Навчальна дисципліна

Розробка мобільних застосувань

Лектор - к.т.н., доцент

Баклан Ігор Всеволодович

Site: baklaniv.at.ua

E-mail: iaa@ukr.net

2016-2017

Лекція №3. Механізми намірів

Для большинства приложений одной активности недостаточно.

До настоящего момента мы рассматривали приложения с одной активностью; для простых приложений это нормально. Однако в более сложной ситуации одна активность попросту не справляется со всеми делами. Мы покажем вам, **как строить приложения с несколькими активностями** и как организовать взаимодействие между активностями с использованием ***интентов***. Также вы узнаете, как использовать интенты **за границами приложения** и как **выполнять *действия при помощи активностей других приложений* на вашем устройстве**. Внезапно перед вами открываются совершенно новые перспективы...

Приложение может содержать несколько активностей

Ранее в книге мы говорили, что активность — одна четко определенная операция, которая может выполняться пользователем, — например, отображение списка рецептов. В очень простом приложении этого может быть достаточно.

Но обычно пользователю требуется выполнять *более* одной операции — например, не только выводить список рецептов, но и добавлять их в базу данных. В таких случаях в приложении используются разные активности: одна для отображения списка рецептов, а другая для добавления рецепта.

Чтобы понять, как работает эта система, лучше всего опробовать ее в деле. Мы построим приложение с двумя активностями. Первая активность позволяет ввести текст сообщения. Щелчок на кнопке в первой активности запускает вторую активность, которой передается сообщение. Далее вторая активность выводит полученное сообщение.

Активность — одна целенаправленная операция, которая может выполняться пользователем. Если объединить несколько активностей для выполнения чего-то более сложного, получится задача.

Последовательность действий

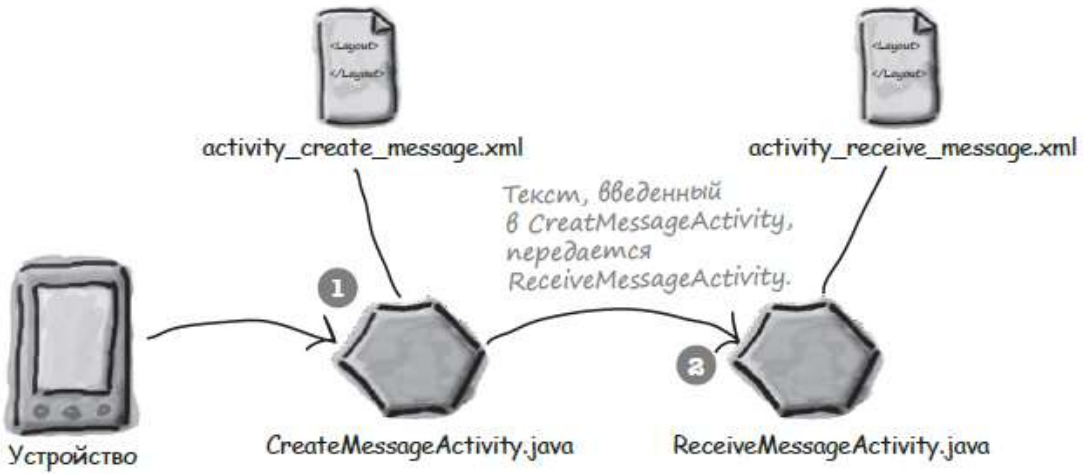
1. Создание базового приложения с одной активностью и макетом.
2. Добавление второй активности и макета.
3. Организация вызова второй активности из первой.
4. Организация передачи данных из первой активности во вторую.

Структура приложения

Приложение состоит из двух активностей и двух макетов.

1. **В начале работы приложения запускается активность CreateMessageActivity.**
   Эта активность использует макет *activity_create_message.xml*.

2. **Пользователь щелкает на кнопке в CreateMessageActivity.**
   Кнопка запускает активность `ReceiveMessageActivity`, которая использует макет *activity_receive_message.xml*.

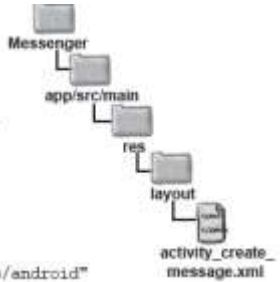


Создание проекта

Проект приложения создается точно так же, как и в предыдущих главах. Создайте в Android Studio новый проект для приложения с именем “Messenger” и именем пакета `com.hfad.messenger`. Выберите минимальный уровень API 15, чтобы приложение работало на большинстве устройств. Чтобы ваш код не отличался от нашего, создайте пустую активность с именем “CreateMessageActivity” и макет с именем “activity_create_message”.

На следующей странице мы отредактируем макет активности.

Обновление макета



Перед вами разметка XML из файла *activity_create_message.xml*. Мы убрали элемент `<TextView>`, автоматически сгенерированный Android Studio, и заменили его элементами `<Button>` и `<EditText>`. Элемент `<EditText>` создает текстовое поле с возможностью редактирования, которое может использоваться для ввода данных.

Приведите файл *activity_create_message.xml* в соответствие со следующей разметкой:

```
<RelativeLayout 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:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context=".CreateMessageActivity" >

    <Button
        android:id="@+id/send"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="36dp"
        android:layout_marginTop="21dp"
        android:onClick="onSendMessage"
        android:text="@string/send" />

    <EditText
        android:id="@+id/message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/send"
        android:layout_below="@+id/send"
        android:layout_marginTop="18dp"
        android:ems="10" />

</RelativeLayout>
```

Замените элемент `<TextView>`, сгенерированный Android Studio, элементами `<Button>` и `<EditText>`.

Щелчок на кнопке запускает метод onSendMessage() из активности.

Строковый ресурс.

Описывает ширину текстового поля `<EditText>`. Ширина поля должна быть достаточной для размещения 10 букв «M».

Элемент `<EditText>` определяет текстовое поле для ввода и редактирования текста. Класс наследует от того же класса Android View, что и классы других компонентов графического интерфейса, встречавшиеся нам до настоящего момента.

Обновление strings.xml...

Добавьте новый строковый ресурс с именем send. Мы присвоили ему значение Send Message, чтобы на кнопке выводился текст "Send Message".

Добавленной нами кнопке назначается текстовое значение `@string/send`. Это означает, что мы должны добавить в *strings.xml* строку с именем "send" и присвоить ей значение — текст, который должен отображаться на кнопке. Выполните следующие действия:

```
...
    <string name="send">Send Message</string>
...
```

...и добавление метода в активность

Следующая строка элемента `<Button>`

```
android:onClick="onSendMessage"
```

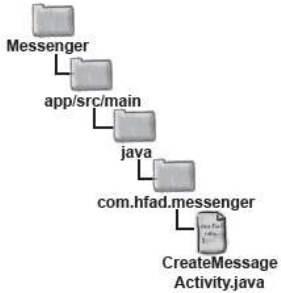
означает, что метод `onSendMessage()` активности будет срабатывать при щелчке на кнопке. Давайте добавим этот метод в активность.

Откройте файл *CreateMessageActivity.java* и замените код, сгенерированный Android Studio, следующим:

```
package com.hfad.messenger;

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

public class CreateMessageActivity extends Activity {

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

    //Вызвать onSendMessage() при щелчке на кнопке
    public void onSendMessage(View view) {
    }
}
```

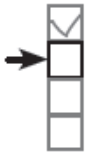
Мы заменяем код, который был сгенерирован Android Studio, так как большая часть этого кода не обязательна.

Метод onCreate() вызывается при создании активности.



Этот метод будет вызываться при щелчке на кнопке. Мы допишем тело метода далее в этой главе.

Итак, первая активность готова; переходим ко второй.



Создание второй активности и макета

В Android Studio имеется мастер для создания новых активностей и макетов в приложениях. По сути это упрощенная версия мастера, используемого при создании приложений; используйте его каждый раз, когда вам потребуется создать новую активность.

Чтобы создать новую активность, выполните команду File → New → Activity и выберите вариант Blank Activity. На экране появляется окно, в котором вы сможете выбрать параметры новой активности.

Каждой создаваемой новой активности и макету необходимо присвоить имя. Задайте для активности имя "ReceiveMessageActivity", а для макета — имя "activity_receive_message". Убедитесь в том, что пакету присвоено имя "com.hfad.messenger". Подтвердите остальные значения по умолчанию, а когда все будет готово — щелкните на кнопке Finish.

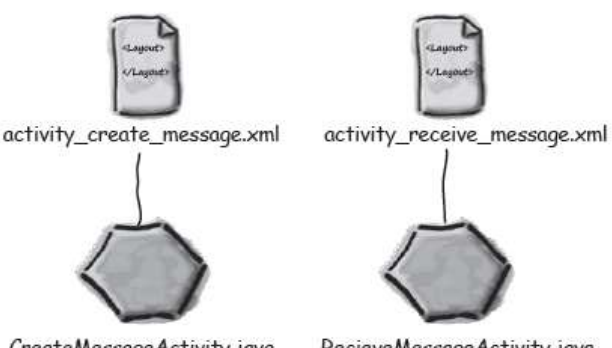
Оставьте остальным параметрам значения по умолчанию, так как нас сейчас интересует только создание новой активности и макета. Мы заменим большую часть кода, сгенерированного Android Studio.

Что произошло?

