Модуль 3. Основы программирования Android приложений

*Тема 3.6. Фрагменты

Оглавление

3.6. Фрагменты

Рассмотрим приложение, которое отображает список, при этом пользователь при выборе элемента списка получает дополнительную информацию. Назовем это приложение «Список/детализация».

Одним из вариантов реализации интерфейса такого приложения является использование двух активностей: одна активность – для управления списком, другая – для детализации.

Однако, если мы возьмем планшет в альбомной ориентации, то представление одного лишь списка на экране будет не эстетично и не рационально, потому что образуется много нефункционального пространства.

Второй вариант построения интерфейса более логичен: список и его детализацию выводить на один экран в следующем виде:

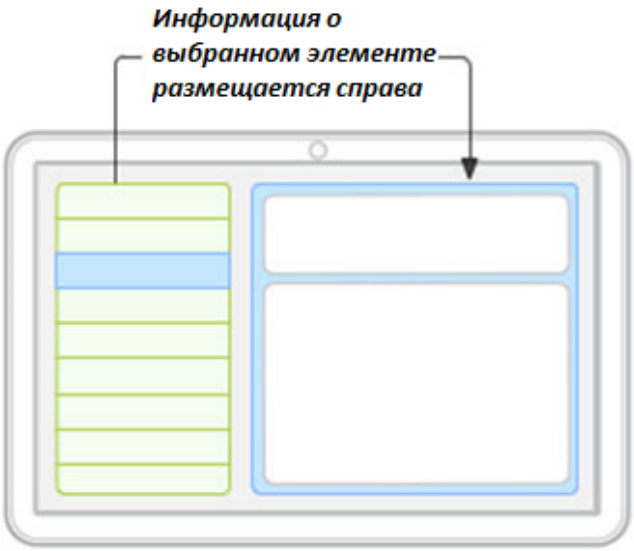


Рисунок 1. Расположение элементов в альбомной ориентации планшета

Рисунок 2. Расположение элементов в книжной ориентации планшета (телефона)

Чтобы построить подобный дизайн используют **фрагменты**. Придуманы были фрагменты именно для того, чтобы строить интерфейс, адаптированный под смартфоны и планшеты разных моделей и размеров. Фрагменты появились в API 11 (Android 3.0) для поддержки на более старых версиях был доработан `Android Support library`.

Фрагменты можно представить, как кусочки пазла, из которых составляется полная картинка нашего приложения. Фрагмент в чем-то схож с Activity - также имеет свой жизненный цикл и свои обработчики событий. На одной Activity может размещаться несколько фрагментов. Главное понять, что фрагменты не являются заменой Activity, они не могут существовать сами по себе, b существуют только вместе с Activity, на котрой размещены.

3.6.1. Создание Фрагментов

Когда фрагмент добавлен как часть разметки активности, он находится в объекте `ViewGroup` внутри иерархии представлений операции и определяет собственный макет представлений. Существует два способа создания фрагментов:

1. Добавление фрагмента в макет активности (как элемент `<fragment>`) – такой способ прост, но недостаточно гибок. Но иногда такие фрагменты полезны. Данный метод позволяет жёстко привязать фрагмент к представлению активности, но в данном случае не получится переключить фрагмент на протяжении жизненного цикла активности.
2. Добавление в код активности (добавить его в существующий объект `ViewGroup`) – такой метод сложен, но позволяет управлять фрагментами во время выполнения. При таком способе разработчик сам определяет, когда фрагмент добавляется в активность и что с ним происходит потом. Он может удалить фрагмент, заменить его другим фрагментом, а потом вернуть первый.

Фрагмент не обязан быть частью разметки активности. Можно использовать фрагмент без интерфейса в качестве невидимого рабочего потока операции.

Упражнение 3.6.1 Указание фрагментов в файле разметки

Создадим проект, в котором рассмотрим создание фрагментов и их использование.

На главной активности (**activity_main.xml)** необходимо чтобы отображались: две кнопки, растянутые по всей длине, ImageView и два элемента CheckBox. В сам проект необходимо добавить две картинки, одну из которых привязать к ImageView.

