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: комплексной безопасности и специального приборостроения ИКБСП
(полное и краткое наименование)

Кафедра: КБ-4 «Прикладные информационные технологии»
(полное и краткое наименование кафедры, реализующей дисциплину (модуль))

Используются в данной редакции с учебного года 2021/22
(учебный год цифрами)

Проверено и согласовано «____» _____20___г. _____ _____
(подпись директора Института/Филиала с расшифровкой)

Москва 2021 г.

ОГЛАВЛЕНИЕ

1 ВВЕДЕНИЕ.

Требуется создать новый проект ru.mirea.«фамилия». practice2

Для мобильных приложений главным ограничением является размер экрана устройства. Очень часто невозможно разместить все элементы полнофункционального приложения так, чтобы их возможно было увидеть одновременно. Очевидным решением этой проблемы является разделение интерфейса на части по какому-либо принципу:

– 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 несколько Activity. Способ универсальный и подходит для любых приложений, однако прежде чем его реализовывать, необходимо продумать структуру будущего приложения. Требуется организовать переключение между различными активностями удобным для пользователя способом.

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

2 ЖИЗНЕННЫЙ ЦИКЛ ACTIVITY

Ключевым компонентом для создания визуального интерфейса в приложении Android является activity (активность). Нередко activity ассоциируется с отдельным экраном или окном приложения, а переключение между окнами производится как перемещение от одной activity к другой.

Кроме Activity в дальнейших занятиях будут изучены другие компоненты построения внешнего вида. В данном занятии упоминаются некоторые компоненты, которые будут изучены в дальнейших занятиях.

Приложение может иметь одну или несколько activity. Как правило, работа приложения начинается с класса MainActivity:

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

Все объекты activity представляют собой объекты класса *android.app.Activity*, который содержит базовую функциональность для всех activity. В примере выше MainActivity наследовалось от класса AppCompatActivity, однако сам класс AppCompatActivity, хоть и не напрямую, наследуется от базового класса Activity.

Все приложения Android имеют строго определенный системой жизненный цикл. При запуске пользователем приложения система присваивает приложению высокий приоритет. Каждое приложение запускается в виде отдельного процесса, что позволяет системе устанавливать одним процессам более высокой приоритет, в отличие от других. Благодаря этому, например, при работе с одними приложениями не блокировать входящие звонки. После прекращения работы с приложением, система освобождает все связанные ресурсы и переводит приложение в разряд низкоприоритетного и закрывает его.

Обычно приложение содержит несколько операций. Каждая операция должна разрабатываться в связи с действием определенного типа, которое пользователь может выполнять, и может запускать другие операции. Например, приложение электронной почты может содержать одну операцию для отображения списка новых

сообщений. Когда пользователь выбирает сообщение, открывается новая операция для просмотра этого сообщения.

Операция может запускать операции, существующие в других приложениях на устройстве. Например, если приложение хочет отправить сообщение электронной почты, вы можете определить намерение для выполнения действия «отправить» и включить в него некоторые данные, например, адрес электронной почты и текст сообщения. После этого открывается операция из другого приложения, которая объявила, что она обрабатывает намерения такого типа. В этом случае намерение состоит в том, чтобы отправить сообщение электронной почты, поэтому в приложении электронной почты запускается операция «составить сообщение» (если одно намерение может обрабатываться несколькими операциями, система предлагает пользователю выбрать, какую из операций использовать). После отправки сообщения электронной почты ваша операция возобновляет работу, и все выглядит так, будто операция отправки электронной почты является частью вашего приложения. Хотя операции могут быть частями разных приложений, система Android поддерживает удобство работы пользователя, сохраняя обе операции в одной задаче.

Задача — это коллекция 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 для большинства задач является главный экран устройства. Когда пользователь касается значка в средстве запуска приложений (или ярлыка на главном экране), эта задача приложения переходит на передний план. Если для приложения нет задач (приложение не использовалось в последнее время), тогда создается новая задача и открывается «основная» операция для этого приложения
в качестве корневой операции в стеке.

Когда текущая операция запускает другую, новая операция помещается в вершину стека и получает фокус. Предыдущая операция остается в стеке, но ее выполнение останавливается. Когда операция останавливается, система сохраняет

текущее состояние ее пользовательского интерфейса. Когда пользователь нажимает кнопку «Назад», текущая операция удаляется из вершины стека (операция уничтожается) и возобновляется работа предыдущей операции (восстанавливается предыдущее состояние ее 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. По существу, стек переходов назад работает по принципу «последним вошел — первым вышел». На рисунке 2.1 это поведение показано 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. При запуске новой activity она помещается поверх стека и выводится на экран устройства, пока не появится новая activity. Когда текущая activity заканчивает свою работу (например, пользователь уходит из приложения), то она удаляется из стека, и возобновляет работу та activity, которая ранее была второй в стеке.

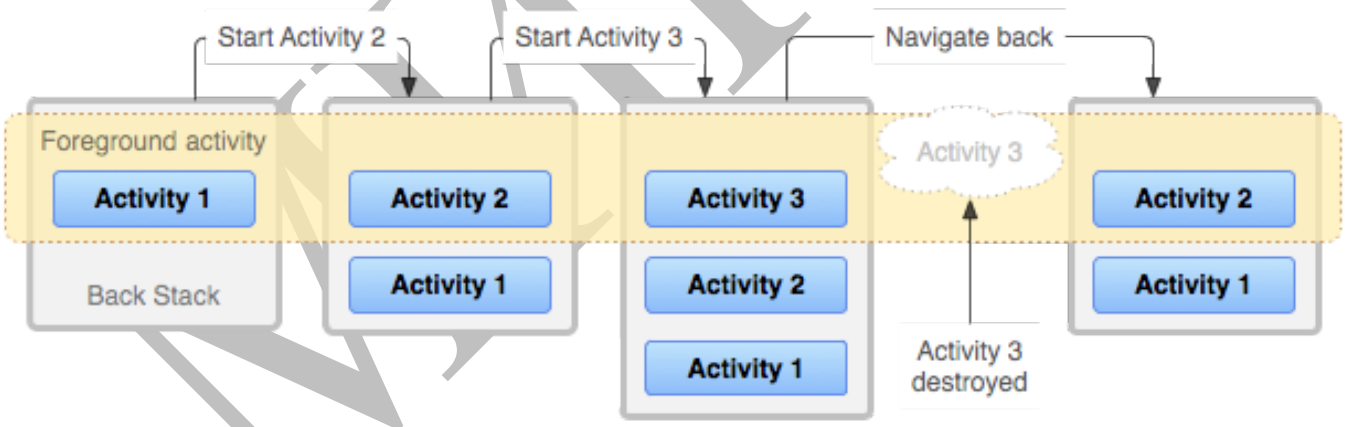


Рисунок 2.1 – Структура стека при вызове нового activity

Если пользователь продолжает нажимать кнопку «Назад», операции поочередно удаляются из стека, открывая предыдущую операцию, 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.

После запуска приложения производится, как правило, вызов 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.

