# 7 UI-фрагменты и FragmentManager

В этой главе мы начнем строить приложение CriminalIntent. Оно предназначено для хранения информации об «офисных преступлениях»: грязной посуде, оставленной в раковине, или пустом лотке общего принтера после печати документов.

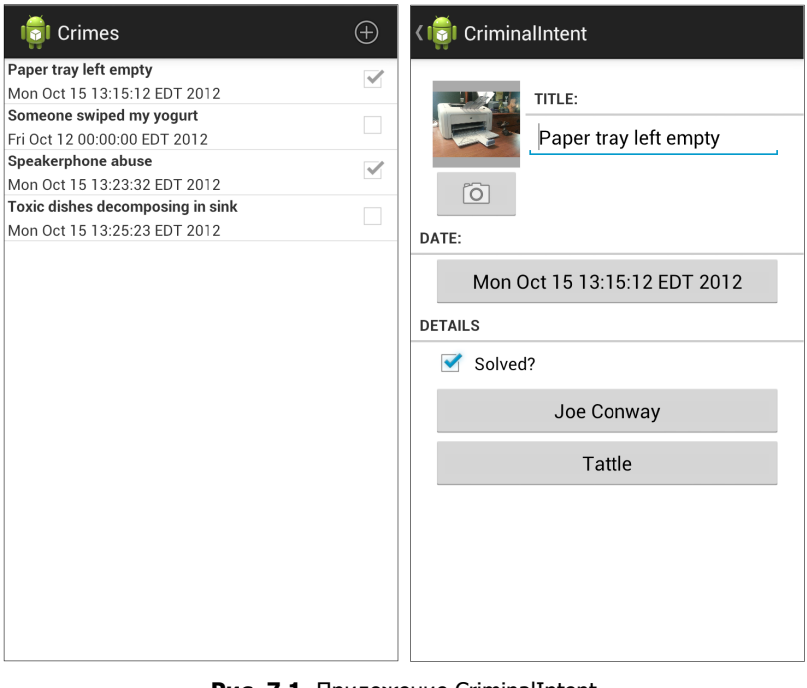


**Рис. 7.1.** Приложение CriminalIntent

В приложении CriminalIntent пользователь создает запись о преступлении с заголовком, датой и фотографией. Также можно выбрать подозреваемого в списке контактов и отправить жалобу по электронной почте, опубликовать ее в Twitter, Facebook или другом приложении. Сообщив о преступлении, пользователь освобождается от негатива и может сосредоточиться на текущей задаче.

CriminalIntent — сложное приложение, для построения которого нам понадобится целых 13 глав. В нем используется интерфейс типа «список/детализация»: на главном экране выводится список зарегистрированных преступлений. Пользователь может добавить новое или выбрать существующее преступление для просмотра и редактирования информации.

## Гибкость пользовательского интерфейса

Разумно предположить, что приложение типа «список/детализация» состоит из двух активностей: для управления списком и для управления детализированным представлением. Щелчок на преступлении в списке запускает экземпляр детализированной активности.

Нажатие кнопки Back уничтожает активность детализации и возвращает на экран список, в котором можно выбрать другое преступление.

Такая архитектура работает, — но что делать, если вам потребуется более сложная схема представления информации и навигации между экранами?

- Допустим, пользователь запустил приложение CriminalIntent на планшете. Экраны планшетов и некоторых больших телефонов достаточно велики для одновременного отображения списка и детализации — по крайней мере в альбомной ориентации.

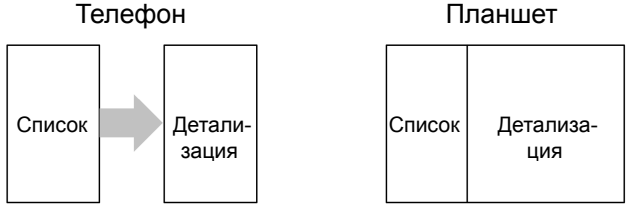


**Рис. 7.2.** Идеальный интерфейс «список/детализация» для телефонов и планшетов

- Пользователь просматривает описание преступления на телефоне и хочет увидеть следующее преступление в списке. Было бы удобно, если бы пользователь мог провести пальцем по экрану, чтобы перейти к следующему преступлению без возвращения к списку. Каждый жест прокрутки должен обновлять детализированное представление информацией о следующем преступлении.