Для того чтобы показать преимущества использования фрагментов, добавим в проект еще три новые разметки активности (**File -> New -> Layout resource file**).

Разметки данных layout'ов будет выглядеть следующим образом:

- первый фрагмент содержит две кнопки. Файл разметки **button_fragment.xml**

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

        <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:text="Кнопка 1" />
        <Button
        android:id="@+id/button2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Кнопка 2" />

</LinearLayout>
```

- второй фрагмент содержит один элемент ImageView. Файл разметки **image_fragment.xml**

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

        <ImageView
        android:id="@+id/imageView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_marginTop="20dp"
        android:src="@drawable/android_img" />

</LinearLayout>
```

- третий фрагмент содержит элементы checkbox. Файл разметки **chechbox_fragment.xml**

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

        <CheckBox
        android:id="@+id/checkBox"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_marginTop="20dp"
        android:checked="false"
        android:text="Новый компонент 1" />
        <CheckBox
        android:id="@+id/checkBox2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:checked="false"
        android:text="Новый компонент 2" />

</LinearLayout>
```

Теперь в разметке основной активности (**activity_main.xml**) подключим новые фрагменты.

```
<?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:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical" >

        <fragment
        android:id="@+id/button_fragment"
        android:name="com.myfragmentexam.app.fragments.ButtonFragment"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        tools:layout="@layout/button_fragment" />
        <fragment
        android:id="@+id/checkbox_fragment"
        android:name="com.myfragmentexam.app.fragments.CheckBoxFragment"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        tools:layout="@layout/checkbox_fragment" />
        <fragment
       android:id="@+id/image_fragment"
        android:name="com.myfragmentexam.app.fragments.ImageFragment"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        tools:layout="@layout/image_fragment" />

</LinearLayout>
```

Такими образом мы разнесли три блока функциональности (кнопки, картинки и checkbox'ы) по разным layout'ам: по сути разобрали наш activity по кусочкам – эти кусочки и есть наши фрагменты.

Чтобы понять преимущества использования фрагментов, создадим папку layout-land для хранения разметки в альбомной ориентации. В данной папке создадим файл **activity_main.xml** и сделаем разметку под альбомную ориентацию устройства

Папка `layout-land` (чтобы показать различную компоновку при разных ориентациях)

Файл **layout-land/activity_main.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="horizontal" >

        <fragment
        android:id="@+id/button_fragment"
        android:name="com.myfragmentexam.app.fragments.ButtonFragment"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="15dp"
        android:layout_toLeftOf="@+id/image_fragment"
```

```
        tools:layout="@layout/button_fragment" />
        <fragment
        android:id="@+id/checkbox_fragment"
        android:name="com.myfragmentexam.app.fragments.CheckBoxFragment"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
         android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_below="@+id/button_fragment"
        android:layout_marginLeft="15dp"
        android:layout_toLeftOf="@+id/image_fragment"
        tools:layout="@layout/checkbox_fragment" />
        <fragment
        android:id="@+id/image_fragment"
        android:name="com.myfragmentexam.app.fragments.ImageFragment"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentEnd="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="15dp"
        android:layout_marginRight="15dp"
        tools:layout="@layout/image_fragment" />

</RelativeLayout>
```

Можно сразу увидеть плюсы: используя те же фрагменты мы реализовали различный интерфейс для различной ориентации устройства.

Обрабатывать элементы, содержащиеся во фрагментах, создаем дополнительные java-классы для каждого фрагмента

Например, рассмотрим файл **ButtonFragment.java**

```
import android.os.Bundle;
import android.support.v4.app.Fragment; // используем данную библиотеку
//для обратной совместимости
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.Toast;

public class ButtonFragment extends Fragment {

        @Override
        // Переопределяем метод onCreateView
        public View onCreateView(LayoutInflater inflater, ViewGroup container,
                Bundle savedInstanceState) {

                // менеджер компоновки, который позволяет
                // получать доступ к layout с наших ресурсов
                View view = inflater
                        .inflate(R.layout.button_fragment, container, false);

                // теперь можем достучаться до наших элементов, расположенных во
                // фрагменте
                Button button = (Button) view.findViewById(R.id.button);
```

```
                button.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                        Toast.makeText(getActivity(), "Сообщение из фрагмента",
                                        Toast.LENGTH_LONG).show();
                }
                });
                return view;
        }
}
```

Для того, чтобы фрагменты заработали на основной активности, необходимо наследовать наш класс от **FragmentActivity**

Файл **MainActivity.java**

```
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;

public class MainActivity extends FragmentActivity {

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

        }
}
```

3.6.2. Класс Fragment и его методы

Класс Fragment представляет поведение или часть пользовательского интерфейса в классе Activity.

