

Технологии разработки мобильных приложений

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 3

УЧЕБНЫЕ ВОПРОСЫ

1	Намерения	2
1.1	Action.....	3
1.2	Data.....	5
1.3	Category	6
1.4	Передача данных с помощью Intent	7
1.5	Методы использования Intent.....	8
1.6	Возвращение результата. Метод startActivityForResult.....	11
1.7	Задание	11
1.8	Что такое Uri. Вызов системных приложений.....	15
1.9	Задание	16
2	Фрагменты	19
2.1	Задание	21
3	КОНТРОЛЬНОЕ ЗАДАНИЕ – Веб-браузер	24

1 Намерения.

На прошлом практическом занятии был изучен способ вызова Activity, основой которого являются action, data, category и Intent Filter. 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 (рис. 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 этих 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 передать ей текст, который нужно отправить. Вместо того, чтобы создавать собственные методы для отправки электронной почты, можно 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?

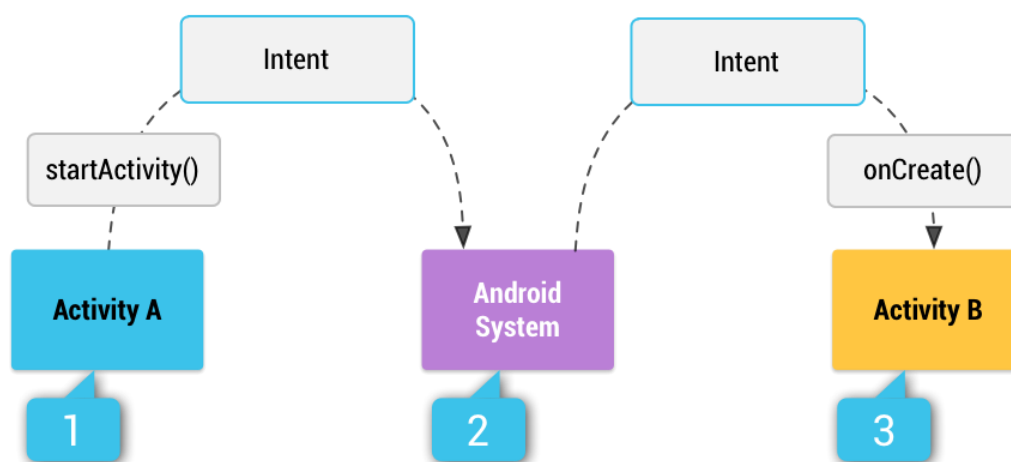


Рисунок 1.1 Схематическое изображение процесса передачи неявного объекта Intent

Чтобы зарегистрировать компонент приложения в качестве потенциального обработчика намерений, нужно добавить тег `<intent-filter>` в узел компонента в манифесте. 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 (рис. 1.2):

```
<activity-alias
    android:name=".activities.MainActivity"
    android:enabled="false"
    android:icon="@mipmap/ic_launcher_pink"
    android:roundIcon="@mipmap/ic_launcher_pink"
    android:targetActivity=".activities.SplashActivity">

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

Рисунок 1.2 - Использование *intent-filter*

Фильтр такого вида в элементе `<action>` помечает активность, как запускаемую по умолчанию. Элемент `<category>` заставляет значок и метку для деятельности отображаться на панели Application Launcher, давая пользователям возможность запускать задание и возвращаться к этому заданию в любое время после того, как оно было запущено.

1.1 Action

Все эти вопросы решаются при помощи действий (actions). 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 нашего фильтра, то наше 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 будет, и активность

запустится автоматически. Какие действия активностей можно 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 Вашего 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:

- Intent Filter может содержать в себе несколько action. Тем самым Activity оповещает систему о возможности выполнения нескольких функций. Например, не только запись аудио, но и редактирование этой 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 — Открывает активность, которая связана с входящими звонками. Это действие обрабатывается стандартным экраном для приема звонков;

ACTION_CALL — инициализирует обращение по телефону;

ACTION_DELETE — Запускает активность, с помощью которой можно удалить данные, указанные в пути URI внутри намерения;

ACTION_EDIT — Отображает данные для редактирования пользователем;

ACTION_INSERT — Открывает 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 - Загружает дочернюю Активность, позволяющую 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 — Запускает 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. Используйте 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`. Используйте действие *ACTION_SEND* только в тех случаях, когда данные нужно передать удаленному адресату (а не другой программе на том же устройстве);

ACTION_SENDTO — Открывает активность для отправки сообщений контакту, указанному в пути URI, который передаётся через намерение;

ACTION_SYNC — Синхронизирует 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 — Открывает 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 можно задать несколько тегов `data`. Чтобы указать, какие именно данные

поддерживает ваш компонент, используйте сочетание следующих атрибутов:

- *android:host* — задает 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 составлена 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 вам не нужно декларировать специфику данных Uri(например, когда операция использует другие виды дополнительных данных вместо URI), вы должны указать только атрибут android:mimeType для декларирования типа данных, с которыми работает ваша операция, например, text/plain или image/jpeg. Если в Фiltro намерений не указано ни одного параметра data, его действие будет распространяться на любые данные.

1.3 Category

– дополнительный способ описания 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 в основном используется 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("сообщение", "значение");
```