Эти сценарии объединяет гибкость пользовательского интерфейса: возможность формирования и изменения представления активности во время выполнения в зависимости от того, что нужно пользователю или устройству.

Подобная гибкость в активности не предусмотрена. Активность тесно связана со своим представлением. Представление, которое заполняется активностью при вызове `setContentView(…)`, привязывается к активности до самого конца. Вы не можете переключить все представление активности (не уничтожая саму активность) или передать представление от одной активности к другой во время выполнения. Таков закон Android.

## Знакомство с фрагментами

Закон Android нельзя нарушить, но можно обойти, передав управление пользовательским интерфейсом приложения от активности одному или нескольким *фрагментам* (fragments).

Фрагмент представляет собой объект контроллера, которому активность может доверить выполнение операций. Чаще всего такой операцией является управление пользовательским интерфейсом — целым экраном или его частью.

Фрагмент, управляющий пользовательским интерфейсом, называется *UI-фрагментом*. UI-фрагмент имеет собственное представление, которое заполняется на основании файла макета. Представление фрагмента содержит элементы пользовательского интерфейса, с которыми будет взаимодействовать пользователь.

Пользователь активности содержит точку, в которой вставляется представление фрагмента, или несколько точек для представлений нескольких фрагментов.

Фрагменты, связанные с активностью, могут использоваться для формирования и изменения экрана в соответствии с потребностями приложения и пользователей. Представление активности формально остается неизменным на протяжении жизненного цикла, и законы Android не нарушаются.

Давайте посмотрим, как эта схема работает в приложении «список/детализация». Представление активности строится из фрагмента списка и фрагмента детализации. Представление детализации содержит подробную информацию о выбранном элементе списка.

При выборе другого элемента на экране появляется новое детализированное представление. С фрагментами эта задача решается легко; активность заменяет фрагмент детализации другим фрагментом детализации (рис. 7.3). Это существенное изменение представления происходит без уничтожения активности.

Применение UI-фрагментов позволяет разделить пользовательский интерфейс вашего приложения на структурные блоки, а это полезно не только для приложений «список/детализация». Работа с отдельными блоками упрощает построение интерфейсов со вкладками, анимированных боковых панелей и многих других.

Впрочем, за такую гибкость пользовательского интерфейса приходится платить: повышением сложности, увеличением количества «подвижных частей», увеличением объема кода.

Выгода из использования фрагментов проявится в главах 11 и 22, зато разбираться со сложностью придется прямо сейчас.


**Рис. 7.3.** Переключение фрагмента детализации

# Начало работы над CriminalIntent

В этой главе мы возьмемся за представление детализации CriminalIntent. На рис. 7.4 показано, как будет выглядеть приложение CriminalIntent к концу главы.

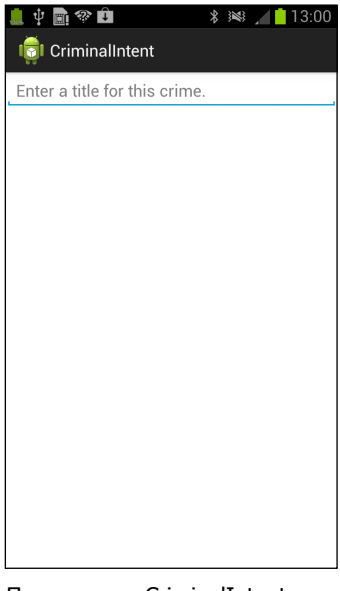


**Рис. 7.4.** Приложение CriminalIntent к концу главы

На первый взгляд результат не впечатляет. Однако не забывайте, что эта глава всего лишь закладывает основу для более серьезных дел в будущем.