Жизненный цикл фрагмента схож с жизненным циклом активности и зависит от него.

Методы жизненного цикла фрагмента

Для того, чтобы удачно манипулировать методами жизненного цикла фрагмента, необходимо понимать какие циклы после каких происходят.

Переопределять данные методы (по аналогии с жизненным циклом активности) необходимо только если необходимо модифицировать логику фрагментов.

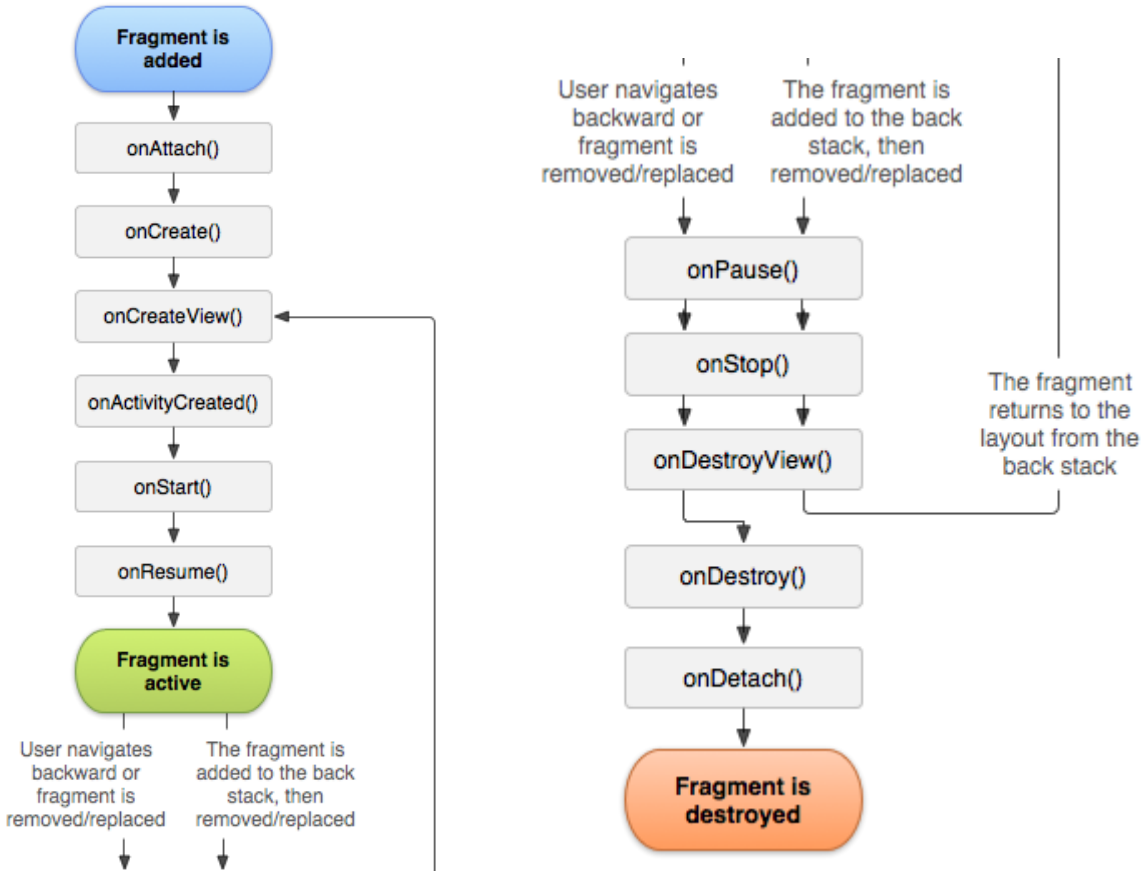


Рисунок 3. Жизненный цикл фрагмента (во время выполнения операции)

