МИНОБРНАУКИ РОССИИ

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

РТУ МИРЭА

Практическое занятие № 3/4ч.

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

1 НАМЕРЕНИЯ.

Требуется создать новый проект «*ru.mirea.«фамилия».Lesson3*».

На прошлом практическом занятии был изучен способ вызова Activity, основой которого являются «*action*», «*data*», «*category*» и «*IntentFilter*». Активность приложения передает «*intent*» операционной системе (ОС) «*Android*», которая проверяет его, а затем вызывает требуемую активность, даже если данная активность находится в другом приложении (рисунок 1.1).

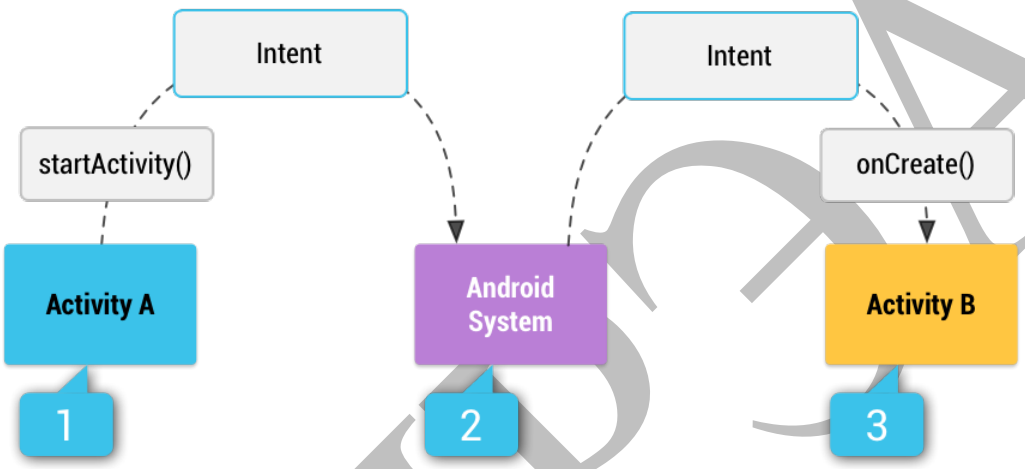


Рисунок 1.1 – Схематическое изображение процесса передачи неявного объекта Intent

Если намерение запрашивает выполнение какого-либо действия с определенным набором данных, то системе нужно уметь выбрать приложение (или компонент) для выполнения данного запроса. Для решения данной задачи используются фильтры намерений («*Intent Filter*»), которые используются для регистрации активностей, сервисов и широковещательных приёмников в качестве компонентов, способных выполнять заданные действия с конкретным видом данных. С помощью данных фильтров также регистрируются широковещательные приёмники, настроенные на трансляцию намерением заданного действия или события. Использование «*Intent Filter*» позволяют приложениям объявлять, что они могут отвечать на действия, запрашиваемые любой другой программой, установленной на устройстве. Например, возможно использовать «*intent*» для запуска активности «*Gmail*», отправляющей сообщения, и передачу текста, который нужно отправить. Вместо того, чтобы создавать собственные методы для отправки

электронной почты, возможно воспользоваться готовым приложением «*Gmail*». Прежде чем вызывать активности из других приложений, требуется узнать:

– какие активности доступны на устройстве пользователя;
– какие из этих активностей подходят для выполнения задачи;
– каким образом использовать эти активности.

Чтобы зарегистрировать компонент приложения в качестве потенциального обработчика намерений, требуется добавить тег <*intent-filter*> в узел компонента в манифест-файле. В фильтре намерений декларируется только три составляющих объекта «*Intent*»: действие, данные, категория. Дополнения и флаги не играют никакой роли в принятии решения, какой компонент получает намерение. Например, в любом приложении существует главная активность, которая устанавливается как точка входа для задания:

```
<activity android:name=".MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Фильтр такого вида в элементе <*action*> помечает активность, как запускаемую по умолчанию. Элемент <*category*> определяет метку, отображающуюся на панели «*Application Launcher*» (главный экран), давая пользователям возможность запускать задание и возвращаться к этому заданию в любое время после того, как оно было запущено.

1.1 Действие

Действия — стандартный механизм, при помощи которого ОС «*Android*» узнает о том, какие стандартные операции могут выполняться активностями. Например, ОС «*Android*» знает, что все активности, зарегистрированные для действия «*ACTION_SEND*», могут отправлять сообщения. Иными словами, если параметры намерения совпадают с условиями фильтра, то приложение (активность) будет вызвано. Система сканирует активности всех установленных приложений, и если находится несколько подходящих активностей, то ОС «*Android*» предоставляет пользователю выбор, какой именно программой следует воспользоваться. Если найдётся только одна подходящая активность, то никакого

диалога для выбора не вызовется, и активность запустится автоматически. Какие действия активностей возможно использовать в программах и какую дополнительную информацию они поддерживают, можно узнать в справочных материалах для разработчиков Android: https://developer.android.com/reference/android/content/Intent.

Для того, чтобы ОС знала фильтры «*Intent*» приложения и добавила информацию во внутренний каталог намерений, поддерживаемый всеми установленными приложениями, требуется добавить в файл манифеста элемент <*intent-filter*> для соответствующего элемента <*activity*>. Стоит отметить:

– «*IntentFilter*» может содержать в себе несколько действий. Тем самым активность оповещает систему о возможности выполнения нескольких функций. Например, не только запись аудио, но и редактирование данной записи. Таким образом получается, что активность может подойти разным «*Intent*» с разными «*action*»;

– активность, которая была вызвана с помощью намерения, имеет доступ к нему и может прочесть его атрибуты. Т.е. может узнать какой action использовался.

Основные константы действия перечислены ниже:

– *«ACTION_ANSWER»* — открывается активность, которая связана с входящими звонками. Это действие обрабатывается стандартным экраном для приема звонков;

– *«ACTION_CALL»* – инициализируется обращение по телефону;

– *«ACTION_DELETE»* – запускается активность, с помощью которой возможно удалить данные, указанные в пути URI внутри намерения;

– *«ACTION_EDIT»* – отображаются данные для редактирования пользователем;

– *«ACTION_INSERT»* – открывается активность для вставки в Курсор (Cursor) нового элемента, указанного с помощью пути URI. Дочерняя активность, вызванная с этим действием, должна вернуть URI, ссылающийся на вставленный элемент;

– *«ACTION_HEADSET_PLUG»* – подключение наушников;

– *«ACTION_MAIN»* – запускается как начальная активность задания;

– *«ACTION_PICK»* – загружается дочерняя активность, позволяющая выбрать элемент из источника данных, указанный с помощью пути URI. При закрытии должен возвращаться URI, ссылающийся на выбранный элемент. Активность, которая будет запущена, зависит от типа выбранных данных, например, при передаче пути «*content://contacts/people»* вызовется системный список контактов;

– *«ACTION_SEARCH»* – запускается активность для выполнения поиска. Поисковый запрос хранится в виде строки в дополнительном параметре намерения по ключу «*SearchManager.QUERY*»;

– *«ACTION_SEND»* – загружает экран для отправки данных, указанных в намерении. Контакт-получатель должен быть выбран с помощью полученной активности. Используется метод «*setType*», чтобы указать тип MIME для передаваемых данных. Эти данные должны храниться в параметре намерения «*extras*» с ключами «*EXTRA_TEXT*» или «*EXTRA_STREAM*», в зависимости от типа. В случае с электронной почтой стандартное приложение в ОС «*Android»* также принимает дополнительные параметры по ключам «*EXTRA_EMAIL», «EXTRA_CC», «EXTRA_BCC»* и «*EXTRA_SUBJECT»*. Используются действия «*ACTION_SEND*» только в тех случаях, когда данные нужно передать удаленному адресату (а не другой программе на том же устройстве);

– *«ACTION_SENDTO»* – открывается активность для отправки сообщений контакту, указанному в пути URI, который передаётся через намерение;

– *«ACTION_SYNC»* – синхронизируются данные сервера с данными мобильного устройства;

– *«ACTION_TIMEZONE_CHANGED»* – смена часового пояса;

– *«ACTION_VIEW»* – наиболее распространенное 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; данные о контакте отображаются в приложении для управления контактной информацией;

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

1.2 Информация

Идентификатор «*Data*» позволяет указать тип данных, с которым может взаимодействовать компонент приложения. При необходимости возможно задать несколько тегов «*data*». Чтобы указать, какие именно данные поддерживает компонент, используется сочетание следующих атрибутов:

– *«android:host»* – задается доступное имя удаленного сервера (например, «*google.com*»);

– *«android:mimetype»* – позволяет указать тип данных, которые компонент способен обрабатывать. Для примера: *<type android: value="vnd.android.cursor.dir/*"/>* будет соответствовать любому типу Курсора;

– *«android:path»* – задает доступные значения для пути URI (например, «*/transport/boats/*»). Uri – это объект, который берет строку, разбирает ее на составляющие и хранит в себе эту информацию. Строка составляется в соответствии с документом RFC 2396. Uri имеет набор методов, которые позволяют извлекать из разобранной строки отдельные элементы.

– *«android:port* «– указывает доступные порты для заданного сервера;

– *«android:scheme»* – это протокольная часть пути URI, например «*http*», «*https*», «*mailto:*» или «*tel:*».).

✅ **Пример:** Регистрация Activity по Intent.ACTION_VIEW