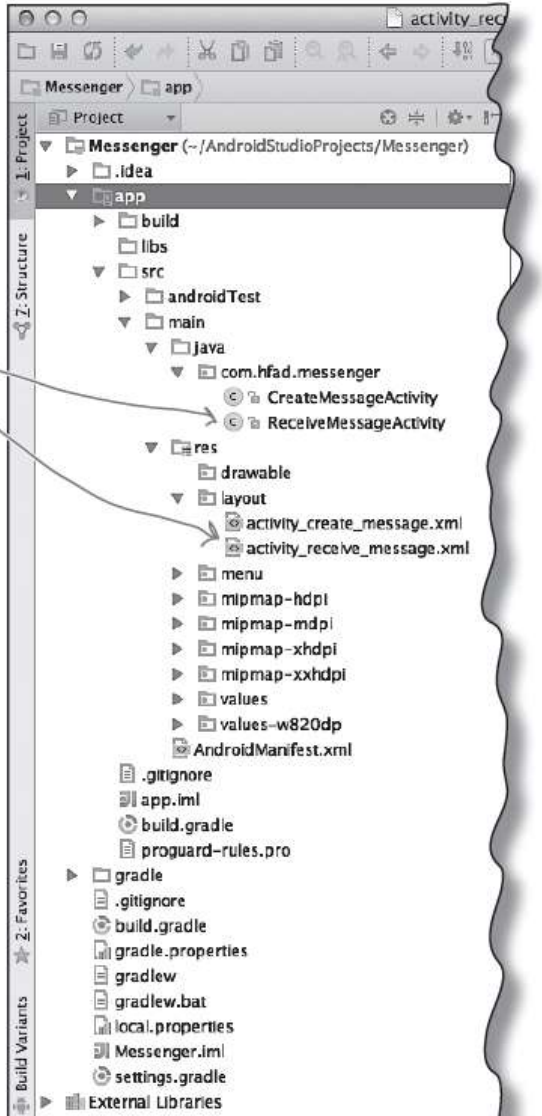
Когда вы щелкнули на кнопке Finish, Android Studio создает для вас новый файл активности вместе с новым макетом. Заглянув на панель структуры проекта, вы увидите, что в папке *app/src/main/java* появился новый файл с именем *ReceiveMessageActivity.java*, а в папке *app/src/main/res/layout* — файл с именем *activity_receive_message.xml*.

Новая активность и макет, которые мы только что создали. Теперь приложение содержит две активности и два макета.

Каждая активность использует свой отдельный макет. `CreateMessageActivity` использует макет *activity_create_message.xml*, а `ReceiveMessageActivity` — макет *activity_receive_message.xml*.



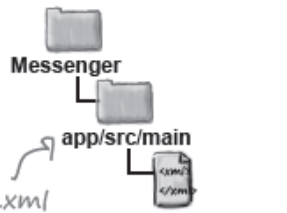
Android Studio также незаметно изменяет конфигурацию приложения в файле с именем *AndroidManifest.xml*. Давайте разберемся повнимательнее.



Знакомьтесь: файл манифеста Android

Каждое Android-приложение должно содержать файл с именем *AndroidManifest.xml*. Вы можете найти его в папке *app/src/main* вашего проекта. Файл *AndroidManifest.xml* содержит важнейшую информацию о приложении: какие активности оно содержит, какие библиотеки ему необходимы и другие объявления. Android создает этот файл при создании приложения. Если вы вспомните настройки, которые были выбраны при создании проекта, то часть содержимого этого файла может показаться знакомой.

Наш экземпляр *AndroidManifest.xml* выглядит так:



Файл AndroidManifest.xml находится в этой папке.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.hfad.messenger" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name=".CreateMessageActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity
            android:name=".ReceiveMessageActivity"
            android:label="@string/title_activity_receive_message" >
        </activity>

    </application>
</manifest>
```

Имя пакета, которое мы указали.

Android Studio назначает приложению значок по умолчанию. Мы еще вернемся к этой теме позднее.

Тема влияет на оформление приложения. Этот вопрос тоже будет рассмотрен позднее.

Первая активность, Create Message Activity.

Это означает, что активность является главной активностью приложения.

А это означает, что активность может использоваться для запуска приложения.

Вторая активность, Receive Message Activity.

Android Studio добавляет эти строки при добавлении второй активности.

Будьте осторожны!

Если вы разрабатываете Android-приложения без IDE, файл придется создать вручную.

Каждая активность должна быть объявлена

Все активности должны быть объявлены в файле *AndroidManifest.xml*. Если активность не объявлена в файле, то система не будет знать о его существовании. А если система не знает об активности, то активность не будет выполняться.

Активности объявляются в манифесте включением элемента `<activity>` в элемент `<application>`. Собственно, *каждая* активность в приложении должна иметь соответствующий элемент `<activity>`. Общий формат объявления выглядит так:

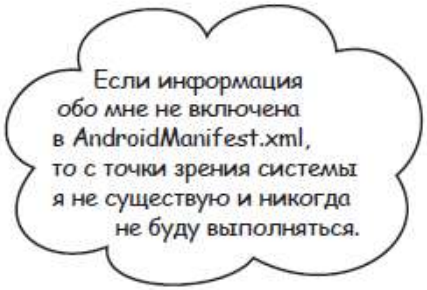
```
<application
    ...
    ...>
    <activity
        android:name="имя_класса_активности"
        android:label="@string/метка_активности"
        ...
        ...>
    ...
    </activity>
    ...
</application>
```

Каждая активность должна быть объявлена в элементе <application>.

Эта строка обязательна.

Эта строка не обязательна, но Android Studio генерирует ее за нас.

Активность также может обладать другими свойствами.



Следующая строка является обязательной и используется для определения имени класса активности:

```
android:name="имя_класса_активности"
```

`имя_класса_активности` — имя класса с префиксом ".", в данном случае `.ReceiveMessageActivity`. Имя класса снабжается префиксом ".", потому что Android строит *полное* имя класса, объединяя имя класса с именем пакета.

Следующая строка не является обязательной; она задает метку активности, удобную для пользователя:

```
android:label="@string/метка_активности"
```

Метка выводится в верхней части экрана при выполнении активности. Если убрать ее, то Android будет использовать вместо нее имя приложения.

В объявление активности также могут включаться и другие свойства — например, разрешения безопасности или возможность использования активностями других приложений.

Будьте осторожны!

Вторая активность была объявлена автоматически, потому что она была добавлена с использованием мастера Android Studio.

Если вы добавляете новые активности вручную, то вам придется редактировать файл AndroidManifest.xml *самостоятельно. Если вы используете другую среду разработки, может оказаться, что эта среда не включает информацию в манифест.*

Интент — разновидность сообщения

К настоящему моменту мы создали приложение с двумя активностями, каждая из которых имеет собственный макет. При запуске приложения будет выполняться наша первая активность, `CreateMessageActivity`. Следующий шаг — заставить `CreateMessageActivity` вызывать `ReceiveMessageActivity`, когда пользователь щелкает на кнопке Send Message.

Чтобы запустить одну активность из другой, воспользуйтесь **интентом**. Интент можно рассматривать как своего рода «намерение выполнить некую операцию». Это разновидность сообщений, позволяющая связать разнородные объекты (например, активности) на стадии выполнения. Если одна активность хочет запустить другую, она отправляет для этого интент системе Android. Android запускает вторую активность и передает ей интент.

Чтобы запустить активность, создайте интент и используйте его в методе `startActivity()`.

Процедура создания и отправки интента состоит всего из двух строк кода. Для начала создайте интент:

```
Intent intent = new Intent(this, Target.class);
```

При создании интента указывается активность, которая должна его получить, — словно адрес, написанный на конверте.

Первый параметр сообщает Android, от какого объекта поступил интент; для обозначения текущей активности используется ключевое слово `this`. Во втором параметре передается имя класса активности, которая должна получить интент.

После того как интент будет создан, он передается Android следующим вызовом:

```
startActivity(intent);
```

startActivity() запускает активность, указанную в интенте...

Этот вызов приказывает Android запустить активность, определяемую интентом.

При получении интента Android убеждается в том, что все правильно, и приказывает активности запуститься. Если найти активность не удалось, инициируется исключение `ActivityNotFoundException`.

Использование интента для запуска второй активности

А теперь применим эту схему на практике и используем интент для вызова `ReceiveMessageActivity`. Активность должна запускаться, когда пользователь щелкает на кнопке Send Message, поэтому мы добавляем две строки кода в метод `onSendMessage()`.

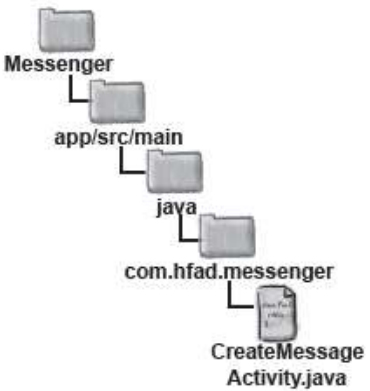
Внесите изменения, выделенные жирным шрифтом:

```
package com.hfad.messenger;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class CreateMessageActivity extends Activity {

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

    //Вызвать onSendMessage() при щелчке на кнопке
    public void onSendMessage(View view) {
        Intent intent = new Intent(this, ReceiveMessageActivity.class);
        startActivity(intent);
    }