Экраном, показанным на рис. 7.4, будет управлять UI-фрагмент с именем `CrimeFragment`. *Хостом* (host) экземпляра `CrimeFragment` является активность с именем `CrimeActivity`.

Пока считайте, что хост предоставляет позицию в иерархии представлений, в которой фрагмент может разместить свое представление (рис. 7.5). Фрагмент не может

вывести представление на экран сам по себе. Его представление отображается только при размещении в иерархии активности.

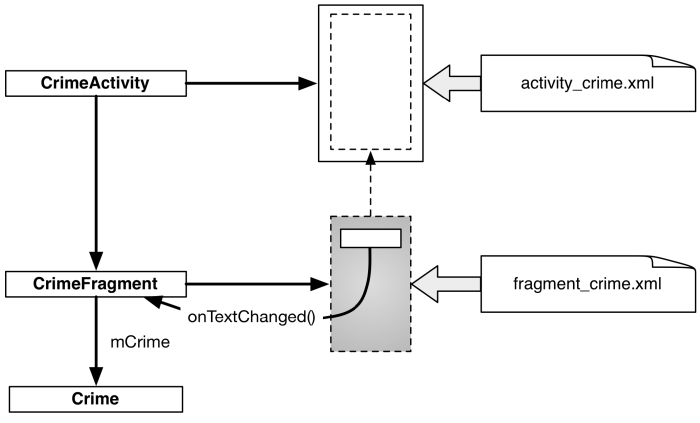


**Рис. 7.5.** CrimeActivity как хост CrimeFragment

Проект CriminalIntent будет большим; диаграмма объектов поможет понять логику его работы. На рис. 7.6 изображена общая структура CriminalIntent. Запоминать все объекты и связи между ними не обязательно, но прежде чем выходить в путь, полезно хотя бы в общих чертах понимать, куда вы направляетесь.

Мы видим, что класс `CrimeFragment` делает примерно то же, что в GeoQuiz делали активности: он создает пользовательский интерфейс и управляет с ним, а также обеспечивает взаимодействие с объектами модели.

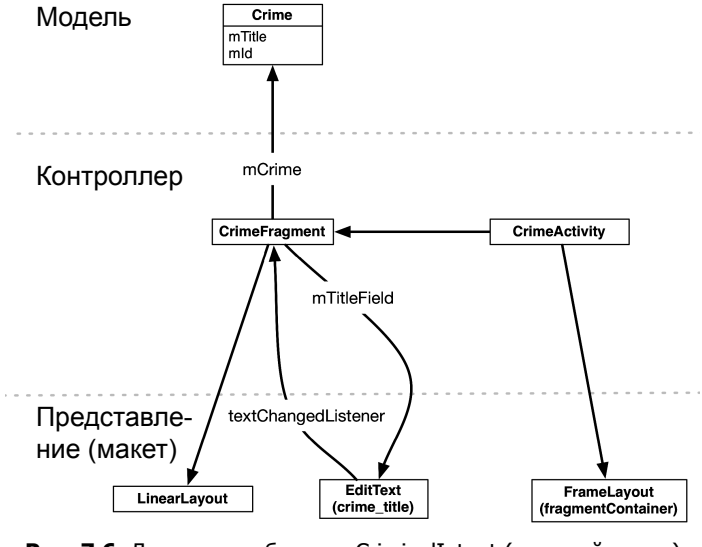


**Рис. 7.6.** Диаграмма объектов CriminalIntent (для этой главы)

Мы напишем три класса, изображенных на рис. 7.6: `Crime`, `CrimeFragment` и `CrimeActivity`.

Экземпляр `Crime` представляет одно офисное преступление. В этой главе описание преступления будет состоять только из заголовка и идентификатора. Заголовок содержит содержательный текст (например, «Свалка химических отходов в раковине» или «Кто-то украл мой йогурт!»), а идентификатор однозначно идентифицирует экземпляр `Crime`.