где `сообщение` — имя ресурса для передаваемой информации, а `значение` — само 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`.

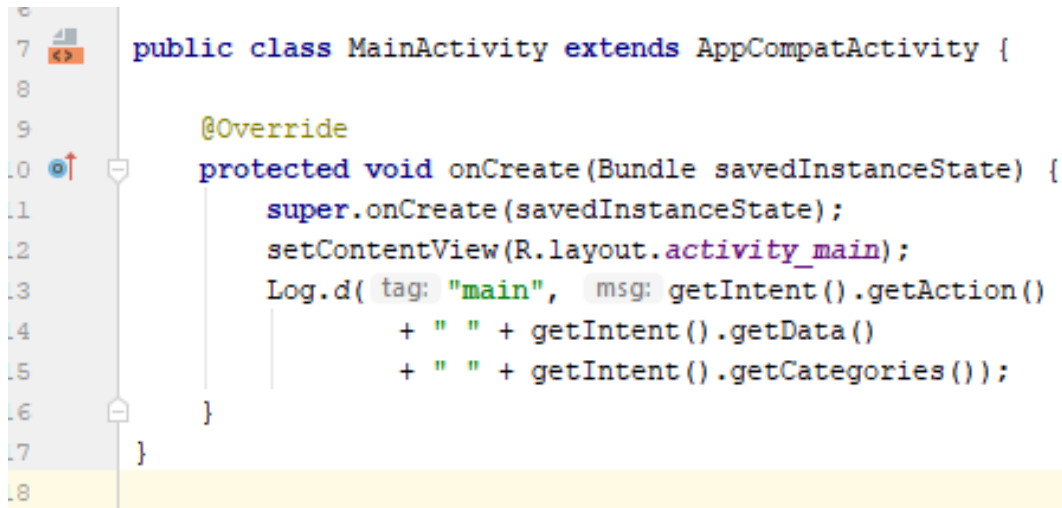
При вызове новой Activity данные следует извлечь из `intent`. В решении этой задачи используется метод `getIntent()`. Данный метод возвращает интент, запустивший активность и из полученного объекта возможно прочитать информацию, отправленную вместе с ним. Конкретный способ чтения зависит от типа отправленной 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 Методы использования Intent

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

Когда программный компонент запускается с помощью неявного намерения, он должен найти действие, которое необходимо осуществить, и данные для него.



```
7 public class MainActivity extends AppCompatActivity {  
8  
9     @Override  
10    protected void onCreate(Bundle savedInstanceState) {  
11        super.onCreate(savedInstanceState);  
12        setContentView(R.layout.activity_main);  
13        Log.d( tag: "main", msg: getIntent().getAction()  
14            + " " + getIntent().getData()  
15            + " " + getIntent().getCategories());  
16    }  
17 }  
18
```

Рисунок 1.3 – Информация о намерении, с помощью которого запущен компонент

Применяйте методы `getAction()` и `getData()`, чтобы найти действие и данные, связанные с намерением (рис. 1.3). Для извлечения дополнительной информации, хранящейся в параметре `extras`, используйте типизированные методы `get<тип>Extra`.

Какой результат Вы увидели в консоли?

До сих пор были рассмотрены намерения или действия, которые активируют другую активность, не ожидая получить в ответ на это результат. Рассмотрим более сложное действие, возвращающее значение после того, как будет активировано.

