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: 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 НАМЕРЕНИЯ.

Создать новый проект *ru.mirea.«фамилия».practice3*

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 вторую активность — несмотря на то, что эта активность находится в другом приложении (рисунок 1.1). Если намерение запрашивает выполнение какого-либо действия с определенным набором данных, то системе нужно уметь выбрать приложение (или компонент) для обслуживания этого запроса. Для решения этой 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;
- как использовать эти активности?

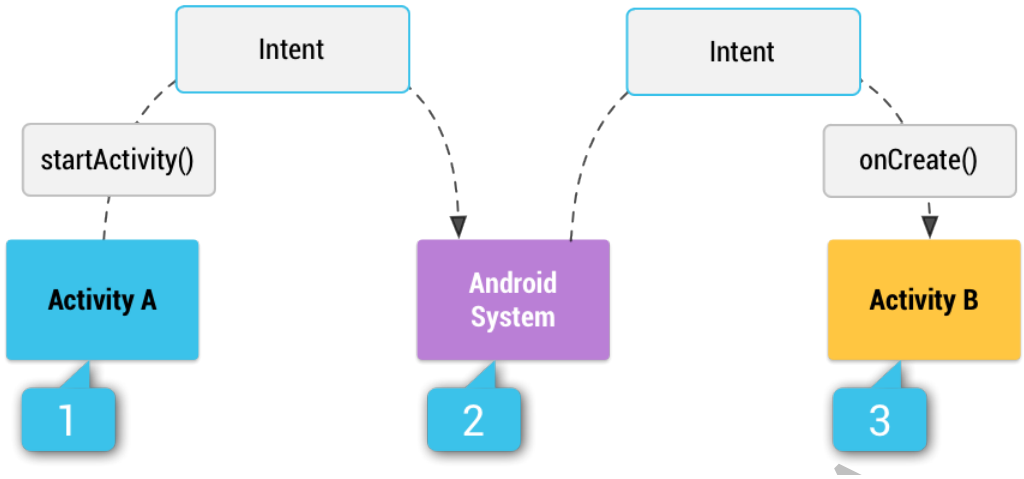


Рисунок 1.1 – Схематическое изображение процесса передачи неявного объекта Intent

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 есть 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 Action

Действия — стандартный механизм, при помощи которого Android узнает о том, какие стандартные операции могут выполняться активностями. Например, 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 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 action. Тем самым Activity оповещает систему о возможности выполнения нескольких функций. Например, не только запись аудио, но и редактирование данной записи. Таким образом получается, что Activity может подойти разным Intent с разными action;

– Activity, которое было вызвано с помощью Intent, имеет доступ к этому Intent 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 можно 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 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, а 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 Data

Данный тег дает возможность указать тип данных, с которым может взаимодействовать компонент приложения. При необходимости возможно задать несколько тегов data. Чтобы указать, какие именно данные поддерживает ваш компонент, используйте сочетание следующих атрибутов:

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

– *android:mimetype* — позволяет указать тип данных, которые ваш компонент способен обрабатывать. Для примера: <type android: value="vnd.android.cursor.dir/*"/> будет соответствовать любому Курсору в Android;

– *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:, mailto: или tel:.).

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 Category

Применение данного поля 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 Intent тэгом <category>. Далее перечислены основные типы категорий:

- CATEGORY_DEFAULT – используется в основном для неявных объектов Intent;
- 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 при нажатии клавиши HOME;
- CATEGORY_LAUNCHER — активность может быть начальной деятельностью задания из списка приложений в группе Application Launcher устройства

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;

1.4 Передача данных с помощью Intent

Для передачи данных между двумя Activity используется объект Intent. Через его метод putExtra() возможно добавить ключ и связанное с ним значение. Область extraData - это 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 включить в intent несколько экземпляров дополнительных данных. При таком способе передачи данных следует обратить внимание, чтобы каждому экземпляру было присвоено уникальное имя.

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

При вызове новой Activity данные следует извлечь из intent. В решении этой задачи используется метод getIntent(). Данный метод возвращает интент, запустивший активность и из полученного объекта возможно прочитать информацию, отправленную вместе с ним. Конкретный способ чтения зависит от типа отправленной информации. Например, если известно, что intent включает строковое значение с именем «message», используется следующий вызов:

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

1.5 Задание

Создать новый модуль/переименовать app. В меню File> New> New Module> Phone & Tablet Module> Empty Activity. Имя модуля IntentApp.

Создайте 2 activity. Проверьте наличие записи о новом activity в manifest - файле. В первом activity требуется получить системное время с помощью следующей функции:

```
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));
```

Передайте время из одного activity в другой и отобразите его в textView во второй activity.

1.6 Share

Создать новый модуль. В меню File> New> New Module> Phone & Tablet Module> Empty Activity. Имя модуля Share.

ACTION_PICK (это обобщённое название для действий с возвращающим значением) назначение 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. Ниже приведённый код выведет диалоговое окно со списком всех возможных программ, которые могут запустить активность с данными (рисунок 1.2), так не указывается конкретный тип (setType("*/*")):

```
Intent intent = new Intent(Intent.ACTION_PICK);
intent.setType("*/*");
startActivityForResult(intent, 1);
```

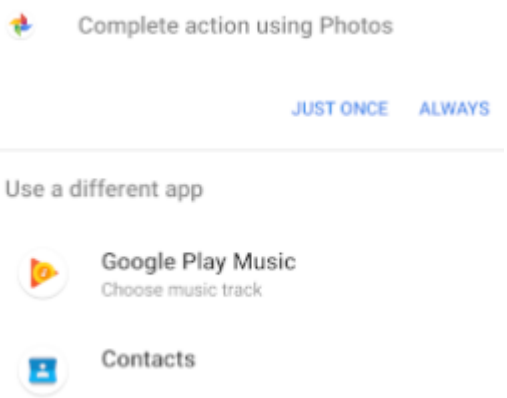


Рисунок 1.2 – Диалоговое окно выбора приложения

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

ACTION_SEND - используется для отправки различных сообщений: электронное письмо, SMS, MMS и т.д. 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. В случае с электронной почтой стандартное приложение в Android также принимает дополнительные параметры по ключам EXTRA_EMAIL, EXTRA_CC, EXTRA_BCC и EXTRA_SUBJECT.

Если есть установленное приложение с фильтром, который соответствует ACTION_SEND и MIME-типу text/plain, система Android запустит его, а если будет найдено более одного приложения, то система отобразит диалог выбора, который позволяет пользователю выбрать приложение.

