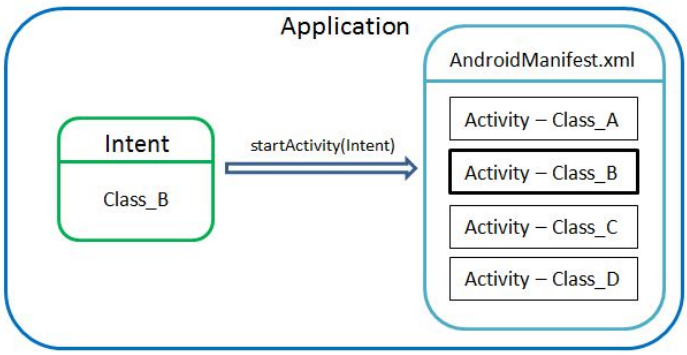


Рисунок 3.2 – Явный вызов «*Activity*»

Существует также неявный вызов «*Activity*». Он отличается тем, что при создании «*Intent*» используется не имя класса, а указываются параметры «*action*», «*data*», «*category*». Комбинация данных значений определяет цель, которую приложение хочет достичь. Например, отправка письма, открытие гиперссылки, редактирование текста, просмотр картинки, звонок по определенному номеру и т.д. В свою очередь для «*Activity*» указывается «*Intent Filter*» – набор параметров «*action*», «*data*», «*categor*y». В случае, когда параметры «*Intent*» совпадают с условиями фильтра «*Activity*» производится ее выбор. Поиск выполняется по всем «*Activity*» всех приложений в системе. В случае обнаружения нескольких вариантов система предоставляет выбор пользователю, каким именно приложением требуется воспользоваться. Порядок действий неявного вызова «Activity» схематично изображен на рисунке 3.3.

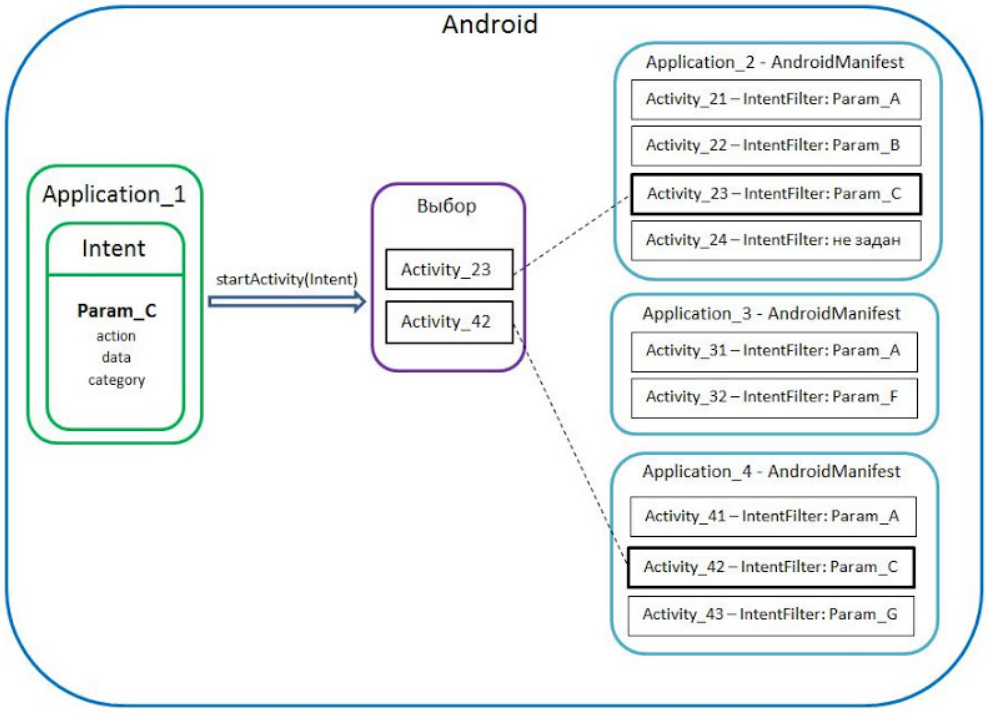


Рисунок 3.3 – Неявный вызов Activity

В «*Application_1*» создается «*Intent*» и указываются значения параметров «*action», «data», «category»*. Для удобства обозначения, получившемуся набору параметров присваивается идентификатор «*Param_C*». Далее производится вызов метода «*startActivity*» вместе с переданным «*Intent*», что приводит к созданию задачи поиска «*Activity*», соответствующей параметрам вызова (т.е. то, что определено с

помощью «*Param_C*»). В системе могут находиться разные приложения с различным количеством «*Activity*» и для некоторых может быть определен «*Intent Filter*» (наборы «*Param_A», «Param_B»* и т.д.), для некоторых нет. Метод «*startActivity*» сверяет набор параметров «*Intent*» и наборы параметров «*Intent Filter*» для каждой «*Activity»* и в случае совпадения считается подходящей. Если результатом поиска является единственная «*Activity*», то она и будет отображена пользователю. В случае обнаружения нескольких подходящих «*Activity*» пользователю выводится список, где он может сам выбрать какое приложение ему использовать. Например, если в системе установлено несколько музыкальных плееров, и запускается файл расширением «.*mp3*», система выведет список приложений с «*Activity*», которые умеют воспроизводить музыку. Приложения с «*Activity»*, которые умеют редактировать текст, показывать картинки, звонить и т.п. будут проигнорированы. Стоит отметить, что если для «*Activity*» не задан «*Intent Filter*» (*Activity_24* на рисунке 3.3), то «*Intent*» с параметрами не подойдет, и оно также будет проигнорировано.