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, можно многократно использовать функцию UI для выбора нескольких элементов определенного типа. Код на рис. 1.4 выведет диалоговое окно со списком всех возможных программ, которые могут запустить активность с данными, так не указывается конкретный тип (`setType("*/*")`):

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Intent intent = new Intent(Intent.ACTION_PICK);
    intent.setType("*/*");
    startActivityForResult(intent, requestCode: 1);
}
```

Рисунок 1.4 - ActionPick

Какой результат Вы увидели на экране эмулятора?

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`.

Для того, чтобы разрабатываемая активность имела возможность обрабатывать подобные намерения, требуется содержание следующих значений в манифесте (рис. 1.5):

```

<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>

```

Рисунок 1.5 - Объявления Intent в манифесте

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 этом случае диалоговое окно выводиться не будет. Но можно принудительно выводить диалоговое окно при помощи метода `createChooser()` и пользователю придётся каждый раз выбирать нужную активность:

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Intent intent = new Intent(Intent.ACTION_SEND);
    intent.setType("text/plain");
    intent.putExtra(Intent.EXTRA_TEXT, value: "мирэа");
    Intent chosenIntent = Intent.createChooser(intent, title: "Заголовок в диалоговом окне");
    startActivity(chosenIntent);
}

```

Рисунок 1.6 - Вызов диалогового окна

Какой результат Вы увидели на экране эмулятора?

1.6 Возвращение результата. Метод `startActivityResult`

В некоторых случаях после запуска операции может потребоваться получить результат. Например – при создании SMS, производится выбор адресата, система показывает экран со списком из адресной книги, вы выбираете нужного вам абонента и возвращаетесь в экран создания SMS номер контакта. Т.е. вы вызвали экран выбора абонента, а он вернул вашему экрану результат.

Для данной реализации необходимо вызвать метод `startActivityResult()`. Метод `startActivityResult(Intent, int)` со вторым параметром, идентифицирующим запрос, позволяет возвращать результат. Разница между методами `startActivity` и `startActivityResult` заключается в дополнительном параметре `requestCode`. По сути это просто целое число, которое вы можете сами придумать. 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` мы видим следующие параметры:

`requestCode` – тот же идентификатор, что и в `startActivityResult`. По нему определяем, с какого Activity пришел результат.

`resultCode` – код возврата. Определяет успешно прошел вызов или нет.

`data` – `Intent`, в котором возвращаются данные

1.7 Задание

Примечание: Создать новый модуль. В меню `File > New > New Module > Phone & Tablet Module > Empty Activity`. Имя модуля `ActivityResultApp`.

Создадим приложение с двумя экранами. На первом экране будет отображаться 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 создадим новое activity: `New > Activity > Empty activity > DataActivity`.

Требуется привести `activity_main` и `activity_data` к виду как на рис. 1.7 и 1.8. 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.

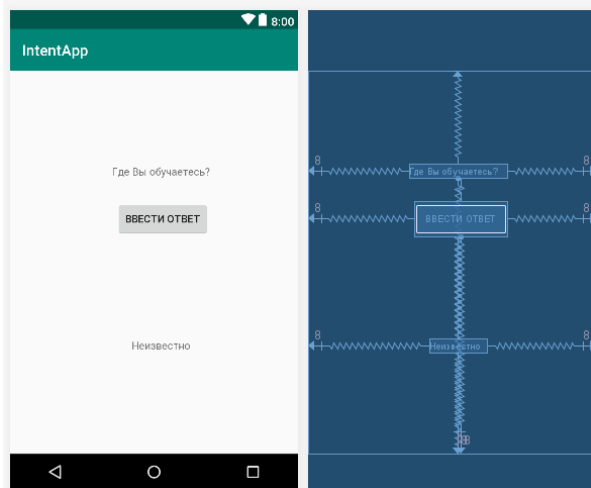


Рисунок 1.7 - activity_main

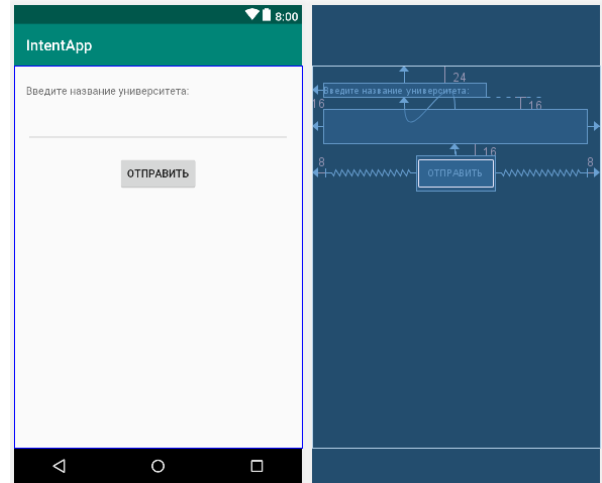


Рисунок 1.8 - activity_data

```
public class MainActivity extends AppCompatActivity {
    private TextView textViewResult;
    private final static int requestCode = 150;

    @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: "name");
            setUniversityInTextView(university);
        }
    }

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

    // установка значения для текстового поля
    public void setUniversityInTextView(String university) { textViewResult.setText(university); }
}
```