}
```

Необходимо импортировать класс интента android.content.Intent, так как он используется в onSendMessage().

Этот метод не изменился.

Запустить активность ReceiveMessageActivity.



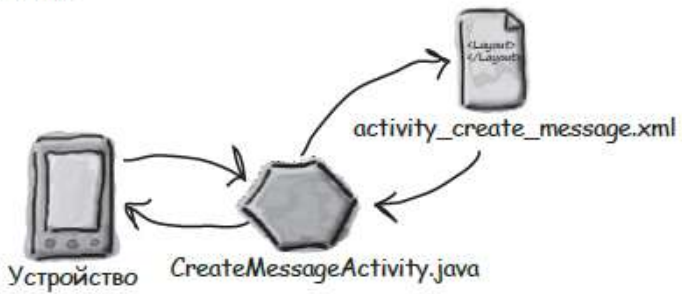
Что же произойдет, если запустить приложение?

Создание 1-й активности
Создание 2-й активности
Вызов 2-й активности
Передача данных

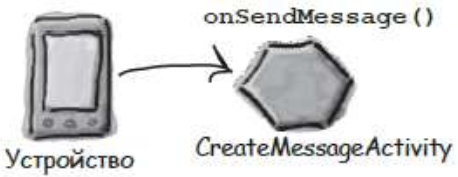
Что происходит при запуске приложения

Прежде чем тестировать приложение, еще раз посмотрим, как будет функционировать версия, построенная нами к настоящему моменту:

1. **При запуске приложения начинает работать его главная активность CreateMessageActivity.**
   В конфигурации запускаемой активности указано, что она использует макет *activity_create_message.xml*. Этот макет отображается в новом окне.

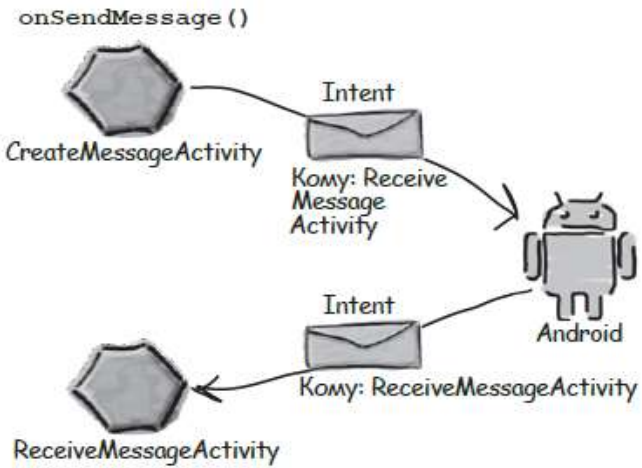


2 **Пользователь щелкает на кнопке.** Метод `onSendMessage()` класса `CreateMessageActivity` реагирует на щелчок.



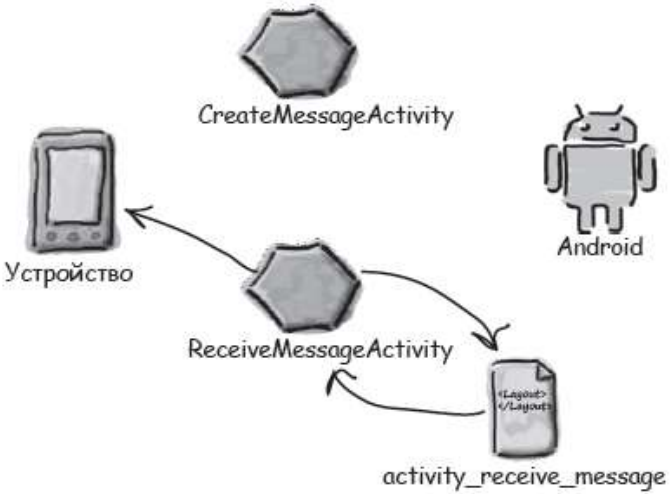
---

3 **Метод onSendMessage() приказывает Android запустить активность ReceiveMessageActivity при помощи интента.** Android убеждается в том, что интент правилен, и после этого приказывает `ReceiveMessageActivity` запуститься.



История продолжается...

4 При запуске активность ReceiveMessageActivity сообщает, что она использует макет *activity_receive_message.xml*; этот макет отображается в новом окне.



Тест-драйв

Сохраните изменения и запустите приложение. Активность `CreateMessageActivity` запускается, а при щелчке на кнопке Send Message она запускает `ReceiveMessageActivity`.

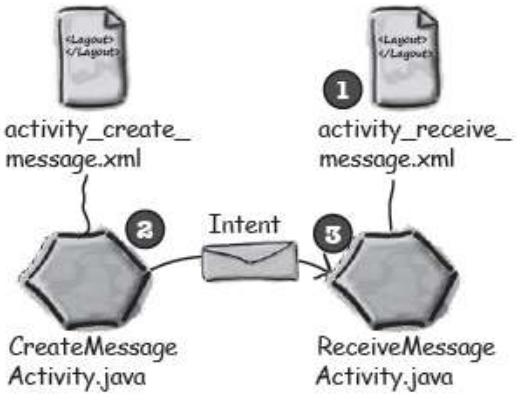
Создание 1-й активности
Создание 2-й активности
Вызов 2-й активности
Передача данных

Создание 1-й активности
Создание 2-й активности
Вызов 2-й активности
Передача данных

Передача текста второй активности

К настоящему моменту мы запрограммировали активность `CreateMessageActivity` так, чтобы она запускала `ReceiveMessageActivity` при щелчке на кнопке Send Message. Теперь необходимо обеспечить передачу текста из `CreateMessageActivity` в `ReceiveMessageActivity`, чтобы активность `ReceiveMessageActivity` могла вывести полученный текст. Для этого необходимо сделать три вещи:

1. Изменить макет *activity_receive_message.xml* так, чтобы он мог использоваться для вывода текста. Сейчас он представляет собой макет по умолчанию, сгенерированный мастером.
2. Обновить активность *CreateMessageActivity.xml*, чтобы она получала введенный пользователем текст. Этот текст должен быть добавлен в интент перед его отправкой.
3. Обновить код *ReceiveMessageActivity.java*, чтобы он выводил текст, отправленный в интенте.

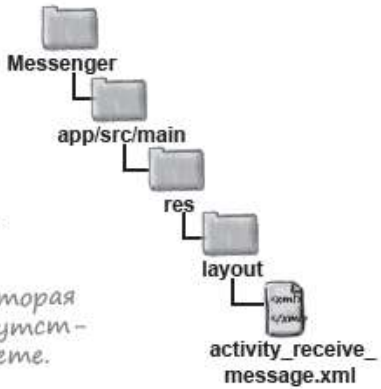


Начнем с макета

Вот как выглядит макет *activity_receive_message.xml*, сгенерированный Android Studio:

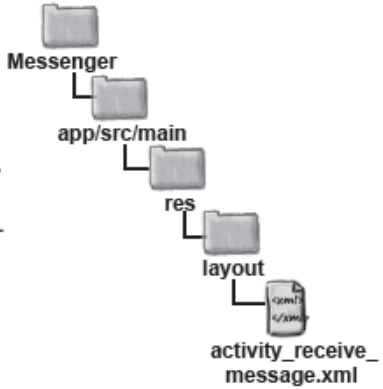
```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context="com.hfad.messenger.ReceiveMessageActivity">

    <TextView
        android:text="@string/hello_world"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</RelativeLayout>
```

Надпись, которая сейчас присутствует в макете.



В макет необходимо внести пару изменений. Во-первых, нужно присвоить элементу `<TextView>` идентификатор "message", чтобы к нему можно было обращаться из кода активности, а во-вторых, нужно заблокировать вывод текста "Hello world!". Как изменить макет? Попробуйте сами, прежде чем переходить к следующей странице.

Обновление свойств надписи



В макет необходимо внести пару изменений.

Прежде всего необходимо назначить идентификатор элементу <TextView>. Идентификаторы должны назначаться всем компонентам графического интерфейса, с которыми вы хотите работать в коде активности, — идентификатор используется для обращения к компоненту из кода Java. Также необходимо отменить вывод в макете текста “Hello world!”.

Чтобы внести оба изменения, приведите макет к следующему виду:

```
<RelativeLayout 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:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context="com.hfad.messenger.ReceiveMessageActivity">

    <TextView
        android:id="@+id/message"
        android:text="@string/hello_world"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</RelativeLayout>
```

В этой строке элементу <TextView> назначается идентификатор message.

Удалите строку, в которой атрибуту text присваивается значение @string/hello_world.

Вместо того, чтобы удалять строку

```
android:text="@string/hello_world"
```

также можно было бы в файле *strings.xml* присвоить строковому ресурсу `hello_world` пустое значение. Мы не стали так делать, потому что единственный текст, который будет выводиться в этой надписи — это сообщение, передаваемое из `CreateMessageActivity`. Итак, после внесения изменений в макет можно переходить к работе с активностями.

Часто Задаваемые Вопросы

В: Обязательно ли использовать интенты? Разве я не могу просто создать экземпляр второй активности в коде первой активности?

О: Хороший вопрос, но... Нет, в Android так дела не делаются. Одна из причин заключается в том, что при передаче интентов Android знает последовательность запуска активностей. В частности, при нажатии кнопки Back на вашем устройстве Android будет точно знать, в какую точку следует вернуться.

putExtra() Включает в интент дополнительную информацию

Вы уже видели, как создать новый интент командой

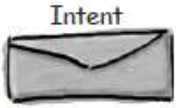
```
Intent intent = new Intent(this, Target.class);
```

В интент также можно добавить дополнительную информацию, которая должна передаваться получателю. В этом случае активность, получившая интент, сможет на него как-то среагировать. Для этого используется метод `putExtra()`

```
intent.putExtra("сообщение", значение);
```

где `сообщение` — имя ресурса для передаваемой информации, а `значение` — само значение. Перегрузка метода `putExtra()` позволяет передавать `значение` многих возможных типов. Например, это может быть примитив (скажем, `boolean` или `int`), массив примитивов или `String`. Многократные вызовы `putExtra()` позволяют включить в интент несколько экземпляров дополнительных данных. Если вы решите действовать так, проследите за тем, чтобы каждому экземпляру было присвоено уникальное имя.

Вызов putExtra() включает дополнительную информацию в отправляемое сообщение.



В аргументе value могут передаваться значения разных типов. Полный перечень этих типов приведен в документации Google Android. Кроме того, Android Studio отображает список вариантов во время ввода кода.

Как получить дополнительную информацию из интента

Впрочем, это еще не все. Когда Android приказывает `ReceiveMessageActivity` запуститься, активность должна каким-то образом получить дополнительную информацию, которая была отправлена `CreateMessageActivity` системе Android в интенте.

Существует пара удобных методов, которые помогают в решении этой задачи. Первый метод:

```
getIntent();
```

`getIntent()` возвращает интент, запустивший активность; из полученного интента можно прочитать любую информацию, отправленную вместе с ним. Конкретный способ чтения зависит от типа отправленной информации. Например, если вы знаете, что интент включает строковое значение с именем "message", используйте следующий вызов:


```
Intent intent = getIntent();
```
← Получить интент.

```
String string = intent.getStringExtra("message");
```
← Получить переданную с интентом строку с именем "message".

Конечно, из интента можно читать не только строковые значения. Например, вызов

```
int intNum = intent.getIntExtra("name", default_value);
```

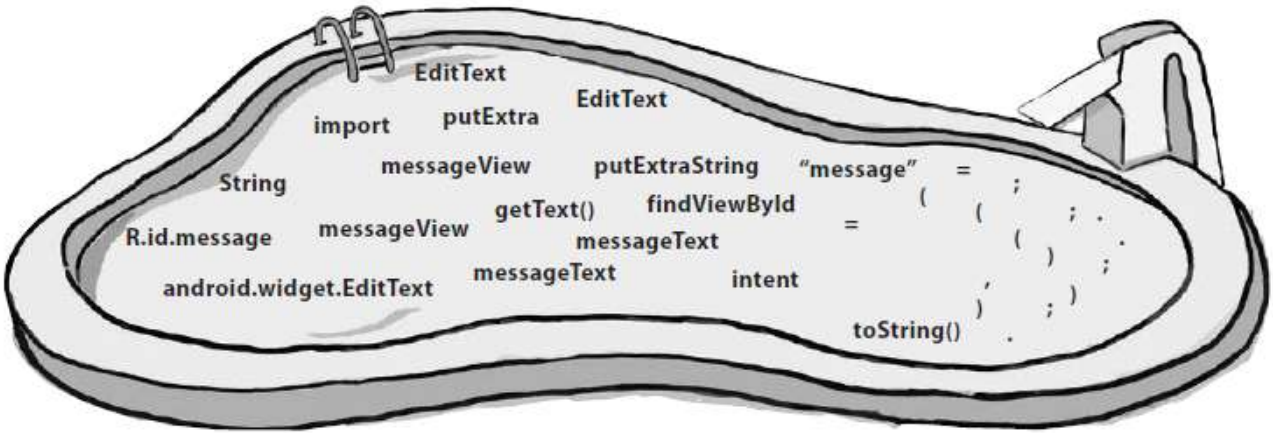
может использоваться для получения значения `int` с именем `name`. Параметр `default_value` указывает, какое значение `int` должно использоваться по умолчанию.

У бассейна

Выловите из бассейна фрагменты кода и расставьте их в пустых строках *CreateMessageActivity.java*. Каждый фрагмент кода может использоваться только один раз, при этом все фрагменты использовать не обязательно.

Ваша **задача** — сделать так, чтобы активность прочитала текст сообщения из `<EditText>` и добавила его в интент.