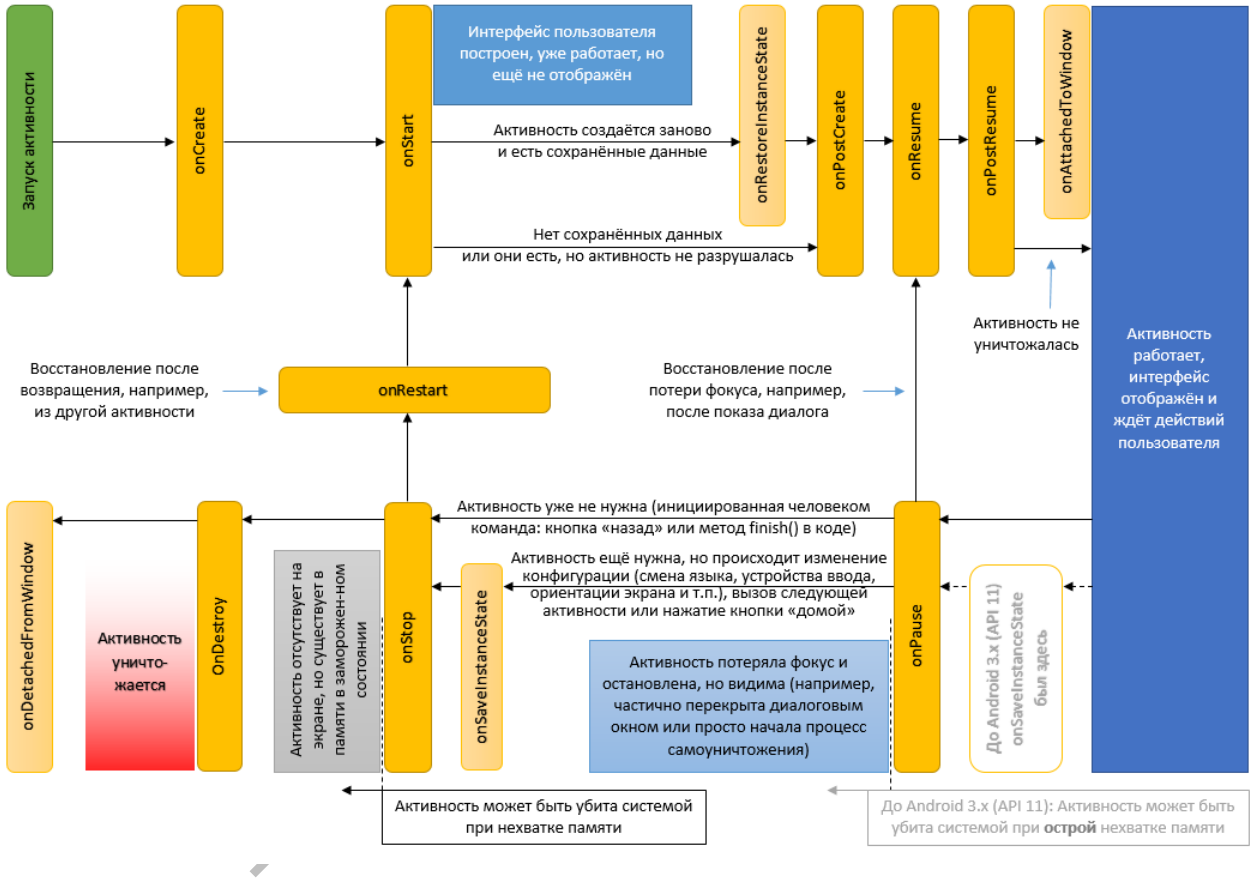


Рисунок 2.2 – Жизненный цикл Activity

2.1 onCreate

первый метод, с которого начинается выполнение activity. В этом методе activity переходит в состояние Created. Этот метод обязательно должен быть

определен в классе activity. В нем производится первоначальная настройка activity. В частности, создаются объекты визуального интерфейса. Этот метод получает объект Bundle, который содержит прежнее состояние activity, если оно было сохранено. Если activity заново создается, то данный объект имеет значение null. Если же activity уже ранее была создана, но находилась в приостановленном состоянии, то bundle содержит связанную с activity информацию.

```
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(int) для получения доступа к визуальным элементам.

2.2 onStart

В методе onStart() осуществляется подготовка к выводу activity на экран устройства. Как правило, этот метод не требует переопределения, а всю работу производит встроенный код. После завершения работы метода activity отображается на экране, вызывается метод onResume, а activity переходит в состояние Resumed.

2.3 onRestoreInstanceState

После завершения метода onStart() вызывается метод onRestoreInstanceState, который призван восстанавливать сохраненное состояние из объекта Bundle, который передается в качестве параметра. Но следует учитывать, что этот метод вызывается только тогда, когда Bundle не равен null и содержит ранее сохраненное состояние. Так, при первом запуске приложения этот объект Bundle будет иметь значение null, поэтому и метод onRestoreInstanceState не будет вызываться.

2.4 onPostCreate

Метод предназначен для системных классов для проведения окончательной инициализации. Разработчиками переопределение данного метода, как правило, не используется.

2.5 onResume

При вызове метода onResume activity переходит в состояние Resumed, а пользователь может с ней взаимодействовать. Activity остается в этом состоянии, пока она не потеряет фокус, например, вследствии переключения на другую activity или просто из-за выключения экрана устройства.

Применяется для инициализации компонентов, регистрации любых широковещательных приемников или других процессов, которые были освобождены/приостановлены в onPause() и выполнять любые другие инициализации, которые должны происходить, когда активность вновь активна.

Пытайтесь размещать относительно быстрый и легковесный код, чтобы ваше приложение оставалось отзывчивым при скрытии с экрана или выходе на передний план. Не требуется перезагружать состояние пользовательского интерфейса внутри данного метода, так как эти функции возложены на обработчики onCreate() и onRestoreInstanceState.

Например, после метода onPause(), в котором приостанавливается работа камеры (см. ниже) снова производится запуск камеры:

```
@Override
public void onResume() {
   super.onResume();

   // Get the Camera instance as the activity achieves full user focus
   if (mCamera == null) {
      initializeCamera(); // Local method to handle camera init
   }
}
```

2.6 onPostResume

Метод предназначен для системных классов для проведения окончательной инициализации после выполнения кода метода onResume. Разработчиками переопределение данного метода, как правило, не используется.

2.7 onAttachedToWindow

Метод вызывается, когда главное окно активности подключается к оконному менеджеру.

2.8 onPause

Если пользователь решит перейти к другую activity, то система вызывает метод onPause. В этом методе возможно освобождать используемые ресурсы, приостанавливать процессы, например, воспроизведение аудио, анимации, останавливать работу камеры (если она используется) и т.д., чтобы они меньше сказывались на производительности системы. Но надо учитывать, что на работу данного метода отводится очень мало времени, поэтому не стоит здесь сохранять какие-то данные, особенно если при этом требуется обращение к сети, например, отправка данных по интернету, или обращение к базе данных.