В этой главе мы для простоты будем использовать один экземпляр `Crime`. В класс `CrimeFragment` включается поле (`mCrime`) для хранения этого отдельного инцидента.

Представление `CrimeActivity` состоит из элемента `FrameLayout`, определяющего место, в котором будет отображаться представление `CrimeFragment`.

Представление `CrimeFragment` будет состоять из элементов `LinearLayout` и `EditText`. `CrimeFragment` определяет поле для виджета `EditText` (`mTitleField`) и назначает для него слушателя, обновляющего уровень модели при изменении текста.

## Создание нового проекта

Но довольно разговоров; пора построить новое приложение. Создайте новое приложение Android (New ▸ Android Application Project). Введите имя приложения `CriminalIntent` и имя пакета `com.bignerdranch.android.criminalintent`, как показано на рис. 7.7. Укажите для построения новейшие версии API и убедитесь в том, что приложение совместимо с устройствами на базе Froyo.

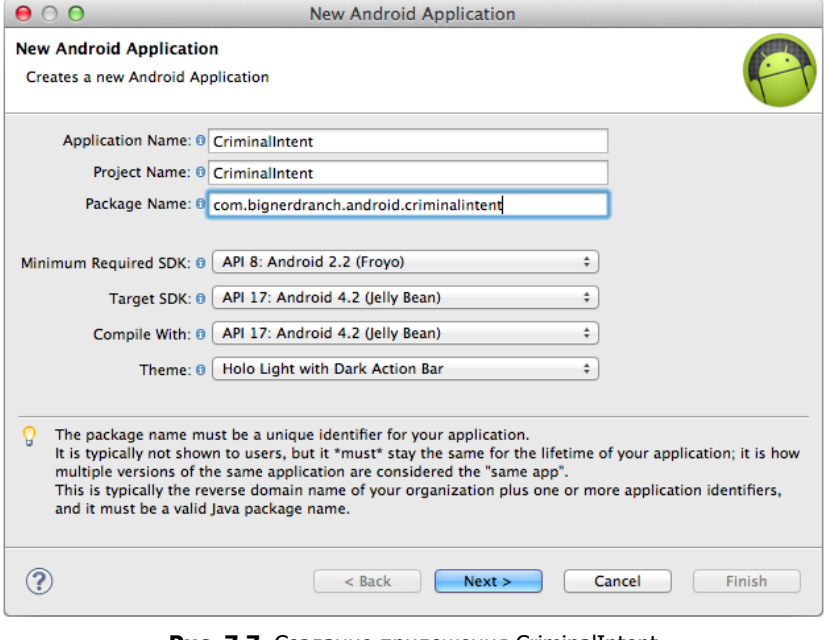


**Рис. 7.7.** Создание приложения CriminalIntent

В следующем диалоговом окне снимите флажок, чтобы создать пользовательский значок лаунчера, и щелкните на кнопке Next. Выберите создание активности на базе пустого шаблона активности и щелкните на кнопке Next.

Наконец, введите имя активности `CrimeActivity` и щелкните на кнопке Finish (рис. 7.8).