```
package com.hfad.messenger;

import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.View;
.....

public class CreateMessageActivity extends Activity {

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

    //Вызвать onSendMessage() при щелчке на кнопке
    public void onSendMessage(View view) {
        .....
        .....
        Intent intent = new Intent(this, ReceiveMessageActivity.class);
        .....
        startActivity(intent);
    }
}
```


EditText
EditText
import
putExtra
messageView
putExtraString
"message"
=
;
String
(
(
;
.
getText()
findViewById
=
R.id.message
messageView
messageText
(
.
)
;
messageText
intent
,
android.widget.EditText
)
)
;
toString()
.

У бассейна. Решение

Выловите из бассейна фрагменты кода и расставьте их в пустых строках *CreateMessageActivity.java*. Каждый фрагмент кода может использоваться только один раз, при этом все фрагменты использовать не обязательно.

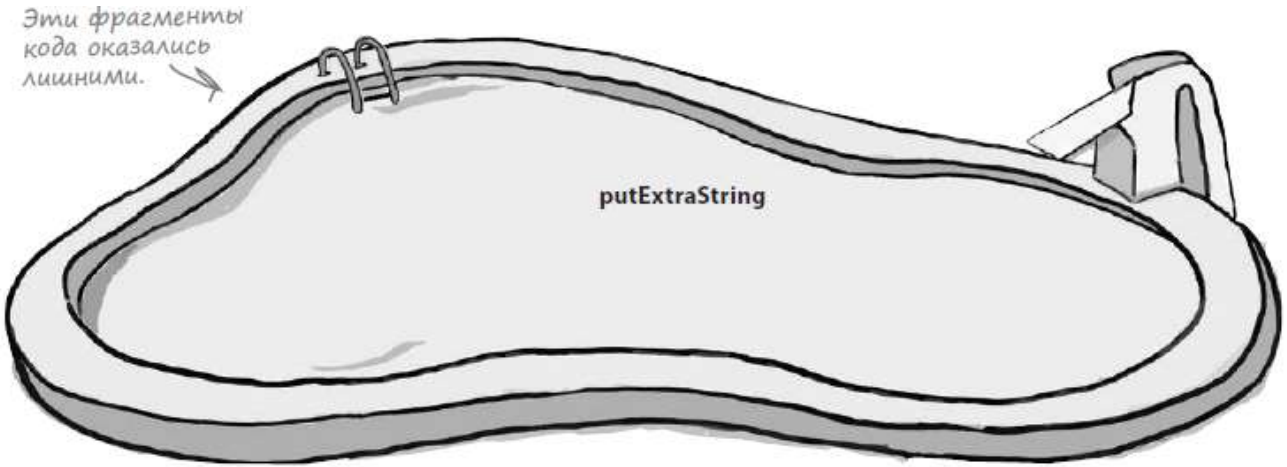
Ваша **задача** — сделать так, чтобы активность прочитала текст сообщения из `<EditText>` и добавила его в интент.

```
package com.hfad.messenger;

import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.View;
import android.widget.EditText;

public class CreateMessageActivity extends Activity {

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

    //Вызвать onSendMessage() при щелчке на кнопке
    public void onSendMessage(View view) {
        EditText messageView = (EditText) findViewById(R.id.message);
        String messageText = messageView.getText().toString();
        Intent intent = new Intent(this, ReceiveMessageActivity.class);
        intent.putExtra("message", messageText);
        startActivity(intent);
    }
}
```

Необходимо импортировать класс EditText.

Получить текст из текстового поля с идентификатором message.

Добавить текст в интент под именем "message".

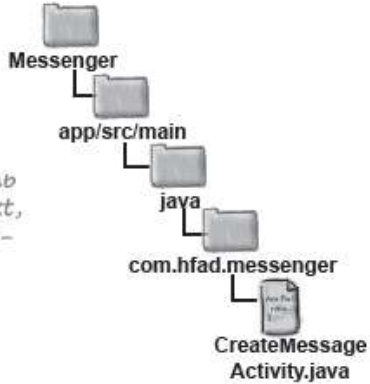

Эти фрагменты кода оказались лишними.
putExtraString

Обновление кода CreateMessageActivity

Мы обновили код *CreateMessageActivity.java*, чтобы активность получала текст, введенный пользователем на экране, и добавляла его в интент. Ниже приведен полный код (не забудьте изменить свою версию и включить изменения, выделенные жирным шрифтом):

Создание 1-й активности
Создание 2-й активности
Вызов 2-й активности
Передача данных

```
package com.hfad.messenger;

import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.View;
import android.widget.EditText;

public class CreateMessageActivity extends Activity {

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

    //Вызвать onSendMessage() при щелчке на кнопке
    public void onSendMessage(View view) {
        EditText messageView = (EditText)findViewById(R.id.message);
        String messageText = messageView.getText().toString();
        Intent intent = new Intent(this, ReceiveMessageActivity.class);
        intent.putExtra(ReceiveMessageActivity.EXTRA_MESSAGE, messageText);
        startActivity(intent);
    }
}
```



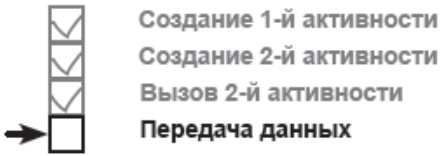
Необходимо импортировать класс android.widget.EditText, используемый в коде активности.

Получить текущий текст из EditText.

Мы создаем интент, а затем добавляем в него текст. В качестве имени дополнительной информации используется константа – в этом случае мы можем быть уверены в том, что CreateMessageActivity и ReceiveMessageActivity используют одну и ту же строку. Определение константы будет добавлено в ReceiveMessageActivity

Запустить ReceiveMessageActivity при помощи интента.

Теперь, когда активность `CreateMessageActivity` добавила в интент дополнительную информацию, необходимо прочитать эту информацию и использовать ее.



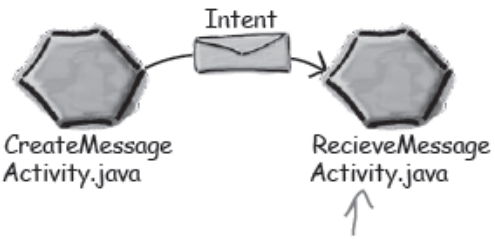
Использование информации из интента в ReceiveMessageActivity

Итак, мы запрограммировали в `CreateMessageActivity` добавление текста в интент; теперь нужно изменить `ReceiveMessageActivity` для использования передаваемого текста.

`ReceiveMessageActivity` будет отображать сообщение в своей надписи при создании активности. Так как метод `onCreate()` активности вызывается сразу же при ее создании, код будет добавлен в этот метод.

Чтобы получить сообщение из интента, мы сначала получим объект интента вызовом `getIntent()`, а затем — сами передаваемые данные вызовом `getStringExtra()`.

Ниже приведен полный код *ReceiveMessageActivity.java* (замените код, сгенерированный Android Studio, и сохраните все изменения):



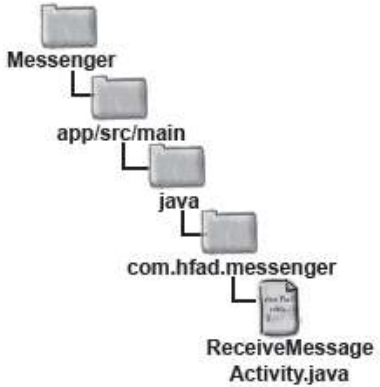
```
package com.hfad.messenger;

import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.widget.TextView;

