Практическая работа №1 “Стартовый проект на Android”

1.1. Android-проект

Приложения для Android могут быть написаны с использованием языков Kotlin, Java и C ++. Инструменты Android SDK компилируют ваш код вместе с любыми файлами данных и ресурсов в APK или пакет приложений для Android.

Пакет Android, представляющий собой архивный файл с расширением .apk. Этот файл содержит необходимые для работы приложения ресурсов и используется для установки приложений в ОС Android.

Пакет приложений для Android, представляющий собой архивный файл с расширением .aab, содержит содержимое проекта приложения для Android, включая некоторые дополнительные метаданные, которые не требуются во время выполнения. AAB - это формат публикации, который нельзя установить на устройства Android, он откладывает создание и подписание APK на более поздний этап. Например, при распространении вашего приложения через Google Play серверы Google Play генерируют оптимизированные APK-файлы, содержащие только те ресурсы и код, которые требуются конкретному устройству, запрашивающему установку приложения.

1.1.1 Структура проекта

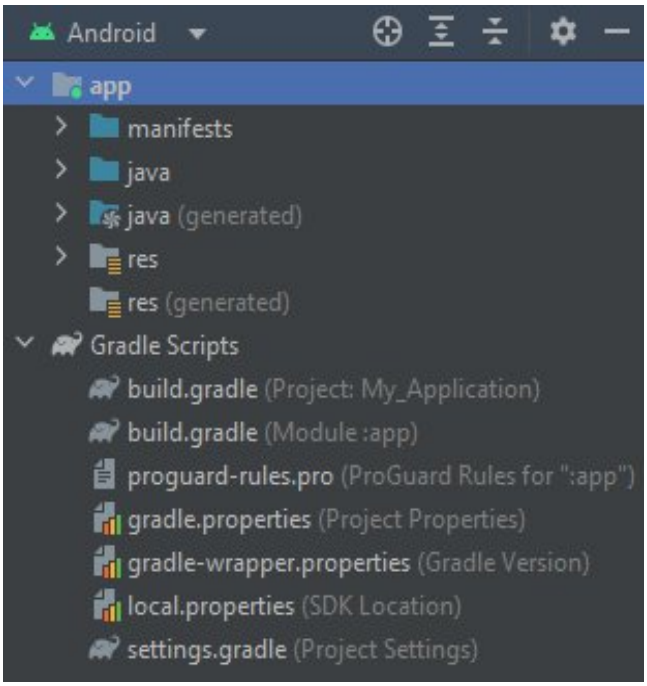

Рисунок 1. Файлы в Android Studio

Что такое AndroidManifest.xml файл

Файл AndroidManifest.xml является одним из самых важных в проекте Android приложения. В нем содержится информация о пакетах приложения, компонентах, таких как Activity, Service и других. Файл AndroidManifest.xml выполняет следующие задачи:

- Описывает разрешения, необходимые приложению для работы, например на доступ к камере или внутреннему хранилищу;
- Определяет как будут запускаться приложение, например, какое Activity должно быть запущено при нажатии на иконку в списке приложений (какие фильтры использовать).

Папка java

Папка java содержит исходный код приложения. Классы могут быть расположены в различных пакетах, но обязательно внутри папки java.

Папка res

В папке res расположены все используемые приложением ресурсы, включая изображения, различные xml файлы, анимации, звуковые файлы и многие другие. Внутри папки res эти все ресурсы распределены по своим папкам:

- Папка drawable содержит файлы с изображениями, которые будет использоваться в приложении;
- Папка layout располагает xml файлами, которые используются для построения пользовательского интерфейса Android приложения;
- В папке menu находятся xml файлы, используемые только для создания меню;
- В mipmap папке хранят только значки приложения. Любые другие drawable элементы должны быть размещены в своей папке;
- values хранит те xml файлы, в которых определяются простые значения типа строк, массивов, целых чисел, размерностей, цветов и стилей.