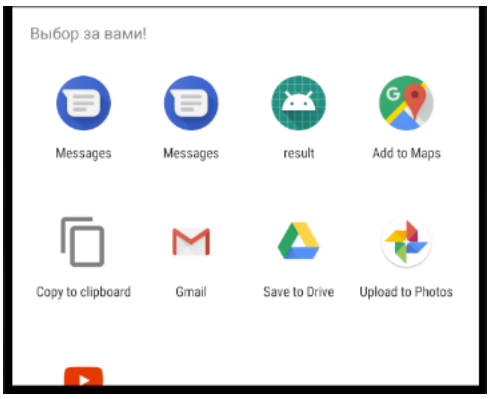
Для того, чтобы разрабатываемая активность имела возможность обрабатывать подобные намерения, требуется содержание следующих значений в манифесте:

```
<activity android:name=".ShareActivity">
        <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>
```

При условии наличия установленного приложения с фильтром, которое соответствует ACTION_SEND и MIME-типу text/plain, система Android запустит его, а если будет найдено более одного приложения, то система отобразит диалог выбора, который позволяет пользователю выбрать приложение. Имя сообщает ОС Android, что активность может обрабатывать ACTION_SEND. Фильтр должен включать категорию DEFAULT, в противном случае он не сможет получать неявные намерения. Указываются типы данных, которые могут обрабатываться активностью.

При выборе из списка программ пользователь может выбрать программу по умолчанию, которая будет автоматически запускаться при выбранном намерении. В данном случае диалоговое окно выводиться не будет. Но можно принудительно выводить диалоговое окно при помощи метода createChooser() и пользователю придётся каждый раз выбирать нужную активность (рисунок1.3):

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



1.7 StartActivityForResult

Рисунок 1.3 – Диалоговое окно выбора приложения

Ранее были представлены намерения или действия, 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.

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. По сути это просто целое число, которое возможно установить произвольно. Оно требуется для того, чтобы различать от кого пришёл результат. Допустим есть пять дополнительных экранов и каждому присваивается значение от 1 до 5. Этот 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.

1.8 Задание

Создать новый модуль. В меню File> New> New Module> Phone & Tablet Module> Empty Activity. Имя модуля ResultActivity.

Создать приложение с двумя экранами. На первом экране отображается вопрос об университете, в котором вы учитесь и кнопка для осуществления перехода к другому экрану. Второй экран предназначена для ввода данных и передачи их в родительских экран.

В созданном модуле создаётся новое activity: New> Activity> Empty activity> DataActivity.

Требуется привести activity_main и activity_data к виду как на рисунках 1.4 и 1.5. На первом экране находится кнопка и два TextView, которые отображают вопрос об учебном заведении и возвращаемое значение.

На втором экране присутствует TextView, Button и EditText для ввода ответа пользователя. При создании разметки кнопкам были установлены значения полей onClick для взаимодействия с java классами.

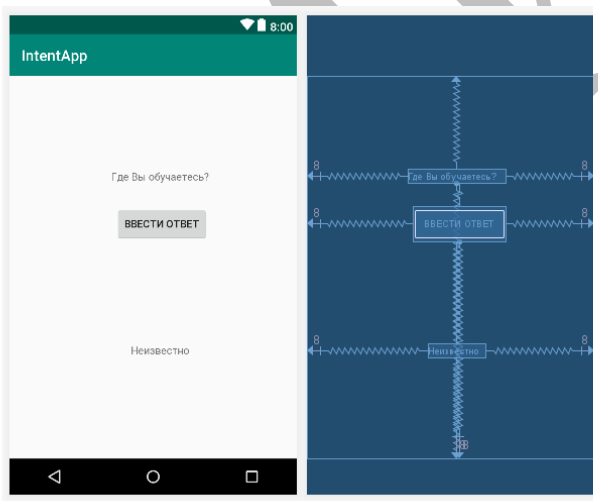


Рисунок 1.5 – activity_main

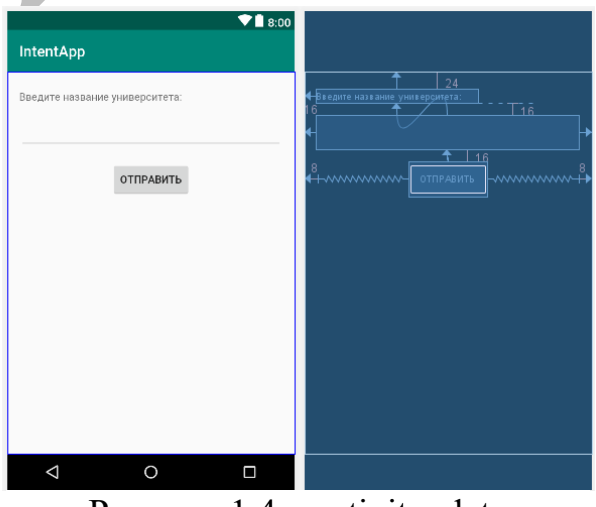


Рисунок 1.4 – activity_data

На рисунке 1.6 представлен алгоритм функционирования приложения.

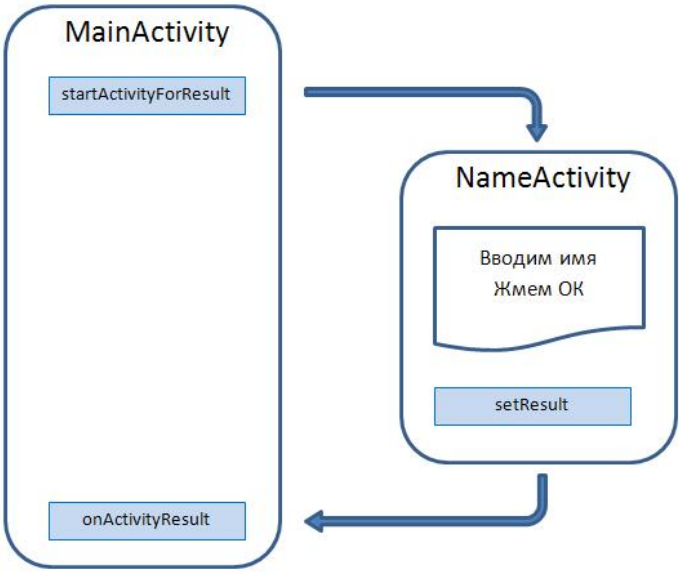


Рисунок 1.6 – Алгоритм функционирования приложения ResultApp

В MainActivity размещается следующий код:

```
public class MainActivity extends AppCompatActivity {
    private TextView textViewResult;
    private final int REQUEST_CODE = 143;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        textViewResult = findViewById(R.id.textViewResult);
    }
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if (data != null){
            String university = data.getStringExtra("name");
            setUniversityTextView(university);
        }
    }

    public void startDataActivityOnClick(View view) {
        Intent intent = new Intent(this, DataActivity.class);
        startActivityForResult(intent, REQUEST_CODE);
    }

    private void setUniversityTextView(String university){
        textViewResult.setText(university);
    }
}
```

Внешний вид главного экрана размещён на рис.1.7. Код MainActivity инициализирует TextView и метод startDataActivityOnClick (вызывается по нажатию кнопки, поле onClick). В методе обработчика startDataActivityOnClick создается Intent с указанным классом DataActivity. Для отправки созданного intent используется startActivityForResult. Отличие от обычного startActivity заключается в

том, что MainActivity становится «родителем» для DataActivity и когда DataActivity закрывается, вызывается метод onActivityResult в MainActivity, тем самым оповещая, что закрылось Activity, которое было вызвано методом startActivityForResult.