public class ReceiveMessageActivity extends Activity {

    public static final String EXTRA_MESSAGE = "message";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_receive_message);
        Intent intent = getIntent();
        String messageText = intent.getStringExtra(EXTRA_MESSAGE);
        TextView messageView = (TextView)findViewById(R.id.message);
        messageView.setText(messageText);
    }
}
```

Необходимо импортировать классы Intent и TextView.

Имя дополнительного значения, передаваемого в интенте.

Получить интент и извлечь из него сообщение вызовом getStringExtra().

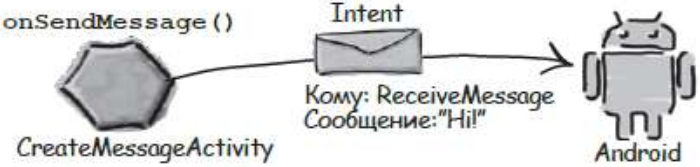
Добавить текст в надпись с идентификатором message.



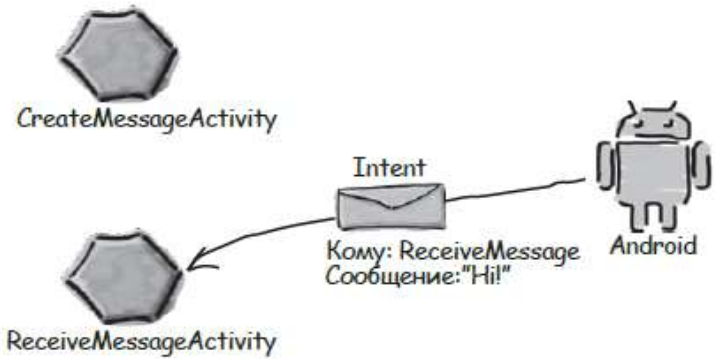
Прежде чем тестировать приложение, еще раз пройдемся по коду и вспомним, что в нем происходит.

Что происходит при щелчке на кнопке Send Message

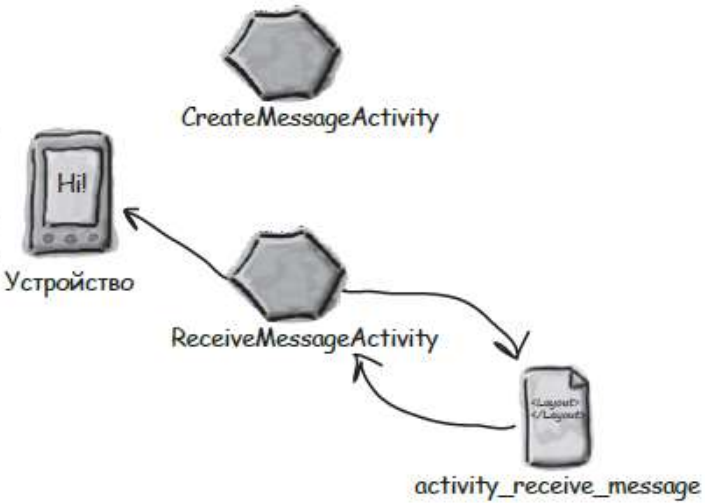
1. **Когда пользователь щелкает на кнопке, вызывается метод onSendMessage().**
Код метода `onSendMessage()` создает интент для запуска активности `ReceiveMessageActivity`, добавляет в интент текст сообщения и передает его Android вместе с инструкцией по запуску активности.



2 **Android проверяет интент на правильность и приказывает ReceiveMessageActivity запуститься.**



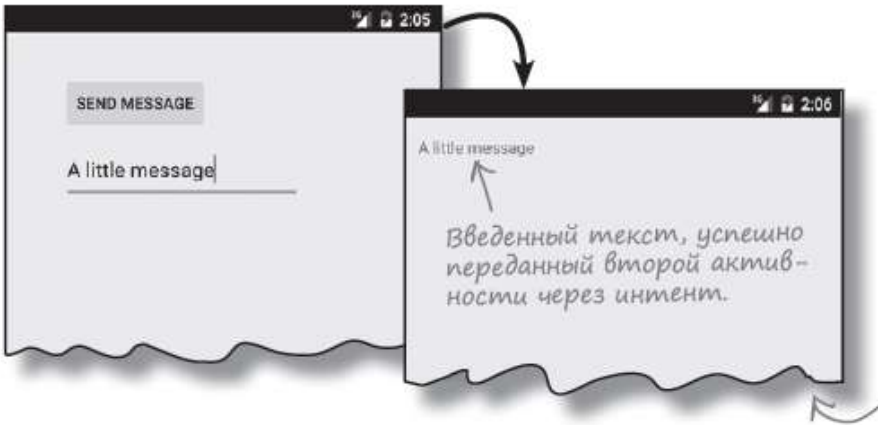
5 **При запуске активность ReceiveMessageActivity указывает, что она использует макет *activity_receive_message.xml*. Этот макет отображается на устройстве.** Активность изменяет макет и выводит в нем текст, полученный из интента.



Тест-драйв

Создание 1-й активности
Создание 2-й активности
Вызов 2-й активности
Передача данных

Проверьте, что вы внесли изменения в обеих активностях, сохраните изменения и запустите приложение. Запускается активность `CreateMessageActivity`; если ввести текст и щелкнуть на кнопке Send Message, запускается `ReceiveMessageActivity`. Текст, введенный в первой активности, появляется в надписи.



Приложение можно изменить так, чтобы сообщения отправлялись другим людям

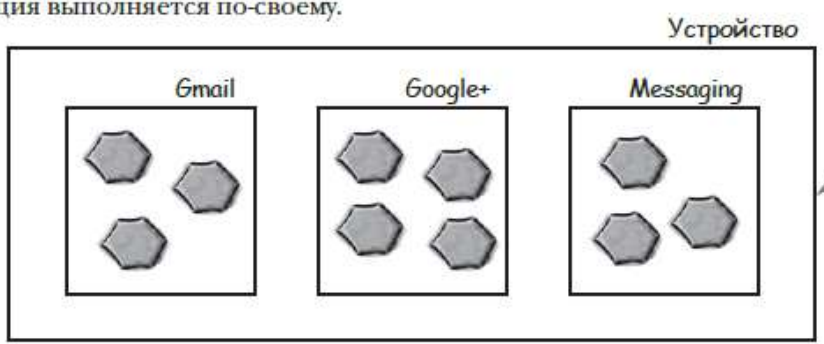
Теперь, когда наше приложение научилось отправлять сообщения другой активности, его можно изменить так, чтобы оно отправляло сообщения другим людям. Для этого приложение интегрируется с другими приложениями, поддерживающими отправку сообщений и уже установленными на устройстве. В зависимости от того, какие приложения установлены у пользователя, можно организовать отправку сообщений из вашего приложения через Gmail, Google+, Facebook, Twitter...

На самом деле все не так сложно, как может показаться, — благодаря особенностям архитектуры Android.

Помните, что говорилось в начале главы о задачах — цепочках из нескольких активностей? Так вот, **при этом совершенно не обязательно ограничиваться активностями вашего приложения.** С таким же успехом можно использовать активности *других* приложений.

Как работают приложения Android

Как вам уже известно, Android-приложения состоят из одной или нескольких активностей, а также других компонентов – например, макетов. Каждая активность представляет одну четко определенную операцию, которая может выполняться пользователем. Например, такие приложения, как Gmail, Google+, Сообщения, Facebook и Twitter, содержат активности, позволяющие отправлять сообщения, хотя в каждом приложении эта операция выполняется по-своему.



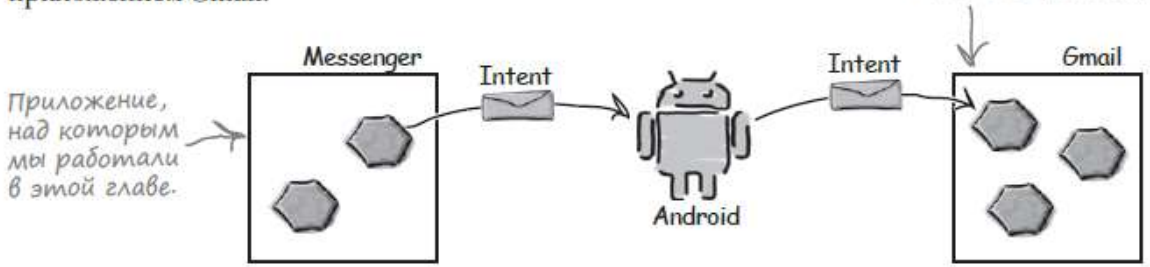
Каждое приложение состоит из активностей. Также существуют и другие компоненты, но сейчас нас интересуют только активности.

Интенты могут запускать активности из других приложений

Вы уже видели, как использовать интент для запуска второй активности из того же приложения. Первая активность передает интент Android; Android проверяет интент, а затем приказывает второй активности запуститься.

Этот принцип относится и к активностям других приложений. Активность вашего приложения передает интент Android, Android проверяет его, а затем приказывает второй активности запуститься — *несмотря на то, что эта активность находится в другом приложении.* Например, можно воспользоваться интентом для запуска активности Gmail, отправляющей сообщения, и передать ей текст, который нужно отправить. Вместо того, чтобы писать собственные активности для отправки электронной почты, можно воспользоваться готовым приложением Gmail.

Вы можете создать интент для запуска другой активности даже в том случае, если активность находится в другом приложении.



Это означает, что объединяя активности на устройстве в цепочку, вы можете строить приложения, обладающие существенно большей функциональностью.

Но мы не знаем, какие приложения установлены на устройстве

Прежде чем вызывать активности из других приложений, необходимо ответить на три вопроса:

- Как узнать, какие активности доступны на устройстве пользователя?
- Как узнать, какие из этих активностей подходят для того, что мы собираемся сделать?
- Как узнать, как использовать эти активности?

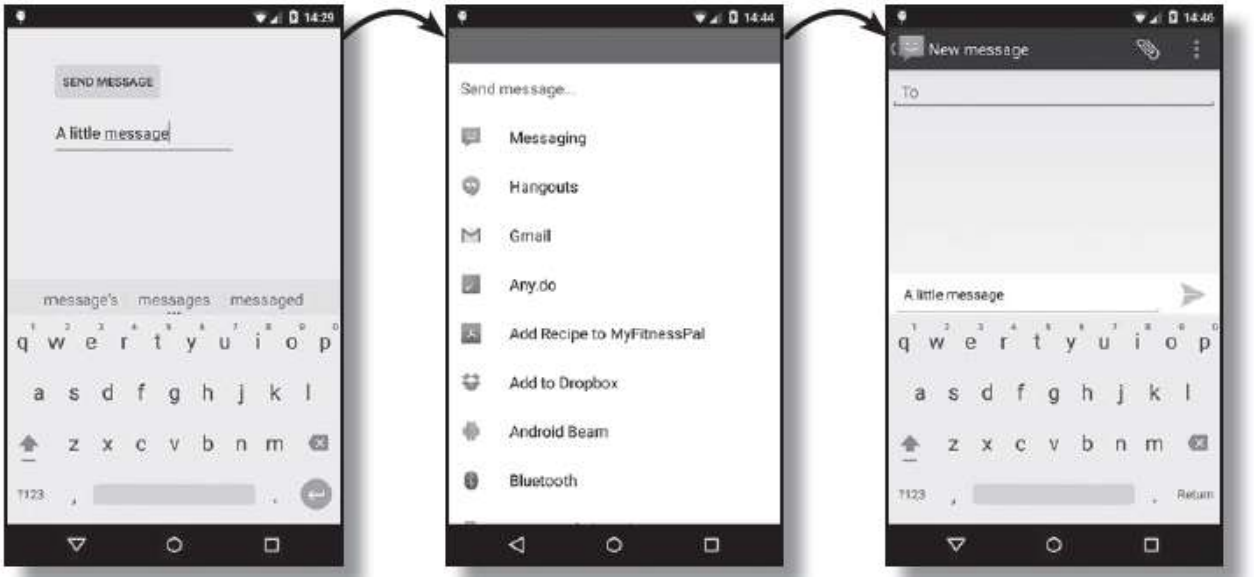
К счастью, все эти проблемы решаются при помощи **действий** (actions). Действия — стандартный механизм, при помощи которого Android узнает о том, какие стандартные операции могут выполняться активностями. Например, Android знает, что все активности, зарегистрированные для действия send, могут отправлять сообщения.

А теперь нужно научиться создавать интенты, использующие действия для получения набора активностей, которые могут использоваться для выполнения стандартных функций — например, для отправки сообщений.

Что мы собираемся сделать

1. **Создать интент с указанием действия.**
Интент сообщит Android, что вам нужна активность, умеющая отправлять сообщения. Интент будет включать текст сообщения.

2. **Разрешить пользователю выбрать используемое приложение.**
Скорее всего, на устройстве установлено сразу несколько приложений, способных отправлять сообщения, поэтому пользователь должен выбрать одно из них. Мы хотим, чтобы пользователь мог выбрать приложение каждый раз, когда он щелкает на кнопке Send Message.



Создание интента с указанием действия

Определение действия
Выбор активности

Ранее вы видели, как создать интент для запуска конкретной активности командой вида