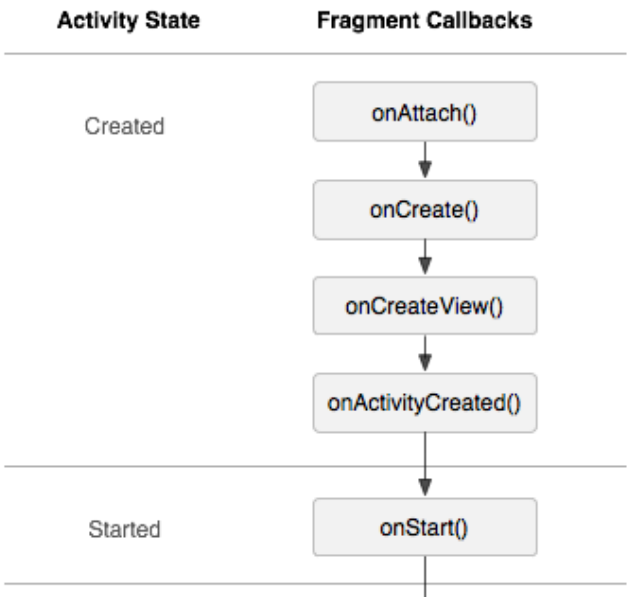
- **`onAttach(Activity)`** – вызывается после того как фрагмент связывается с активностью
- **`onCreate(Bundle)`** – вызывается при создании фрагмента. В своей реализации разработчик должен инициализировать ключевые компоненты фрагмента, которые требуется сохранить, когда фрагмент находится в состоянии паузы или возобновлен после остановки.
- **`onCreateView(LayoutInflater, ViewGroup, Bundle)`** – вызывается при первом отображении пользовательского интерфейса фрагмента на дисплее, позволяет инициализировать компоненты с layout, на данном этапе мы уже имеем всю иерархию компонентов
- **`onActivityCreated(Bundle)`** – вызывается после того как активность завершила свою обработку метода Activity.onCreate. Тут можно выполнять заключительные настройки интерфейса до того, как пользователь увидит фрагмент
- **`onStart()`** – похож на Activity.onStart(). Пользователь в момент вызова данного метода уже видит фрагмент, но еще не может взаимодействовать с ним.
- **`onResume()`** – вызывается после возвращения к нашему фрагменту (аналогично как в активности). После этого пользователь видит наш фрагмент и может выполнить какие-то действия.
- **`onPause()`** - если пользователь уходит на какую-то другую активность, то вызывается данный метод. Работа такая же как в Activity.onPause().– останавливает наш фрагмент. Можно уходя с фрагмента, например, останавливать видео или звук.
- **`onSaveInstanceState()`** – позволяет сохранять состояние фрагмента и восстановить его во время выполнения onCreate(), onCreateView() или onActivityCreated()
- **`onStop()`** – связан с методом на activity, по логике выполняет тоже что и в Activity.onStop(), останавливает работу нашего фрагмента. Но на этом жизненный цикл не

заканчивается. После этого выполняется onDestroyView()

- **`onDestroyView()`** – если фрагмент находится на пути уничтожения или сохранения, то следующим будет вызван именно этот метод – в данном методе мы можем возродить наш фрагмент
- **`onDestroy()`** - можно почистить ресурсы удалить те объекты которые нам не нужно использовать
- **`onDetach()`** – отвязывает фрагмент от активности

Фрагмент всегда должен быть встроен в Activity, и на его жизненный цикл напрямую влияет жизненный цикл Activity.

Например, когда активность приостановлена, в том же состоянии находятся и все фрагменты внутри нее, а когда активность уничтожается, уничтожаются и все фрагменты. Однако пока активность выполняется (это соответствует состоянию возобновлена жизненного цикла), можно манипулировать каждым фрагментом независимо, например, добавлять или удалять их. Когда разработчик выполняет такие транзакции с фрагментами, он может также добавить их в стек переходов назад, которым управляет активность.