Рисунок 1.9 - MainActivity

Разберем код MainActivity, приведенный на рис. 1.9. Определяем TextView и метод startDataActivityOnClick (вызывается по нажатию кнопки, поле onClick). В методе обработчика startDataActivityOnClick создаем Intent, указываем класс DataActivity. Для отправки используем 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.

Рассмотри класс DataActivity (рис. 1.20). В данном классе определяем поле ввода и метод sendResultOnMainActivityOnClick. В данном методе создается Intent и помещается в него данные из поля ввода под именем name. Обратите внимание, мы никак не адресуем этот Intent. Т.е. ни класс,

```
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();  
    }  
}
```

Рисунок 1.10 - DataActivity

ни action мы не указываем, т.е. неизвестно куда пойдет этот Intent. Но метод setResult знает, куда его адресовать - в «родительское» Activity, в котором был вызван метод startActivityForResult. Также в setResult мы передаем константу RESULT_OK, означающую успешное завершение вызова. И именно она передастся в параметр resultCode метода onActivityResult в MainActivity.java. Далее методом finish мы завершаем работу DataActivity, чтобы результат ушел в MainActivity.

Запустим приложение.

Таким образом, в MainActivity мы создали Intent с явным указанием на класс DataActivity. Запустили этот Intent с помощью метода startActivityForResult. DataActivity отобразилось, мы ввели название университета и нажали кнопку. Создался Intent, в который поместилось введенное нами название. Метод setResult знает, что Intent надо вернуть в Activity, которое выполнило вызов startActivityForResult, т.е. – MainActivity. В MainActivity за прием результатов с вызванных Activity отвечает метод

onActivityResult. В нем мы распаковали *Intent* и отображали полученные данные в *TextView*.