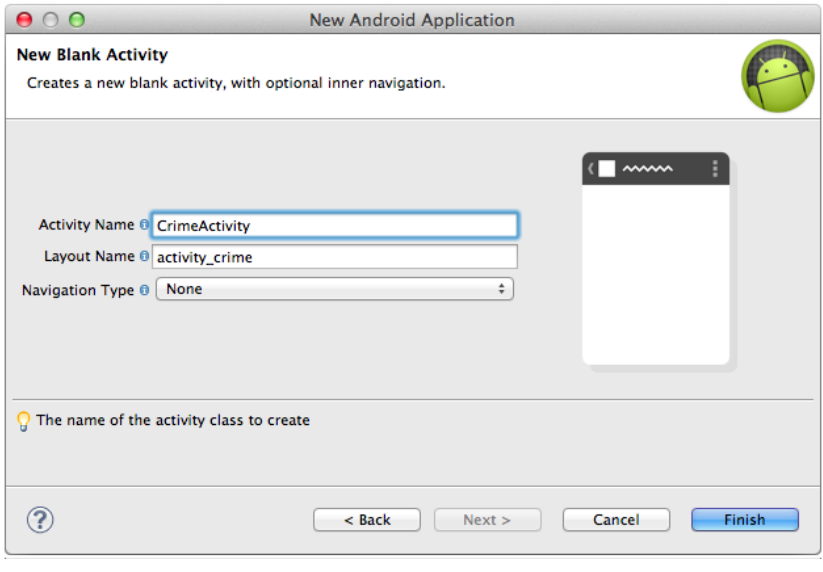


**Рис. 7.8.** Настройка CrimeActivity

## Фрагменты и библиотека поддержки

Фрагменты появились в API уровня 11 вместе с первыми планшетами на базе Android и неожиданной потребностью в гибкости пользовательского интерфейса. При создании `CriminalIntent` был указан минимальный обязательный SDK уровня 8, поэтому мы должны обеспечить совместимость приложения со старыми версиями Android.

К счастью, с фрагментами обратная совместимость достигается относительно легко: достаточно использовать классы фрагментов из библиотеки поддержки Android.

*Библиотека поддержки* (support library) уже является частью вашего проекта. Она находится в файле libs/android-support-v4.jar, куда была добавлена шаблоном. Библиотека поддержки включает класс `Fragment` (`android.support.v4.app.Fragment`), который может использоваться на любом устройстве Android с API уровня 4 и выше.

Класс из библиотеки поддержки используется не только в старых версиях при отсутствии «родного» класса; он также используется вместо него в новых версиях.

В библиотеку поддержки также входит важный класс `FragmentActivity` (`android.support.v4.app.FragmentActivity`). Для использования фрагментов нужны активности, которые умеют управлять фрагментами. В Honeycomb и последующих версиях

Android все субклассы `Activity` поддерживают управление фрагментами. В более ранних версиях `Activity` ничего не знает о фрагментах — как и ваши субклассы `Activity`. Для совместимости с более низкими уровнями API следует субклассировать `FragmentActivity` — субкласс `Activity`, предоставляющий функциональность управления фрагментами нового класса `Activity` даже в старых версиях Android.