```
<activity android:name=".MyActivity">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="https" android:host="example.com" />
    </intent-filter>
</activity>
```

📌 **Как это работает?**

Теперь, если пользователь нажмёт на ссылку https://example.com, система предложит открыть MyActivity.

Если не требуется декларировать специфику данных Uri (например, когда операция использует другие виды дополнительных данных вместо URI), должен быть указан только атрибут «*android:mimeType*» для декларирования типа данных, с которыми работает ваша операция, например, «*text/plain*» или «*image/jpeg*». Если в Фильтре намерений не указано ни одного параметра «*data*», его действие будет распространяться на любые данные.

1.3 Категория

Применение поля «*Category*» позволяет выполнить описание характеристик операции, обрабатывающей объект «*Intent*». Система поддерживает несколько разных категорий, но большинство из них используется редко. Однако по умолчанию все неявные объекты «*Intent*» определяются с «*CATEGORY_DEFAULT*». Обозначается в фильтре намерений тэгом <*category*>. Далее перечислены основные типы категорий:

Категория	Описание
Intent.CATEGORY_DEFAULT	Используется по умолчанию для обработки неявных Intent
Intent.CATEGORY_LAUNCHER	Активность может быть начальной деятельностью задания из списка приложений в группе «Application Launcher» устройства
Intent.CATEGORY_BROWSABLE	Позволяет открывать ссылки в браузере или приложениях
Intent.CATEGORY_HOME	Fктивность отображает домашний экран, первый экран, который пользователь видит после включения устройства и загрузки системы или при нажатии клавиши «HOME»
Intent.CATEGORY_APP_EMAIL	Запуск почтового клиента
Intent.CATEGORY_APP_MESSAGING	Запуск приложения для сообщений
Intent.CATEGORY_APP_GALLERY	Запуск галереи

Для работы с категориями в классе «*Intent*» определена группа методов:

– «*addCategory*» – помещает категорию в объект Intent;

– «*removeCategory*» – удаляет категорию, которая была добавлена ранее;

– «*getCategories*» – получает набор всех категорий, находящихся в настоящее время в объекте Intent;

✅ **Пример:** Объявление Activity, которая может открывать ссылки

```
<activity android:name=".MyActivity">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE" />
        <data android:scheme="https" android:host="example.com" />
    </intent-filter>
</activity>
```

Теперь, если пользователь нажмёт ссылку https://example.com, система предложит открыть MyActivity.

1.4 Передача данных с помощью Intent

Для передачи данных между двумя активностями используется объект «*Intent*». С помощью вызова метода «*putExtra*» возможно добавить ключ и связанное с ним значение. Область «*extraData*» содержит список пар ключ/значение, который передаётся вместе с намерением. В качестве ключей используются строки, а для значений любые примитивные типы данных, массивы примитивов, объекты класса «*Bundle*» и др. Пример кода приведен ниже:

```
intent.putExtra("message", "value");
```

где «message» — имя ресурса/ключ для передаваемой информации;

«value» — само передаваемое значение.

Многократные вызовы «*putExtra*» позволяют добавить в «*intent*» наборы данных. При таком способе передачи данных следует обратить внимание, чтобы каждому экземпляру было присвоено **уникальный ключ**.

Метод «*putExtra*» позволяет передать данные простейших типов - *String, int, float, double, long, short, byte, char*, массивы этих типов, либо объект интерфейса «*Serializable*».

✅ **Пример:** Передача данных через Intent

```
Intent intent = new Intent(this, SecondActivity.class);
intent.putExtra("username", "JohnDoe");
intent.putExtra("university", "MIREA");
startActivity(intent);
```

При вызове новой активности данные следует извлечь из намерения. В решении этой задачи используется метод «*getIntent*». Данный метод возвращает намерение, запустившую активность и из полученного объекта возможно прочитать информацию, отправленную вместе с ним. Конкретный способ чтения зависит

от типа отправленной информации. Например, если известно, что намерение включает строковое значение с именем «*message*», используется следующий вызов:

```
Intent intent = getIntent();
String username = intent.getStringExtra("username");
```

1.5 Задание

Создать новый модуль/переименовать app. В меню «*File | New | New Module | Phone & Tablet Module | Empty Views Activity* ». Имя модуля «*IntentApp*».

Создайте 2 activity. Проверьте наличие записи о новом активности в manifest - файле. В первой активности требуется получить системное время с помощью следующего участка кода:

```
long dateInMillis = System.currentTimeMillis();
String format = "yyyy-MM-dd HH:mm:ss";
final SimpleDateFormat sdf = new SimpleDateFormat(format);
String dateString = sdf.format(new Date(dateInMillis));
```

Далее требуется передать время из одной активности в другую и отобразить во второй activity в «textView» следующую строку: «КВАДРАТ ЗНАЧЕНИЯ МОЕГО НОМЕРА ПО СПИСКУ В ГРУППЕ СОСТАВЛЯЕТ *ЧИСЛО*, а текущее время *ВРЕМЯ*».

1.6 Обмен данными

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

1.6.1 Передача данных

Для отправки различных сообщений типа электронного письма, SMS, MMS и т.д. применяется действие «*ACTION_SEND*». Метод «*setType*», требуется для установки типа MIME для передаваемых данных, хранящихся в параметре намерения *«extras»* с ключами EXTRA_TEXT или EXTRA_STREAM, в зависимости от типа. Если на устройстве установлено приложение с *intent*-фильтром, которое соответствует «*ACTION_SEND*» и MIME-типу «*text/plain*», система «*Android*» запустит его, а если будет найдено более одного приложения, то система отобразит диалоговое окно со списком удовлетворяющих запросу приложений.

Для того, чтобы разрабатываемая активность имела возможность обрабатывать подобные намерения, требуется указать следующие значения в манифест-файле:

```
<activity
    android:name=".ShareActivity"
    android:exported="true" >
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
        <data android:mimeType="image/*"/>
    </intent-filter>
</activity>
```

Тэг «*action*» сообщает ОС «*Android*» о том, что активность может обрабатывать действие «*ACTION_SEND*». Фильтр «*category*» должен включать категорию «*DEFAULT*», в противном случае он не сможет получать неявные намерения. В поле «*mimeType*» указываются типы данных, которые могут обрабатываться активностью.

```
Intent intent = new Intent(android.content.Intent.ACTION_SEND);
intent.setType("*/*");
intent.putExtra(Intent.EXTRA_TEXT, "Mirea");
startActivity(Intent.createChooser(intent, "Выбор за вами!"));
```

При выборе из списка программ пользователь может установить программу по умолчанию, которая будет автоматически запускаться при выбранном намерении. **В данном случае диалоговое окно выводиться не будет**. Возможно принудительно выводить диалоговое окно при помощи метода «*createChooser*» и пользователю придётся каждый раз выбирать нужную активность (рисунок1.2):

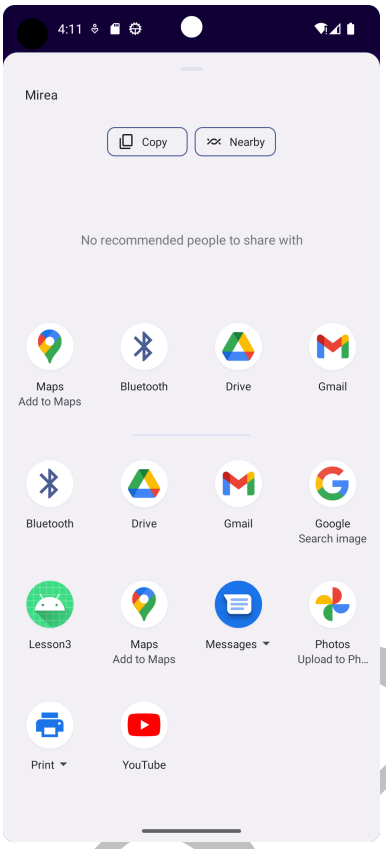


Рисунок 1.2 – Диалоговое окно выбора приложения

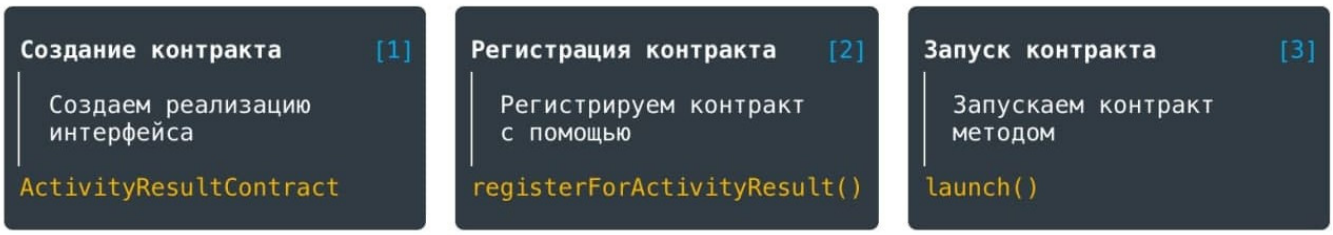
1.6.2 Получение данных

Ранее были представлены намерения, которые активируют другую активность, не ожидая получить в ответ на это результат. Далее рассматриваются более сложные действия, возвращающие значения. Например – при создании SMS, производится выбор адресата, система показывает экран со списком из адресной книги, пользователь выбирает нужного абонента и возвращается в экран создания SMS номер контакта. Т.е. пользователь вызвал экран выбора абонента, возвращающий контакт.

Предназначение действия «ACTION_PICK» (обобщённое название для действий с возвращающим значением) заключается в том, чтобы запустить активность, отображающую список элементов. После этого активность должна предоставлять пользователю возможность выбора элемента из этого списка. Когда пользователь выберет элемент, активность возвратит URI выбранного элемента вызывающей стороне. Таким образом, возможно многократно использовать

функцию UI для выбора нескольких элементов определенного типа. Для примера требуется вызвать приложение «Камера» и произвести снимок.

Приложения разработчиков могут не только передавать данные запускаемой активностью, но и ожидать от нее некоторого результата работы. Для получения результата работы запускаемой активности необходимо использовать «*Activity Result API*». «*Activity Result API*» предоставляет компоненты для регистрации, запуска и обработки результатов от другой активности. Главным преимуществом применения «*Activity Result API*» является то, что результат активности не связан с самой «*Activity*». Это позволяет получить и обработать результат, даже в случаях, когда активность по какой-то из причин завершила свою работу. Применение «*Activity Result*» состоит из трех шагов:



Для регистрации функции, которая будет обрабатывать результат, «*Activity Result API*» предоставляет метод «*registerForActivityResult*». Данный метод в качестве параметров принимает объекты «*ActivityResultContract*» и «*ActivityResultCallback*» и возвращает объект «*ActivityResultLauncher*», который применяется для запуска другой активности с помощью метода «*launch*». «*ActivityResultCallback*» представляет интерфейс с единственным методом «*onActivityResult*», который определяет обработку полученного результата. Когда вторая активность закончит работу и будет возвращен результат, сработает данный метод. Результат передается в метод в качестве параметра. При этом тип параметра должен соответствовать типу результата, определенного в «*ActivityResultContract*». Экземпляр объекта «*ActivityResultContract*» определяет контракт: данные какого типа будут подаваться на вход и какой тип будет представлять результат.

Ниже приведённый код выведет диалоговое окно со списком всех возможных программ, которые могут запустить активность с обработкой любого типа данных, т.к. «*setType("*/*")*» с помощью применения «*Activity Result API*».