Рисунок 1.7 – Внешний вид экрана activity_main.xml

Далее требуется создать класс DataActivity. Внешний вид экрана и последовательность выполнения программы представлены на рисунке 1.8.

```
public class DataActivity extends AppCompatActivity {
    private EditText universityEditText;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_data);
        universityEditText = findViewById(R.id.universityEditText);
    }

    public void sendResultOnMainActivityOnClick(View view) {
        Intent intent = new Intent();
        intent.putExtra("name", universityEditText.getText().toString());
        setResult(RESULT_OK, intent);
        finish();
    }
}
```

В данном классе определяются поле ввода и метод sendResultOnMainActivityOnClick. В данном методе создается Intent и помещается в него данные из поля ввода под именем name. Стоит обратить внимание, что никак не адресуется данный Intent. Метод setResult адресовывает intent в «родительское»

Activity, в котором был вызван метод startActivityForResult. Также в setResult передаётся константа RESULT_OK, означающая успешное завершение вызова. Именно данная константа передастся в параметр resultCode метода onActivityResult в MainActivity.java. Далее методом finish завершается работа DataActivity, чтобы результат ушел в MainActivity.

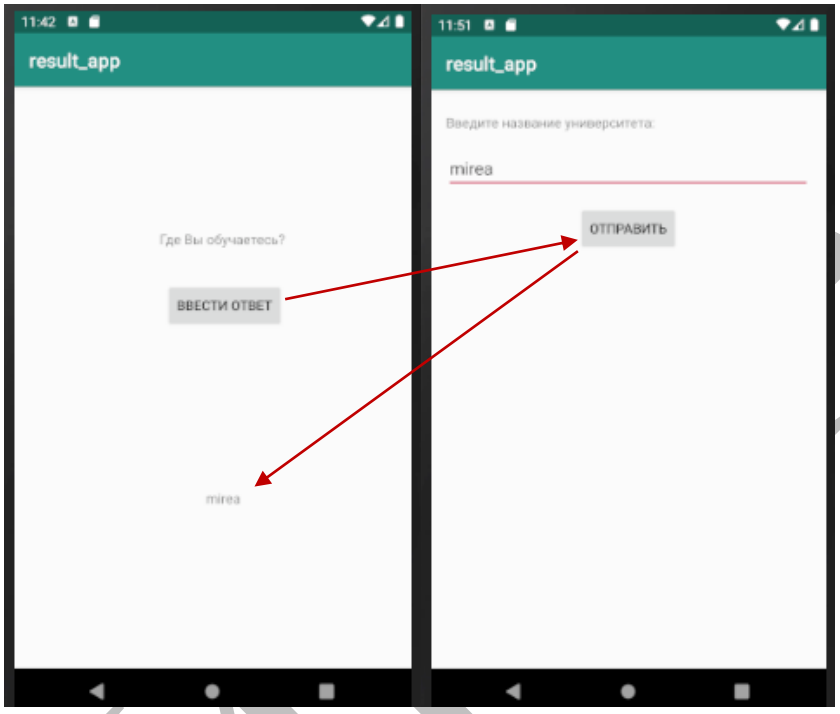


Рисунок 1.8 – Разметка activity_data.xml и activity_main.xml

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.
resultCode – статус вызова. Задается в методе setResult, и проверяется в onActivityResult, чтобы понять насколько успешно прошел вызов. Если при вызове что-то пошло не так, то вернется системная константа RESULT_CANCELED.

1.9 Что такое Uri. Вызов системных приложений.

Intent имеет атрибут action. С помощью данного атрибута указывается действие activity (например, просмотр или редактирование), но действие

совершаются над данными. Значит кроме указания действия, возможно указывать на объект 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.

Ниже представлена таблица с примерами разбора строк, выполненному согласно RFC 2396.

Таблица 1.1 – Пример значений строки URI

URL	**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	**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	**Uri.parse("geo:56.111,37.111");** uri.getScheme(): **geo** uri.getSchemeSpecificPart(): **55.111,37.111**
Number	**Uri.parse("tel:12345");** uri.getScheme(): **tel** uri.getSchemeSpecificPart():**12345**
Contact	**Uri.parse("content://contacts/people/1");** uri.getScheme(): **content** uri.getSchemeSpecificPart(): **//contacts/people/1** uri.getAuthority(): **contacts** uri.getPath(): **/people/1** uri.getLastPathSegment(): **1**

1.10 Задание

Создать новый модуль. В меню File> New> New Module> Phone & Tablet Module> Empty 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. Нужен образ, название которого начинается с "Google APIs". На рис. 1.9 приведен алгоритм установки.