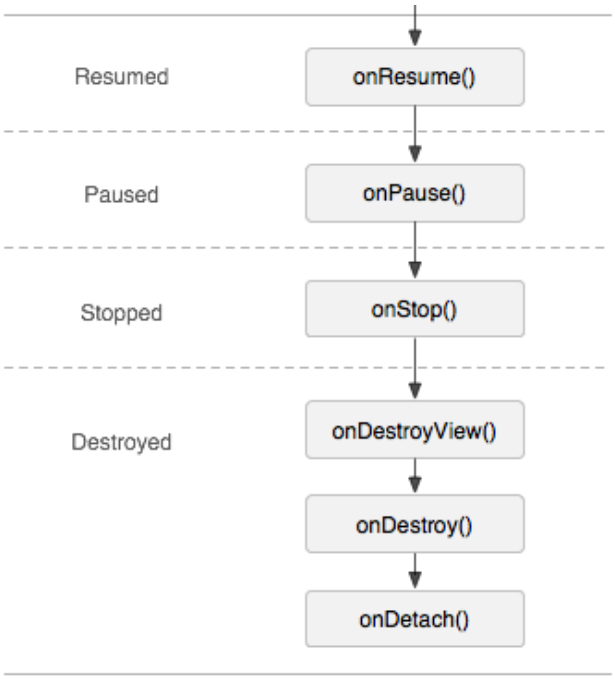


Рисунок 4. Влияние жизненного цикла операции на жизненный цикл фрагмента

3.6.3. Управление Фрагментами

Объекты FragmentManager

Для управления фрагментами в активности необходим класс `FragmentManager`. Данный класс также отвечает за добавление фрагментов представлений в иерархию представлений активности.

Чтобы получить его, следует вызвать метод `getFragmentManager()` из кода активности.

```
FragmentManager fragmentManager = getFragmentManager()
```

FragmentManager позволяет выполнить следующие действия:

- получать фрагменты, имеющиеся в активности, с помощью метода findFragmentById() (для фрагментов, предоставляющих пользовательский интерфейс в макете операции) или findFragmentByTag() (как для фрагментов, имеющих пользовательский интерфейс, так и для фрагментов без него);
- снимать фрагменты со стека переходов назад методом popBackStack() (имитируя нажатие кнопки Назад пользователем);
- регистрировать процесс-слушатель изменений в стеке переходов назад при помощи метода addOnBackStackChangedListener().

Можно использовать класс FragmentManager для открытия FragmentTransaction, что позволяет выполнять транзакции с фрагментами, например, добавление и удаление.

FragmentTransaction и их методы

Большим достоинством использования фрагментов в активности является возможность добавлять, удалять, заменять их и выполнять другие действия с ними в ответ на действия пользователя. Любой набор изменений, вносимых в активность, называется **транзакцией**. Ее можно выполнить при помощи API-интерфейсов в FragmentTransaction. Каждую транзакцию можно сохранить в стеке переходов назад, которым управляет активность. Это позволит пользователю перемещаться назад по изменениям во фрагментах (аналогично перемещению назад по активности).

Экземпляр класса `FragmentTransaction` можно получить от `FragmentManager`, например, так:

```
FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
```

Каждая транзакция является набором изменений, выполняемых одновременно. Разработчик может указать все изменения, которые ему нужно выполнить в данной транзакции, вызывая методы `add`(), `remove`() и `replace`(). Затем, чтобы применить транзакцию к активности, следует вызвать метод `commit`().

Впрочем, до вызова метода `commit`() у разработчика может возникнуть необходимость вызвать метод `addToBackStack`(), чтобы добавить транзакцию в стек переходов назад по транзакциям фрагмента. Этим стеком переходов назад управляет активность, что позволяет пользователю вернуться к предыдущему состоянию фрагмента, нажав кнопку Назад.