```
@Override
public void onPause() {
    super.onPause();

    // Release the Camera because we don't need it when paused
    // and other activities might need to use it.
    if (mCamera != null) {
        mCamera.release()
        mCamera = null;
    }
}
```

После выполнения данного метода activity становится невидимой, не отображается на экране, но она все еще активна. И если пользователь решит вернуться к этой activity, то система вызовет снова метод onResume, и activity снова появится на экране. При нажатии пользователем на кнопку Back (Назад) у activity вызывается метод onStop. На более свежих версиях жизнь активности гарантируется до завершения выполнения метода onStop, а значит, сохранение можно реализовать и там. Тем не менее, общая рекомендация — сохранять данные именно в onPause. Также здесь останавливают анимацию и другие процессы, нагружающие процессор.

2.9 onSaveInstanceState

Метод onSaveInstanceState вызывается после метода onPause(), но до вызова onStop(). В onSaveInstanceState производится сохранение состояния приложения

в передаваемый в качестве параметра объект Bundle. В объект Bundle возможно записать 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(), которые вызываются после onStart(), чтобы один из них или они оба могли установить активность в предыдущее состояние. Прежде чем передавать изменённый параметр Bundle в обработчик родительского класса, производится сохранение значения с помощью методов putXXX() и восстановление с помощью getXXX(). Используется обработчик onSaveInstanceState() для сохранения состояния интерфейса (например, состояния флажков, текущего выделенного элемента или введенных, но не сохранённых данных), чтобы объект Activity при следующем входе в активное состояние мог вывести на экран тот же UI. Рассчитывайте, что перед завершением работы процесса во время активного состояния будут вызваны обработчики onSaveInstanceState и onPause.

В отличие от базовых методов, методы onSaveInstanceState() и onRestoreInstanceState() не относятся к методам жизненного цикла активности. Система будет вызывать их не во всех случаях. Например, 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). В этом случае нет никаких причин для сохранения состояния активности.

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);
}
```

2.10 onStop

В этом методе activity переходит в состояние Stopped. В методе onStop следует освобождать используемые ресурсы, которые не требуются пользователю, когда он не взаимодействует с activity. Здесь также возможно сохранять данные, например, в базу данных. При этом во время состояния Stopped activity остается в памяти устройства, сохраняется состояние всех элементов интерфейса. К примеру, если в текстовое поле EditText был введен какой-то текст, то после возобновления работы activity и перехода ее в состояние Resumed мы вновь увидим в текстовом поле ранее введенный текст.

Если после вызова метода onStop пользователь решит вернуться к прежней activity, тогда система вызовет метод onRestart. Если же activity вовсе завершила свою работу, например, из-за закрытия приложения, то вызывается метод onDestroy().

2.11 onDestroy

Завершается работа активности вызовом метода onDestroy, который возникает либо, если система решит убить activity, либо при вызове метода finish().
Также следует отметить, что при изменении ориентации экрана система завершает activity и затем создает ее заново, вызывая метод onCreate.

2.12 onDetachedFromWindow

Метод вызывается, когда главное окно активности отключается от оконного менеджера.

2.13 onRestart

Если окно возвращается в приоритетный режим после вызова onStop(), то в этом случае вызывается метод onRestart(). Т.е. вызывается после того, как активность была остановлена и снова была запущена пользователем. Всегда сопровождается вызовом метода onStart().

onRestart предшествует вызовам метода onStart() (кроме самого первого). Используется для специальных действий, которые должны выполняться только при повторном запуске активности в рамках «полноценного» состояния.

Механизм организации Activity в Android очень схож по реализации с навигацией в браузере. Вы находитесь в одной вкладке(Task) и открываете страницы (Activity) переходя по ссылкам (Intent). В любой момент можете вернуться на предыдущую страницу, нажав кнопку Назад. Но кнопка Вперед отсутствует, т.к. страница, на которой была нажата кнопка Назад, стирается из памяти. И надо снова нажимать ссылку, если хотим попасть на нее. Если вам надо открыть что-то новое, вы создаете новую вкладку и теперь уже в ней открываете страницы, переходите по ссылкам, возвращаетесь назад. В итоге у вас есть несколько вкладок. Большинство из них на заднем фоне, а одна (активная, с которой сейчас работаете) – на переднем.

В итоге список аналогий браузера и Android таков:

- браузер – Android;
- вкладка с историей посещений – Task;
- страница – Activity;
- ссылка – Intent.

3 LOGCAT

В Android SDK входит набор инструментов, предназначенных для отладки. Важным инструментом при выполнении отладки является LogCat, отображающий сообщения логов (журнал логов), рассылаемые при помощи различных методов.

Часто программисту требуется вывести промежуточные результаты, чтобы понять, почему программа не работает.

Класс android.util.Log позволяет разбивать сообщения по категориям в зависимости от важности. Для разбивки по категориям используются специальные методы, которые легко запомнить по первым буквам, указывающие на категорию:

- Log.e() - ошибки (error)
- Log.w() - предупреждения (warning)
- Log.i() - информация (info)
- Log.d() - отладка (degub)
- Log.v() - подробности (verbose)
- Log.wtf() - серьезная ошибка! (What a Terrible Failure!, работает начиная с Android 2.2)

Далее приведён пример использования данной утилитой:

```
Log.d(TAG, "Код выполняется тут!");
```

В первом параметре метода используется строка, называемая тегом. Обычно принято объявлять глобальную статическую строковую переменную TAG в начале класса:

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

Разработчик имеет возможность увидеть данное сообщение через программу LogCat (рис.3.1), доступный через ADB или DDMS.

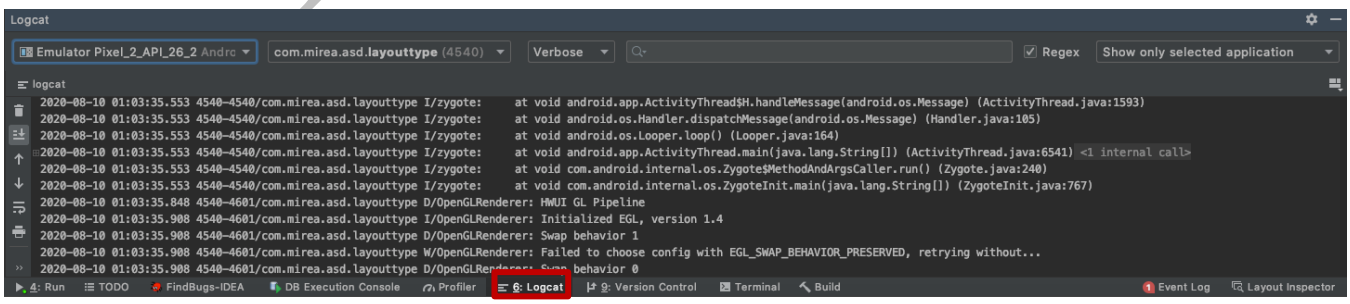


Рисунок 3.1 – Программа LogCat в Android Studio

В LogCat возможно отфильтровать сообщение по заданному тегу, чтобы видеть на экране только определённые сообщения. Для этого требуется выбрать нужный тип тега из выпадающего списка Log Level.

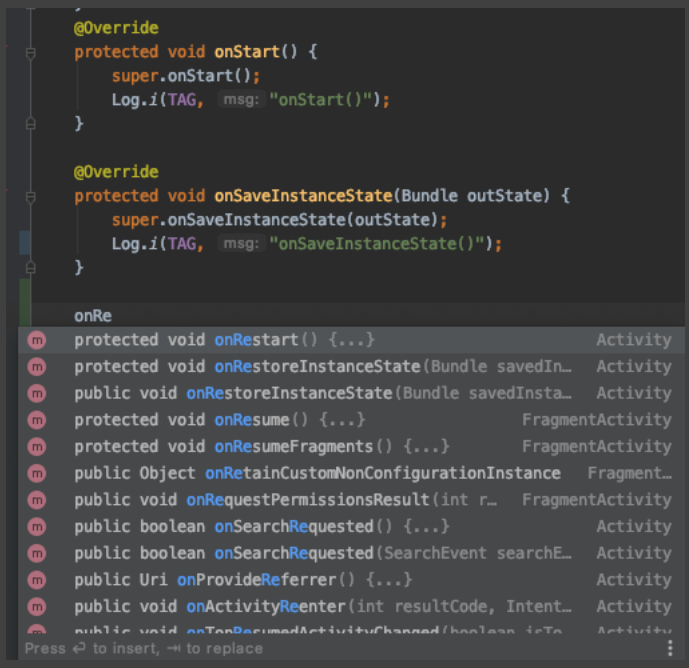
LogCat также возможно запустить из командной строки:

```
#adb logcat
```

Параметры командной строки требуется самостоятельно изучить в документации.