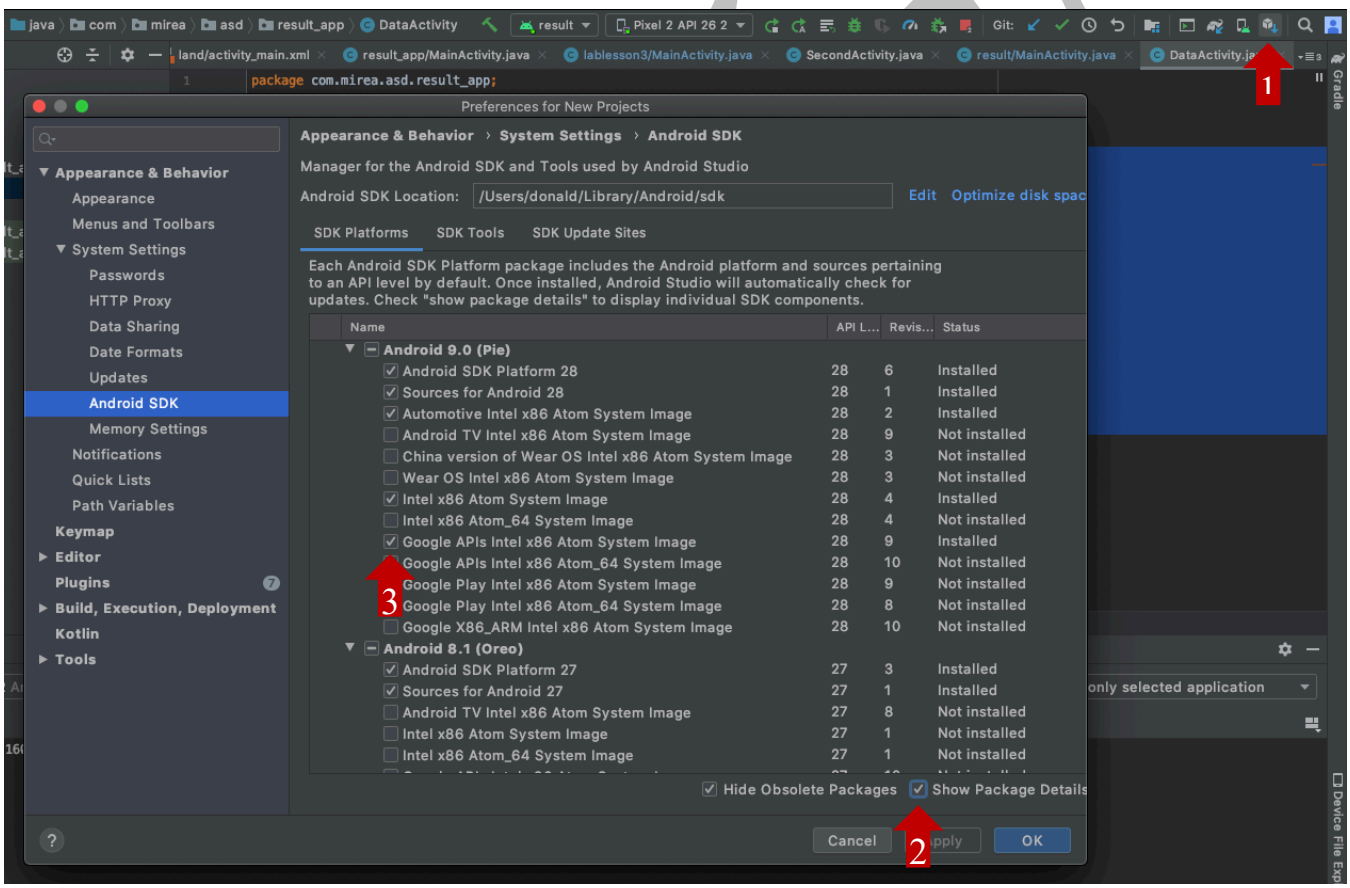


Рисунок 1.9 – Загрузка Google APIs

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

Рисунок 1.10 – Внешний вид приложения SystemIntentsApp

На экране размещены три кнопки:

– «позвонить», метод onClick = «onClickCall»;

– «открыть браузер», onClick = «onClickOpenBrowser»;

– «открыть карту», onClick = «onClickOpenMaps».

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 action - 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 *12345*.

В методе «onClickOpenBrowser» создается Intent и на вход принимается action и data. Используется стандартный системный action – ACTION_VIEW. Это константа в классе Intent – означает, что требуется просмотреть что-либо. 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, что намерение посмотреть карту с указанными координатами в центре экрана.

2 ФРАГМЕНТЫ

Одной из особенностей создания приложений для 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.

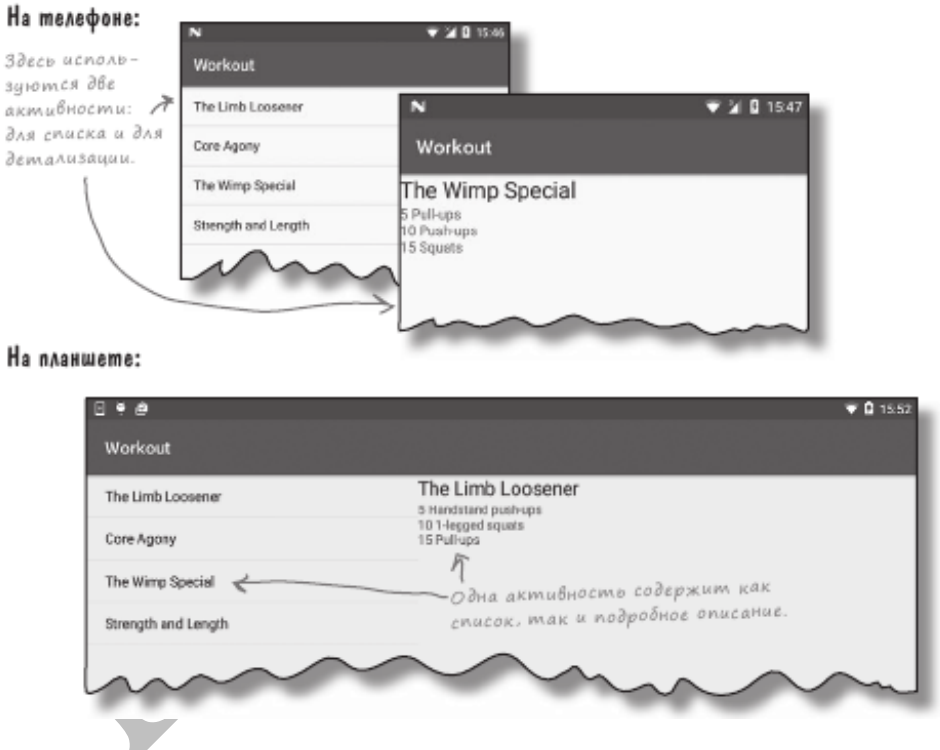


Рисунок 2.1 – Внешний вид приложения на разных устройствах

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). Что же собой представляет фрагмент?

Фрагменты — нечто вроде компонентов, предназначенных для повторного использования, или вторичных активностей. 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.