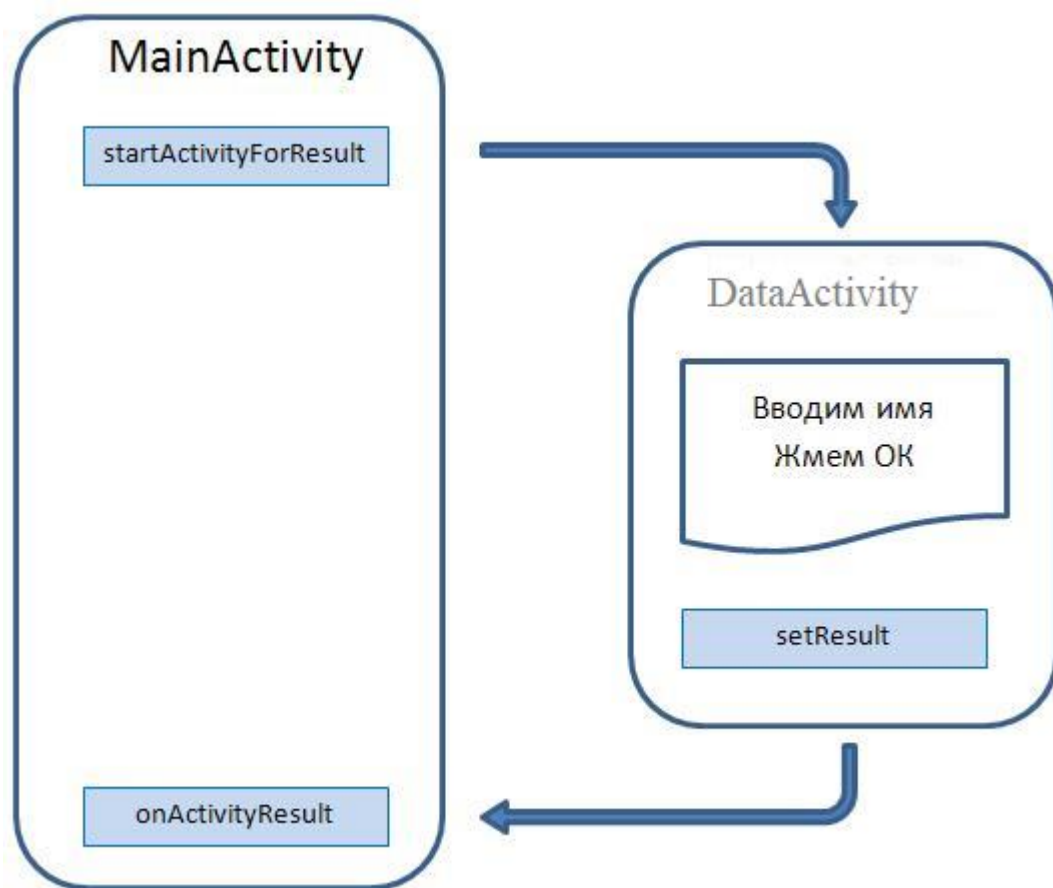


Рисунок 1.11 – Схема взаимодействия activity через *startActivityResult*

Итак, подведем итог:

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 *startActivityResult*, 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*.

1.8 Что такое Uri. Вызов системных приложений.

Intent имеет атрибут action. С помощью данного атрибута указывается действие нашего 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 эту информацию. Строка составляется в соответствии с документом RFC 2396. Uri имеет кучу методов, которые позволяют извлекать из разобранной строки отдельные элементы.

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

Таблица 1

URL	<pre>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</pre>
FTP	<pre>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</pre>
Geo	<pre>Uri.parse("geo:56.111,37.111"); uri.getScheme(): geo uri.getSchemeSpecificPart(): 55.111,37.111</pre>

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

В таблице 1 в примере с «Contact» scheme равен content. Это особый тип данных – Content Provider. Он позволяет любой программе давать доступ к своим данным, а другим программам – читать и менять эти данные.

Примеры показывают, что Uri можно создать из абсолютно разных строк: http-адрес, ftp-адрес, координаты, номер телефона, контакт из адресной книги.

Тип содержимого можно определить по Scheme. И этот же Scheme можно настроить в Intent Filter и отсеивать Intent, только с требуемым типом данных в Uri, например, только http.

1.9 Задание

Создадим приложение, позволяющее просмотреть следующее: http-адрес, координаты на карте и окно набора номера.

Примечание: Создать новый модуль. В меню File> New> New Module> Phone & Tablet Module> Имя модуля SystemIntentsApp. **Указываем minimumSDK – API 23: Google APIs, Android 23 > Empty Activity..**

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. 1.12 приведен алгоритм установки.