```
Intent intent = new Intent(Intent.ACTION_PICK);
intent.setType("*/*");

ActivityResultCallback<ActivityResult> callback = new
ActivityResultCallback<ActivityResult>() {
    @Override
    public void onActivityResult(ActivityResult result) {
        if (result.getResultCode() == Activity.RESULT_OK) {
            Intent data = result.getData();
            Log.d(MainActivity.class.getSimpleName(), "Data:" + (data != null ?
data.getDataString() : null));
        }
    }
};
ActivityResultLauncher<Intent> imageActivityResultLauncher =
registerForActivityResult(
        new ActivityResultContracts.StartActivityForResult(),
        callback);
imageActivityResultLauncher.launch(intent);
```

После запуска приложения должно быть вызвано диалоговое окно выбора приложения для обработки намерения. Пример окна приведен на рисунке 1.3.

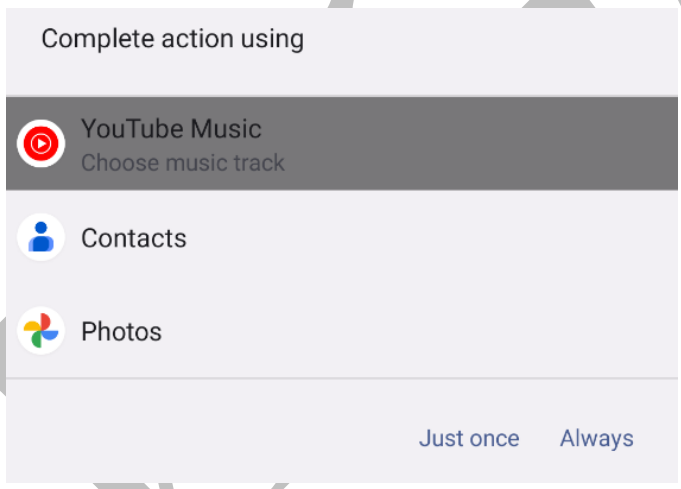


Рисунок 1.3 – Диалоговое окно выбора приложения

При выборе конкретного типа данных отобразится список программ, отвечающих заданным требованиям. Например, при установке «*intent.setType("image/*")*» результатом сканирования ОС Android будут программы для просмотра изображений.

1.6.3 StartActivityForResult

Для данной реализации необходимо вызвать метод *startActivityForResult*. Метод *startActivityForResult(Intent, int)* со вторым параметром, идентифицирующим запрос, позволяет возвращать результат. Разница между методами *startActivity* и *startActivityForResult* заключается в дополнительном параметре *RequestCode*. Представляет собой целое число, значение которого устанавливается произвольно

и необходимо для того, чтобы понимать от кого пришёл результат. Допустим имеется пять дополнительных экранов и каждому присваивается значение от 1 до 5. Данный код позволяет определить владельца возвращаемого результата. Когда дочерняя активность закрывается, то в родительской активности срабатывает метод *onActivityResult(int, int, Intent)*, который содержит возвращённый результат, определённый в родительской активности.

В onActivityResult отображаются следующие параметры:

- requestCode – тот же идентификатор, что и в startActivityForResult. По нему определяется, с какого Activity пришел результат.
- resultCode – код возврата. Определяет успешно прошел вызов или нет.
- data – Intent, в котором возвращаются данные.

1.7 Задание

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

Создать приложение с двумя экранами. Основное предназначение приложения заключается в отображении на экране названия любимой книги разработчика и пользователя приложением с использованием двух активностей.

Компоненты первого экрана:

– поле отображения «*TextView*» имеет несколько состояний: начальное значение «Тут появится название вашей любимой книги и любимая цитата из нее!», «Название Вашей любимой книги: КНИГА. Цитата: Цитата»;

– кнопка «*Button*» с текстом «Открыть экран ввода данных» предназначена для открытия второй активности.

Компоненты второго экрана:

– поле отображения «*TextView*» – «Любимая книга разработчика»;

– поле отображения «*TextView*» – «Цитата из книги»;

– поле ввода «*EditText*», со значением свойства «Hint»: «Введите название Вашей любимой книги»;

– поле ввода «*EditText*», со значением свойства «Hint»: «Введите цитату из Вашей любимой книги»;

– кнопка «*Button*» предназначена для отправки введенных данных пользователя на первый экран.

В созданном модуле создаётся новое activity: «*New* | *Activity* | *Empty activity* | *ShareActivity*». Исходный код класса «*MainActivity*» представлен ниже:

```
public class MainActivity extends AppCompatActivity {
    private ActivityResultLauncher<Intent> activityResultLauncher;
    static final String BOOK_NAME_KEY = "book_name";
    static final String QUOTES_KEY = "quotes_name";
    static final String USER_MESSAGE="MESSAGE";
    private TextView textViewUserBook;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        textViewUserBook = findViewById(R.id.textViewBook);

        ActivityResultCallback<ActivityResult> callback = new ActivityResult-
Callback<ActivityResult>() {
            @Override
            public void onActivityResult(ActivityResult result) {
                if (result.getResultCode() == Activity.RESULT_OK) {
                    Intent data = result.getData();
                    String userBook = data.getStringExtra(USER_MESSAGE);
                    textViewUserBook.setText(userBook);
                }
            }
        };
        activityResultLauncher = registerForActivityResult(
                new ActivityResultContracts.StartActivityForResult(),
                callback);
    }

    public void getInfoAboutBook(View view) {
        Intent intent = new Intent(this, ShareActivity.class);
        intent.putExtra(BOOK_NAME_KEY, !!!ВАША КНИГА, А НЕ МОЯ!!!);
        intent.putExtra(QUOTES_!!!, !!!ВАША ЦИТАТА, А НЕ МОЯ!!!);
        activityResultLauncher.launch(intent);
    }
}
```

В методе обработчика «*getInfoAboutBook*» создается «*Intent*» с указанным классом «*ShareActivity*». Для вызова второй активности используется явные намерения, а для получения результата «*Activity Result API*» (п.1.6.2). После отправки «*Intent*» «*MainActivity*» становится родительским для «*ShareActivity*». Далее требуется самостоятельно реализовать класс «*ShareActivity*». Пример получения данных и отправка приведен ниже:

```
// Получение данных из MainActivity
Bundle extras = getIntent(.getExtras();
if (extras != null) {
    TextView ageView = findViewById(R.id.textViewBook);
    String book_name = extras.getString(MainActivity.BOOK_NAME_KEY);
    String quotes_name = extras.getString(MainActivity.QUOTES_KEY)
    textView.setText(String.format("Моя любимая книга: %s и цитата %s",
book_name, quotes_name));
}