3.2 Константы действия

Неявные намерения позволяют запрашивать анонимные компоненты приложений с помощью действий, что позволяет запускать активность, выполняющую заданное действие, не зная ничего ни о самой активности, ни о её приложении. Для определения действия требуется в намерение передать определенную константу, описывающую требование. Ниже приведены наиболее распространенные значения action:

- ACTION_CALL – инициализирует обращение по телефону;
- ACTION_MAIN – запускается как начальная активность приложения;
- ACTION_VIEW – предназначено для данных, передаваемых с помощью пути «URI» в намерении, ищется наиболее подходящий способ вывода. Выбор приложения зависит от схемы (протокола) данных. Web-адреса «*http*» будут открываться в браузере, адреса «tel» – в приложении для обработки телефонных звонков, «*geo*» – в программах типа «Google Maps», а данные о контакте — отображаться в приложении для управления контактной информацией

– ACTION_WEB_SEARCH – вызывает активность, которая ведет поиск в интернете, основываясь на тексте, переданном с помощью пути URI (как правило, при этом запускается браузер)

3.3 Константы категорий

Дополнительная информации о намерении содержится в константе «Category». В ней размещена информация о виде компонента, который должен её обработать. Класс «Intent» определяет несколько констант «Category»:

– CATEGORY_BROWSABLE – активность может быть вызвана браузером с целью отображения ссылочных данных (изображение, почтовое сообщение и т.д.);

– CATEGORY_HOME – активность отображает домашний экран/первый экран, который отображается пользователю после включения мобильного устройства и загрузки системы, или при нажатии кнопки «HOME»;

– CATEGORY_LAUNCHER – активность может быть начальной деятельностью списка приложений в группе «Application Launcher» устройства.

3.4 Задание. Вызов активности

3.4.1 Явные намерения

Создать новый модуль. В меню «*File* | *New* | *New Module* | *Phone & Tablet Module* | *Empty Views Activity*». Название проекта «*MultiActivity*».

В разметке activity_main.xml требуется добавить кнопку и реализовать обработчик нажатия на кнопку:

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onClickNewActivity"
    android:text="Start new activity!"
    app:layout_constraintBottom_toTopOf="@+id/textView"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

«*Android Studio*» при создании модуля создаёт «*MainActivity*» и размещает в манифест-файле данные о нем. Если требуется создание нового «*Activity*» вручную, то среда разработки также предоставит менеджер создания компонента, который автоматически добавит создаваемую «Activity» в манифест. Для создания

активности требуется вызвать пункт меню: «*File* | *New* | *Activity* | *Basic Activity*». На рис 3.4 представлен экран создания «*activity*».

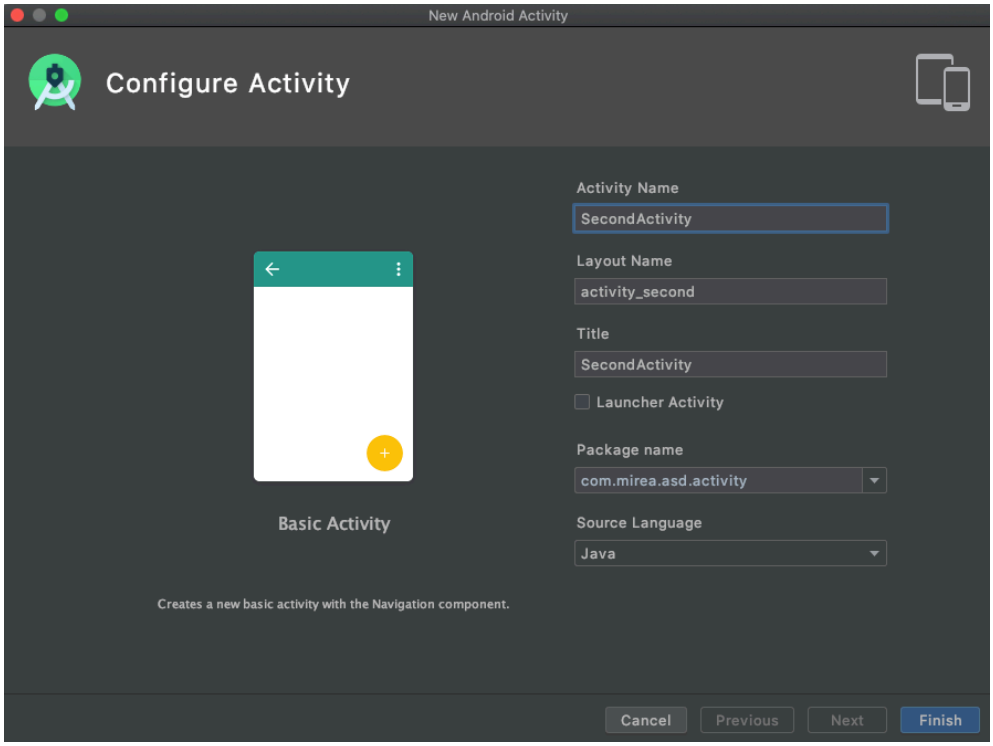


Рисунок 3.4 – Менеджер создания Activity

Создание новой активности сопровождается добавлением записи в манифест-файл о новом компоненте приложения:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.mirea.asd.multiactivity">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".SecondActivity"
            android:label="SecondActivity"
            android:theme="@style/AppTheme.NoActionBar"></activity>
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Внутри тега «*application*» располагается тег «*activity*» с атрибутом «*name=".MainActivity"*». Активность может содержать «*intent-filter*» с определенными параметрами. В активности «.*MainActivity*» определены значения «*android.intent.action.MAIN*» – устанавливает «Activity» точкой входа в приложение, «*android.intent.category.LAUNCHER*» – приложение будет отображено в общем списке приложений «*Android*». Т.е. данный манифест-файл является конфигуратором приложения. В данном файле указываются различные параметры отображения и запуска «*Activity*» или целого приложения. В случае отсутствия информации в манифесте о запускаемом «*Activity*», будет возвращена ошибка.