Например, следующий код демонстрирует, как можно заменить один фрагмент другим, сохранив при этом предыдущее состояние в стеке переходов назад:

```
        // Создание нового фрагмента и транзакции
        Fragment newFragment = new ExampleFragment();
        FragmentTransaction transaction = getFragmentManager().beginTransaction();
        // Замена контейнер в разметке на фрагмент
        // и добавляем транзакцию в стек обратного вызова
        transaction.replace(R.id.fragment_container, newFragment);
        transaction.addToBackStack(null);
        // выполнение транзакции
        transaction.commit();
```

В этом коде объект `newFragment` замещает фрагмент (если таковой имеется), находящийся в контейнере макета, на который указывает идентификатор `R`.id.fragment_container. В результате вызова метода `addToBackStack()` транзакция замены сохраняется в стеке переходов назад, чтобы пользователь мог обратить транзакцию и вернуть предыдущий фрагмент, нажав кнопку Назад.

Если в транзакцию добавить несколько изменений (например, еще раз вызвать `add`() или `remove`()), а затем вызвать `addToBackStack`(), все изменения, примененные до вызова метода `commit`(), будут добавлены в стек переходов назад как одна транзакция, и кнопка Назад обратит их все вместе.

Совет. К каждой транзакции с фрагментом можно применить анимацию перехода, вызвав `setTransition`() до фиксации.

Упражнение 3.6.2 Указание фрагментов в коде (динамическое создание фрагментов)

Рассмотрим на примитивном примере. Создадим две активности со следующей разметкой.

Файл **activity_main.xml**

```
<?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:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical" >

        <Button
        android:id="@+id/btn_Add"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="OnClickFragment"
        android:text="New Button" />
/* Для динамической загрузки фрагмента используем контейнер, например LinearLayout
*/
    <LinearLayout
        android:id="@+id/fragmentContainer"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_below="@+id/tool_bar"
        android:layout_marginTop="48dp"
        android:orientation="vertical" />

</LinearLayout>
```

Файл **first_fragment.xml**

```
<?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">

        <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Первый фрагмент"
        android:id="@+id/textView" />
</LinearLayout>
```

Для того, чтобы динамически загрузить фрагмент на активность, необходимо использовать FragmentManager и FragmentTransaction.

Файл **MainActivity.java**

```
import android.app.Fragment;
import android.os.Bundle;
import android.support.design.widget.FloatingActionButton;
```

```
import android.support.design.widget.Snackbar;
import android.support.v4.app.FragmentActivity;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentTransaction;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.View;
import android.view.Menu;
import android.view.MenuItem;

public class MainActivity extends FragmentActivity {

        private FirstFragment firsfragment;
        private FragmentManager manager;
        private FragmentTransaction transaction;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);
                manager = getSupportFragmentManager();// инициализация менеджера
                firsfragment = new FirstFragment(); // создаем объект (фрагмент),
но
                // не показываем его. Фрагмент по
                // являться после нажатия кнопки на главной активности
        }

        // Обработчик нажатия кнопки на главной активности, которая загружает на
        // главную активность
        public void OnClickFragment(View view) {
                // Накапливаем изменения фрагемнта
                transaction = manager.beginTransaction();
                switch (view.getId()) {
                case R.id.btn_Add:
                transaction.replace(R.id.fragmentContainer, firsfragment);
                }
                // реализуем изменения
                transaction.commit();
        }
}
```

Подклассы класса Fragment

- DialogFragment

Отображение перемещаемого диалогового окна. Использование этого класса для создания диалогового окна является хорошей альтернативой вспомогательным методам диалогового окна в классе Activity. Дело в том, что он дает возможность вставить диалоговое окно фрагмента в управляемый операцией стек переходов назад для фрагментов, что позволяет пользователю вернуться к закрытому фрагменту.

Создадим два диалога, соответственно нам понадобятся два фрагмента. Первый будет создаваться из разметки (dialog_fragment.xml), а второй динамически.