// Отправка введенных пользователем данных по нажатию на кнопку
Intent data = new Intent();
data.putExtra(MainActivity.USER_MESSAGE, text);
setResult(Activity.RESULT_OK, data);
finish();
```

Стоит обратить внимание, что при отправке данных создается «*Intent*» без указания вызываемого класса. Для возврата результата необходимо вызвать метод «*setResult*», в который передается два параметра: числовой код результата и отправляемые данные. Константа «*RESULT_OK*» означает успешное завершение вызова. Именно данная константа передастся в параметр «*resultCode*» метода «*onActivityResult*» в «*MainActivity*». Вызов метода «*finish*» завершает работу «*ShareActivity*».

Основное:
«requestCode» = 1 – успешное завершение вызова
«*resultCode*» = 0, это значение константы «*RESULT_CANCELED*», значит вызов прошел неудачно
Ограничений на значение статуса в методе «*setResult*» нет. «*RESULT_OK*» и «*RESULT_CANCELED*» – системные общепринятые константы. Возможно использование отличных значений, если в этом есть необходимость.
«*requestCode*» – ID запроса. Задается в методе startActivityForResult и проверяется потом в onActivityResult, чтобы точно знать, на какой вызов пришел ответ.

1.8 Вызов системных приложений.

С помощью атрибута «action» указывается действие «*activity*» (например, просмотр или редактирование данных). Также возможно указывать объекты, с которым эти действия нужно произвести. Для этого «*Intent*» имеет атрибут «*data*». Один из способов присвоения значения этому атрибуту – метод «*setData (Uri data)*» у объекта «*Intent*». На вход данному методу подается объект Uri.

Uri – это объект, который берет строку, разбирает ее на составляющие и хранит в себе данную информацию. Строка составляется в соответствии с документом RFC 2396. Uri имеет набор методов, которые позволяют извлекать из

разобранной строки отдельные элементы. В таблице 1.1 приведены примеры разбора строк, выполненных согласно RFC 2396.

Таблица 1.1 – Пример значений строки URI

Тип URI	Описание

URL	Пример: Uri.parse("http://developer.android.com/reference/android/net/Uri.html");
	
	uri.getScheme(): http
	uri.getSchemeSpecificPart(): //developer.android.com/reference/android/net/Uri.html
	uri.getAuthority(): developer.android.com
	uri.getHost(): developer.android.com
	uri.getPath(): /reference/android/net/Uri.html
	uri.getLastPathSegment(): Uri.html

FTP	Пример: Uri.parse("ftp://user@google.com:80/data/files");
	
	uri.getScheme(): ftp
	uri.getSchemeSpecificPart(): //user@google.com:80/data/files
	uri.getAuthority(): user@google.com:80
	uri.getHost(): google.com
	uri.getPort(): 80
	uri.getPath(): /data/files
	uri.getLastPathSegment(): files
	uri.getUserInfo(): user

Geo	Пример: Uri.parse("geo:56.111,37.111"); uri.getScheme(): geo uri.getSchemeSpecificPart(): 55.111,37.111
Number	Пример: Uri.parse("tel:12345"); uri.getScheme(): tel uri.getSchemeSpecificPart():12345

Contact	Пример: Uri.parse("content://contacts/people/1");
	
	uri.getScheme(): content
	uri.getSchemeSpecificPart(): //contacts/people/1
	uri.getAuthority(): contacts
	uri.getPath(): /people/1
	uri.getLastPathSegment(): 1

1.9 Задание

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

Требуется создать приложение, позволяющее отображать:

- страницу web - ресурса;
- координаты на карте;
- окно набора номера.

Для просмотра координат на карте, требуется приложение «*Google Maps*». Его нет в стандартных образах *Android* систем. Требуется убедиться, что установленный образ в эмуляторе содержит идентификатор «*Google Play*». На рис. 1.4 приведен алгоритм установки образа. Затем требуется создать эмулятор с ОС «*Google APIs*»

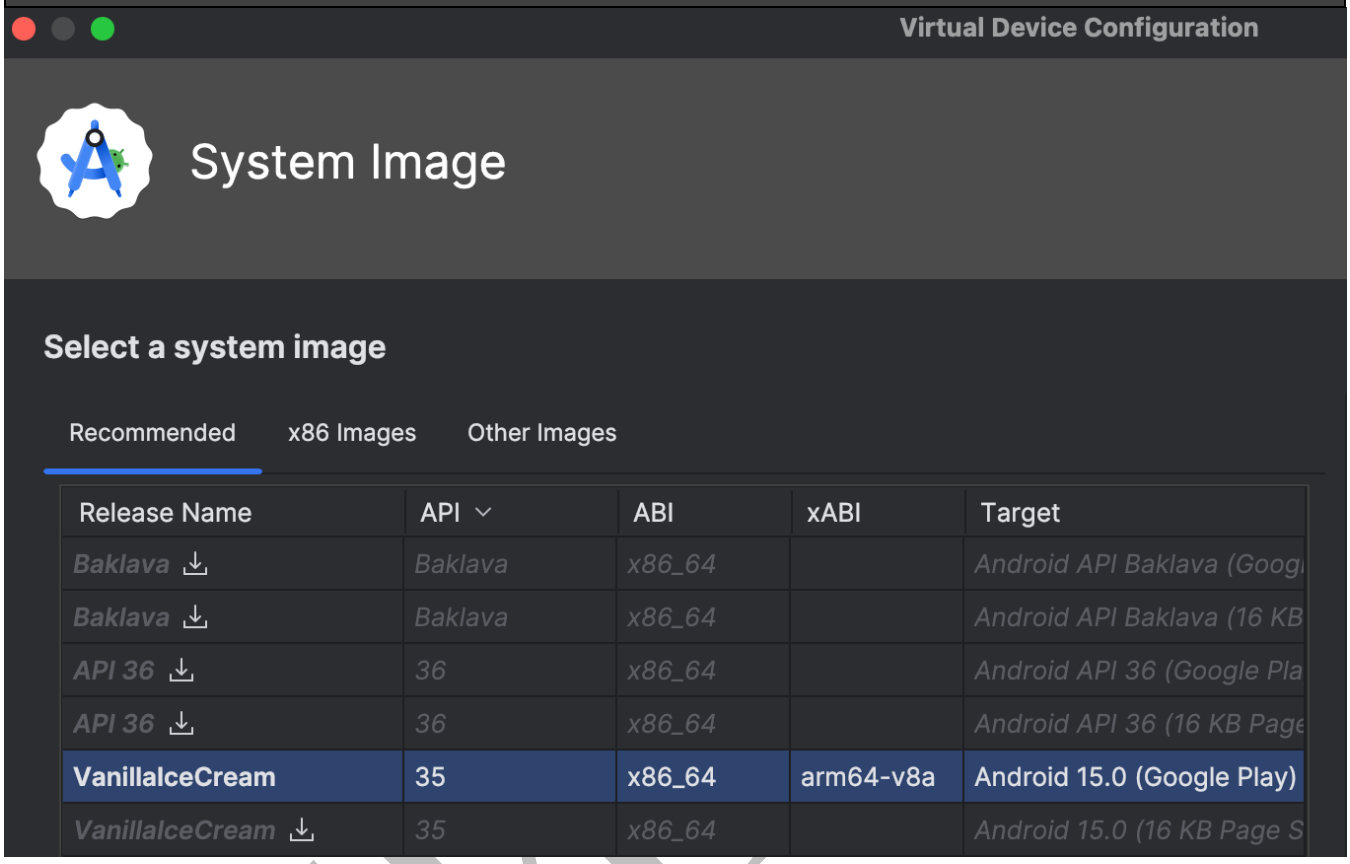


Рисунок 1.4 – Диалоговое окно выбора образа ОС Android для эмулятора

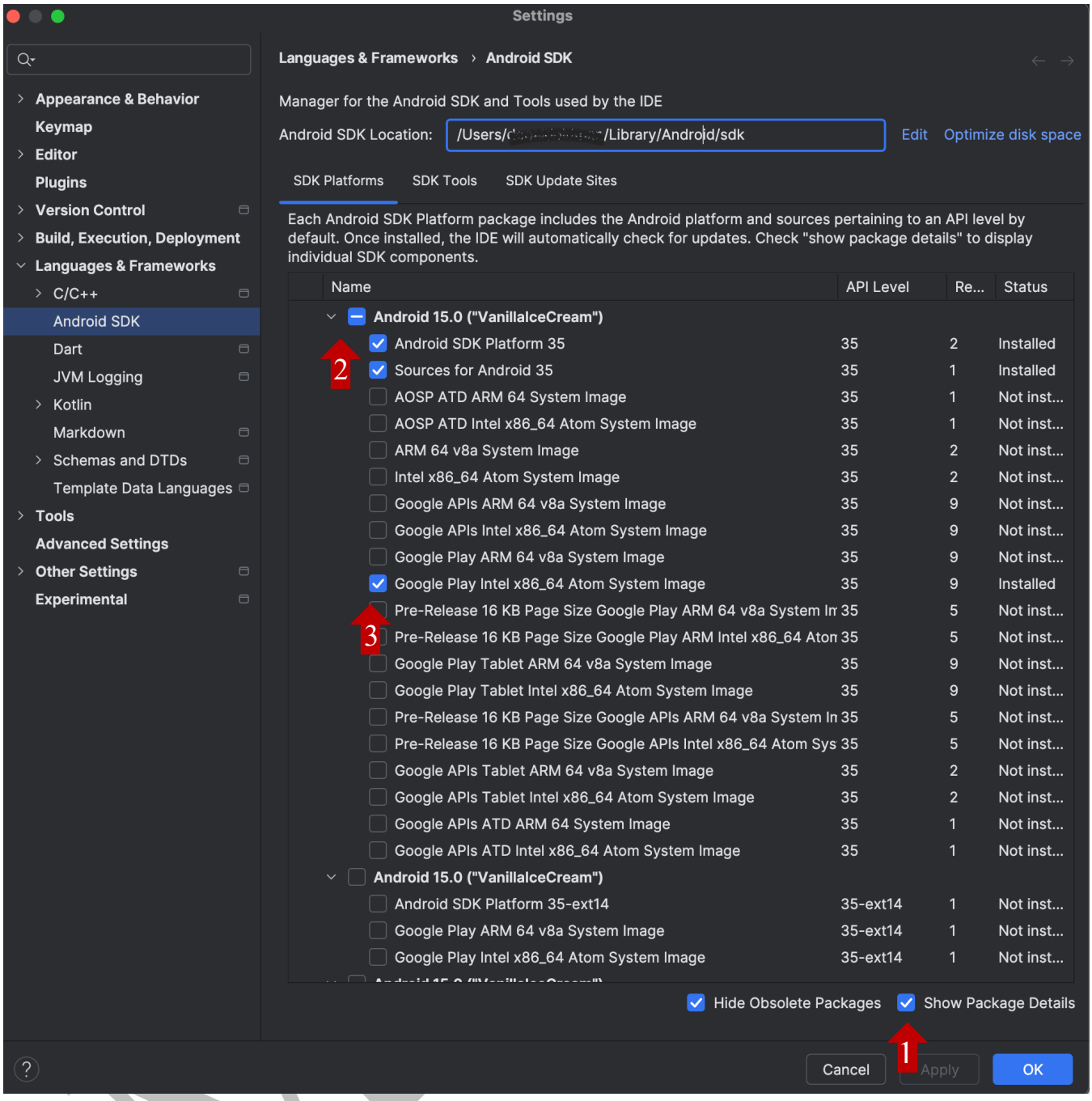


Рисунок 1.5 – Окно настроек SDK Manager

Требуется сформировать экран *activity_main.xml* в соответствии с рисунком1.10.

Рисунок 1.6 – Внешний вид приложения «*SystemIntentsApp*»

На экране размещены три кнопки (обработку нажатий реализовать через *setOnClickListener*):

– «позвонить», метод «*onClick*»;

– «открыть браузер», «*onClick*»;

– «открыть карту», «*onClick*».

В классе *MainActivity* реализованы 3 метода, реагирующие на нажатие кнопок:

```
public class MainActivity extends AppCompatActivity {

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

    public void onClickCall(View view) {
        Intent intent = new Intent(Intent.ACTION_DIAL);
        intent.setData(Uri.parse("tel:89811112233"));
        startActivity(intent);
    }

    public void onClickOpenBrowser(View view) {
        Intent intent = new Intent(Intent.ACTION_VIEW);
        intent.setData(Uri.parse("http://developer.android.com"));
        startActivity(intent);
    }