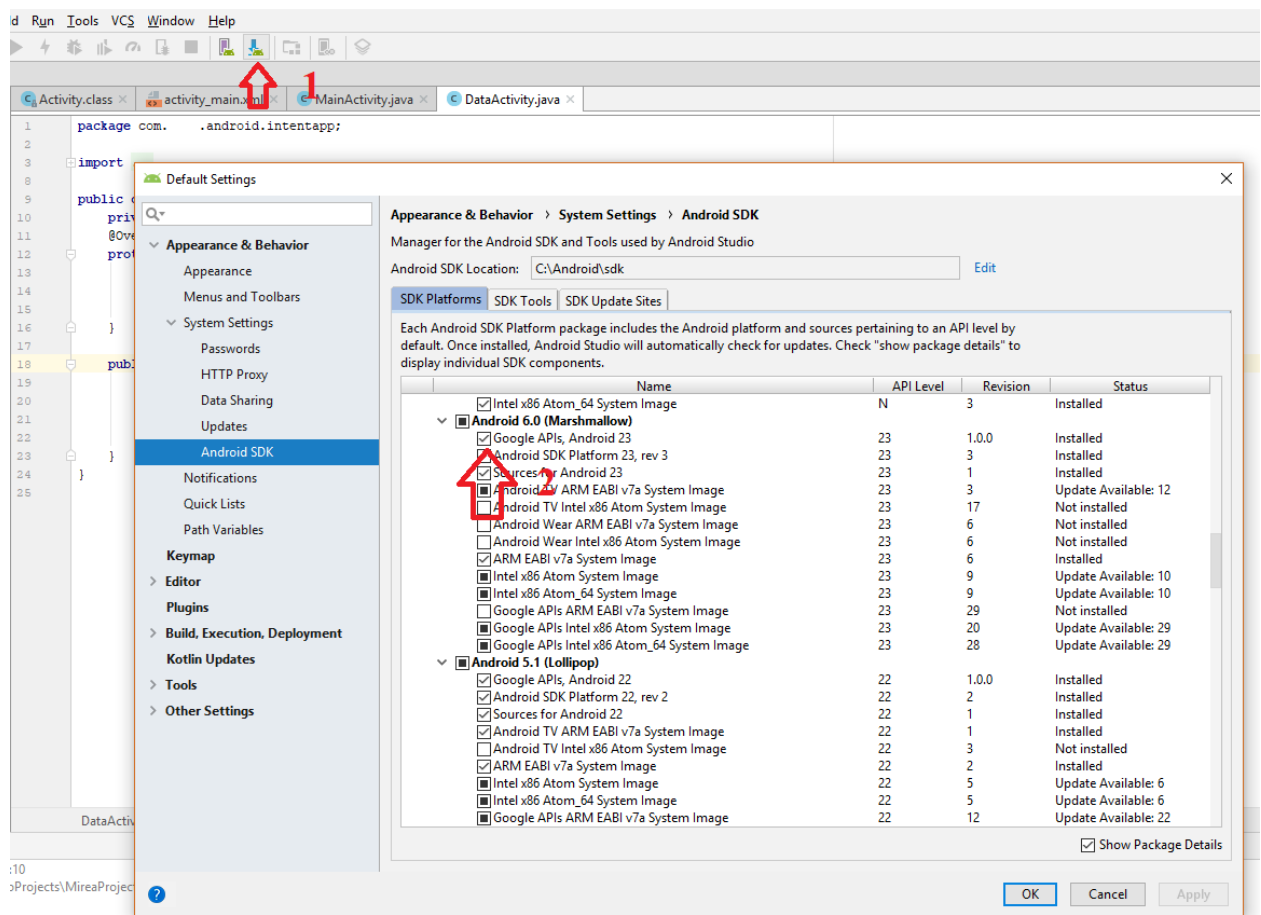


Рисунок 1.12 - Установка Google APIs

Сформируем экран activity_main.xml

На экране три кнопки:

- «позвонить», метод onClick="onClickCall"
- «открыть браузер», onClick="onClickBrowser"
- «открыть карту», onClick="onClickMaps"

Рассмотрим код MainActivity (рис. 1.14) использовались три различных способа создания Intent-а и задания его атрибутов.

В случае onClickBrowser использовался конструктор Intent (String action, Uri uri). Он создает Intent и на вход сразу принимает action и data. Мы используем стандартный системный action – ACTION_VIEW. Это константа в классе Intent – означает, что требуется просмотреть



Рисунок 1.13- activity_main.xml

что-либо. В качестве data мы подаем объект Uri, созданный из веб-ссылки: <http://developer.android.com>.

```
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:12345"));  
        startActivity(intent);  
    }  
  
    public void onClickBrowser(View view) {  
        Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://developer.android.com"));  
        startActivity(intent);  
    }  
  
    public void onClickMaps(View view) {  
        Intent intent = new Intent();  
        intent.setAction(Intent.ACTION_VIEW);  
        intent.setData(Uri.parse("geo:55.749479,37.613944"));  
        startActivity(intent);  
    }  
}
```

В случае `onClickMaps` использовался конструктор `Intent()`. Он просто создает `Intent`, а в следующих строках мы уже присваиваем ему атрибуты `action` и `data`. `action` – снова `ACTION_VIEW`, а в качестве `data` мы создаем `Uri` из пары координат - 55.749479,37.613944. Этот `Intent` означает, что намерение посмотреть карту с указанными координатами в центре экрана.

В случае `onClickCall` используется конструктор `Intent (String action)`. На вход подается `action`, а `data` указывается позже. `action` в данном случае – `ACTION_DIAL` – открывает набор номера и набирает номер, указанный в `data`, но не начинает звонок. В `data` – помещаем `Uri`, созданный из номера телефона 12345.

Три этих способа приводят к одному результату - `Intent` с заполненными атрибутами `action` и `data`. Какой из них использовать - решать вам в зависимости от ситуации.

Запускаем приложение.

2 Фрагменты

Одна из самых замечательных особенностей программирования для Android — 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, можно 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 этом приложении необходимо предоставить одну активность для планшетов и две активности для телефонов (рис. 2.1).