Создадим layout-файл для первого фрагмента **dialog_fragment.xml**

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

        <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_margin="20dp"
        android:text="@string/message_text"
        android:textAppearance="?android:attr/textAppearanceLarge" >
        </TextView>

        <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" >
        <Button
        android:id="@+id/btnYes"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:text="@string/yes" >
        </Button>
        <Button
              android:id="@+id/btnNo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:text="@string/no" >
        </Button>
    </LinearLayout>

</LinearLayout>
```

Файл **strings.xml**

```
<resources>
        <string name="app_name">DialogSample</string>
        <string name="dialog_1">Dialog 1</string>
        <string name="dialog_2">Dialog 2</string>
        <string name="message_text">Сообщение</string>
        <string name="yes">Да</string>
        <string name="no">Нет</string>
</resources>
```

Файл **activity_main.xml**

```
<?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">
        <Button
        android:id="@+id/btnDlg1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:text="@string/dialog_1">
        </Button>
        <Button
        android:id="@+id/btnDlg2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:text="@string/dialog_2">
        </Button>
</LinearLayout>
```

Так будет выглядеть наш диалог – текст сообщения и две кнопки.

Создаем класс **Dialog1.java**

```
import android.app.DialogFragment;
import android.content.DialogInterface;
import android.os.Bundle;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup;
import android.widget.Button;

public class Dialog1 extends DialogFragment implements OnClickListener {

        final String LOG_TAG = "myLogs";

        public View onCreateView(LayoutInflater inflater, ViewGroup container,
                Bundle savedInstanceState) {
                getDialog().setTitle("Title!"); // получаем dialog с помощью
метода
                // detgialog и устанавливаем заголовок диалога

                View v = inflater.inflate(R.layout.dialog_fragment, null);
                v.findViewById(R.id.btnYes).setOnClickListener(this);
                v.findViewById(R.id.btnNo).setOnClickListener(this);

                return v;
        }
```

```
        public void onClick(View v) {
                Log.d(LOG_TAG, "Dialog 1: " + ((Button) v).getText());
                dismiss(); // закрываем диалог
        }

        // метод onDismiss срабатывает когда диалог закрывается
        public void onDismiss(DialogInterface dialog) {
                super.onDismiss(dialog);
                Log.d(LOG_TAG, "Dialog 1: onDismiss");
        }

        // метод onCancel срабатывает когда диалог отменяют кнопкой Назад
        public void onCancel(DialogInterface dialog) {
                super.onCancel(dialog);
                Log.d(LOG_TAG, "Dialog 1: onCancel");
        }
}
```

Создаем второй фрагмент. Здесь мы будем строить диалог с помощью билдера, поэтому layout-файл не понадобится. Создаем только класс **Dialog2.java**

```
import android.app.AlertDialog;
import android.app.Dialog;
import android.app.DialogFragment;
import android.content.DialogInterface;
import android.content.DialogInterface.OnClickListener;
import android.os.Bundle;
import android.util.Log;

public class Dialog2 extends DialogFragment implements OnClickListener {

        final String LOG_TAG = "myLogs";

        // Создаем динамически диалог с заголовком, сообщением и двумя кнопками
        public Dialog onCreateDialog(Bundle savedInstanceState) {
                AlertDialog.Builder adb = new AlertDialog.Builder(getActivity())
                        .setTitle("Заголовок").setPositiveButton(R.string.no,
this)
                        .setNegativeButton(R.string.yes, this)

                        .setMessage(R.string.message_text);
                return adb.create();
        }

        // Обработчиком для кнопок назначаем текущий фрагмент
        // В onClick определяем, какая кнопка была нажата и выводим
соответствующий
        // текст в лог. В случае создания диалога через билдер, диалог сам
закроется
        // по нажатию на кнопку, метод dismiss здесь не нужен
        public void onClick(DialogInterface dialog, int which) {
                int i = 0;
                switch (which) {
                case Dialog.BUTTON_POSITIVE:
                i = R.string.yes;
                break;
                case Dialog.BUTTON_NEGATIVE:
```

```
                i = R.string.no;
                break;

                }
                if (i > 0)
                Log.d(LOG_TAG, "Dialog 2: " + getResources().getString(i));
        }

        public void onDismiss(DialogInterface dialog) {
                super.onDismiss(dialog);
                Log.d(LOG_TAG, "Dialog 2: onDismiss");
        }

        public void onCancel(DialogInterface dialog) {
                super.onCancel(dialog);
                Log.d(LOG_TAG, "Dialog 2: onCancel");
        }
}
```

MainActivity**.java**