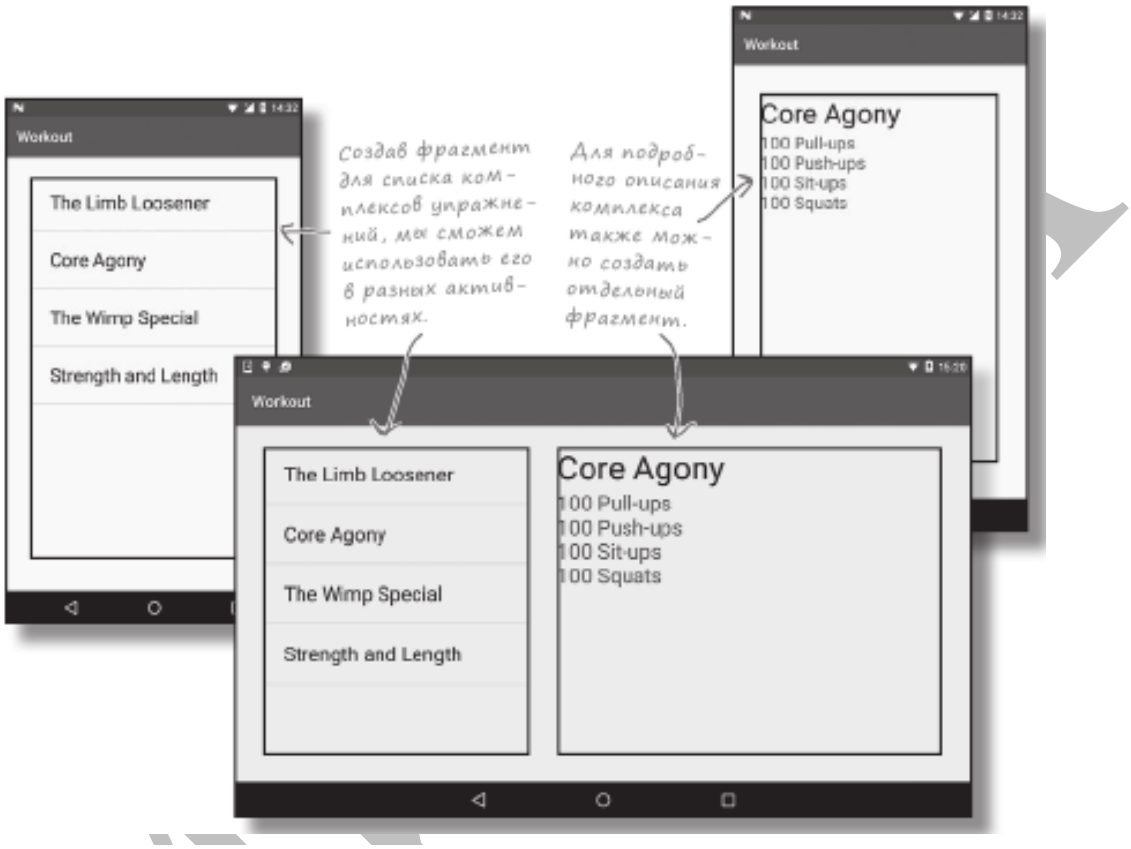


Рисунок 2.2 – Основное назначение фрагментов

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. Но фрагменты - это не замена активности, они не существуют сами по себе, а только в составе активностей. Поэтому в манифесте прописывать их не нужно. Но в отличие от стандартной кнопки, для каждого фрагмента требуется создавать отдельный класс, как для активности.

В составе активности есть 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. Пожалуй, это единственное сходство с активностью.

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 активити, фрагменты должны реализовывать другие методы обратного вызова жизненного цикла, которые позволяют управлять его состоянием, таким как его добавление или удаление из активити, и для обработки переходов между состояниями жизненного цикла активити. Например, когда для активити вызывается onPause() метод, все фрагменты данной активити также получат вызов onPause(). Жизненный цикл компонента Fragment представлен на рисунке 2.4 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).

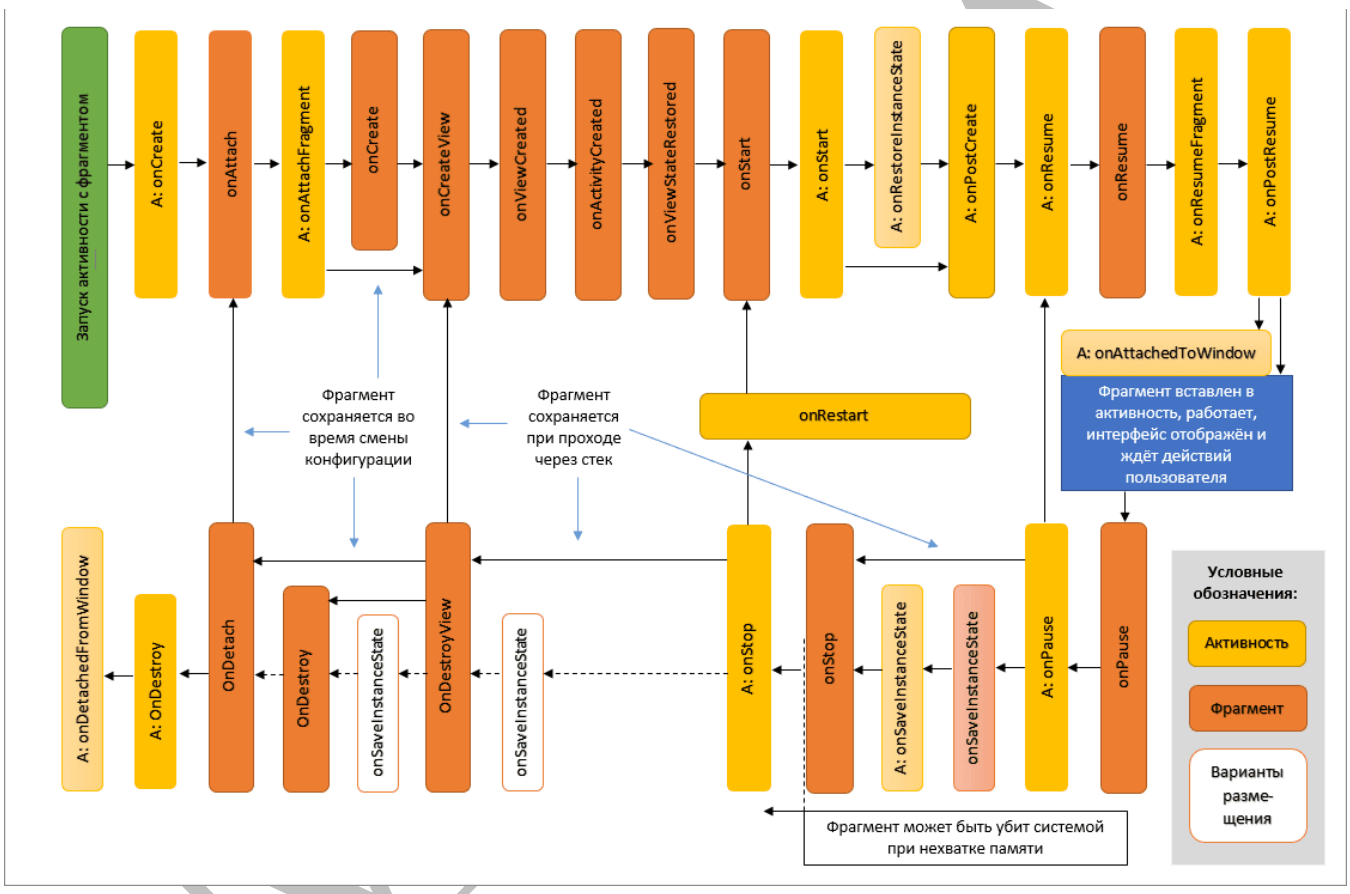


Рисунок 2.3 – Жизненный цикл Fragment совместно с Activity

2.1 Задание

Создать новый модуль. В меню File> New> New Module> Phone & Tablet Module> Empty 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)> name = Fragment1&Fragment2, **set off – include interface callbacks**.