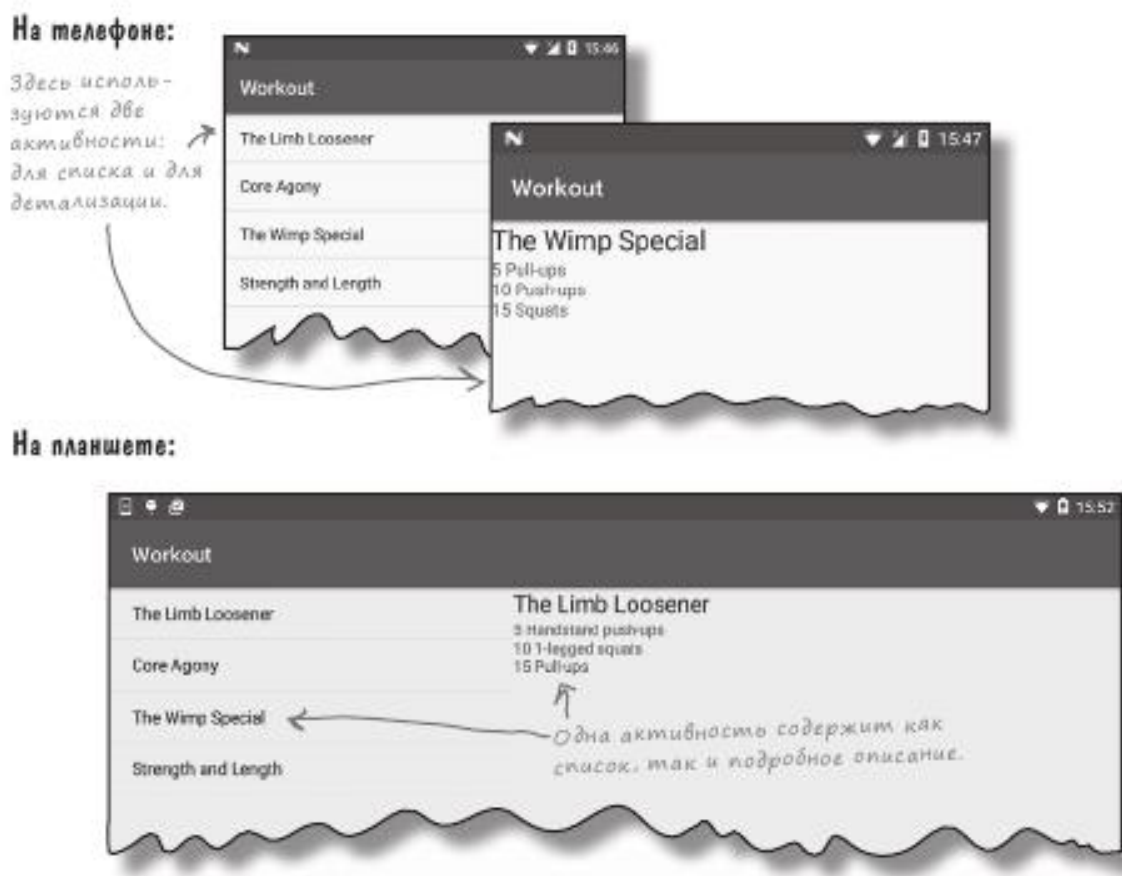


Рисунок 2.1 - Внешний вид приложения на разных устройствах

Но это может привести к дублированию кода. Второй активности, которая работает только на телефонах, потребуется вставить подробное описание в макет. Однако этот же код также должен присутствовать и в основной активности при выполнении приложения на планшете. 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 мы можем создать разные фрагменты для списка комплексов упражнений и для вывода подробного описания одного комплекса. После этого созданные фрагменты можно использовать в разных активностях.



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 вам придётся 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. Вы можете 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, заточенные под определённые задачи - ListFragment, DialogFragment (изучали ранее) и другие.

Есть два варианта использования фрагментов в приложении:

- в разметке сразу указываете фрагмент с помощью тега fragment;
- динамическое подключение фрагмента. В разметку помещается макет из группы ViewGroup, который становится контейнером для фрагмента. Обычно, для этой цели используют FrameLayout, но это не обязательное условие. И в нужный момент фрагмент замещает контейнер и становится частью разметки.