Скриты Gradle

Скрипты Gradle используются для автоматизации сборки проекта. Android Studio выполняет сборку приложения в фоновом режиме без какого-либо вмешательства со стороны разработчика. Этот процесс сборки осуществляется с использованием системы Gradle — инструментария для автоматической сборки с помощью набора конфигурационных файлов. Gradle скрипты написаны на языке groove.

1.1.2 Компоненты приложений

Компоненты приложений являются основными строительными блоками приложения для Android. Каждый компонент - это точка входа, через которую система или пользователь могут войти в ваше приложение. Некоторые компоненты зависят от других.

Существует четыре различных типа компонентов приложений:

1. Activities - основное предназначение заключается в том, что оно служит точкой входа для взаимодействия приложения с пользователем, а также отвечает за то, как пользователь перемещается внутри приложения или между приложениями.
2. Services - это особый тип компонента приложения, который предназначен для обработки повторяющихся и/или длительно выполняемых процессов.
3. Broadcast receivers - компонент Android, позволяющий приложению реагировать на сообщения (Android Intent), которые рассылаются операционной системой Android или приложением.
4. Content providers - компонент, позволяющий приложению предоставлять другим приложениям доступ к данным, которыми оно управляет.

1.2 Жизненный цикл Activity

По мере того, как пользователь перемещается по вашему приложению, выходит из него и возвращается обратно, экземпляры Activity вашего приложения переходят через различные состояния в своем жизненном цикле. Класс Activity предоставляет ряд обратных вызовов, которые позволяют узнать об изменении состояния: что система создает, останавливает или возобновляет Activity или уничтожает процесс, в котором находится Activity.

В методах обратного вызова жизненного цикла вы можете объявить, как ведет себя Activity, когда пользователь покидает Activity и снова открывает его. Например, если вы создаете проигрыватель потокового видео, вы можете приостановить воспроизведение видео и разорвать сетевое соединение, когда пользователь переключится на другое приложение. Когда пользователь вернется, вы можете повторно подключиться к сети и позволить пользователю возобновить просмотр видео с того же места. Другими словами, каждый обратный вызов позволяет выполнять набор операций, подходящих для необходимых изменений в состоянии приложения. Выполнение нужной

работы в нужное время и правильная обработка переходов делают ваше приложение более надежным и производительным. Например, хорошая реализация обратных вызовов жизненного цикла может помочь вашему приложению избежать:

- Сбоев, если пользователь получает телефонный звонок или переключается на другое приложение во время использования вашего приложения;
- Потребления системных ресурсов, когда пользователь не использует приложение;
- Потери состояния Activity, если покинет приложение и вернется к нему позже;
- Сбоев или потерь состояния Activity при повороте экрана между альбомной и портретной ориентацией.

1.2.1. Концепции жизненного цикла деятельности

Для перехода между этапами жизненного цикла класс Activity предоставляет базовый набор из шести обратных вызовов: onCreate(), onStart(), onResume(), onPause(), onStop()и onDestroy(). Система вызывает каждый из этих обратных вызовов, когда Activity переходит в новое состояние. На рис. 2 представлено визуальное представление этой парадигмы.

Когда пользователь покидает Activity, система вызывает методы для уничтожения Activity. В некоторых случаях это уничтожение является лишь частичным: Activity все еще находится в памяти (например, когда пользователь переключается на другое приложение) и все еще может вернуться на передний план. Если пользователь возвращается к этому Activity, оно возобновляется с того места, где пользователь остановился. За некоторыми исключениями приложениям запрещено запускать Activity, когда они работают в фоновом режиме.

Вероятность того, что система уничтожит данный процесс — вместе с Activity в нем — зависит от состояния Activity в данный момент.