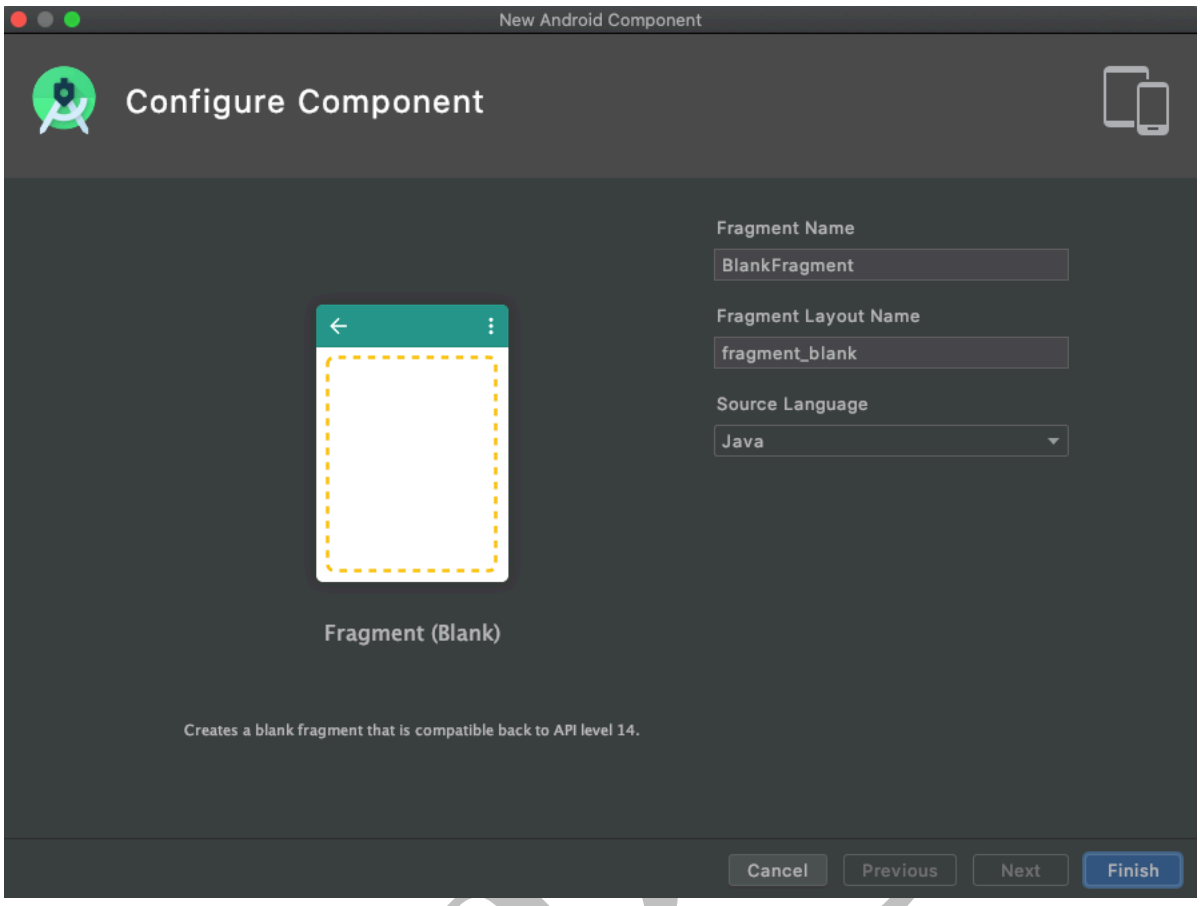


Рисунок 2.4 – Менеджер создания фрагментов

Далее рассматривается файл разметки res>layout>fragment_blank_fragment1.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>
```

Привычное пространство имён xmlns:android позволяет настраивать внешний вид и поведение компонентов в Android-приложении. А новое пространство имён xmlns:tools позволяет среде разработки (в нашем случае, Android Studio) правильно отобразить компоненты для просмотра в режиме дизайна.

Пример. Имется компонент TextView, который получает текст с сервера. Чтобы визуально представить себе, как будет выглядеть текст, не соединяясь 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. Помогает среде сформировать в окне предварительного просмотра внешний вид, устанавливая требуемую тему. Также для наглядности установлен фоновый цвет фрагмента. В файле *res>layout>fragment_blank_fragment2.xml* требуется также изменить цвет фона.

Далее рассмотрен класс BlankFragment1:

```
public class BlankFragment1 extends Fragment {
 ..........

    // TODO: Rename and change types and number of parameters
    public static BlankFragment1 newInstance(String param1, String param2) {
        BlankFragment1 fragment = new BlankFragment1();
        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_blank_fragment1, 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» представления означает получение XML макета и его синтаксический анализ для создания объектов представления и viewgroup из элементов и их атрибутов, указанных внутри, а затем добавление иерархии этих представлений и viewgroups в родительскую ViewGroup. При вызове setContentView() производится присоединение к действию представления, созданные при чтении XML. Также возможно использовать LayoutInflater.

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://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"
    tools:context=".MainActivity">
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        app:layout_constraintBottom_toBottomOf="@+id/fragmentContainer"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="@+id/fragmentContainer">

        <Button
            android:id="@+id/btnFragment2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onClick"
            android:text="fragment1" />

        <Button
            android:id="@+id/btnFragment1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onClick"
            android:text="fragment2" />
    </LinearLayout>
    <FrameLayout
        android:id="@+id/fragmentContainer"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    </FrameLayout>
</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 {
    Fragment fragment1, fragment2;
    FragmentManager fragmentManager;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        fragment1 = new BlankFragment1();
        fragment2 = new BlankFragment2();
    }

    public void onClick(View view) {
        fragmentManager = getSupportFragmentManager();
        switch (view.getId()){
            case R.id.btnFragment1:
                fragmentManager.beginTransaction().replace(R.id.fragmentContainer,
fragment1).commit();
                break;
            case R.id.btnFragment2:
                fragmentManager.beginTransaction().replace(R.id.fragmentContainer,
fragment2).commit();
                break;