```
Intent intent = new Intent(this, ReceiveMessageActivity.class);
```

Такие интенты называются **явными**; вы явно сообщаете Android, какой класс должна запустить система.

Если требуется выполнить некоторое действие и вас не интересует, какой активностью оно будет выполнено, создайте **неявный интент**. При этом вы сообщаете Android, какое действие нужно выполнить, а все подробности по выбору активности, выполняющей это действие, поручаются Android.

Мы сообщили интенту, для какого класса он предназначен, — а если мы этого не знаем?

Как создать интент

Для создания интента с указанием действия применяется следующий синтаксис:

```
Intent intent = new Intent(действие);
```

где `действие` — тип действия, выполняемого активностью. Android предоставляет целый ряд стандартных вариантов действий. Например, действие `Intent.ACTION_DIAL` используется для набора номера, `Intent.ACTION_WEB_SEARCH` — для выполнения веб-поиска, а `Intent.ACTION_SEND` — для отправки сообщений. Итак, если вы хотите создать интент для отправки сообщения, используйте команду следующего вида:

```
Intent intent = new Intent(Intent.ACTION_SEND);
```

О том, какие действия активностей можно использовать в программах и какую дополнительную информацию они поддерживают, можно узнать в справочных материалах для разработчиков Android: *http://tinyurl.com/n57qb5*.

Добавление дополнительной информации

После определения действия в интент можно включить дополнительную информацию. Допустим, вы хотите добавить текст, который образует тело отправляемого сообщения. Задача решается следующим фрагментом кода:

```
intent.setType("text/plain");
intent.putExtra(Intent.EXTRA_TEXT, текст);
```

Эти атрибуты актуальны для Intent.ACTION_SEND, а не для всех возможных действий.

где `текст` — отправляемый текст. Вызов сообщает Android, что активность должна уметь обрабатывать данные с типом данных MIME "text/plain", а также передает сам текст.

Если потребуется добавить несколько видов дополнительной информации, используйте многократные вызовы метода `putExtra()`. Например, если вы хотите также указать тему сообщения, используйте вызов вида

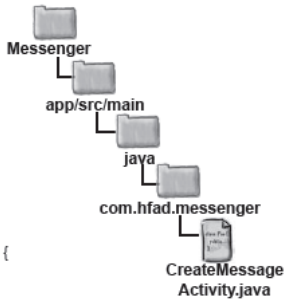
```
intent.putExtra(Intent.EXTRA_SUBJECT, тема);
```

Если информация о теме не актуальна для конкретного приложения, оно просто проигнорирует эту информацию. С другой стороны, любое приложение, которое умеет ее использовать, так и поступит.

где `тема` — тема сообщения.

Изменение интента для использования действия

Мы изменим файл *CreateMessageActivity.java* так, чтобы в нем создавался неявный интент для использования действия отправки. Внесите изменения, выделенные жирным шрифтом, и сохраните свою работу:




```
package com.hfad.messenger;

import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.View;
import android.widget.EditText;

public class CreateMessageActivity extends Activity {

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

    //Вызвать onSendMessage() при щелчке на кнопке
    public void onSendMessage(View view) {
        EditText messageView = (EditText)findViewById(R.id.message);
        String messageText = messageView.getText().toString();
        Intent intent = new Intent(this, ReceiveMessageActivity.class);
        intent.putExtra(ReceiveMessageActivity.EXTRA_MESSAGE, messageText);
        Intent intent = new Intent(Intent.ACTION_SEND);
        intent.setType("text/plain");
        intent.putExtra(Intent.EXTRA_TEXT, messageText);
        startActivity(intent);
    }
}
```

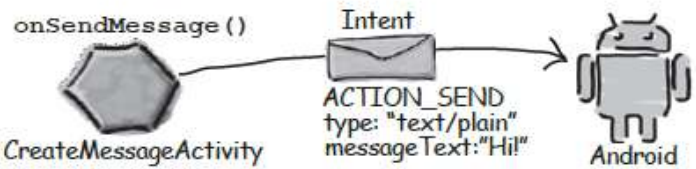
Удалите эти две строки.

Вместо того, чтобы создавать интент, предназначенный конкретно для ReceiveMessageActivity, мы создаем интент с указанием действия отправки.

Давайте подробно проанализируем, что происходит, когда пользователь щелкает на кнопке Send Message.

Что происходит при выполнении кода

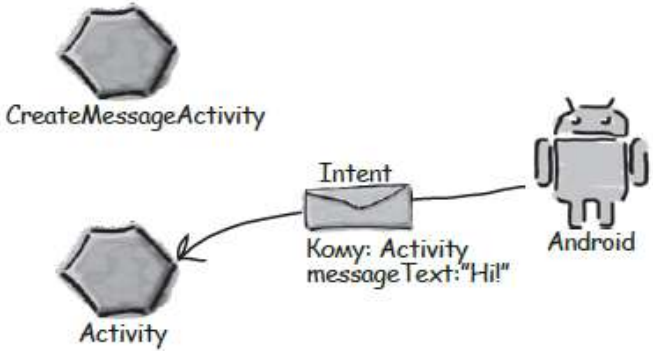
1 **При вызове метода onSendMessage() создается интент. Метод startActivity() передает интент Android.**
Интенту назначается действие `ACTION_SEND` и тип MIME text/plain.



2 **Android видит, что интент может передаваться только активностям, способным обрабатывать действие ACTION_SEND и данные text/plain. Android проверяет все активности и ищет среди них те, которые смогут обработать интент.**

Если ни одно действие не способно обработать интент, инициируется исключение `ActivityNotFoundException`.

5 **Если только одна активность способна обработать интент, Android приказывает этой активности запуститься и передает ей интент.**

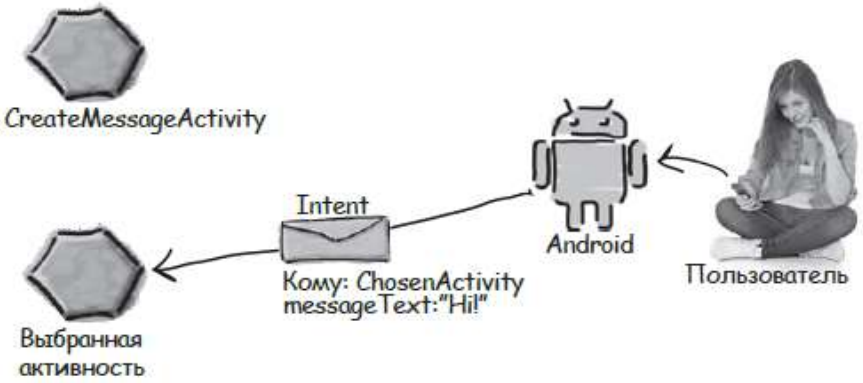


История продолжается...

4 **Если найдется несколько активностей, способных обработать интент, Android открывает диалоговое окно для выбора активности и предлагает пользователю выбрать.**

CreateMessageActivity

5 **Когда пользователь выберет активность, которую он хочет использовать, Android приказывает активности запуститься и передает ей интент.** Активность выводит дополнительный текст, содержащийся в интенте, в теле нового сообщения.



Чтобы создать диалоговое окно для выбора активности, система Android должна знать, какие активности способны получить интент. На ближайшей паре страниц вы узнаете, как это делается.

Фильтр интентов сообщает Android, какие активности могут обработать те или иные действия

При получении интента система Android должна определить, какая активность (или активности) может этот интент обработать. Этот процесс называется **разрешением интента**.

При использовании *явного* интента процесс разрешения тривиален: в самом интенте явно указано, для какого компонента он предназначен, поэтому у Android имеются четкие инструкции, что с ним делать. Например, следующий код явно приказывает Android запустить `ReceiveMessageActivity`:

```
Intent intent = new Intent(this, ReceiveMessageActivity.class);
startActivity(intent);
```

При использовании *неявного* интента система Android использует информацию, содержащуюся в интенте, для определения того, какие компоненты могут его получить. Для этого Android проверяет фильтры интентов, содержащиеся в экземплярах *AndroidManifest.xml* всех приложений.

Фильтр интентов указывает, какие типы интентов могут обрабатываться каждым компонентом. Например, следующая запись относится к активности, способной обрабатывать действие `ACTION_SEND`. Эта активность принимает данные с MIME-типами text/plain или image:

```
<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>
```

Сообщает Android, что активность может обрабатывать ACTION_SEND.

Фильтр интентов должен включать категорию DEFAULT; в противном случае он не сможет получать неявные интенты.

Типы данных, которые могут обрабатываться активностью.

Фильтр интентов также включает **категорию**. Категория предоставляет дополнительную информацию об активности: например, может ли она запускаться браузером или является ли она главной точкой входа приложения. Фильтр интентов ***должен*** включать категорию `android.intent.category.DEFAULT`, если он собирается принимать неявные интенты. Если активность не имеет фильтра интентов или не включает категорию с именем `android.intent.category.DEFAULT`, это означает, что активность не может запускаться неявным интентом. Она может быть запущена только *явным* интентом с указанием полного имени компонента.

Определение действия
Выбор активности

Как Android использует фильтр интентов

Получив неявный интент, Android сравнивает информацию из интента с информацией, содержащейся в фильтрах интентов из файла *AndroidManifest.xml* каждого приложения.

Сначала Android рассматривает фильтры интентов, включающие категорию `android.intent.category.DEFAULT`:

```
<intent-filter>
    <category android:name="android.intent.category.DEFAULT"/>
    ...
</intent-filter>
```

Фильтры интентов без этой категории пропускаются, так как они не могут получать неявные интенты.

Затем Android сопоставляет интенты с фильтрами интентов, сравнивая действия и тип MIME из интента с указанными в фильтрах. Допустим, если в интенте указано действие `Intent.ACTION_SEND`:

Также Android будет проверять категорию фильтра интентов, если она указана в интенте. Данная возможность используется нечасто, поэтому мы не рассматриваем добавление категорий в интенты.