4 ЗАДАНИЕ

Создайте модуль/переименуйте текущий. Название *LifecycleActivity*. В созданном классе изначально присутствует только метод onCreate. Требуется переопределить все методы жизненного цикла родительского класса. Для создания метода возможно произвести ввод названия метода, среда разработки отобразит всплывающее меню с доступными методами.



Требуется добавите в разметку *activity_main.xml* поле для текстового ввода EditText. Для отображения состояния activity требуется использовать Log**.** Для каждого метода жизненного цикла устанавливается определенное сообщение. Все методы жизненного цикла, которые переопределяются в классе выделяются аннотацией **@Override.** Пример:

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

Переопределение метода (англ. Method overriding) - позволяет заменять реализацию метода родительского класса в дочернем.

Осуществите запуск проекта и следите за сообщениями в окне logcat. Они будут всплывать в определенной последовательности, давая представление о жизненном цикле приложения.

Обратите внимание на следующий момент: когда приложение запущено измените текст в EditText. Затем нажмите кнопку Home (не Back!), чтобы попасть на Домашний экран. После чего снова запустите ваше приложение. Вы увидите, что приложение не вызывает метод onCreate(), а текст в EditText будет свидетельствовать, что приложение не было закрыто, а только свёрнуто. Это очень важный момент. Понимание данных вещей поможет правильно выстраивать логику приложения.

5 СОЗДАНИЕ И ВЫЗОВ ACTIVITY.

На предыдущих практических работах были созданы приложения, содержащие только один экран (Activity). Если рассмотреть, например, почтовое приложение, то в нем есть следующие экраны: список аккаунтов, список писем, просмотр письма, создание письма, настройки и т.д. В данной части работы будут рассмотрены вопросы создания многоэкранного приложения.

Intent – это асинхронные сообщения, позволяющие компонентам приложения запрашивать функциональность от других компонентов Android. Intents позволяют взаимодействовать с другими компонента из тех же приложений, так же как и с компонентами созданные другими приложениями. Например, один Activity может вызвать внешний Activity, чтобы произвести фотосъёмку.

Таким образом возможно реализовать задуманное на рисунке 3.1:

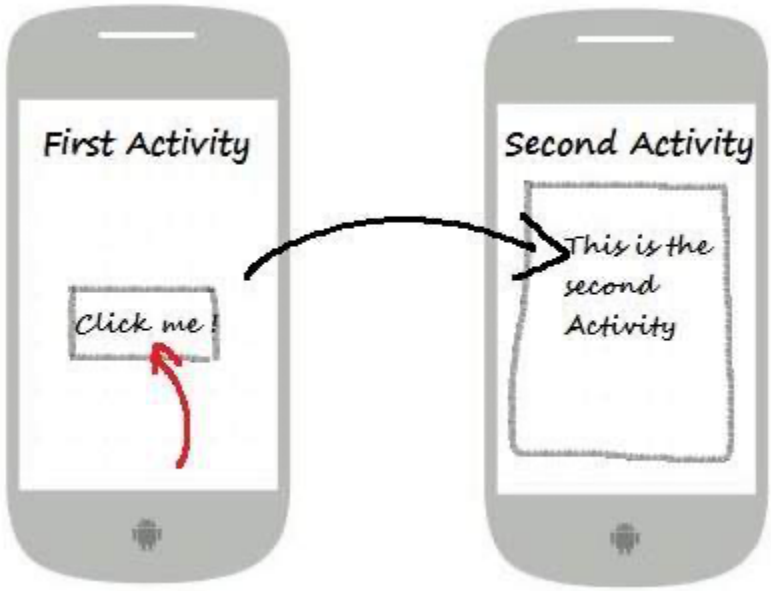


Рисунок 5.1 – Алгоритм вызова нового activity

Intent это объект класса android.content.Intent. В данном случае Intent – это объект, в котором указывается, какое Activity необходимо вызвать. После чего данный Intent-объект передаётся методу startActivity, который находит соответствующее Activity и показывает его. При создании Intent используется конструктор Intent (Context packageContext, Class class) с двумя параметрами.

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

Первый параметр – это Context. Activity является подклассом Context, поэтому используется – this. Context – это объект, который предоставляет доступ к базовым функциям приложения таким как: доступ к ресурсам, к файловой системе, вызов Activiy и т.д.

Второй параметр – имя класса. При создании записи Activity в манифест-файле указывается имя класса. И теперь если будет указан тот же класс в Intent – то система, просмотрев манифест-файл обнаружит соответствие и отобразит соответствующий Activity. Если информация об activity не добавлена в манифест-файл, то система выдаст ошибку следующего характера:

```
ERROR/AndroidRuntime(367): android.content.ActivityNotFoundException: Unable to find explicit activity class
```

Вызов Activity с помощью такого Intent – это явный вызов. Т.е. с помощью класса явно указывается какое Activity требуется отобразить. Это обычно используется внутри одного приложения. Схематично это изображено на рисунке 5.2. В данном примере создаётся Intent, в качестве параметра передается класс Class_B. Далее вызывается метод startActivity с созданным Intent в качестве параметра. Метод проверяет AndroidManifest на наличие Activity связанной с классом Class_B и если находит, то отображает. Все это в пределах одного приложения.

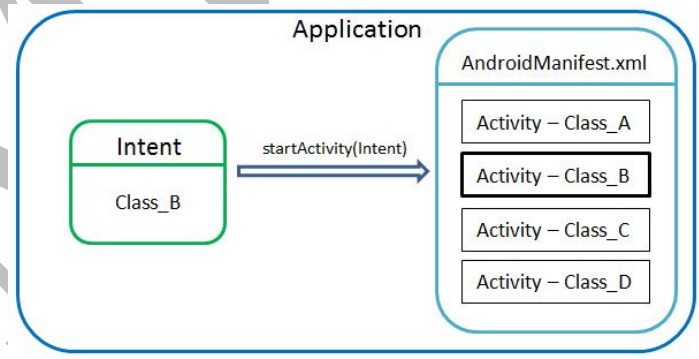


Рисунок 5.2 – Явный вызов Activity

Существует также неявный вызов Activity. Он отличается тем, что при создании Intent используется не класс, а заполняются параметры action, data, category определенными значениями. Комбинация этих значений определяют 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 - это набор тех же параметров: action, data, category (но значения уже свои - зависят от того, что умеет делать Activity). В случае, когда параметры Intent совпадают с условиями этого фильтра Activity вызывается. Поиск выполняется по всем Activity всех приложений в системе. Если находится несколько, то система предоставляет выбор пользователю, какой именно программой требуется воспользоваться. Схематично это изображено на рисунке 5.3.