                default:
                    break;
        }
    }
}
```

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* и установить значения (рисунок 2.5):

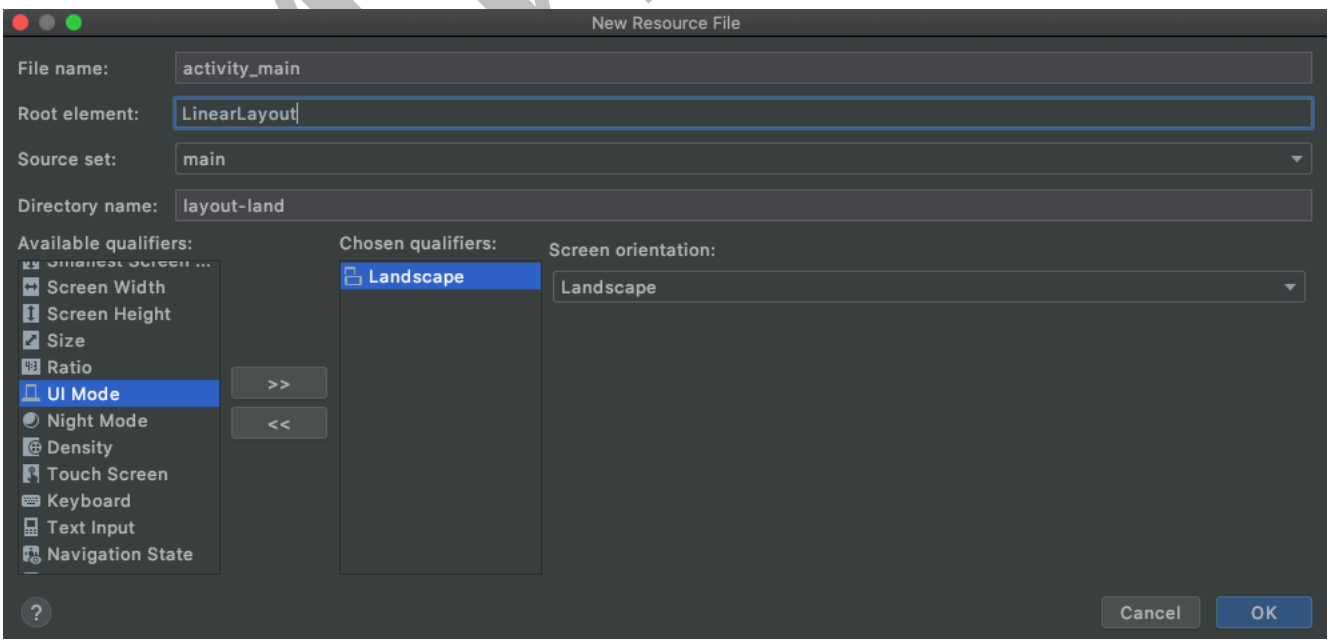


Рисунок 2.5 – Создание горизонтального файла разметки activity_main.xml

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):

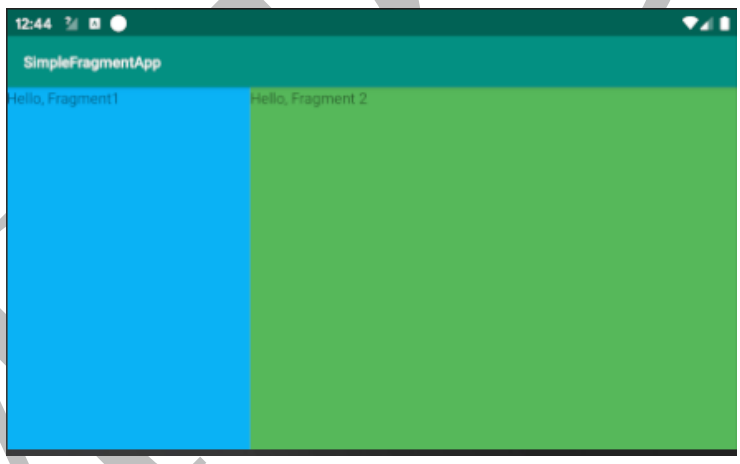


Рисунок 2.6 – Горизонтальная ориентация приложения

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.

– CalculateFragment – **создать простейший калькулятор** с выполнением основных арифметическихы функций;

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

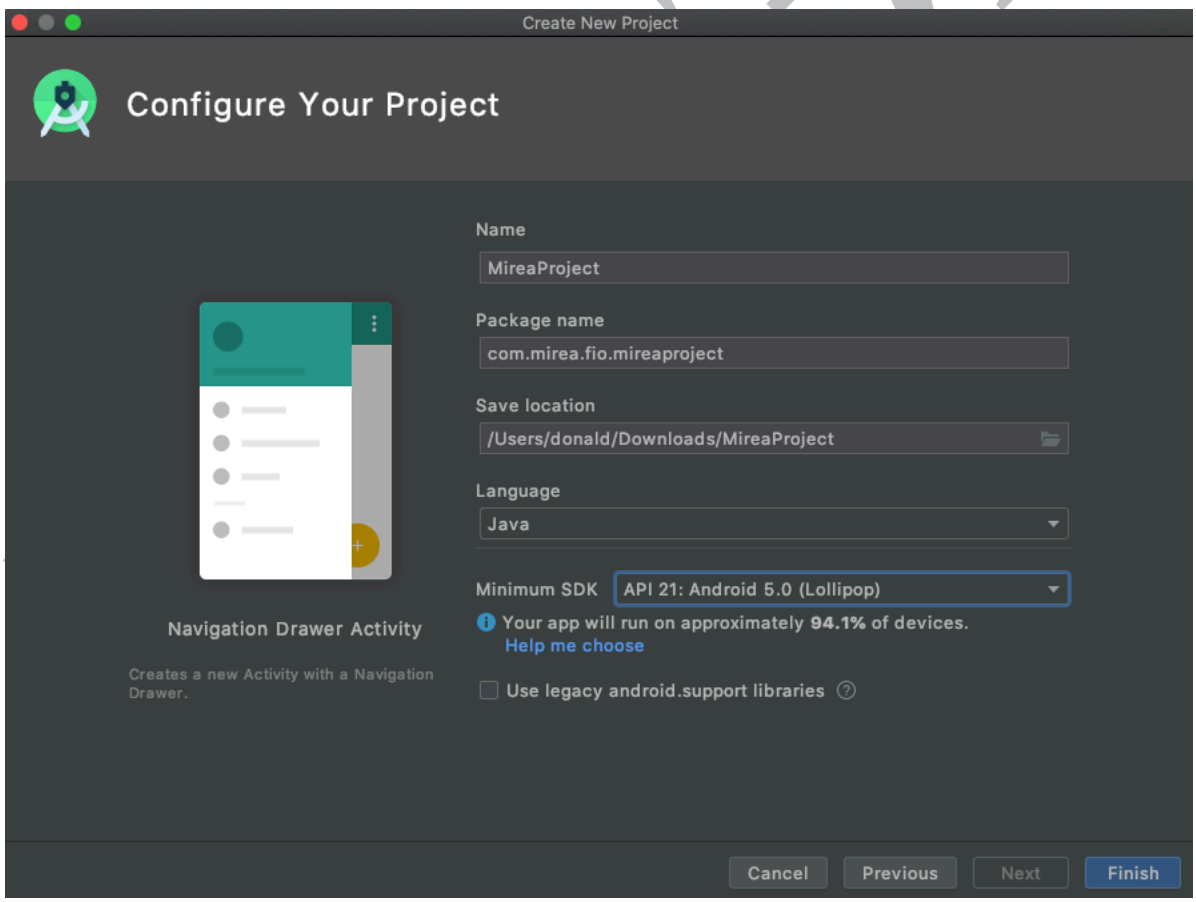


Рисунок 3.1 – Менеджер создания нового проекта

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. Возможно подвигать шторку вперёд-назад, чтобы увидеть, что верхняя кромка шторки в системной области полупрозрачна и не закрывает системные значки. Подобное поведение доступно на устройствах под Android 5 и выше. На старых устройствах шторка находится под системной панелью.

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 позволяет указать студии, что навигационное меню требуется отобразить в раскрытом виде в режиме просмотра разметки.

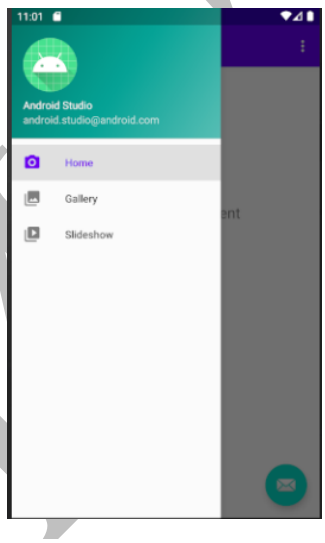


Рисунок 3.2 – Внешний вид навигационного меню

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 3.3). 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