```
Intent intent = new Intent(Intent.ACTION_SEND);
```

Android будет рассматривать только те активности, для которых указан фильтр интентов с действием `android.intent.action.SEND`:

```
<intent-filter>
    <action android:name="android.intent.action.SEND"/>
    ...
</intent-filter>
```

Аналогичным образом, если для интента установлен тип MIME "text/plain":

```
intent.setType("text/plain");
```

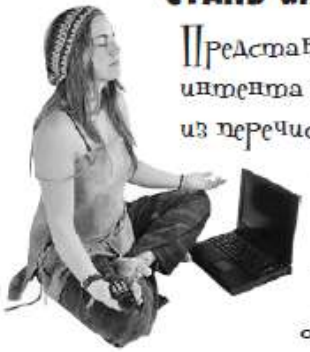
Android будет рассматривать только те активности, которые поддерживают этот тип данных:

```
<intent-filter>
    <data android:mimeType="text/plain"/>
    ...
</intent-filter>
```

Если тип MIME в интенте не указан, то Android пытается вычислить его на основании данных, содержащихся в интенте.

После того как сравнение интента с фильтрами интентов, назначенных компонентам, будет завершено, Android смотрит, сколько совпадений удалось найти. Если найдено только одно совпадение, Android запускает компонент (в нашем случае это активность) и передает ему интент. Если будет найдено несколько совпадений, Android просит пользователя выбрать один из вариантов.

СТАНЬ интентом



Представьте себя на месте интента и скажите, какая из перечисленных ниже активностей совместима с вашим действием и данными. В каждом случае обоснуйте свой ответ.

Это интент.

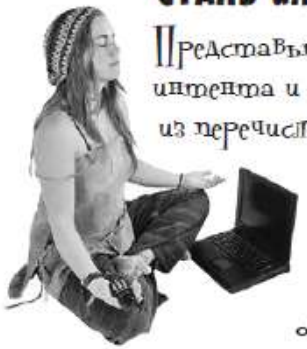
```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.setType("text/plain");
intent.putExtra(Intent.EXTRA_TEXT, "Hello");
```

```
<activity android:name="SendActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="*/*"/>
    </intent-filter>
</activity>
```

```
<activity android:name="SendActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.MAIN"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

```
<activity android:name="SendActivity">
    <intent-filter>
        <action android:name="android.intent.action.SENDTO"/>
        <category android:name="android.intent.category.MAIN"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

СТАНЬ интентом. Решение



Представьте себя на месте интента и скажите, какая из перечисленных ниже активностей совместима с вашим действием и данными. В каждом случае обоснуйте свой ответ.

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.setType("text/plain");
intent.putExtra(Intent.EXTRA_TEXT, "Hello");
```

✓ *Эта активность принимает ACTION_SEND и может обработать данные любого типа MIME, так что она может отреагировать на интент.*

```
<activity android:name="SendActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="*/*"/>
    </intent-filter>
</activity>
```

```
<activity android:name="SendActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.MAIN"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

У этой активности отсутствует категория DEFAULT, поэтому она не сможет получить интент.

```
<activity android:name="SendActivity">
    <intent-filter>
        <action android:name="android.intent.action.SENDTO"/>
        <category android:name="android.intent.category.MAIN"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

Эта активность не принимает интенты ACTION_SEND, только ACTION_SENDTO. Действие ACTION_SENDTO позволяет отправить сообщение получателю, указанному в данных интента.

Запуск приложения на РЕАЛЬНОМ устройстве

До сих пор мы запускали приложения только под управлением эмулятора. Эмулятор включает крайне ограниченную подборку приложений; может оказаться, что на нем есть всего одно приложение, способное обработать ACTION_SEND. Чтобы нормально протестировать приложение, необходимо запустить его на физическом устройстве, на котором заведомо найдется более одного приложения, поддерживающего нужное действие — например, отправку электронной почты или отправку сообщений.

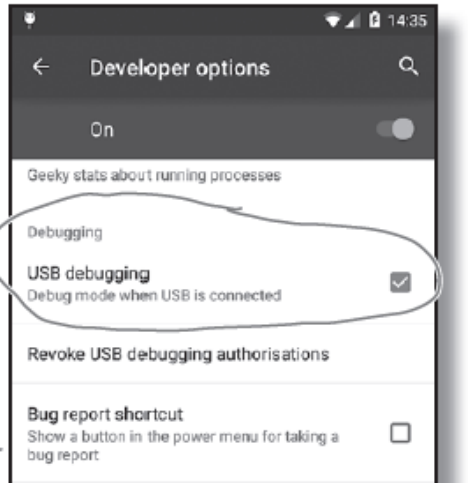
Чтобы протестировать приложение на физическом устройстве, выполните следующие действия:

1. Включите режим отладки через интерфейс USB на устройстве

На устройстве откройте экран "Developer options" (начиная с Android 4.0 по умолчанию этот экран скрыт). Чтобы разрешить его отображение, перейдите в раздел Settings → About Phone и прикоснитесь к номеру сборки семь раз. Когда вы вернетесь к предыдущему экркау, на нем должен появиться раздел "Developer options."

Да, серьезно.

В разделе "Developer options" установите флажок USB debugging.



Необходимо включить отладку через интерфейс USB.

2. Настройте систему для распознавания устройства

Если вы работаете на Mac, пропустите этот шаг.

Если вы работаете в системе Windows, необходимо установить драйвер USB. Самые свежие инструкции можно найти здесь:

http://developer.android.com/tools/extras/oem-usb.html

Если вы работаете в Ubuntu Linux, создайте файл правил `udev`. Самые свежие инструкции относительно того, как это делается, находятся здесь:

http://developer.android.com/tools/device.html#setting-up

3. Подключите свое устройство к компьютеру кабелем USB

Возможно, устройство спросит, хотите ли вы принять ключ RSA, разрешающий отладку через интерфейс USB на вашем компьютере. Если окно с вопросом появится, установите флажок "Always allow from this computer" и щелкните на кнопке OK, чтобы разрешить отладку.

Это сообщение появится в том случае, если ваше устройство работает под управлением Android 4.2.2 и выше.

Запуск приложения на РЕАЛЬНОМ устройстве (продолжение)

4. Запустите приложение в Android Studio, как обычно

Android Studio устанавливает приложение на устройстве и запускает его. Вам будет предложено выбрать, на каком устройстве следует запустить приложение. Выберите свое устройство в списке и щелкните на кнопке OK.

Первое устройство в списке — эмулятор.

А это наше физическое устройство.

Приложение, работающее на физическом устройстве

Приложение на физическом устройстве практически ничем не отличается от приложения, запущенного в эмуляторе. Вероятно, вы заметите, что установка и запуск проходят быстрее.

Теперь, когда вы знаете, как запускать созданные приложения на физическом устройстве, все готово для тестирования новейших изменений в вашем приложении.

Тест-драйв

Сначала запустите приложение в эмуляторе, а потом на физическом устройстве. Полученные результаты будут зависеть от количества активностей на каждом устройстве, поддерживающих действие Send с текстовыми данными.

Если найдена только одна активность

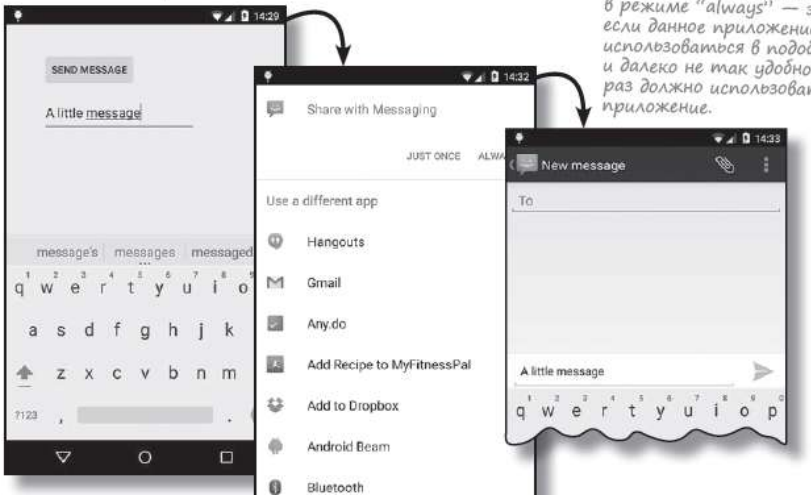
Щелчок на кнопке Send Message переведет вас прямо к этому приложению.

В эмуляторе найдена только одна активность, умеющая отправлять сообщения с текстовыми данными. При щелчке на кнопке Send Message Android запускает эту активность.

Если найдено несколько активностей

Android выводит окно выбора и предлагает указать, какая из активностей должна использоваться. Также предлагается выбрать, должно ли это действие использоваться только в данном случае или всегда. Если выбрать вариант Always, то в дальнейшем при щелчке на кнопке Send Message та же активность всегда будет использоваться по умолчанию.

На нашем физическом устройстве найдено много подходящих активностей. Мы решили использовать приложение Сообщения (Messaging) в режиме "always" — это удобно, если данное приложение должно всегда использоваться в подобных случаях, и далеко не так удобно, если каждый раз должно использоваться новое приложение.



А если вы хотите, чтобы пользователь ВСЕГДА выбирал активность?

Вы уже видели, что при обнаружении на устройстве нескольких активностей, способных принять интент, Android автоматически предлагает выбрать нужную активность. Пользователь даже может указать, когда должна использоваться эта активность — всегда или только в данном случае.

Однако у этого стандартного поведения есть один недостаток: а что если вы хотите *гарантировать*, что пользователь сможет выбрать активность при каждом щелчке на кнопке Send Message? Например, если он приказал всегда использовать Gmail, то в следующий раз Android уже не предложит ему выбрать Twitter. К счастью, у этой проблемы есть обходное решение. Вы можете создать окно выбора, в котором пользователю будет предложено выбрать активность без каких-либо возможностей всегда использовать именно ее.