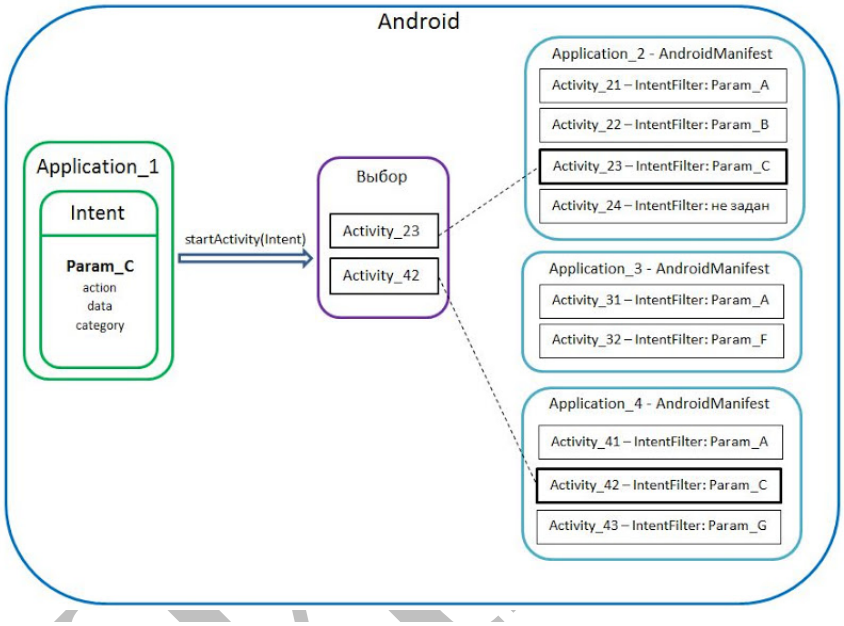


Рисунок 5.3 – Неявный вызов Activity

В Application_1 создается Intent, заполняются параметры *action, data, category*. Для удобства обозначения, получившемуся набору параметров присваивается идентификатор Param_C. С помощью startActivity данный Intent отправляется на поиски подходящей Activity, которая сможет выполнить то, что требуется системе (т.е. то, что определено с помощью Param_C). В системе существуют разные приложения, и в каждом из них несколько Activity. Для некоторых Activity 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. Если наборы совпадают (Param_C для обоих), то Activity считается подходящей.

Если результатом поиска является одна Activity – она и отображается. Если же нашлось несколько подходящих Activity, то пользователю выводится список, где он может сам выбрать какое приложение ему использовать.

Например, если в системе установлено несколько музыкальных плееров, и запускается mp3, то система выведет список Activity, которые умеют играть музыку и попросит выбрать, какое из них использовать. Activity, которые умеют редактировать текст, показывать картинки, звонить и т.п. будут проигнорированы.

Если для Activity не задан Intent Filter (Activity_24 на схеме), то Intent с параметрами не подойдет, и оно также будет проигнорировано. Если проводить аналогии - возможно сравнить Intent с ключом, а Intent Filter с замком, за которым находится Activity.

5.1 Константы действия

Основные константы будут рассмотрены в 3 практическом занятии. Ниже приведены наиболее распространенные значения action:

- ACTION_CALL — инициализирует обращение по телефону;
- ACTION_MAIN – запускается как начальная активность задания
- ACTION_VIEW — наиболее распространённое общее действие. Для данных, передаваемых с помощью пути URI в намерении, ищется наиболее подходящий способ вывода. Выбор приложения зависит от схемы (протокола) данных. Стандартные адреса http: будут открываться в браузере, адреса tel: — в приложении для дозвона, geo: — в программе Google Maps, а данные о контакте — отображатся в приложении для управления контактной информацией
- ACTION_WEB_SEARCH — Открывает активность, которая ведет поиск в интернете, основываясь на тексте, переданном с помощью пути URI (как правило, при этом запускается браузер)

5.2 Константы категорий

- CATEGORY_BROWSABLE — активность может быть безопасно вызвана браузером, чтобы отобразить ссылочные данные, например, изображение или почтовое сообщение

– CATEGORY_HOME — активность отображает Home Screen, первый экран, который пользователь видит после включения устройства и загрузки системы, или когда нажимает клавишу HOME

– CATEGORY_LAUNCHER — активность может быть начальной деятельностью задания из списка приложений в группе Application Launcher устройства

5.3 Практикум Activity

Создать новый модуль. В меню File> New> New Module> Phone & Tablet Module> Empty Activity. Название проекта MultiActivity.

В разметке activity_main.xml требуется добавить кнопку и установить:

```
<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" />
```

Следующим этапом является создание второй Activity: File> New> Activity> Basic Activity. На рис 5.4 приведен экран создания activity/

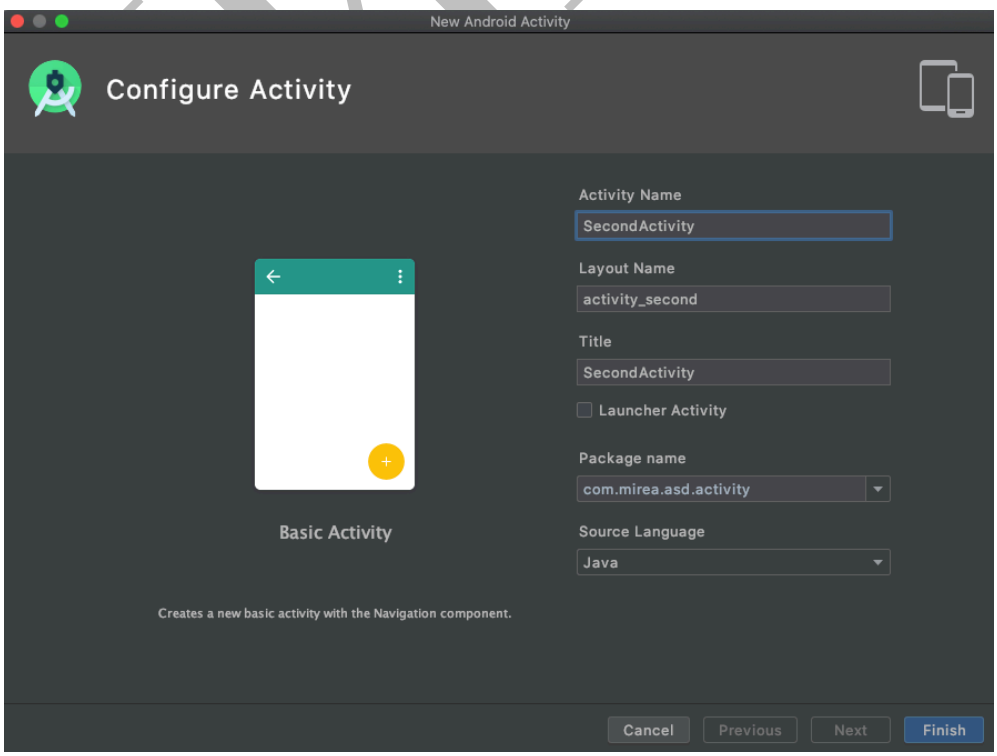


Рисунок 5.4 – Менеджер создания Activity

Следует убедиться в том, что в манифест-файле появилась запись о новой activity:

```xml
<?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* расположен тег *activity* с атрибутом *name=".MainActivity"*. В activity находится тег *intent-filter* с определенными параметрами. Значение *android.intent.action.MAIN* показывает системе, что Activity является основной и будет первой отображаться при запуске приложения, *android.intent.category.LAUNCHER* означает, что приложение будет отображено в общем списке приложений Android.

Т.е. данный манифест-файл – это конфигуратор приложение. В данном файле указываются различные параметры отображения и запуска Activity или целого приложения. Если в этом файле не будет информации об Activity, которое требуется запустить в приложении, то будет возвращена ошибка.

Android Studio при создании модуля создаёт MainActivity и размещает в манифест данные о нем. Если требуется создать новое Activity вручную, то студия также предоставит менеджера, который автоматически добавит создаваемое Activity в манифест.

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".

5.4 Задание. Явные намерения.

Использовать проект из предыдущего практикума. Разобраться с состояниями при переходе от одного activity к другому и назад. Передачу текста из первого activity для установки значения TextView во втором activity осуществить с помощью компонента Bundle.