В файл разметки «*activity_second.xml*», расположенный в ресурсах, требуется добавить «*TextView*». Далее требуется инициализировать вызов данного «*activity*» из «*MainActivity*»:

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void onClickNewActivity(View view) {
        Intent intent = new Intent(this, SecondActivity.class);
        startActivity(intent);
    }
}
```

Ограничение доступа к компонентам

Использование фильтра «*Intent*» не является безопасным способом предотвращения запуска компонентов другими приложениями. Несмотря на то, что после применения фильтров «*Intent*» компонент будет реагировать только на неявные объекты «*Intent*» определенного вида, другое приложение теоретически имеет возможность запустить компонент вашего приложения с помощью явного объекта «*Intent*», если разработчик определит имена ваших компонентов. Если важно, чтобы только ваше собственное приложение могло запускать один из ваших компонентов, требуется установить для атрибута «*exported*» этого компонента значение «*false*».

Для передачи данных из одной активности в другую возможно использовать компонент «*Bundle*». Далее представлен пример передачи текста из первой «activity» во вторую:

```
Intent intent = new Intent(MainActivity.this, SecondActivity.class);
intent.putExtra("key", "MIREA – ФАМИЛИЯ ИМЯ ОТЧЕТСВО СТУДЕНТА");
startActivity(intent);

// У второй активности
String text = (String) getIntent().getSerializableExtra("key");
```

Требуется в первую «*activity*» добавить поле ввода и кнопку «Отправить». На второй «activity» требуется отобразить значение поля первой активности в «*TextView*».

На рисунке 3.5 представлены вызовы основных методов жизненного цикла двух активностей при переходе между друг другом.

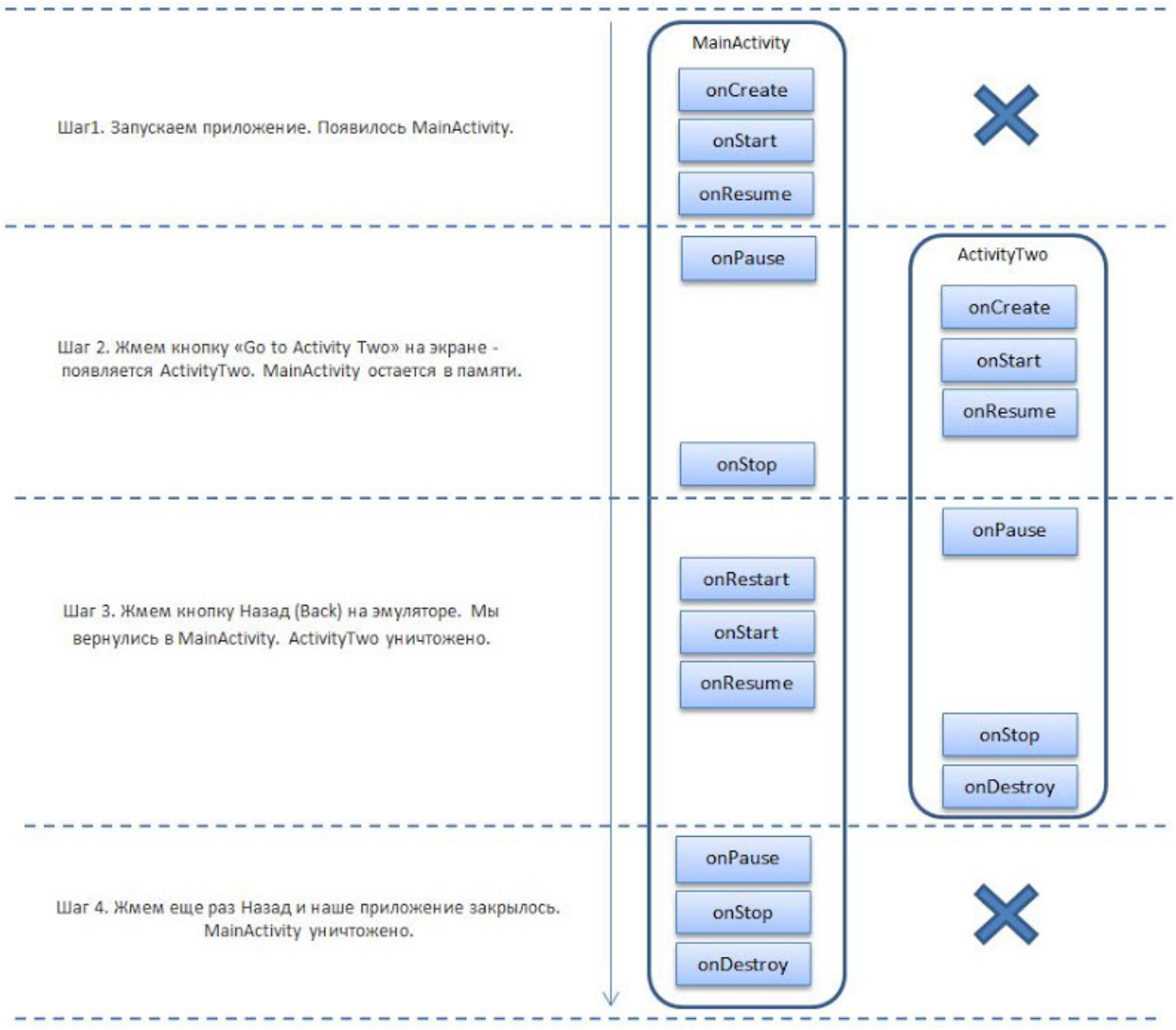


Рисунок 3.5 – Жизненные циклы при переходах между activity

Требуется переопределить основные методы жизненного цикла у обеих «*Activity*». Изучить жизненный цикл двух активностей в соответствии с рисунком 3.5

3.4.2 Неявные намерения.

Создать новый модуль. В меню «*File* | *New* | *New Module* | *Phone & Tablet Module* | *Empty Views Activity*». Название проекта «*IntentFilter*».

Требуется добавить в разметку «*activity_main.xml*» одну кнопку и обработчик нажатия для вызова веб-браузера:

```
Uri address = Uri.parse("https://www.mirea.ru/");
Intent openLinkIntent = new Intent(Intent.ACTION_VIEW, address);
startActivity(openLinkIntent);
```

В приведенном выше коде действие «*ACTION_VIEW*» означает просмотр веб-страницы. Для открытия новой активности (браузера) требуется указать web-адрес страницы. При этом исходная активность приостанавливается и переходит в фоновый режим. Нажатие кнопки «*Back*» приводит к открытию исходной активности. Стоит обратить внимание, что не указывается конкретный идентификатор активности или программа типа «*Chrome*», «*Opera*» и т.п. В каждом случае ОС «*Android*» находит соответствующую активность, чтобы ответить на намерение, инициализируя её в случае необходимости.

Добавить дополнительную кнопку для передачи ФИО студента и университета в другое приложение. Пример реализации передачи данных:

```
Intent shareIntent = new Intent(Intent.ACTION_SEND);
shareIntent.setType("text/plain");
shareIntent.putExtra(Intent.EXTRA_SUBJECT, "MIREA");
shareIntent.putExtra(Intent.EXTRA_TEXT, "ФАМИЛИЯ ИМЯ ОТЧЕСТВО");
startActivity(Intent.createChooser(shareIntent, "МОИ ФИО"));
```

ГЛАВА 4 ДИАЛОГОВЫЕ ОКНА

Диалоговые окна в ОС «Android» представляют собой различного типа активности, частично перекрывающие родительский экран. В ОС «Android» существует три вида диалоговых окон:

– всплывающие подсказки («*toasts*») приведены на рисунке 4.1. Сообщения, которые появляются на экране приложения, перекрывая его интерфейс, и через некоторое время (обычно несколько секунд) автоматически пропадают. Данный вид диалогов рекомендуется использовать для простых уведомлений, не требующих ответа пользователя, но важных для продолжения его работы.

Рисунок 4.1 - Toast

– уведомления («*notifications*») – это сообщения, отображаемые в верхней панели в области уведомлений (рисунок 4.2). Для того чтобы прочитать данное сообщение, необходимо на домашнем экране потянуть вниз верхнюю шторку. Пользователь может это сделать в любой момент времени, следовательно, уведомления стоит использовать, когда сообщение является важным, однако не требует немедленного прочтения и ответа.

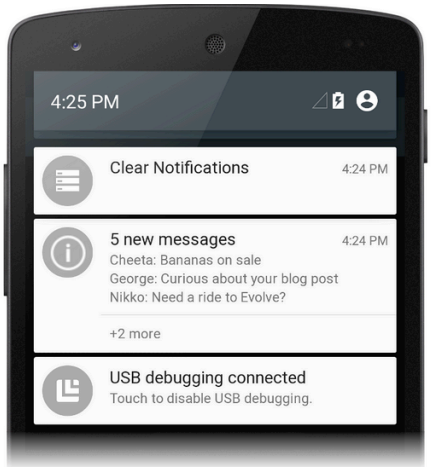


Рисунок 4.2 – Notification

– диалоговые окна, наследуемые от класса Dialog и его производных (рисунок 4.3). Диалоги данного типа не создают новых активностей и их не требуется регистрировать в файле манифеста, что существенно упрощает разработку. Однако они работают в модальном режиме и требуют немедленного ответа пользователя.

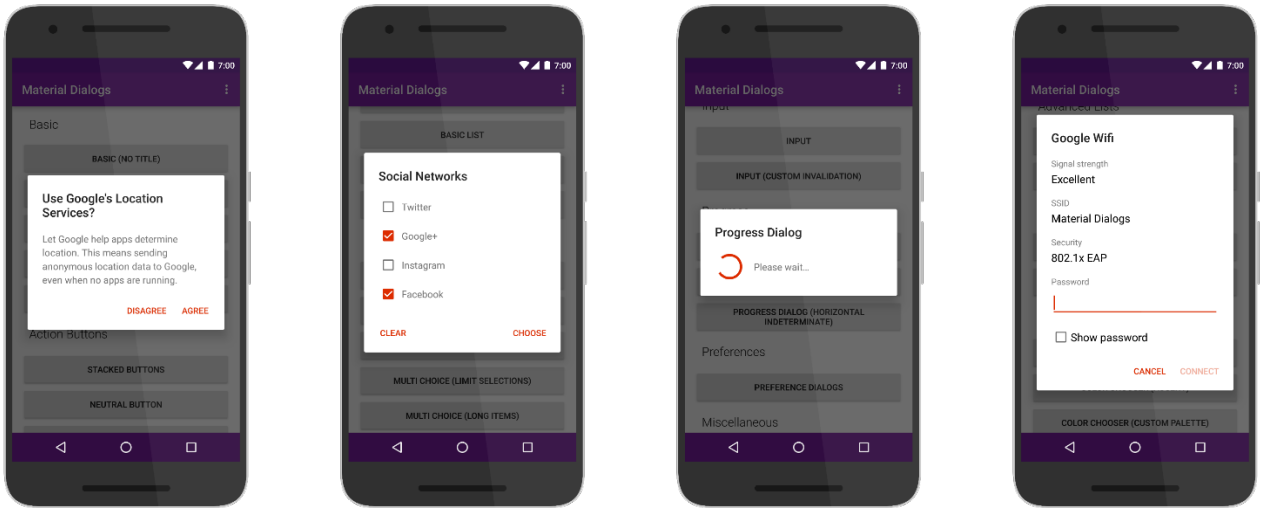


Рисунок 4.3 – Виды Dialog

4.1 Всплывающие уведомления

Всплывающее уведомление (Toast Notification) является сообщением, которое появляется на поверхности окна приложения, заполняя необходимое ему количество пространства, требуемого для сообщения. При этом текущая деятельность приложения остается работоспособной для пользователя. В течение нескольких секунд сообщение плавно закрывается. Всплывающее уведомление также может быть создано службой, работающей в фоновом режиме. Как правило, всплывающее уведомление используется для показа коротких текстовых сообщений.

Для создания всплывающего уведомления необходимо инициализировать экземпляр класса «Toast» при помощи метода «*makeText*», а затем вызвать метод *«show»* для отображения сообщения на экране:

```
Toast toast = Toast.makeText(getApplicationContext(),
        "Здравствуй MIREA!",
        Toast.LENGTH_SHORT);
toast.show();
```

Одна из реализаций метода «*makeText*» принимает в качестве входных аргументов три параметра:

– контекст приложения;

– текстовое сообщение;

– продолжительность времени показа уведомления. Возможно использовать только две константы: «LENGTH_SHORT» – отображение текстового уведомления на короткий промежуток времени (2 секунды) и «LENGTH_LONG» – отображение текстового уведомления в течение длительного периода времени (3.5 секунды).

По умолчанию стандартное всплывающее уведомление появляется в нижней части экрана. Изменить место появления уведомления возможно с помощью метода «*setGravity(int, int, int)*». Метод принимает три параметра:

– стандартная константа для размещения объекта в пределах большего контейнера (например, GRAVITY.CENTER, GRAVITY.TOP и др.);

– смещение по оси X;

– смещение по оси Y.

Создать новый модуль. В меню «*File* | *New* | *New Module* | *Phone & Tablet Module* | *Empty Views Activity*». Название модуля «*ToastApp*». Добавить поле ввода и кнопку. Требуется подсчитать количество символов в поле ввода и отобразить сообщение «*Toast*» «СТУДЕНТ № X ГРУППА X Количество символов - X».

4.2 Уведомления

Тип уведомлений *«Notifications»* – это сообщения, которые возможно отображаются за пределами приложения. На рисунке 4.4 представлен пример уведомления

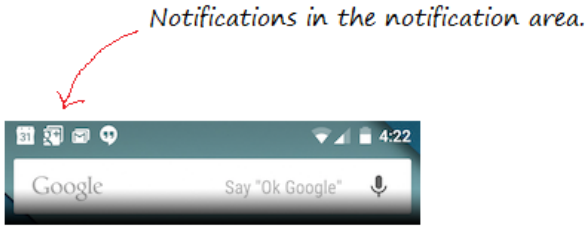


Рисунок 4.4 – Уведомление

Уведомление отображается в виде значка в области строки состояния. Для того, чтобы просмотреть детали уведомления, пользователю требуется открыть список уведомления («*Notification Drawer*»). Область уведомления и список уведомлений являются областями системного управления, которые пользователь может посмотреть в любое время. Для того чтобы создать уведомление в строке состояния используются два класса:

– «*Notification*» – определяет свойства уведомления строки состояния: значок, расширенное сообщение и дополнительные параметры настройки (звук и др.)

– «*NotificationManager*» – системный сервис «*Android*», управляющий всеми уведомлениями. Экземпляр «*NotificationManager*» создается при помощи вызова метода «*getSystemService*», а затем, когда требуется показать уведомление пользователю, вызывается метод «*notify*», либо метод *from()*.

Создать новый модуль. В меню «*File | New | New Module | Phone & Tablet Module | Empty Views Activity*». Название проекта «NotificationApp».

В файле разметки activity_main.xml разместить button и присвоить значение (рисунок 4.5)