    public void onClickOpenMaps(View view) {
        Intent intent = new Intent(Intent.ACTION_VIEW);
        intent.setData(Uri.parse("geo:55.749479,37.613944"));
        startActivity(intent);
    }
}
```

В методе «*onClickCall*» используется конструктор «*Intent (String action)*». Входным аргументом является действие, а «*data*» указывается в следующей строке кода. Значение «*action*» является «*ACTION_DIAL*» – открытие набора номера, указанный в «*data*». В «*data»* размещен Uri, созданный из номера телефона «*89811112233»*.

В методе «*onClickOpenBrowser*» создается «*Intent*» и на вход принимается «*action»* и «*data*». Используется стандартный системный «*action*» – ACTION_VIEW. В качестве «*data*» подаётся объект Uri, созданный из веб-ссылки: «*http://developer.android.com»*.

В методе «*onClickMaps*» используется конструктор «*Intent*» с установленным «*action*» – ACTION_VIEW, а в качестве «*data*» создаётся Uri из пары координат – «*55.749479,37.613944»*. Данный «*Intent»* означает намерение отобразить карту с указанными координатами в центре экрана.

2 ФРАГМЕНТЫ

Одной из особенностей создания приложений для ОС «*Android*» является то, что одно и то же приложение может запускаться на устройствах с разными экранами и процессорами и будет работать на них одинаково. Но это вовсе не означает, что оно будет на них одинаково выглядеть. Чтобы интерфейсы приложения на телефоне и планшете отличались друг от друга, требуется определить разные макеты для больших и малых устройств.

Однако определить разные макеты для разных устройств недостаточно. Чтобы приложение работало по-разному в зависимости от устройства, наряду с разными макетами должен выполняться разный код «*Java*». Например, в приложении на рисунке 2.1 необходимо предоставить одну активность для планшетов и две активности для телефонов.

Рисунок 2.1 – Внешний вид приложения на разных устройствах

Данный способ может привести к дублированию кода второй активности, которая работает только на телефонах. Один код должен выполняться в нескольких активностях. Вместо того, чтобы дублировать код в двух активностях, следует

использовать фрагменты («*fragments*»). Фрагменты являются составной частью компонентов активности. Фрагмент управляет частью экранного пространства и может использоваться на разных экранах. Это означает, что имеется возможность создавать разные фрагменты для списка комплексов упражнений и для вывода подробного описания одного комплекса. После этого созданные фрагменты можно использовать в разных активностях.

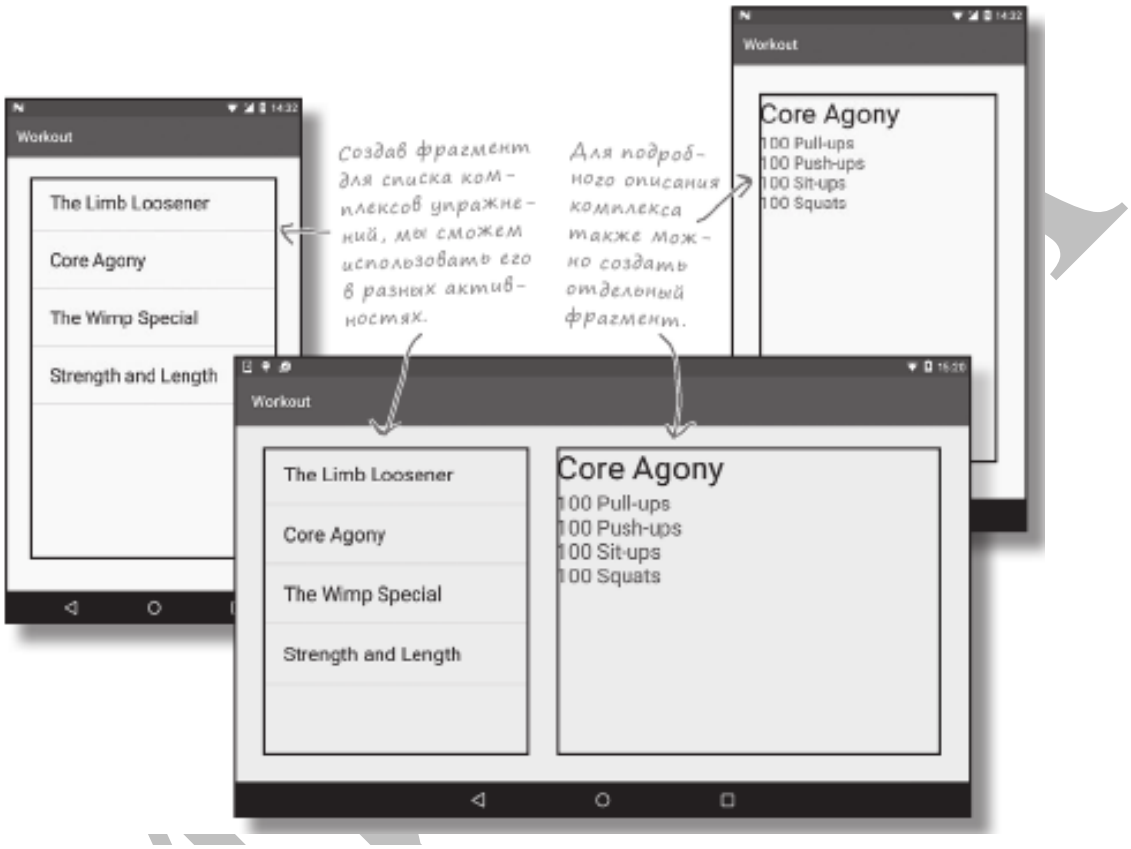


Рисунок 2.2 – Основное назначение фрагментов

Фрагмент, как и активность, связывается с макетом. Если внимательно подойти к его проектированию, управление всеми аспектами интерфейса может осуществляться из кода «*Java*». Если код фрагмента содержит все необходимое для управления его макетом, вероятность того, что фрагмент можно будет повторно использовать в других частях приложения, значительно возрастает. Стоит отметить, что фрагменты не являются заменой активности, они не существуют сами по себе, а только в составе активностей. Поэтому в манифесте прописывать их не нужно. Но в отличие от стандартной графических элементов, для каждого фрагмента требуется создание отдельного класса, как для активности. В составе активности существует

специальный менеджер фрагментов, который может контролировать все классы фрагментов и управлять ими.

Фрагменты являются строительным материалом для приложения. Данный компонент позволяет в нужное время добавить новый фрагмент, удалить ненужный фрагмент или заменить один фрагмент на другой.

Фрагмент может иметь свою разметку, но возможно использование и без неё. Также у фрагмента есть свой жизненный цикл, во многом совпадающий с жизненным циклом активности. Имеются специальные виды фрагментов, предназначенные под определённые задачи – «*ListFragment», «DialogFragment*» (изучали ранее), «*PreferenceFragment*» и другие.

Есть два варианта использования фрагментов в приложении:

– в разметке указывается фрагмент с помощью тега «*fragment*»;

– динамическое подключение фрагмента. В разметку помещается макет из группы «*ViewGroup*», который становится контейнером для фрагмента. Обычно, для данной цели используют «*FrameLayout*», но это не обязательное условие. В нужный момент фрагмент замещает контейнер и становится частью разметки.

Так же как активности, фрагменты должны реализовывать другие методы обратного вызова жизненного цикла, которые позволяют управлять его состоянием, таким как его добавление или удаление из активности, и для обработки переходов между состояниями жизненного цикла. Например, когда для активность вызывается метод «*onPause*», все фрагменты данной активности также получат вызов «*onPause*». Жизненный цикл компонента «*Fragment*» представлен на рисунке 2.3 вместе с циклом Activity.

Как правило, требуется реализовать следующие методы жизненного цикла:

– «*onCreate*» – система вызывает данный метод, когда создает фрагмент. В своей реализации разработчик должен инициализировать ключевые компоненты фрагмента, которые требуется сохранить, когда фрагмент находится в состоянии паузы или возобновлен после остановки.

– «*onCreateView*» – система вызывает данный метод при первом отображении пользовательского интерфейса фрагмента на дисплее. Для прорисовки

пользовательского интерфейса фрагмента следует возвратить из этого метода объект «*View*», который является корневым в макете фрагмента. Если фрагмент не имеет пользовательского интерфейса, можно возвратить «*null*».

– «*onPause*» – система вызывает данный метод как первое указание того, что пользователь покидает фрагмент (это не всегда означает уничтожение фрагмента). Обычно именно в этот момент необходимо фиксировать все изменения, которые должны быть сохранены за рамками текущего сеанса работы пользователя (поскольку пользователь может не вернуться назад).

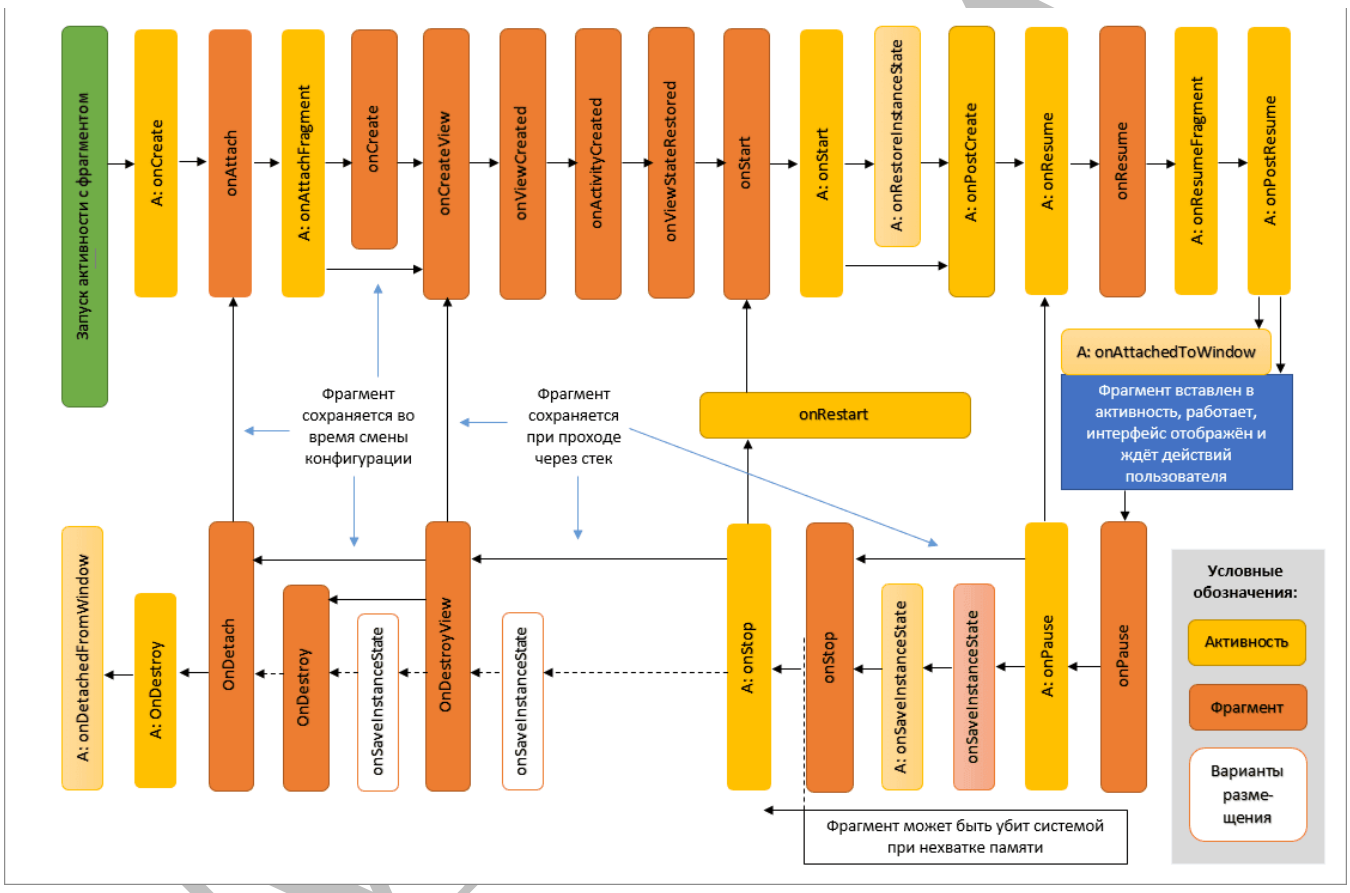


Рисунок 2.3 – Жизненный цикл Fragment совместно с Activity

2.1 Задание

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

Требуется создать приложение на основе «*Fragment*», учитывающее изменение ориентации экрана. Для создания фрагментов используется следующая последовательность: «*File | New | Fragment | Fragment (Blank)*», указать имена «*FirstFragment&SecondFragment*».

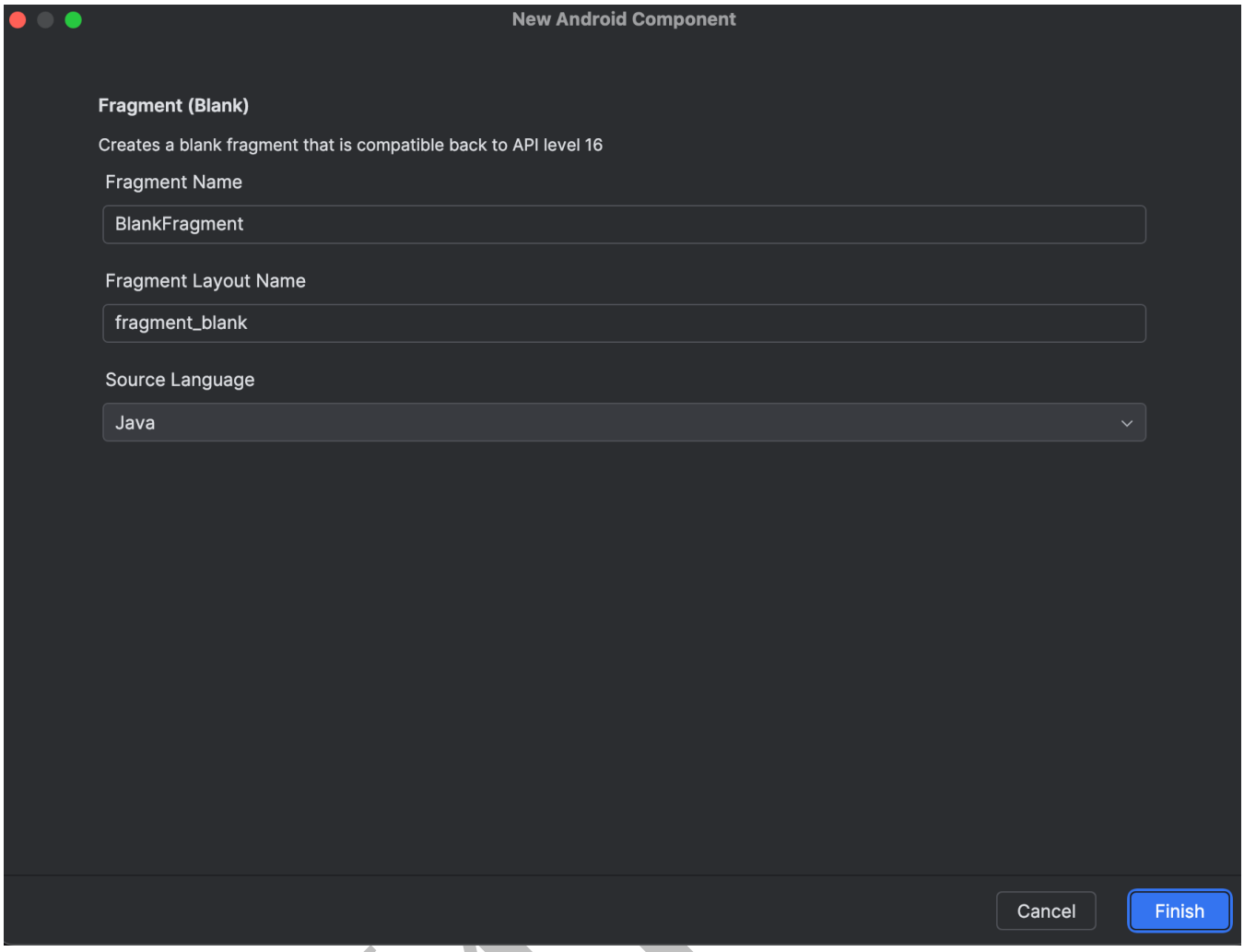


Рисунок 2.4 – Менеджер создания фрагментов

Далее рассматривается файл разметки «*res|layout|fragment_first.xml*»:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".BlankFragment">

    <!-- TODO: Update blank fragment layout -->
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="@string/hello_blank_fragment" />

</FrameLayout>
```