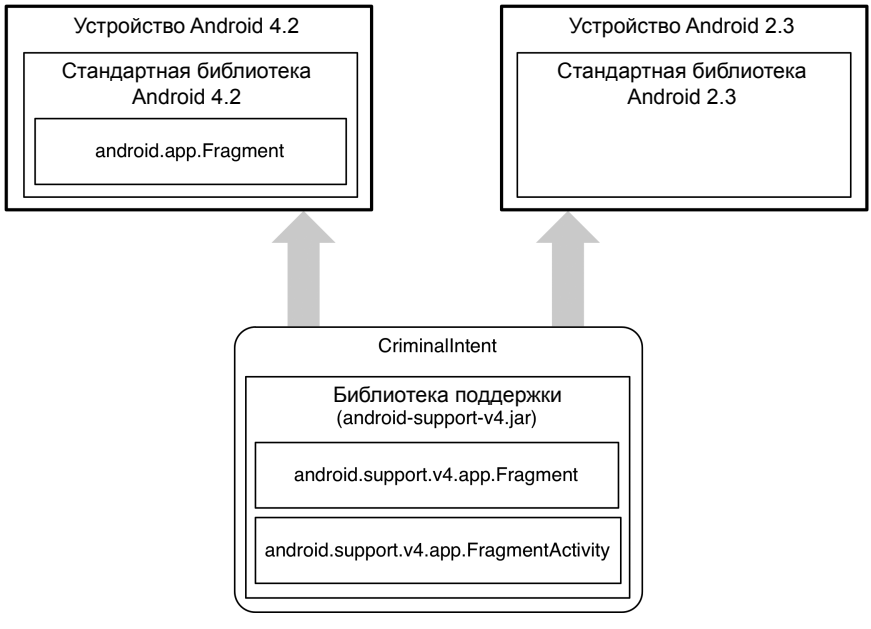


**Рис. 7.9.** Местонахождение разных классов фрагментов

На рис. 7.9 приведены имена этих классов и их местонахождение. Поскольку библиотека поддержки (и `android.support.v4.app.Fragment`) находится в вашем приложении, ее можно безопасно использовать независимо от того, где работает приложение.

На панели Package Explorer найдите и откройте файл CrimeActivity.java. Замените суперкласс `CrimeActivity` на `FragmentActivity`. Заодно удалите шаблонную реализацию `onCreateOptionsMenu(Menu)`. (Мы создадим меню CriminalIntent «с нуля» в главе 16.)

**Листинг 7.1.** Настройка шаблонного кода (CrimeActivity.java)

```
public class CrimeActivity extends Activity FragmentActivity {

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

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
```

*продолжение ⇩*

**Листинг 7.1** (*продолжение*)

```
        getMenuInflater().inflate(R.menu.activity_crime, menu);
        return true;
    }
}
```

Прежде чем продолжать с `CrimeActivity`, мы создадим уровень модели CriminalIntent. Для этого мы напишем класс `Crime`.

## Создание класса Crime

На панели Package Explorer щелкните правой кнопкой мыши на пакете `com.bignerdranch.android.criminalintent` и выберите команду New ▸ Class. Введите имя класса `Crime`, оставьте суперкласс `java.lang.Object` и щелкните на кнопке Finish.

Добавьте в Crime.java следующий код.

**Листинг 7.2.** Добавление кода в класс Crime (Crime.java)

```
public class Crime {

    private UUID mId;
    private String mTitle;

    public Crime() {
        // Генерирование уникального идентификатора
        mId = UUID.randomUUID();
    }
}
```

Затем для свойства `mId`, доступного только для чтения, необходимо сгенерировать только `get`-метод, а для свойства `mTitle` — get- и `set`-методы.

Щелкните правой кнопкой мыши после конструктора и выберите команду Source ▸ Generate Getters and Setters. Чтобы сгенерировать только `get`-метод для `mId`, щелкните на стрелке слева от имени переменной, чтобы раскрыть список методов, и установите флажок только рядом с `getId()`, как показано на рис. 7.10.

**Листинг 7.3.** Сгенерированные get- и set-методы (Crime.java)

```
public class Crime {
    private UUID mId;

    private String mTitle;

    public Crime() {
        mId = UUID.randomUUID();
    }

    public UUID getId() {
        return mId;
    }

    public String getTitle() {
```

```
        return mTitle;
    }

    public void setTitle(String title) {
        mTitle = title;
    }
}
```