2.1 Задание

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

Создадим 2 фрагмента File> New> Fragment> Fragment (Blank)> name = Fragment1&Fragment2, set off – include interface callbacks.

Фрагмент, как и активность, состоит из разметки и класса. Сначала займёмся разметкой. В каждой разметке создан файл с TextView – сделайте их отличными друг от друга (смените цвет фона разметки).

```
<?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"
    android:background="@color/colorAccent"
    tools:context=".Fragment1">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="@string/hello_blank_fragment"/>

</FrameLayout>
```

Рассмотрим java классы Fragment1 и Fragment2:

public class Fragment1 extends Fragment {...}

Обратите внимание, что класс должен наследоваться от класса Fragment. В активностях разметка подключается в методе onCreate() через метод setContentView(). В фрагментах метод onCreate() служит для других задач. А для подключения разметки используется отдельный метод onCreateView().

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

У метода используются три параметра. В первом параметре используется объект класса LayoutInflater, который позволяет построить нужный макет, считывая информацию из указанного XML-файла. Остальные два параметра container, false используются в связке и указывают на возможность подключения фрагментов в активность через контейнер. Мы обойдёмся без контейнеров, а создадим собственные блоки для фрагментов, поэтому у нас используется значение false.

Рассмотрим файл activity_main.xml (рис. 2.2)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical">
        <Button
            android:id="@+id/ btnFragment2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onClick"
            android:text="fragment1">
        </Button>
        <Button
            android:id="@+id/btnFragment1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onClick"
            android:text="fragment2">
        </Button>
    </LinearLayout>
</LinearLayout>
```

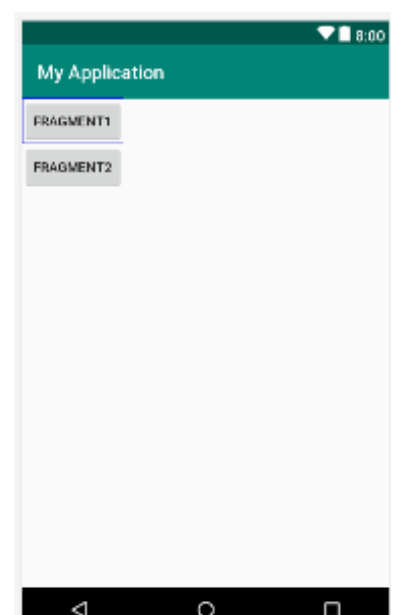


Рисунок 2.2 -
activity_main.xml

```

        <FrameLayout
            android:id="@+id/frgmCont"
            android:layout_width="match_parent"
            android:layout_height="match_parent">
    </FrameLayout>
</LinearLayout>

```

Две кнопки для выбора фрагмента. `FrameLayout` – это контейнер, в котором будет происходить вся работа с фрагментами. Он должен быть типа `ViewGroup`. Следите, чтобы импортировался класс `android.support.v4.app.Fragment` и `android.support.v4.app.FragmentManager`, а не `android.app.*`, которые предназначен для новых устройств. Фрагменты

```

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 Fragment1();
        fragment2 = new Fragment2();
    }

    public void onClick(View v) {
        fragmentManager = getSupportFragmentManager();
        switch (v.getId()) {
            case R.id.btnAdd:
                fragmentManager.beginTransaction().replace(R.id.fragmentContainer, fragment1).commit();
                break;
            case R.id.btnReplace:
                fragmentManager.beginTransaction().replace(R.id.fragmentContainer, fragment2).commit();
            default:
                break;
        }
    }
}

```

Рисунок 2.3 - MainActivity

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. А сами фрагменты должны реализовать необходимые интерфейсы, которые активность будет использовать в своих целях. Транзакция позволяет выполнить все операции скопом. Вы сообщаете менеджеру про все операции, запускаете их в методе `beginTransaction()` и подтверждаете своё намерение через метод `commit()`.

Запустите проект!

3 КОНТРОЛЬНОЕ ЗАДАНИЕ – Веб-браузер.

В созданном ранее приложении с калькулятором создать новый фрагмент:

File> New> Fragment> Fragment (Blank)> name = BrowserFragment, set off – include interface callbacks.

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.

Задание:

Сделать простейший браузер, который будет реагировать на:

action = ACTION_VIEW

data = Uri-объект с http-адресом

Требуется настроить Intent Filter, использовать компонент WebView в разметке, указать разрешение в манифесте, переопределить класс WebViewClient.