```
import android.app.Activity;
import android.app.DialogFragment;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends Activity {

        DialogFragment dlg1;
        DialogFragment dlg2;

        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);
                dlg1 = new Dialog1();
                dlg2 = new Dialog2();
        }

        // Создаем диалоги и запускаем их методом show, который на вход требует
        // FragmentManager и строку-тэг. Транзакция и коммит происходят внутри
этого
        // метода, нам об этом думать не надо.
        public void onClick(View v) {
                switch (v.getId()) {
                case R.id.btnDlg1:
                dlg1.show(getFragmentManager(), "dlg1");
                break;
                case R.id.btnDlg2:
                dlg2.show(getFragmentManager(), "dlg2");
                break;
                default:
                break;
                }
        }
}
```

- ListFragment

Отображение списка элементов, управляемых адаптером (например, SimpleCursorAdapter), аналогично классу ListActivity. Этот класс предоставляет несколько методов для управления списком представлений, например, метод обратного вызова onListItemClick() для обработки нажатий.

ListFragment - это просто Fragment, в котором есть методы, упрощающие доступ к ListView и некоторым его операциям.

activity_main.xml:

```
<?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" >

        <fragment
        android:name="com.samsung.denikina.listfragmentsample.List"
        android:layout_width="match_parent"
        android:layout_height="match_parent" >
        </fragment>

</LinearLayout>
```

List_fragmen.xml:

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

        <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/number_list" >
        </TextView>

        <ListView
        android:id="@id/android:list"
        android:layout_width="match_parent"
        android:layout_height="match_parent" >
        </ListView>

        <TextView
        android:id="@id/android:empty"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:text="@string/empty" >
        </TextView>
```

```
</LinearLayout>
```

Создадим класс фрагмента, наследующий не android.app.Fragment как обычно, а android.app.ListFragment.

List.java - создаем адаптер и используем метод setListAdapter, чтобы передать его списку. Обратите внимание - мы даже не создаем или не находим (findViewById) список (ListView), он уже есть где-то внутри фрагмента и метод setListAdapter сам знает, как до него добраться. В принципе, это и есть основное преимущество ListFragment - нам не надо работать с ListView.

```
import android.app.ListFragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;

public class List extends ListFragment {

        String data[] = new String[] { "one", "two", "three", "four" };

        @Override
        public void onActivityCreated(Bundle savedInstanceState) {
                super.onActivityCreated(savedInstanceState);
                ArrayAdapter<String> adapter = new
ArrayAdapter<String>(getActivity(),
                        android.R.layout.simple_list_item_1, data);
                setListAdapter(adapter);
        }

        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup container,
                Bundle savedInstanceState) {
                return inflater.inflate(R.layout.list_fragment, null);
        }
}
```

MainActivity.java оставляем без каких-либо изменений.

3.6.4. Взаимодействия между Фрагментами и Активностями

Хотя Fragment реализован как объект, независимый от класса Activity, и может быть использован внутри нескольких операций, конкретный экземпляр фрагмента напрямую связан с содержащей его операцией.

В частности, фрагмент может обратиться к экземпляру Activity с помощью метода getActivity() и без труда выполнить такие задачи, как поиск представления в макете операции:

```
View listView = getActivity().findViewById(R.id.list);
```

Аналогичным образом операция может вызывать методы фрагмента, получив ссылку на объект

Fragment от FragmentManager с помощью метода findFragmentById() или findFragmentByTag().

Например,

```
ExampleFragment fragment = (ExampleFragment)
getFragmentManager().findFragmentById(R.id.example_fragment);
```

Ссылки

1. http://www.bubelov.com/2012/12/dialogfragment.html
2. http://startandroid.ru/

Благодарности

Компания Samsung Electronics выражает благодарность за участие в подготовке данного материала преподавателю IT ШКОЛЫ SAMSUNG Деникину Антону Витальевичу и Деникиной Наталье Владимировне.