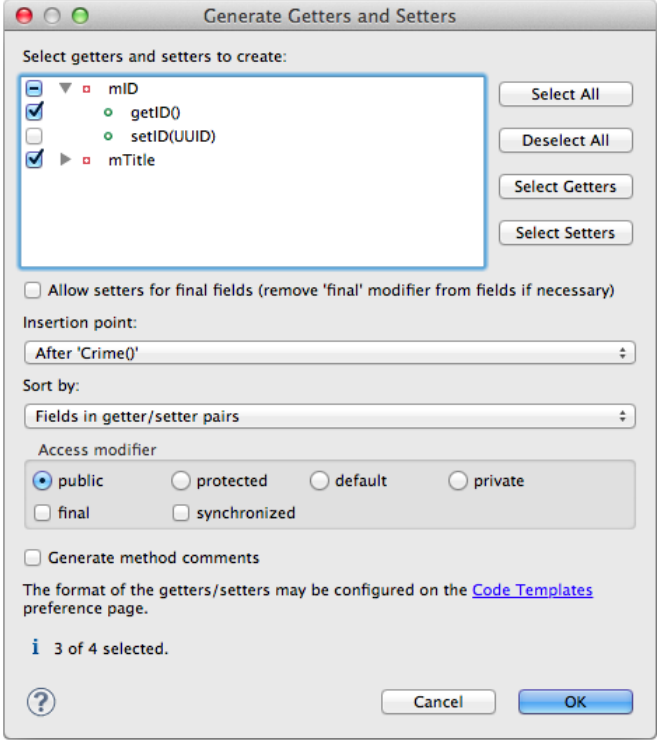


**Рис. 7.10.** Генерирование двух get-методов и одного set-метода

Вот и все, что нам понадобится для класса `Crime` и уровня модели CriminalIntent в этой главе.

Итак, мы создали уровень модели и активность, которая выполняет хостинг фрагмента в режиме совместимости с версиями Froyo и Gingerbread. А теперь более подробно рассмотрим, как активность выполняет свои функции хоста.

## Хостинг UI-фрагментов

Чтобы стать хостом для UI-фрагмента, активность должна:

- определить место представления фрагмента в своем макете;
- управлять жизненным циклом экземпляра фрагмента.

## Жизненный цикл фрагмента

На рис. 7.11 показан жизненный цикл фрагмента. Он имеет много общего с жизненным циклом активности: он тоже может находиться в состоянии остановки, приостановки и выполнения, он тоже содержит методы, переопределяемые для выполнения операций в критических точках, — многие из которых соответствуют методам жизненного цикла активности.

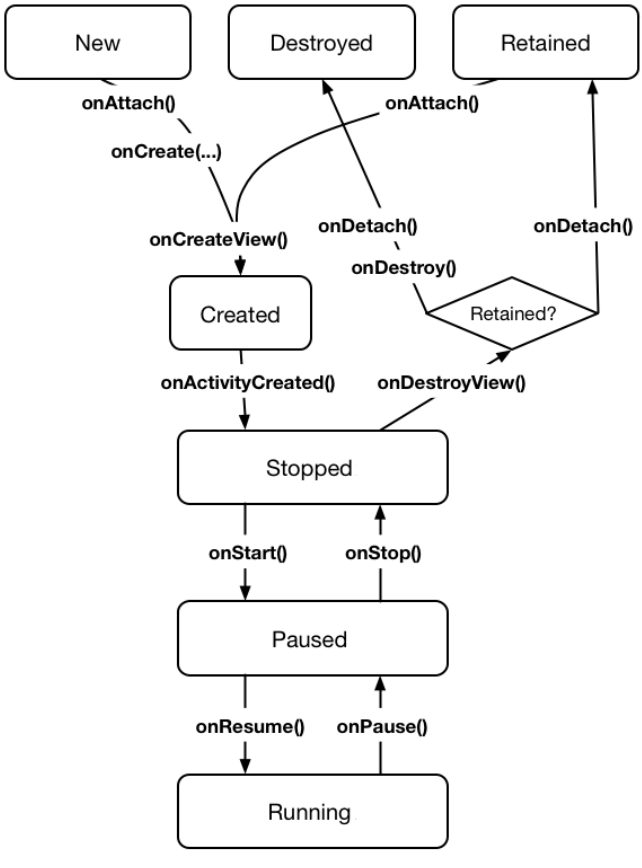


**Рис. 7.11.** Жизненный цикл фрагмента

Эти соответствия играют важную роль. Так как фрагмент работает по поручению активности, его состояние должно отражать текущее состояние активности. Следовательно, ему необходимые соответствующие методы жизненного цикла для выполнения работы активности.

Принципиальное различие между жизненными циклами фрагмента и активности заключается в том, что методы жизненного цикла фрагмента вызываются активностью-хостом, а не ОС. ОС ничего не знает о фрагментах, используемых активностью; фрагменты — внутреннее дело самой активности.