```
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 изучить рисунок 5.5. Переопределить все методы жизненного цикла у обоих Activity. Где размещено Activity, пока его не видно? И откуда оно извлекается при нажатии кнопки назад.

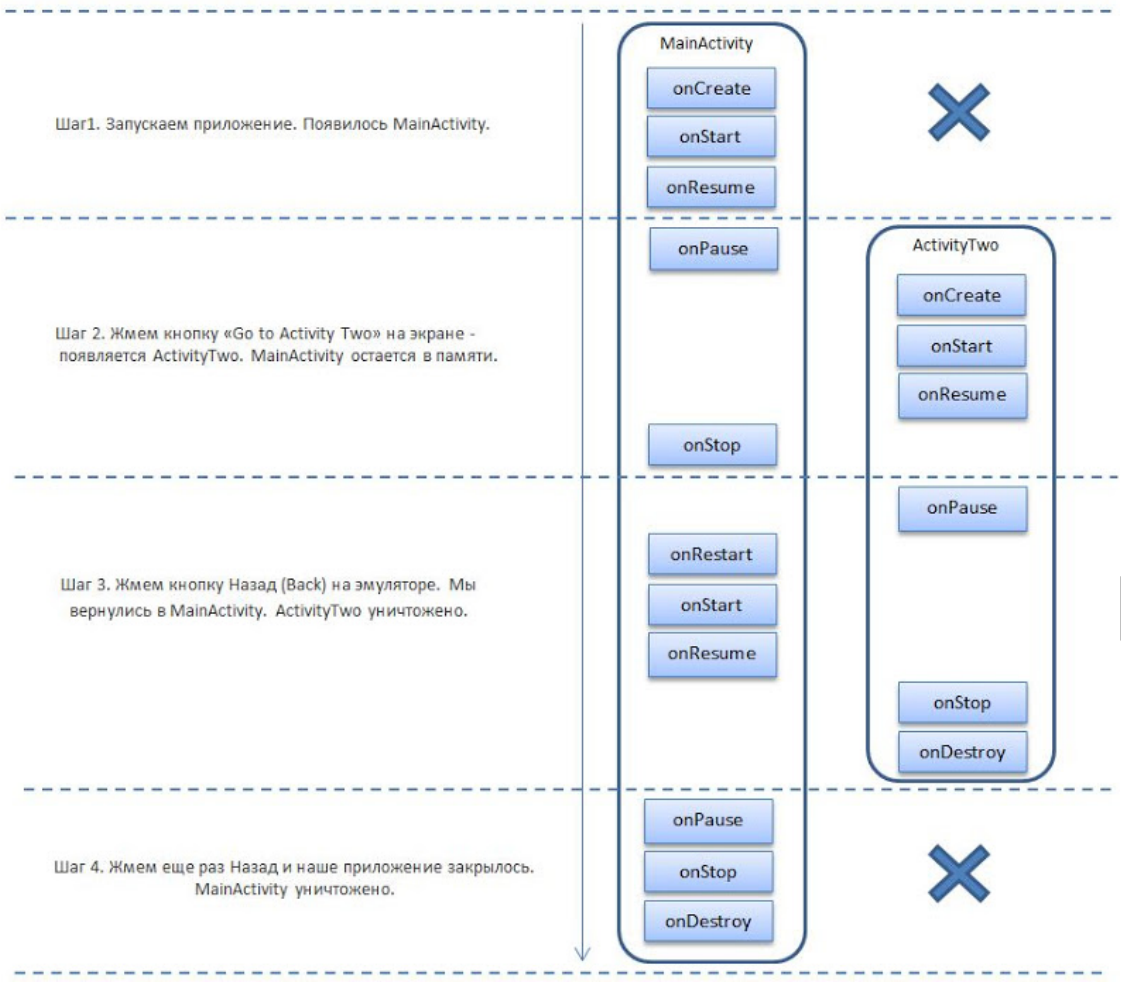


Рисунок 5.5 – Жизненные циклы при переходах между activity

5.5 Задание. Неявные намерения.

Создать новый модуль. В меню File> New> New Module> Phone & Tablet Module> Empty Activity. Название проекта IntentFilter.

В разметке activity_main.xml требуется добавить одну кнопку для вызова веб-браузера:

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

if (openLinkIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(openLinkIntent);
} else {
    Log.d("Intent", "Не получается обработать намерение!");
}
```

В данном случае действие ACTION_VIEW означает просмотр веб-страницы. Указываются нужные данные (адрес), и происходит запуск новой активности (браузера). При этом исходная активность приостанавливается и переходит в фоновый режим. Когда пользователь нажимает на кнопку Back, то он возвращается к исходной активности. Требуется обратить внимание, что нигде не указываем конкретную программу-браузер типа Chrome, Opera и т.п.

В каждом случае Android находит соответствующую активность, чтобы ответить на намерение, инициализируя её в случае необходимости.

Добавить дополнительные кнопки. Передать сообщение в другое приложение.