```
android:onClick="onClickNewMessageNotification"
```

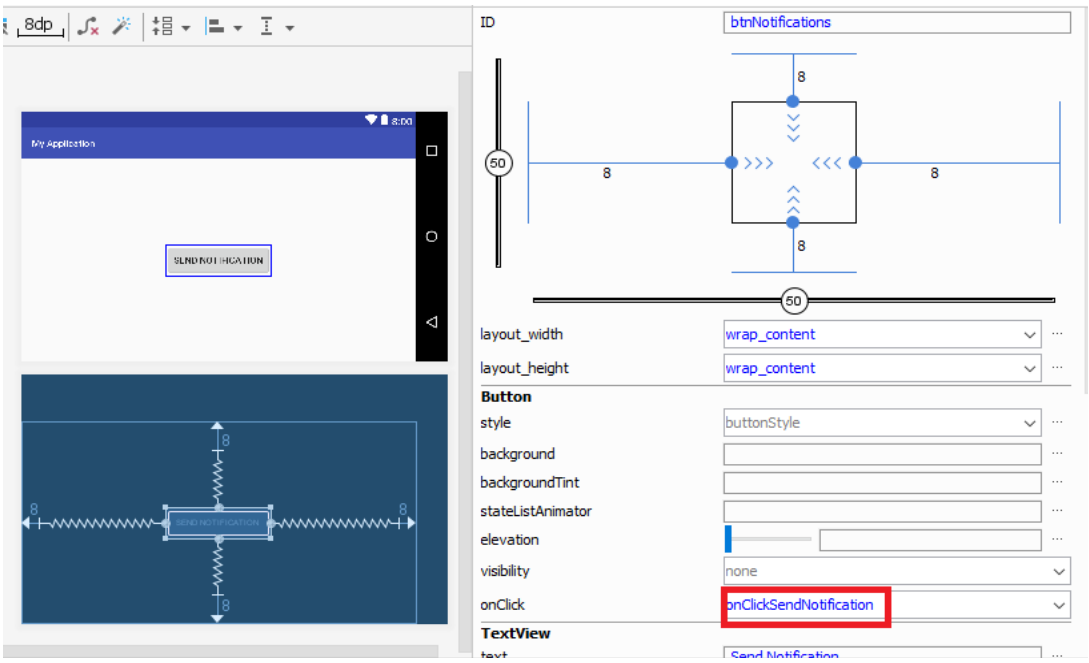


Рисунок 4.5 – Экран activity_main.xml

Заданное имя события должно использоваться произвольное, но отвечающее логическому смыслу. Далее требуется указать в классе активности созданное имя метода, который будет обрабатывать нажатие.

```
public void onClickSendNotification(View view) {

}
```

Операционная система «*Android*» устроена таким образом, что для выполнения некоторых операций или доступа к определенным ресурсам, приложение должно иметь разрешение на это. Основными двумя типами разрешений, используемых разработчиками приложений, являются «*normal*» и «*dangerous*». Начиная с ОС «Android» 13 версии требуется добавлять разрешение на показ уведомлений в манифест-файл модуля перед тегом «application»:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <uses-permission android:name="android.permission.POST_NOTIFICATIONS" />
```

В методе «*onCreate*» представлен пример проверки наличия разрешения. В первом блоке условия с помощью метода «*ContextCompat.checkSelfPermsission*» производится сравнения возвращаемого значения с константой «*PackageManager.Permission_Granted*». В случае наличия разрешения оба значения будут равны нулю, иначе произойдет переход к следующему блоку, в котором произойдет вывод окна запроса разрешений.

```
private int PermissionCode = 200;
@RequiresApi(api = Build.VERSION_CODES.TIRAMISU)
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    if (ContextCompat.checkSelfPermission(this, POST_NOTIFICATIONS) == Pack-
ageManager.PERMISSION_GRANTED) {
        Log.d(MainActivity.class.getSimpleName().toString(), "Разрешения полу-
чены");
    } else {
        Log.d(MainActivity.class.getSimpleName().toString(), "Нет разрешений!");
        ActivityCompat.requestPermissions(this, new String[]{Manifest.permis-
sion.POST_NOTIFICATIONS}, PermissionCode);
    }
}
```

После получения требуемого разрешения необходимо создать идентификатор уведомления. Он предназначен для возможности различать уведомления друг от

друга. Если имеется один идентификатор, то каждое новое уведомление затрёт предыдущее. Для идентификатора используется число.

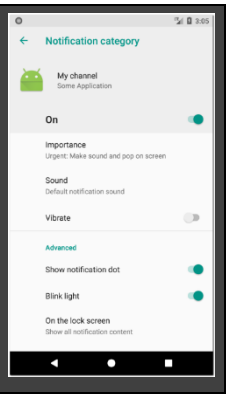
```
public class MainActivity extends AppCompatActivity {
    private static final String CHANNEL_ID = "com.mirea.asd.notification.ANDROID";
```

Далее рассматривается метод «*onClickSendNotification*» разделенный на 3 логических части.

```
public void onClickSendNotification (View view) (View view) {
        if (ActivityCompat.checkSelfPermission(this, Manifest.permis-
sion.POST_NOTIFICATIONS) != PackageManager.PERMISSION_GRANTED) {
            return;
        }

        NotificationCompat.Builder builder = new NotificationCom-
pat.Builder(this, CHANNEL_ID)
                .setContentText("Congratulation!")
                .setSmallIcon(R.drawable.baseline_1x_mobiledata_24)
                .setPriority(NotificationCompat.PRIORITY_HIGH)
                .setStyle(new NotificationCompat.BigTextStyle()
                        .bigText("Much longer text that cannot fit one
line..."))
                .setContentTitle("Mirea");

        int importance = NotificationManager.IMPORTANCE_DEFAULT;
        NotificationChannel channel = new NotificationChannel(CHANNEL_ID, "Stu-
dent FIO Notification", importance);
        channel.setDescription("MIREA Channel");

        NotificationManagerCompat notificationManager = NotificationManagerCom-
pat.from(this);
        notificationManager.createNotificationChannel(channel);
        notificationManager.notify(1, builder.build());
    }
```

В начале метода вызывается класс «*NotificationCompat.Builder*» (паттерн строитель) для создания объекта «*NotificationCompat*», в котором указывается иконка, заголовок и текст для уведомления, т.е. формируется внешний вид и поведение уведомления. Методом «*build*» создаётся готовое уведомление и выводится уведомление с помощью метода «*notify*».