Пространство имён «*xmlns:android*» позволяет настраивать внешний вид и поведение компонентов в «*Android*»-приложении, а «*xmlns:tools*» позволяет среде разработки «*Android Studio*» правильно отображать компоненты для просмотра в режиме дизайна.

Пример. Имеется компонент «*TextView*», который отображает информацию с сервера. Чтобы визуально отобразить, как будет выглядеть текст, не обмениваясь данными с сервером, разработчику требуется временно присвоить какой-нибудь текст, а потом не забыть удалить его при создании релиза. Возможно использовать инструмент «*tools*», дублирующий многие визуальные атрибуты пространства имён «*android*»:

```
android:text=""
tools:text="@string/hello_blank_fragment2"
```

Атрибут «*tools:context*» у корневого элемента позволяет определить связь между макетом и классом активности, в которой данный макет будет реализован. Также для наглядности установлен фоновый цвет фрагмента. В файле «*res|layout|fragment_blank_fragment2.xml*» требуется также изменить цвет фона.

Далее рассмотрен класс FirstFragment:

```
public class FirstFragment extends Fragment {
 ……….

    // TODO: Rename and change types and number of parameters
    public static FirstFragment newInstance(String param1, String param2) {
        FirstFragment fragment = new FirstFragment ();
        Bundle args = new Bundle();
        args.putString(ARG_PARAM1, param1);
        args.putString(ARG_PARAM2, param2);
        fragment.setArguments(args);
        return fragment;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (getArguments() != null) {
            mParam1 = getArguments().getString(ARG_PARAM1);
            mParam2 = getArguments().getString(ARG_PARAM2);
        }
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_first, container, false);
    }

}
```

Данный класс должен наследоваться от класса Fragment. Чтобы создать макет для фрагмента, разработчик должен реализовать метод обратного вызова «*onCreateView*», который система «*Android*» вызывает, когда для фрагмента

наступает время отобразить свой макет. Реализация данного метода должна возвращать объект «*View*», который является корневым в макете фрагмента.

Раздувание («*inflate*») — это процесс создания View из XML-файла разметки. Макет в res/layout является описанием интерфейса в виде XML. Чтобы использовать его в коде (например, добавить кнопку или список), нужно «раздуть» (inflate) этот XML в объект View.

Метод «*inflate*» принимает три аргумента:

– идентификатор ресурса макета, «раздувание» которого следует выполнить;

– объект класса «*ViewGroup*», который должен стать родительским для макета после раздувания. Передача параметра «*container*» необходима для того, чтобы система смогла применить параметры макета к корневому представлению раздутого макета, определяемому родительским представлением, в которое направляется макет;

– логическое значение, показывающее, следует ли прикрепить макет к объекту «*ViewGroup*» (второй параметр) во время раздувания. (В данном случае это «*false*», потому что система уже разместила «раздутый» макет в объекте «*container*», передача значения «*true*» создаёт лишнюю группу представлений в окончательном макете).

Далее следует модифицировать activity_main.xml. Следует добавить 2 кнопки для переключения между фрагментами. Также в разметку требуется добавить FrameLayout – это контейнер, в котором будет происходить вся работа с фрагментами. Данный контейнер должен быть типа ViewGroup.

Далее приведён код разметки activity_main.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://sche-
mas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <androidx.fragment.app.FragmentContainerView
        android:id="@+id/fragmentContainerView"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/btnFirstFragment"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginBottom="16dp"
        android:text="First screen"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

    <Button
        android:id="@+id/btnSecondFragment"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="183dp"
        android:layout_marginEnd="16dp"
        android:layout_marginBottom="16dp"
        android:text="Second screen"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
       />

</androidx.constraintlayout.widget.ConstraintLayout>
```

В классе «*MainActivity*» требуется реализовать методы обработки нажатия кнопок и реализовать логику загрузки фрагментов. Фрагменты ничего не должны знать о существовании друг друга. Любой фрагмент существует только в активности и только активность через свой специальный менеджер фрагментов должна управлять ими. Сами фрагменты должны реализовать необходимые интерфейсы, которые активность будет использовать в своих целях. Транзакция позволяет выполнить все операции скопом. Разработчик сообщает менеджеру про все операции, который в свою очередь запускает их в методе «*beginTransaction*»

и подтверждает своё намерение через метод «*commit*». Далее приведён код класса «*MainActivity.xml*».