```
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"));
```

6 ДИАЛОГОВЫЕ ОКНА

В ОС Android существует три вида диалоговых окон:

– всплывающие подсказки (toasts) приведены на рисунок 6.1. Сообщения, которые появляются на экране приложения, перекрывая его интерфейс, и через некоторое время (обычно несколько секунд) автоматически пропадают.

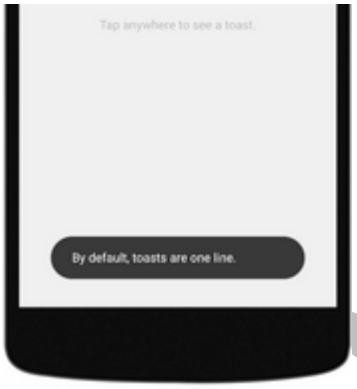


Рисунок 6.1 - Toast

Данный вид диалогов рекомендуется использовать для простых уведомлений, не требующих ответа пользователя, но важных для продолжения его работы.

– уведомления (notifications) – это сообщения, отображаемые в верхней панели в области уведомлений (рисунок 6.2). 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.

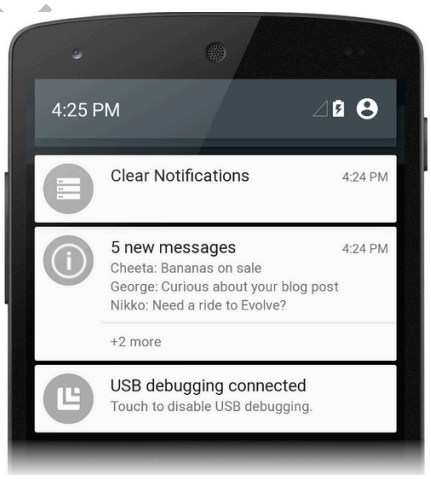


Рисунок 6.2 – Notification

– диалоговые окна, наследуемые от класса Dialog и его производных (рисунок 6.3). 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.

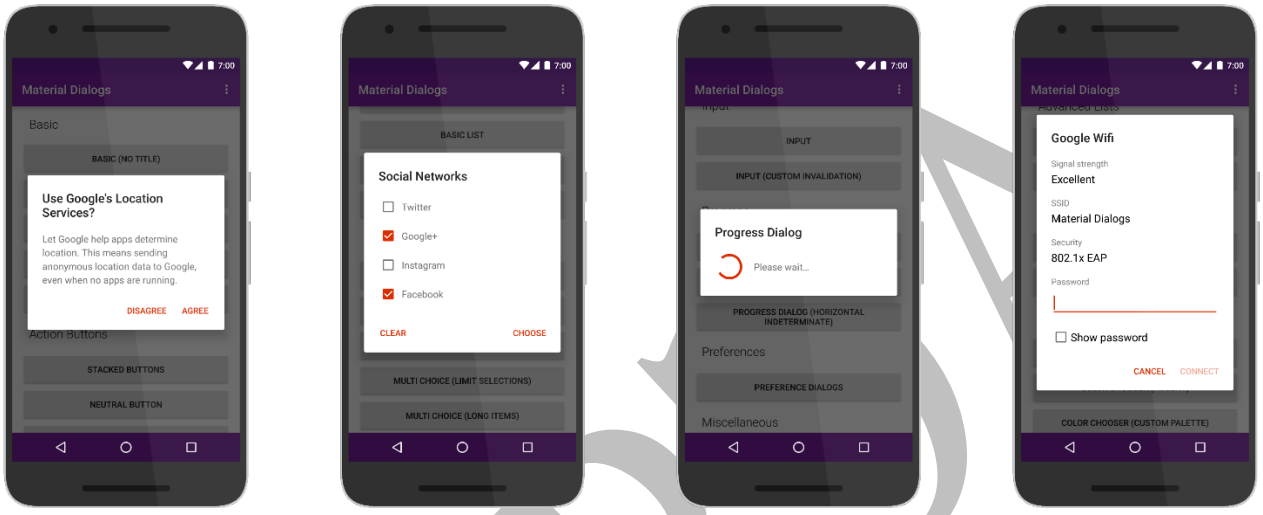


Рисунок 6.3 – Виды Dialog

6.1 Toast - всплывающие сообщения

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 объект Toast при помощи метода Toast.makeText(), а затем вызвать метод show() для отображения сообщения на экране:

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

У метода makeText() существует 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.

6.2 Задание:

Создать новый модуль. В меню File> New> New Module> Phone & Tablet Module> Empty Activity. Название проекта ToastApp. Требуется создать Toast посередине экрана с изображением.

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toast toast = Toast.makeText(getApplicationContext(),
                "Здравствуй MIREA! ФИО",
                Toast.LENGTH_SHORT);
        toast.setGravity(Gravity.BOTTOM, 0, 0);
        LinearLayout toastContainer = (LinearLayout) toast.getView();
        ImageView catImageView = new ImageView(getApplicationContext());
        catImageView.setImageResource(R.drawable.ic_launcher_background);
        toastContainer.addView(catImageView, 0);
        toast.show();
    }
}
```

6.3 Уведомления

Notifications – это сообщение, которое возможно показать пользователю за пределами приложения.

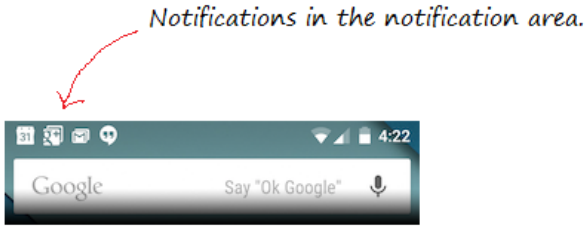


Рисунок 6.4 - Уведомление

Уведомление отображается в виде значка в области уведомления (рис.6.4). 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. Рассмотрим основы - отображение/ обновление/удаление уведомления и обработка нажатия на него.

Чтобы создать уведомление в строке состояния, необходимо использовать два класса:

- Notification — определяем свойства уведомления строки состояния: значок, расширенное сообщение и дополнительные параметры настройки (звук и др.)
- NotificationManager — системный сервис Android, который управляет всеми уведомлениями. Экземпляр NotificationManager создается при помощи вызова метода getSystemService(), а затем, когда требуется показать уведомление пользователю, вызывается метод notify(). Недавно появился более простой способ через метод **from()**.

6.4 Задание:

Создать новый модуль. В меню File> New> New Module> Phone & Tablet Module> Empty Activity. Название проекта NotificationApp.

В файле разметки activity_main.xml разместить button и присвоить значение (рисунок 6.5)

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

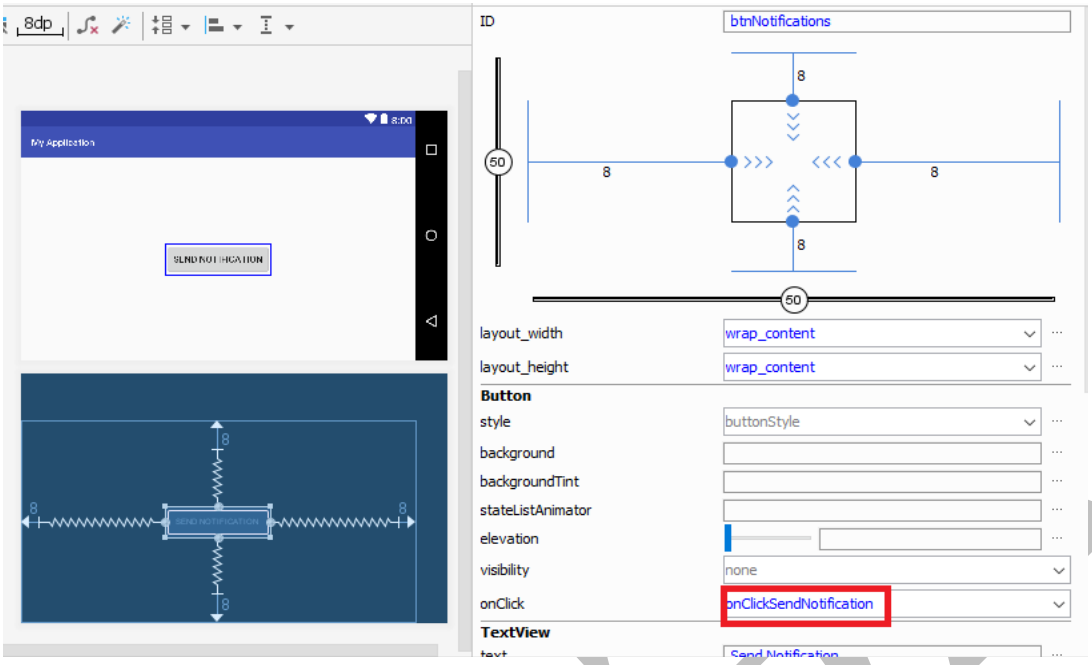


Рисунок 6.5 – Экран activity_main.xml

Заданное имя события должно использоваться произвольное, но отвечающее логическому смыслу. Далее требуется указать в классе активности созданное имя метода, который будет обрабатывать нажатие. Метод должен быть открытым (public) и с одним параметром, использующим объект View.

```
public void onClickSendNotification(View view) {

}
```

При подобном подходе не требуется объявлять кнопку через конструкцию findViewById(R.id.button1). Данный способ применим не только к кнопке, но и к другим элементам и позволяет сократить количество строк кода.