В версии Android 8 появилась возможность создавать каналы для уведомлений. Для каждого приложения пользователь может настроить уведомления: важность, звук, вибрацию и прочее. Каналы актуальны только для «Android Oreo» и выше, поэтому используется проверка версии Android. Методом createNotificationChannel создается канал.



Результат выполненной работы представлен на рисунке 4.6.

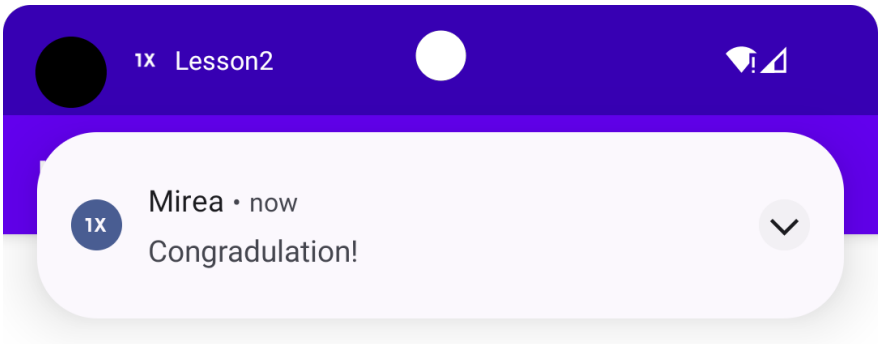


Рисунок 4.6 – Пример уведомления

4.3 Диалоговые окна

К данным типам относятся всплывающие окна, использующиеся для подтверждения каких-либо операций или ввода небольшого количества данных (рисунок 4.7).

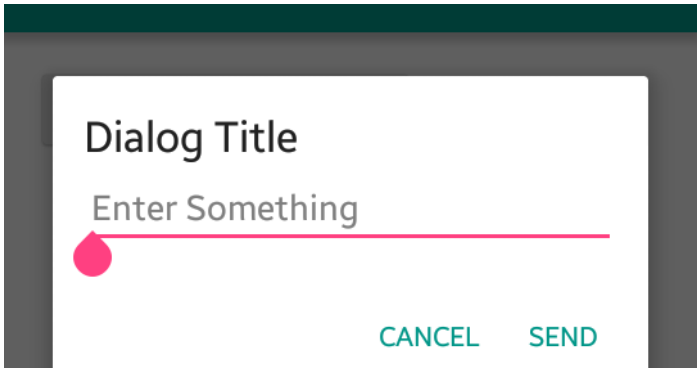


Рисунок 4.7 – Внешний вид Dialog

Диалоговое окно занимает часть экрана и обычно используется в модальном режиме. Это означает, что работа приложения приостанавливается до момента, пока пользователь не закроет диалоговое окно. При этом, возможно, потребуется ввести какие-либо данные или просто выбрать один из вариантов ответа. В «*Android 3.0*» (API 11) появилась новинка – класс «*android.app.DialogFragment*» и его аналог «*android.support.v4.app.DialogFragment*», а чуть позже и «*android.support.v7.app.AppCompatDialogFragment*» из библиотеки совместимости, позволяющие выводить диалоговое окно поверх своей активности. На текущий момент используется класс «*androidx.fragment.app.DialogFragment*» – рекомендуемый стандарт для вывода диалоговых окон в новых проектах.

Использование фрагментов для диалоговых окон в силу своей архитектуры является удобным вариантом в приложениях, которым требуется обрабатывать

поворот устройства, нажатие кнопки «Назад», масштабирование под разные экраны и т.д.

Создать новый модуль. В меню «*File | New | New Module | Phone & Tablet Module | Empty Views Activity*». Название модуля Dialog.

Далее производится переход в директорию (рисунок 4.8), в которой находится MainActivity и правой кнопкой мыши вызывается контекстное меню: File-> New-> Java Class.

Рисунок 4.8 – Вызов менеджера создания Java классов

На рисунке 4.9 приведен пример создания класса. Для того, чтобы задать родительский класс требуется в поле «Superclass» указать ключевое слово Dialog и выбрать требуемый родительский класс.

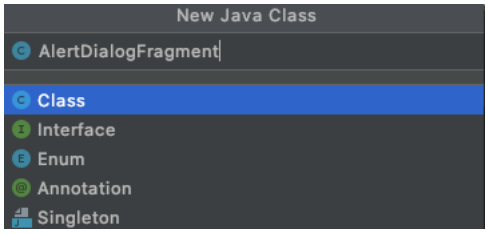


Рисунок 4.9 – Создание класса DialogFrament

```
import androidx.fragment.app.DialogFragment;

public class MyDialogFragment extends DialogFragment {

}
```

В файле «*activity_main.xml*» требуется добавить кнопку и установить значение следующих полей:

```
<Button
…
    android:layout_marginBottom="120dp"
    android:onClick="onClickShowDialog"
    android:text="Show dialog"
… />
```

Для вызова диалогового окна требуется создание экземпляра класса и вызов метода «*show*». Метод принимает два параметра: объект класса «FragmentManager», получаемый через метод «*getSuportFragmentManager*», и тег - идентификатор диалога в виде строковой константы, по которому возможно идентифицировать диалоговое окно, если их будет много в проекте.