```
public class MainActivity extends AppCompatActivity {
    private Fragment fragment1, fragment2;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main);
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v,
insets) -> {
            Insets systemBars =
insets.getInsets(WindowInsetsCompat.Type.systemBars());
            v.setPadding(systemBars.left, systemBars.top, systemBars.right,
systemBars.bottom);
            return insets;
        });

        fragment1 = new BlankFragment();
        fragment2 = new BlankFragment2();
        FragmentManager fragmentManager = getSupportFragmentManager();

        Button btnFirstFragment = (Button) findViewById(R.id.btnFirstFragment);
        btnFirstFragment.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {

fragmentManager.beginTransaction().replace(R.id.fragmentContainerView,
fragment1).commit();
            }
        });
        Button btnSecondFragment = (Button) findViewById(id.btnSecondFragment);
        btnSecondFragment.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {

fragmentManager.beginTransaction().replace(R.id.fragmentContainerView,
fragment2).commit();
            }
        });

    }
}
```

Далее требуется собрать проект и убедиться в том, что производится загрузка требуемых фрагментов.

Следующим этапом является добавление горизонтальной ориентации в разрабатываемое приложение. Для этого следует создать дополнительный файл разметки для activity_main.xml: *New-> Android Resource File.* В меню выбрать *«Orientation»* и установить значения (рисунок 2.5):

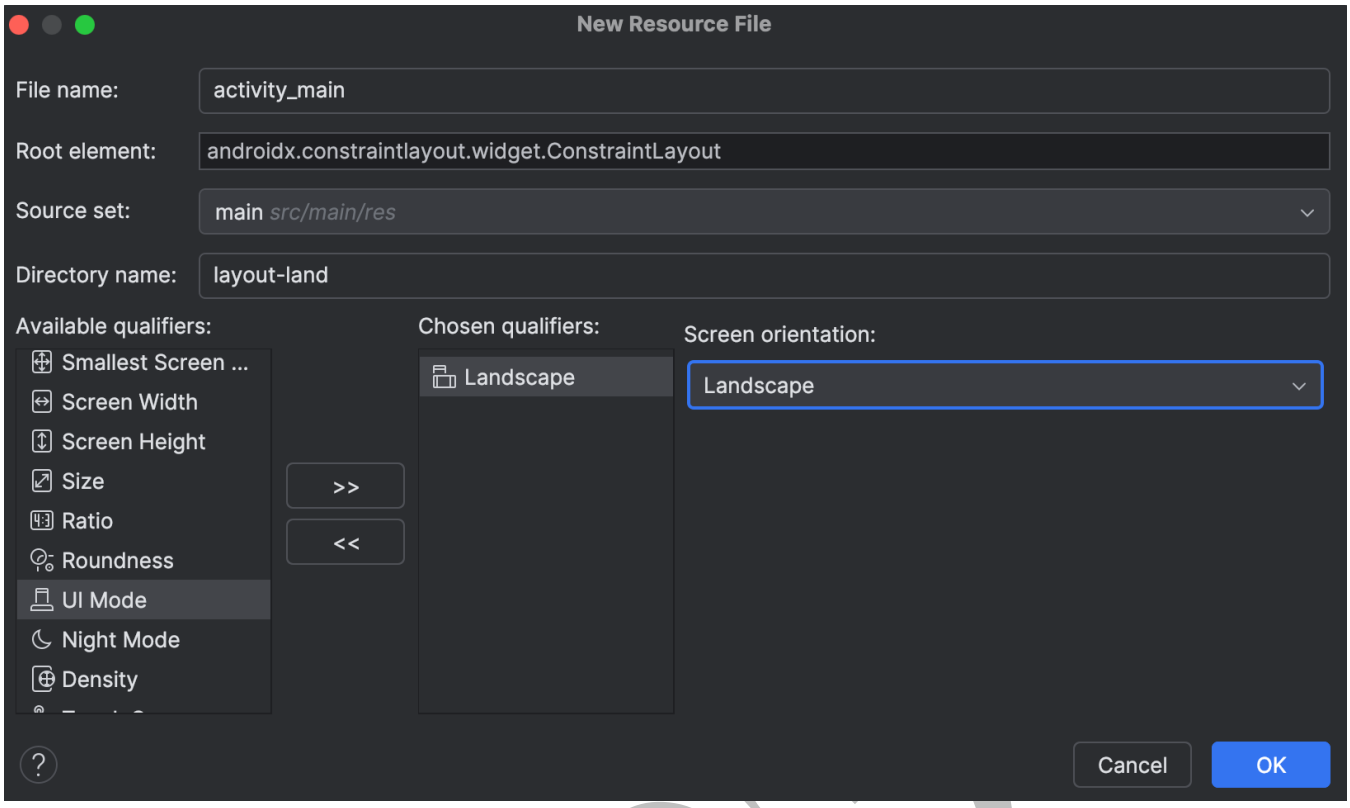


Рисунок 2.5 – Создание горизонтального файла разметки activity_main.xml

В данном файле требуется указать созданные фрагменты:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="horizontal">
        <fragment android:name="com.mirea.asd.simplefragmentapp.BlankFragment1"
            android:id="@+id/list"
            android:layout_weight="1"
            android:layout_width="0dp"
            android:layout_height="match_parent" />

        <fragment android:name="com.mirea.asd.simplefragmentapp.BlankFragment2"
            android:id="@+id/viewer"
            android:layout_weight="2"
            android:layout_width="0dp"
            android:layout_height="match_parent" />

    </LinearLayout>
</LinearLayout>
```

Далее требуется скомпилировать приложение и запустить. После изменения положения устройства на горизонтальное, должен отобразится следующий экран (рисунок 2.6):

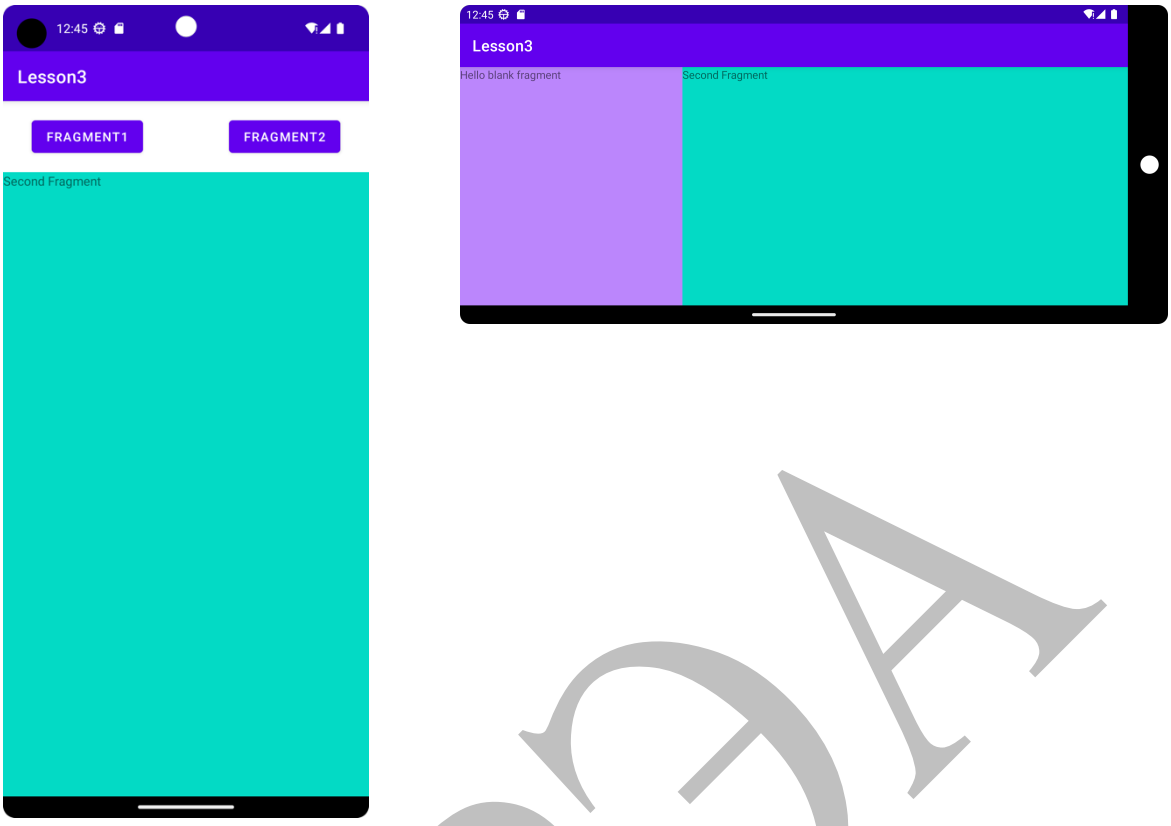


Рисунок 2.6 – Экран приложения «SimpleFragmentApp» в горизонтальной и вертикальной ориентации

3 КОНТРОЛЬНОЕ ЗАДАНИЕ.

Создать новое приложение «*File | New | New Project | Navigation Drawer Activity*. Название проекта MireaProject. (рисунок 3.1)

Android позволяет создать собственное окно для просмотра веб-страниц и клон браузера при помощи элемента WebView. Сам элемент использует движок «*WebKit*» или движок от «*Chromium*» и имеет множество свойств и методов.

Требуется создать/модифицировать два фрагмента.

– «*DataFragment»* – **создать фрагмент** с уникальной информацией об интересующей Вас отрасли. Отразить навыки разметки, цветовая схема и отступы должны быть выполнены согласно требованиям «Material You»;

– *«WebViewFragment»* – **создать простейший браузер** со страницей по умолчанию.

После сборки проекта требуется запустить проект и ознакомиться с логикой работы данного шаблона. Значок в виде трёх горизонтальных полосок в заголовке называется «гамбургером» («*Hamburger menu*»). При нажатии слева отображается навигационная шторка.

Если открыть файл «*activity_main.xml*» в режиме «*Design*», то возможно увидеть, как будет выглядеть приложение с открытой шторкой (рисунок 3.2). За выдвигающую шторку отвечает элемент «*NavigationView*», который входит последним в контейнере «*DrawerLayout»* и представляет собой навигационное меню. А перед меню находится вставка «*include*», указывающая на разметку «*app_bar_main.xml*». Атрибут «*tools:openDrawer ="start"*» позволяет указать среде разработки, что навигационное меню требуется отобразить в раскрытом виде в режиме просмотра разметки.