```
NavController navController = Navigation.findNavController(this, R.id.nav_host_fragment);
NavigationUI.setupActionBarWithNavController(this, navController, mAppBarConfiguration);
NavigationUI.setupWithNavController(navigationView, navController);
```

Navigation состоит из:

- NavController - основной механизм Navigation Component, отображающий на экране фрагменты. Для этого он должен обладать спискам 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.

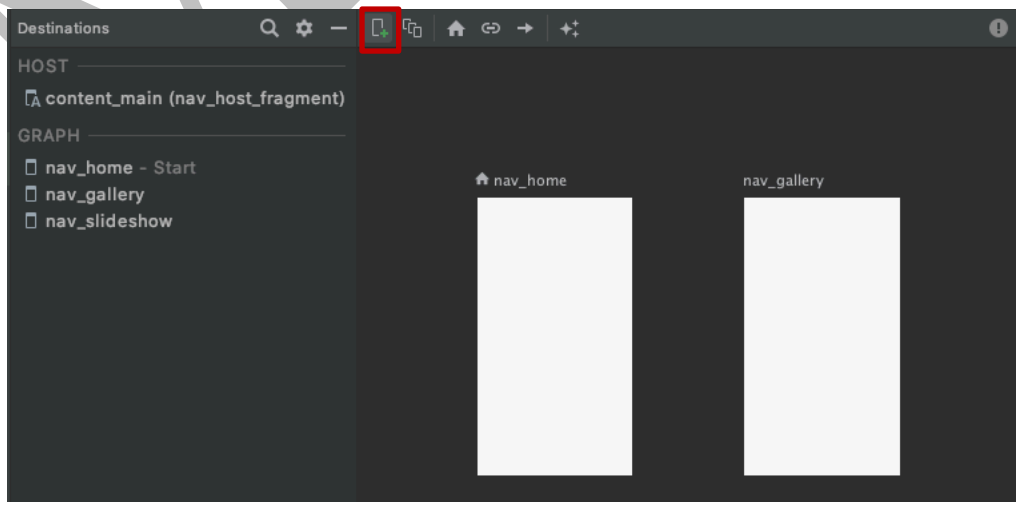


Рисунок 3.3 - Navigation Architecture Component

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».

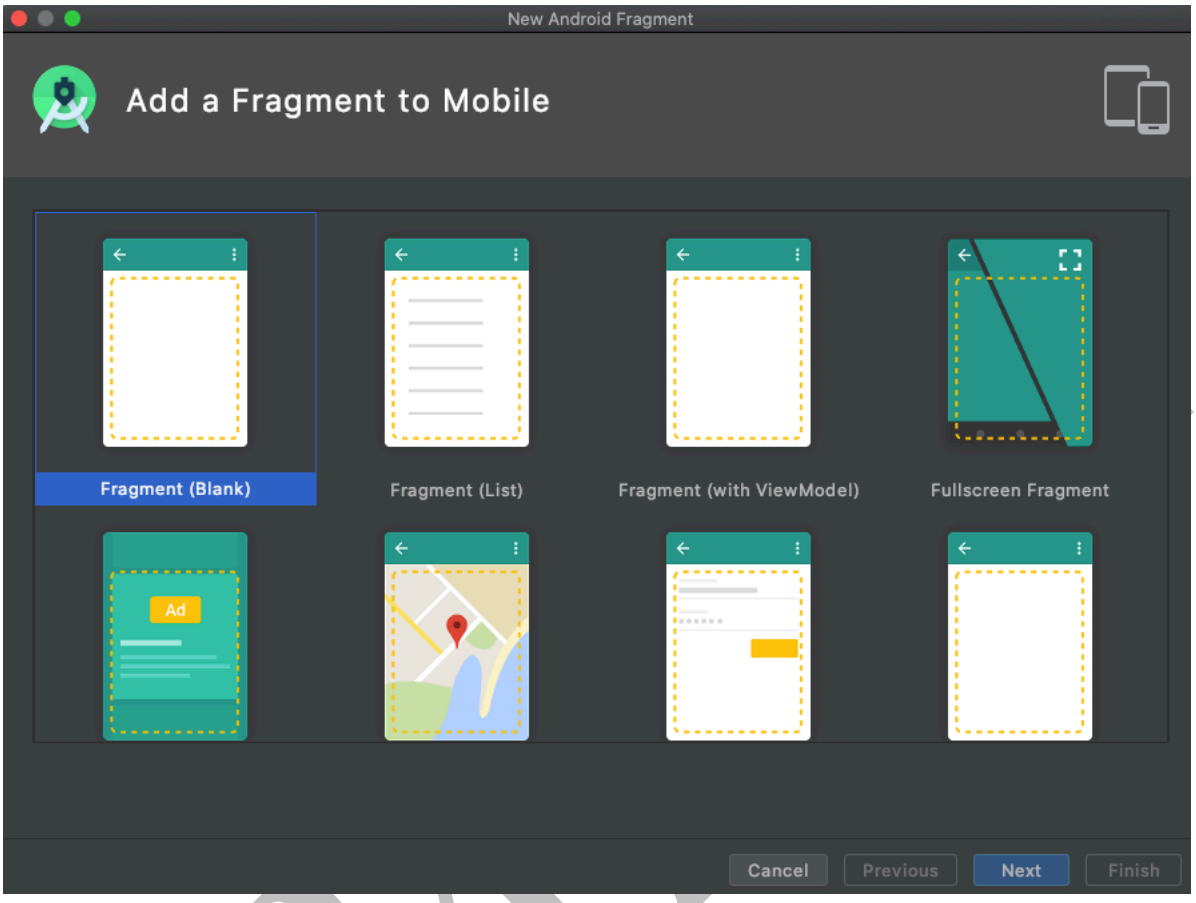


Рисунок 3.4 – Менеджер создания фрагментов

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();
```