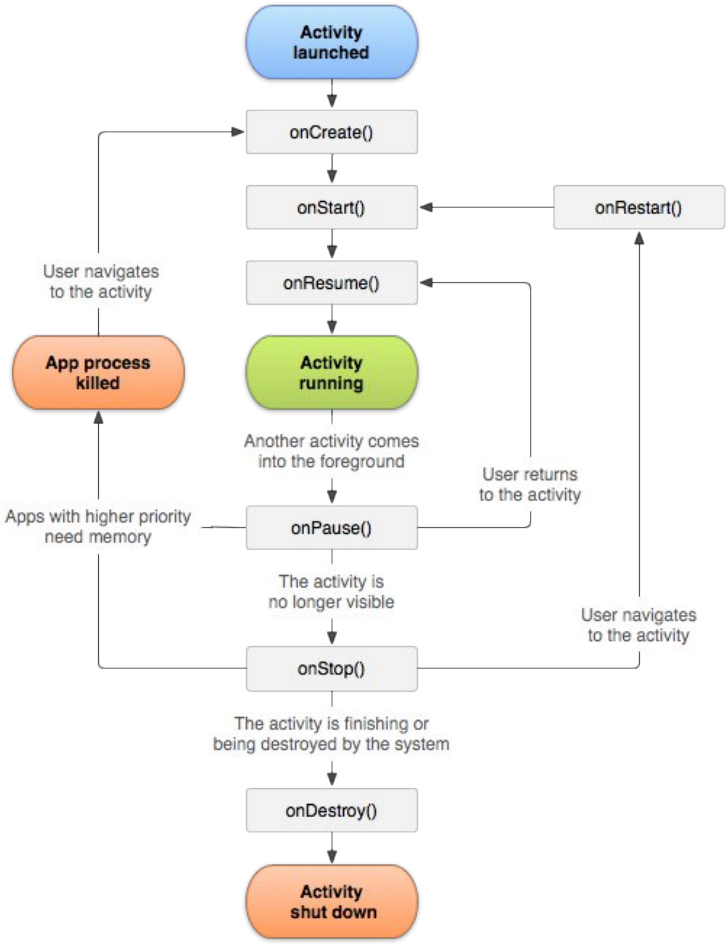

Рисунок 2. Упрощенная иллюстрация жизненного цикла активности

В зависимости от сложности вашего Activity вам, вероятно, не нужно реализовывать все методы жизненного цикла. Однако важно, чтобы вы понимали каждый из них и реализовывали те из них, которые гарантируют, что ваше приложение ведет себя так, как ожидают пользователи.

1.2.2. Обратные вызовы жизненного цикла

Некоторые действия, такие как вызов setContentView(), относятся к методам жизненного цикла активности. Однако код, реализующий действия зависимого компонента, должен быть размещен в самом компоненте. Для этого необходимо сделать так, чтобы зависимый компонент учитывал жизненный цикл.

onCreate( )

Метод onCreate() вызывается при создании или перезапуска Activity. Система может запускать и останавливать текущие окна в зависимости от происходящих событий. Внутри данного метода настраивают статический интерфейс Activity. Инициализирует статические данные Activity, связывают

данные со списками и т.д. Связывает с необходимыми данными и ресурсами. Задаёт внешний вид через метод setContentView().

onStart()

За onCreate() всегда следует вызов onStart(), но перед onStart() не обязательно должен идти onCreate(), так как onStart() может вызываться и для возобновления работы приостановленного приложения (приложение останавливается методом onStop()). При вызове onStart() окно ещё не видно пользователю, но вскоре будет видно. Вызывается непосредственно перед тем, как Activity становится видимой пользователю. Сопровождается вызовом метода onResume(), если Activity получает передний план, или вызовом метода onStop(), если становится скрытой.

onResume()

Метод onResume() вызывается после onStart(), даже когда окно работает в приоритетном режиме и пользователь может его наблюдать. В этот момент пользователь взаимодействует с созданным вами окном. Приложение получает монопольные ресурсы. Запускает воспроизведение анимации, аудио и видео. Также может вызываться после onPause().

onPause()