Для начала требуется 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";
    private int IDENTIFICATE_MSG = 0;
```

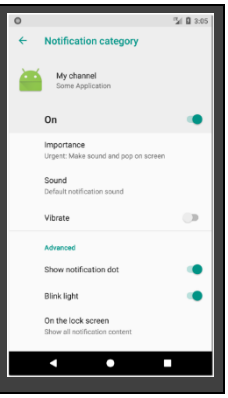
Далее рассматривается метод onClickSendNotification разделенный на 3 логических части.

```
public void onClickSendNotification(View view) {
    NotificationManager notificationManager =
            (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
    // Create PendingIntent
    Intent resultIntent = new Intent(this, MainActivity.class);
    PendingIntent resultPendingIntent = PendingIntent.getActivity(this, 0, resultIntent,
            PendingIntent.FLAG_UPDATE_CURRENT);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        NotificationChannel notificationChannel = new NotificationChannel(CHANNEL_ID, "My
Notifications", NotificationManager.IMPORTANCE_DEFAULT);
        // Configure the notification channel.
        notificationChannel.setDescription("Channel description");
        notificationChannel.enableLights(true);
        notificationChannel.setLightColor(Color.RED);
        notificationChannel.setVibrationPattern(new long[]{0, 1000, 500, 1000});
        notificationChannel.enableVibration(true);
        notificationManager.createNotificationChannel(notificationChannel);
    }

    // Create Notification
    NotificationCompat.Builder builder =
            new  NotificationCompat.Builder(this, CHANNEL_ID)
                    .setSmallIcon(R.mipmap.ic_launcher)
                    .setContentTitle("Title")
                    .setContentText("Notification text for MIREA")
                    .setWhen(System.currentTimeMillis())
                    .setProgress(100, 50, false)
                    .setContentIntent(resultPendingIntent);
    Notification notification = builder.build();
    // Show Notification
    notificationManager.notify(IDENTIFICATE_MSG++, notification);
}
```

В начале метода создаётся объект класса NotificationManager с помощью вызова getSystemService(), передав ему в качестве параметра строковую константу NOTIFICATION_SERVICE, определённую в классе Context. Далее определяется, какое действие требуется выполнить по нажатию на уведомление. Необходимо использовать PendingIntent (это контейнер для Intent). Этот контейнер может быть использован для последующего запуска, вложенного в него Intent. В данном случае используется Intent для запуска Activity. Упаковывается этот Intent и передается PeningIntent в уведомление. По нажатию на уведомление, система достанет из него PedningIntent и использует вложенный в него Intent, чтобы запустить Activity.

Для создания объекта NotificationCompact используется билдер, 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.



Запустите приложение на эмуляторе-> нажмите button-> сверните приложение-> нажмите на ваше уведомление (рис. 6.6). Если все правильно выполнено, то произойдет переход на Activity, которое указано в Intent.

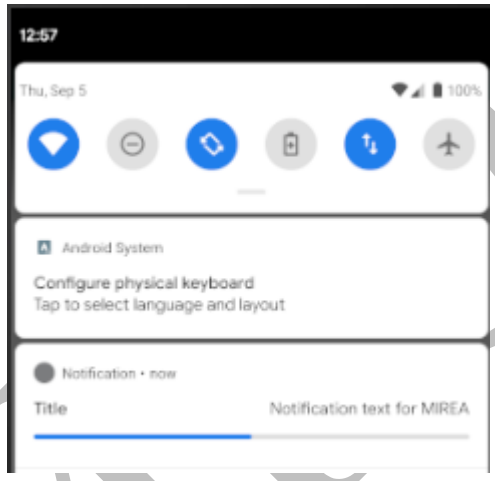


Рисунок 6.6 – Notification

6.5 Диалоговые окна

– всплывающее окно, использующееся для подтверждения каких-либо операций или ввода небольшого количества данных (рисунок 6.7).

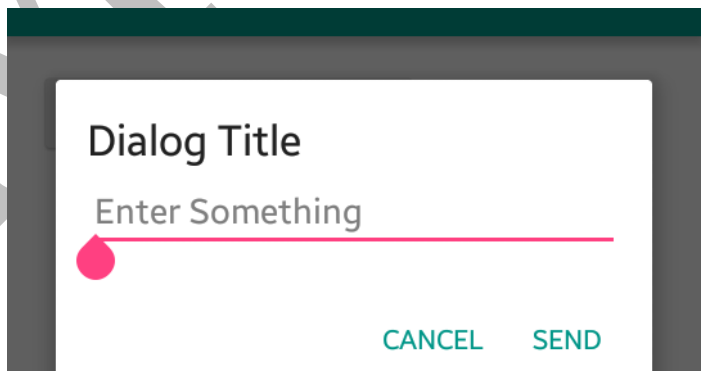


Рисунок 6.7 – Внещний вид Dialog

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 ему, возможно, потребуется

ввести какие-то 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. Раннее использовался класс Dialog и его производные, например, AlertDialog. Данный тип диалогов используется в фрагментах, который выступает в качестве контейнера. Таким образом создавать диалоговые окна возможно двумя способами – использовать DialogFragment, либо Dialog, AlertDialog и другими диалоговыми окнами.

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.

6.6 Задание

Создать новый модуль. В меню File> New> New Module> Phone & Tablet Module> Empty Activity. Название проекта Dialog.

Далее производится переход в директорию (рисунок 6.8), в которой находится MainActivity и правой кнопкой мыши вызывается контекстное меню: File-> New-> Java Class.

Рисунок 6.8 – Вызов менеджера создания Java классов

На рис.6.9 приведен пример создания класса. Для того, чтобы задать родительский класс требуется в поле «Superclass» указать ключевое слово Dialog и выбрать требуемый родительский класс.

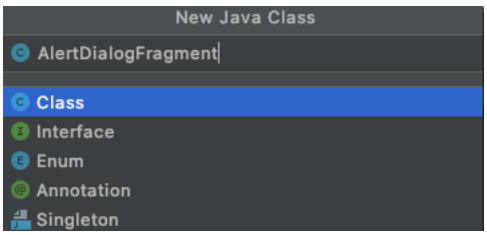


Рисунок 6.9 – Создание класса DialogFrament

```
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");
    }
}
```

Запустите проект. 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(), как и у обычных фрагментов.

Самый распространённый вариант диалогового окна – это AlertDialog.

Диалоговое окно AlertDialog является расширением класса Dialog, и это наиболее используемое диалоговое окно в практике программиста.

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 возможное количество кнопок в диалоге — три. 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();
    }
}
```

Внешний вид диалогового окна представлен на рисунке 6.10.

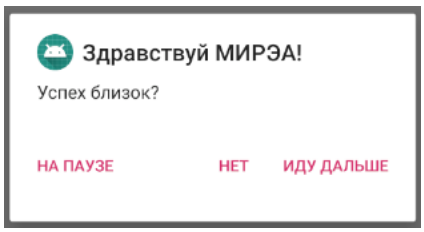


Рисунок 6.10 – Внешний вид AlertDialog

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)

6.7 Самостоятельная работа

Требуется изучить 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.