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void onClickShowDialog(View view) {
        MyDialogFragment dialogFragment = new MyDialogFragment();
        dialogFragment.show(getSupportFragmentManager(), "mirea");
    }
}
```

После запуска проекта, скорее всего, отобразится пустой прямоугольник или квадрат, либо потемнеет экран активности. Таким образом был получен пустой фрагмент. Следующий этап – это его конструирование. В созданном классе требуется переопределить метод «*onCreateDialog*». Если используется разметка, то также используется метод «onCreateView».

Самый распространённый вариант диалогового окна – это «*AlertDialog*», являющееся расширением класса «*Dialog*», и это наиболее используемое диалоговое окно. В создаваемых диалоговых окнах возможно задавать заголовок, текстовое сообщение, кнопки (от одной до трёх), список, флажки, переключатели. Редактирование класса «*MyDialogFragment*» включает в себя создание объекта класса «*AlertDialog.Builder*» с передачей в качестве параметра ссылки на активность. Используя методы класса «*Builder*», задаётся для создаваемого диалога заголовок (метод «*setTitle*»), текстовое сообщение в теле диалога (метод «*setMessage*»), значок (метод «*setIcon*»), а также кнопки через метод с названием «*setPositiveButton*», «*setNeutralButton*» и «*setNegativeButton*».

В «*AlertDialog*» возможно добавить только по одной кнопке каждого типа: «*Positive», «Neutral» и «Negative*», т. е. максимально количество кнопок в диалоге данного типа составляет три единицы. Названия кнопок не играют роли, а только определяют порядок вывода. В разных версиях «*Android*» порядок менялся, поэтому на одних устройствах кнопка «*Positive*» может быть первой, а на других последней. Для каждой кнопки используется один из методов с префиксом «*set...Button*», которые принимают в качестве параметров надпись для кнопки и интерфейс «*DialogInterface.OnClickListener*», определяющий действие при нажатии.

```
public class MyDialogFragment extends DialogFragment {
    @NonNull
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        builder.setTitle("Здравствуй МИРЭА!")
                .setMessage("Успех близок?")
                .setIcon(R.mipmap.ic_launcher_round)
                .setPositiveButton("Иду дальше", new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        // Закрываем окно
                    }
                })
                .setNeutralButton("На паузе",
                        new DialogInterface.OnClickListener() {
                            public void onClick(DialogInterface dialog,
                                                int id) {
                            }
                        })
                .setNegativeButton("Нет",
                        new DialogInterface.OnClickListener() {
                            public void onClick(DialogInterface dialog,
                                                int id) {
                            }
                        });
        return builder.create();
    }
}
```

Результатом выполнения кода является диалоговое окно, внешний вид которой представлен на рисунке 4.10.

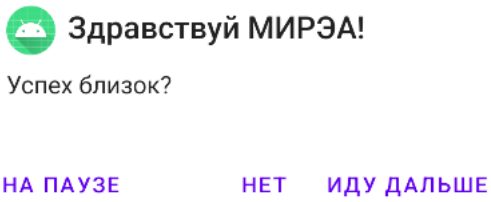


Рисунок 4.10 – Внешний вид AlertDialog

Для передачи данных в «*Activity*» из «*Dialog*» возможно использовать один из методов, для которого требуется указывать родительскую «*Activity*» и название методов в ней, которые будут отвечать за обработку нажатий кнопок диалога. Для этого требуется добавить в «*MainActivity*» 3 метода:

```
public void onOkClicked() {
    Toast.makeText(getApplicationContext(), "Вы выбрали кнопку \"Иду дальше\"!",
            Toast.LENGTH_LONG).show();
}

public void onCancelClicked() {
    Toast.makeText(getApplicationContext(), "Вы выбрали кнопку \"Нет\"!",
            Toast.LENGTH_LONG).show();
}

public void onNeutralClicked() {
    Toast.makeText(getApplicationContext(), "Вы выбрали кнопку \"На паузе\"!",
            Toast.LENGTH_LONG).show();
}
```

В классе «*MyDialogFragment*» требуется добавить внутрь методов следующие строки:

```
.setPositiveButton("Иду дальше", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        // Закрываем окно
        ((MainActivity)getActivity()).onOkClicked();
        dialog.cancel();
    }
})
```

…

```
((MainActivity)getActivity()).onNeutralClicked();
dialog.cancel();
```

…

```
((MainActivity)getActivity()).onCancelClicked();
dialog.cancel();
```

Если методы не созданы в Activity, то имена методов будут подчёркнуты красной линией и среда разработки предложит создать данные методы в классе активности (используйте комбинацию клавиш «*Alt+Enter*»)

4.4 Самостоятельная работа

Требуется изучить «*snackbar*», «*TimePickerDialog*», «*DatePickerDialog*» и «*ProgressDialog*».

Создать 3 класса и сконструировать диалоговые окна:

– *MyTimeDialogFragment*;

– *MyDateDialogFragment*;

– *MyProgressDialogFragment*.

Добавить в activity_main.xml 3 кнопки и реализовать вызов данных окон.