Когда пользователь решает перейти к работе с новым окном, система вызовет для прерываемого окна метод onPause(). По сути происходит свёртывание активности. Сохраняет незафиксированные данные. Деактивирует и выпускает монопольные ресурсы. Останавливает воспроизведение видео, аудио и анимацию. От onPause() можно перейти к вызову либо onResume(), либо onStop().

onStop()

Метод onStop() вызывается, когда окно становится невидимым для пользователя. Это может произойти при её уничтожении, или если была запущена другая активность (существующая или новая), перекрывшая окно текущей активности. Всегда сопровождает любой вызов метода onRestart(), если Activity возвращается, чтобы взаимодействовать с пользователем, или метода onDestroy(), если эта активность уничтожается.

onRestart()

Если окно возвращается в приоритетный режим после вызова onStop(), то в этом случае вызывается метод onRestart(). Т.е. вызывается после того, как

Activity была остановлена и снова была запущена пользователем. Всегда сопровождается вызовом метода onStart().

onDestroy()

Метод вызывается по окончании работы Activity, при вызове метода finish() или в случае, когда система уничтожает этот экземпляр активности для освобождения ресурсов. Эти два сценария уничтожения можно определить вызовом метода isFinishing(). Вызывается перед уничтожением Activity. Это последний запрос, который получает активность от системы. Если определённое окно находится в верхней позиции в стеке, но невидимо пользователю и система решает завершить это окно, вызывается метод onDestroy(). В этом случае метод удаляет все статические данные Activity. Отдаёт все используемые ресурсы.

Переопределение методов, существующих в классе Activity помечается аннотацией @Override и выглядит следующим образом:

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

@Override
protected void onStart() {
    super.onStart();
}

@Override
protected void onStop() {
    super.onStop();
}

@Override
protected void onDestroy() {
    super.onDestroy();
}

@Override
protected void onPause() {
    super.onPause();
}

@Override
protected void onResume() {
    super.onResume();
}
```

1.3 Всплывающие сообщения

Класс Toast предоставляет простую обратную связь об операции в небольшом всплывающем окне. Он заполняет только пространство, необходимое для сообщения, а текущая активность остается видимой и интерактивной. Toast’ы автоматически исчезают по истечении тайм-аута.

Например, нажатие кнопки « Отправить » в сообщении электронной почты вызывает всплывающее уведомление «Отправка сообщения...», как показано на следующем скриншоте:

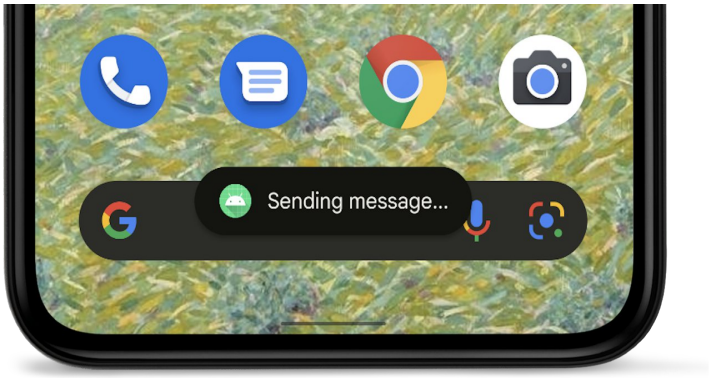

Рисунок 3. Пример вывода Toast

Если ваше приложение предназначено для Android 12 (уровень API 31) или выше, всплывающее уведомление ограничено двумя строками текста и показывает значок приложения рядом с текстом. Имейте в виду, что длина строки этого текста зависит от размера экрана, поэтому лучше сделать текст как можно короче.

1.3.1 Создание объекта Toast

Используйте makeText() метод, который принимает следующие параметры:

1. Контекст приложения.
2. Текст, который должен отображаться для пользователя.
3. Время, в течение которого всплывающее уведомление должно оставаться на экране.

Метод makeText() возвращает правильно инициализированный Toast объект.

1.3.2 Отображение Toast

Чтобы отобразить всплывающее уведомление, вызовите show()метод, как показано в следующем примере:

```
Context context = getApplicationContext();
CharSequence text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(context, text, duration);
toast.show();
```

1.3.3 Последовательный вызовов метода отображения уведомления

Вы можете вызвать метод отображения Toast сразу при его создании, чтобы не удерживать Toast объект, как показано в следующем фрагменте кода:

```
Toast.makeText(context, text, duration).show();
```

1.4 Логирование

Окно Logcat в Android Studio помогает отлаживать ваше приложение, отображая журналы с вашего устройства в режиме реального времени — например, сообщения, которые вы добавили в свое приложение с помощью класса, сообщения от служб, работающих на Android, или системные сообщения, например, при сборке мусора. Когда приложение выдает исключение, Logcat показывает сообщение, за которым следует соответствующая трассировка стека, содержащая ссылки на строку кода.

Очень часто программисту нужно вывести куда-то промежуточные результаты, чтобы понять, почему программа не работает. Особо хитрые временно размещают на экране текстовую метку и выводят туда сообщение при помощи метода textView.setText(" "). Но есть способ лучше. В Android есть специальный класс android.util.Log для подобных случаев.

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).

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

```
private static final String TAG = "MyApp";
```

Некоторые в сложных проектах используют следующий вариант, чтобы понимать, в каком классе происходит вызов:

```
private static final String TAG = this.getClass().getSimpleName();
```

Далее уже в любом месте вашей программы вы вызываете нужный метод журналирования с этим тегом:

```
Log.i(TAG, "Это мое сообщение для записи в журнале");
```

Также используется в исключениях:

```
try {
// ...
} catch (Exception exception) {
    Log.e(TAG, "Получено исключение", exception);
}
```

Рисунок 4. Пример «логирования»

Пользователи не видят этот журнал. Но, вы, как разработчик, можете увидеть его через программу LogCat, доступный в Android Studio.

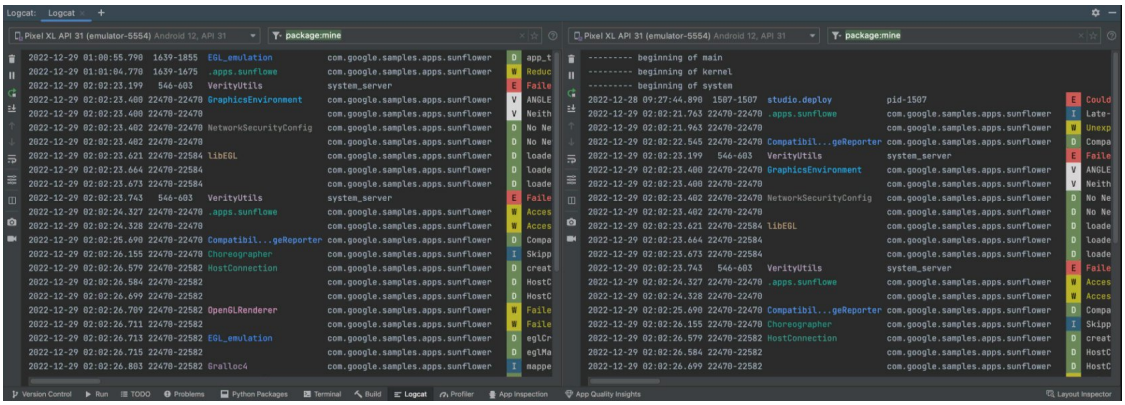

Рисунок 5. Разделение окон Logcat в Android Studio Electric Eel.

Задание:

1. Создать проект;
2. Описать в отчете наполнение проекта;
3. На всех этапах жизненного цикла MainActivity отобразить Toast, а также вывести в Logcat строки логов при помощи методов Log.d, Log.e и т. д.