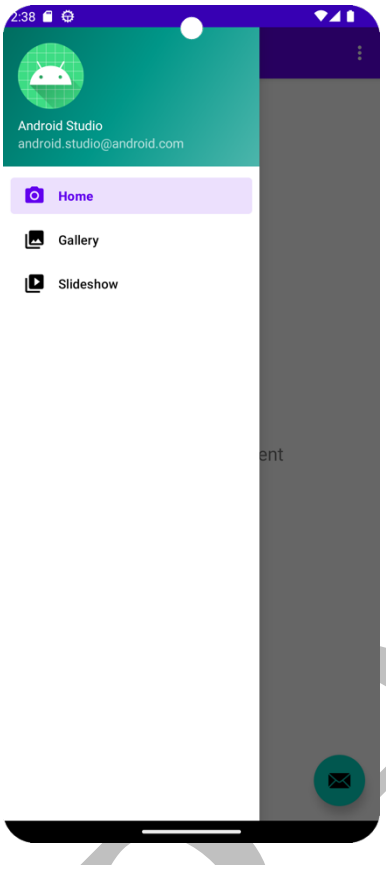


Рисунок 3.1 – Внешний вид навигационного меню

Элементы меню размещены в ресурсах «res | menu | activity_main_drawer.xml»:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    tools:showIn="navigation_view">

    <group android:checkableBehavior="single">
…
        <item
            android:id="@+id/nav_slideshow"
            android:icon="@drawable/ic_menu_slideshow"
            android:title="@string/menu_slideshow" />
    </group>
</menu>
```

В меню стоит обратить внимание на идентификатор пункта меню (напр. *android:id="@+id/nav_slideshow*"). Данное поле будет связано с идентификатором фрагмента в файле, отвечающим за логику отображения элементов. Вся логика переходов между фрагментами размещена в файле *res/navigation/mobile_navigation.xml*, содержащий список всех фрагментов, требующих отображения с идентификатором (рисунок 3.2). Данный файл является навигационным графом, позволяющий отслеживать логику работы приложения. В данном файле указан идентификатор меню, связанный с классом и разметкой экрана

```
<fragment
    android:id="@+id/nav_slideshow"
    android:name="com.mirea.fio.mireaproject.ui.slideshow.SlideshowFragment"
    android:label="@string/menu_slideshow"
    tools:layout="@layout/fragment_slideshow" />
```

Инициализация инструмента Navigation производится в «MainActivity»

```
NavigationView navigationView = binding.navView;
// Passing each menu ID as a set of Ids because each
// menu should be considered as top level destinations.
mAppBarConfiguration = new AppBarConfiguration.Builder(
    R.id.nav_home, R.id.nav_gallery, R.id.nav_slideshow)
    .setOpenableLayout(drawer)
    .build();
NavController navController = Navigation.findNavController(this, R.id.nav_host_fragment_content_main);
NavigationUI.setupActionBarWithNavController(this, navController, mAppBarConfiguration);
NavigationUI.setupWithNavController(navigationView, navController);
```

Предупреждение. Вы можете столкнуться с проблемами при создании **`NavHostFragment`** с помощью **`FragmentContainerView`** или при добавлении **`NavHostFragment`** вручную в свою активность с помощью **`FragmentTransaction`** . Если вы это сделаете, вы можете вызвать сбой **`Navigation.findNavController(Activity, @IdRes int)`** при попытке получить **`NavController`** в **`onCreate()`** . Вместо этого вам следует получить **`NavController`** непосредственно из **`NavHostFragment`**.

```
NavHostFragment navHostFragment = (NavHostFragment)
getSupportFragmentManager().findFragmentById(R.id.nav_host_fragment_content_ma
in);
NavController navController = navHostFragment.getNavController();
```

Navigation состоит из:

– «*NavController*» – основной механизм Navigation Component, отображающий на экране фрагменты. Для этого он должен обладать списками фрагментов и контейнер, в котором он будет эти фрагменты отображать.

– «*NavGraph*» – предназначен для хранения списка фрагментов. NavController отображает фрагменты только из этого списка. Далее обозначается как граф. (res>layout>content_main.xml)

– «*NavHostFragment*» - это контейнер. Внутри него NavController отображает фрагменты.

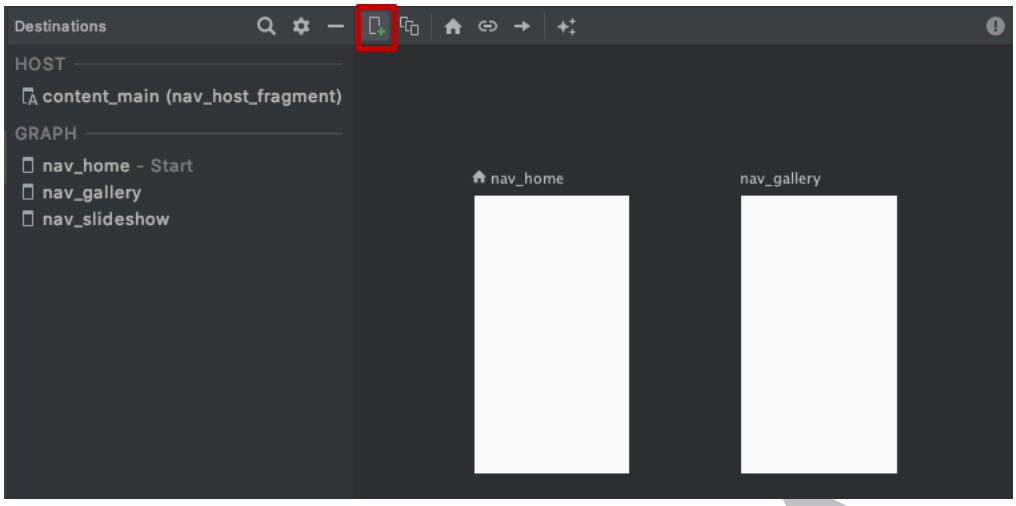


Рисунок 3.2 - Navigation Architecture Component

Для создания нового фрагмента меню требуется в файле mobile_navigation.xml нажать на кнопку выделенной на рисунке 3.3 и выбрать пункт «Create new destination».

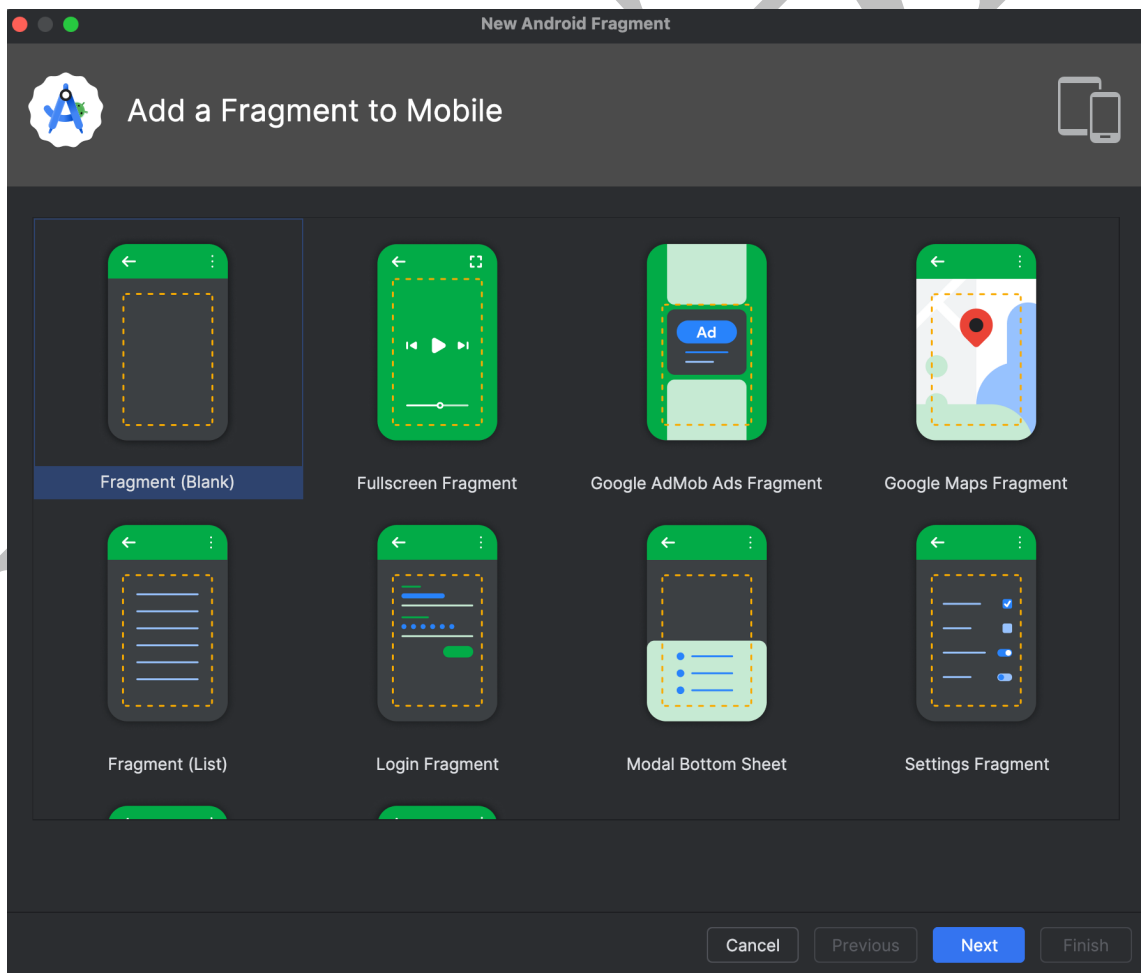


Рисунок 3.3 – Менеджер создания фрагментов

В результате создастся фрагмент в папке «*app|java|com….project|…Fragment*». Данному фрагменту требуется установить идентификатор, который позже будет прописан в файле меню.

Файл mobile_navigation.xml:

```
<fragment
    android:id="@+id/nav_brouser"
    android:name="com.mirea.asd.mireaproject2019.ui.brouser.BrouserFragment"
```

Файл res/menu/activity_main_drawer.xml. Значение иконки и заголовка требуется создать:

```
<item
    android:id="@+id/nav_brouser"
    android:icon="@drawable/ic_menu_share"
    android:title="@string/brouser" />
```

После редактирования файлов, требуется добавить изменения в конфигурацию объекта NavigationUI в MainActivity:

```
mAppBarConfiguration = new AppBarConfiguration.Builder(
        R.id.nav_home, R.id.nav_gallery, R.id.nav_slideshow, R.id.nav_brauser)
        .setDrawerLayout(drawer)
        .build();
```

Для обеспечения возможности создания экранов с пролистыванием требуется заменить «*ConstarintLayout*» на «*NestedScrollView*» в файле разметки «*content_main.xml*»:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.core.widget.NestedScrollView xmlns:android="http://schemas.an-
droid.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:showIn="@layout/app_bar_main">

    <fragment
        android:id="@+id/nav_host_fragment_content_main"
        android:name="androidx.navigation.fragment.NavHostFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:defaultNavHost="true"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:navGraph="@navigation/mobile_navigation" />
</androidx.core.widget.NestedScrollView>
```

В созданном проекте по умолчанию используется подход связывания разметки и кода с помощью класас Binding. Инициализация производится следующим образом:

```
binding = ActivityMainBinding.inflate(getLayoutInflater());
setContentView(binding.getRoot());

setSupportActionBar(binding.appBarMain.toolba
```

«ActivityMainBinding» является сгенерированным классом. Имя данного класса берется из имени «*layout*»-файла (т.е. «*activity_main*») и добавляется слово «*Binding*». Инициализацию графических компонентов производится с помощью вызова экземпляра класса «*Binding*» и поиска по имени.

```
final TextView textView = binding.textGallery;
```