Intent.createChooser() выводит диалоговое окно выбора

В этом вам поможет метод `Intent.createChooser()`. Он получает уже созданный интент и «упаковывает» его в диалоговое окно выбора. Главная отличительная особенность этого метода — он не предоставляет возможности выбора активности по умолчанию, то есть пользователю придется каждый раз выбирать нужную активность.

Метод createChooser( ) позволяет задать заголовок окна выбора и не дает возможности выбрать активность, используемую по умолчанию. Если ни одной подходящей активности не найдено, то пользователь будет оповещен об этом при помощи сообщения.

Вызвать метод `createChooser()` можно так:

```
Intent chosenIntent = Intent.createChooser(intent, "Send message...");
```

Интент, созданный ранее.

Вы можете передать заголовок окна выбора, который будет отображаться у верхнего края экрана.

Метод получает два параметра: интент и необязательный заголовок диалогового окна выбора в формате `String`. Параметр `Intent` должен описывать типы активностей, которые должны выводиться в окне выбора. Вы можете использовать интент, созданный ранее, так как он указывает, что для его обработки требуется поддержка `ACTION_SEND` с текстовыми данными.

Метод `createChooser()` возвращает новый объект `Intent`. Он представляет собой новый явный интент, предназначенный для активности, выбранной пользователем. Он содержит всю дополнительную информацию, передававшуюся в исходном интенте, включая весь текст.

Чтобы запустить активность, выбранную пользователем, нужно вызвать:

```
startActivity(chosenIntent);
```

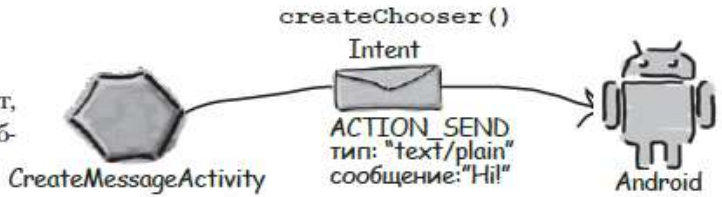
Сейчас вы узнаете, что происходит при вызове метода `createChooser()`.

Что происходит при вызове createChooser()

Давайте разберемся, что происходит при выполнении следующих двух строк кода:

```
Intent chosenIntent = Intent.createChooser(intent, "Send message...");
startActivity(chosenIntent);
```

1. **Вызывается метод createChooser().**
При вызове указывается интент, определяющий действие, и необходимый тип MIME.



2 **Android проверяет, какие активности могут обработать интент, по их фильтрам интентов.**
Совпадения проверяются по действиям, типам данных и поддерживаемым категориям.

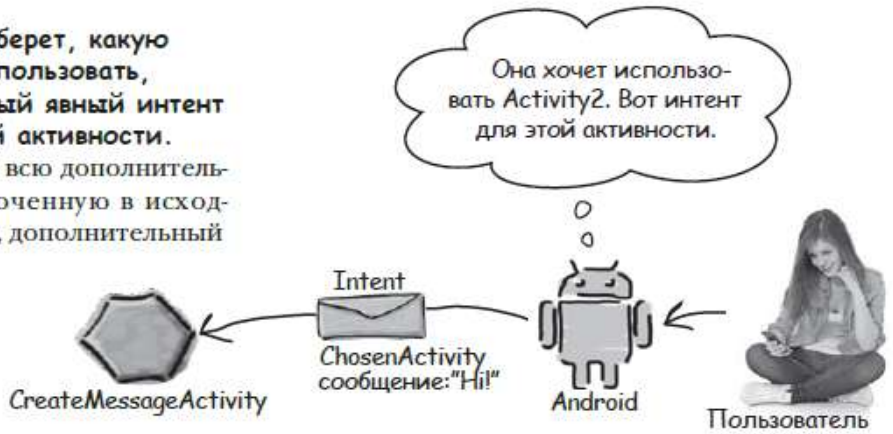
5 **Если интент может быть обработан несколькими активностями, Android отображает диалоговое окно для выбора активности. В этом окне пользователь указывает, какую активность следует использовать.**

На этот раз у пользователя нет возможности выбрать активность по умолчанию, а в заголовке отображается строка "Send message...".

Если Android не находит ни одной подходящей активности, то окно все равно отображается, но пользователь получает сообщение об отсутствии приложений, способных выполнить действие.

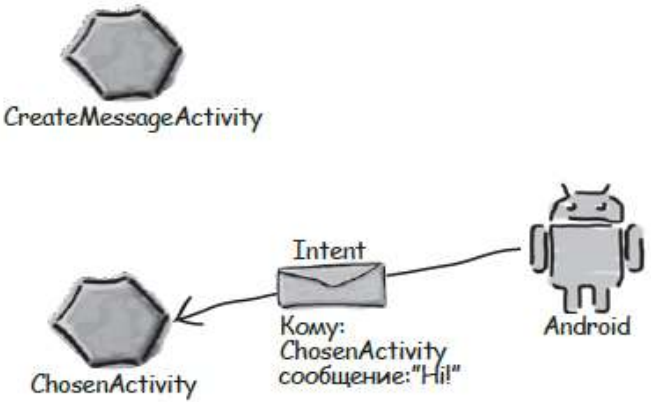
История продолжается...

4 **Когда пользователь выберет, какую активность он хочет использовать, Android возвращает новый явный интент с описанием выбранной активности.** Новый интент содержит всю дополнительную информацию, включенную в исходный интент (в частности, дополнительный текст).



5 **Исходная активность приказывает Android запустить активность, указанную в интенте.**

6 **Android запускает активность, указанную в интенте, и передает ей интент.**



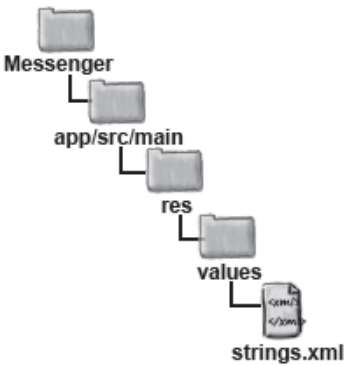
Изменение кода выбора активности

Мы изменим код так, чтобы при каждом щелчке на кнопке Send Message пользователю предлагалось выбрать активность, используемую для отправки сообщения. Мы обновим метод `onSendMessage()` из файла *CreateMessageActivity.java* так, чтобы он вызывал метод `createChooser()`, а также добавим в *strings.xml* строковый ресурс для текста в заголовке окна выбора.

Обновление strings.xml...

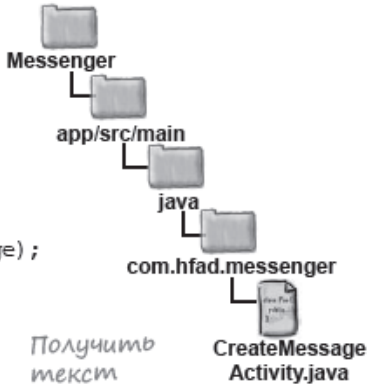
В заголовке диалогового окна выбора должен отображаться текст "Send message...". Добавьте в *strings.xml* строку с именем "chooser" и присвойте ей значение "Send message..." (не забудьте сохранить изменения):

```
...
<string name="chooser">Send message...</string>
...
```



...и обновление метода onSendMessage()

Метод `onSendMessage()` необходимо изменить так, чтобы он получал значение ресурса chooser из файла *strings.xml*, вызывал метод `createChooser()`, после чего запускал активность, выбранную пользователем. Приведите код к следующему виду:




```
...
    //Вызвать onSendMessage() при щелчке на кнопке
    public void onSendMessage(View view) {
        EditText messageView = (EditText)findViewById(R.id.message);
        String messageText = messageView.getText().toString();
        Intent intent = new Intent(Intent.ACTION_SEND);
        intent.setType("text/plain");
        intent.putExtra(Intent.EXTRA_TEXT, messageText);
        String chooserTitle = getString(R.string.chooser);
        Intent chosenIntent = Intent.createChooser(intent, chooserTitle);
        startActivity(intent);
        startActivity(chosenIntent);
    }
...
```

Получить текст заголовка.

Вывести диалоговое окно выбора.

Запустить активность, выбранную пользователем.

Метод `getString()` используется для получения значений строковых ресурсов. Он получает один параметр — идентификатор ресурса (в нашем случае `R.string.chooser`):

```
getString(R.string.chooser);
```

Заглянув в файл R.java, вы найдете chooser во внутреннем классе с именем string.

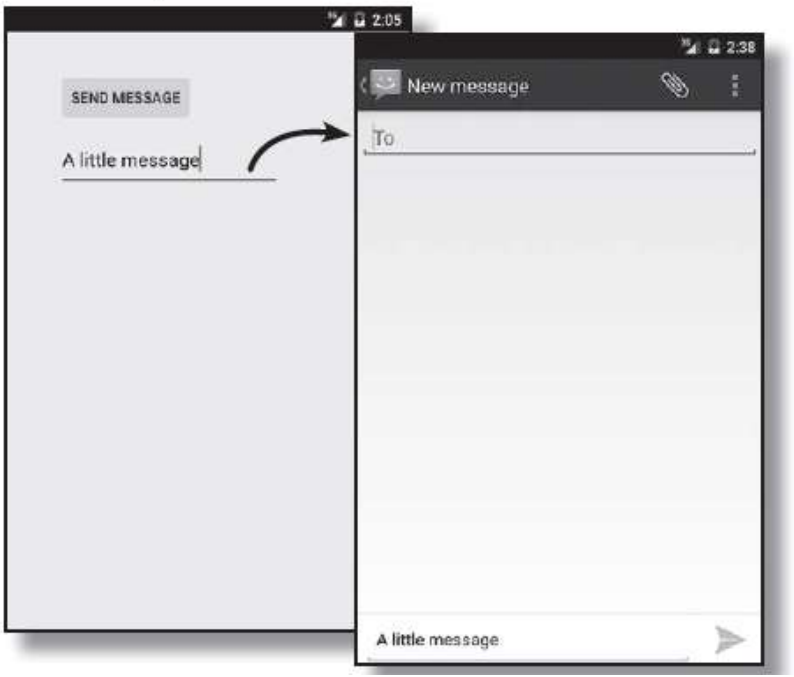
После того как в приложение будут внесены все необходимые изменения, запустите приложение и посмотрите, как работает окно выбора.

Сохраните изменения и попробуйте снова запустить приложение.

Если найдена только одна активность

Щелчок на кнопке Send Message приведет вас сразу к приложению, как и прежде.

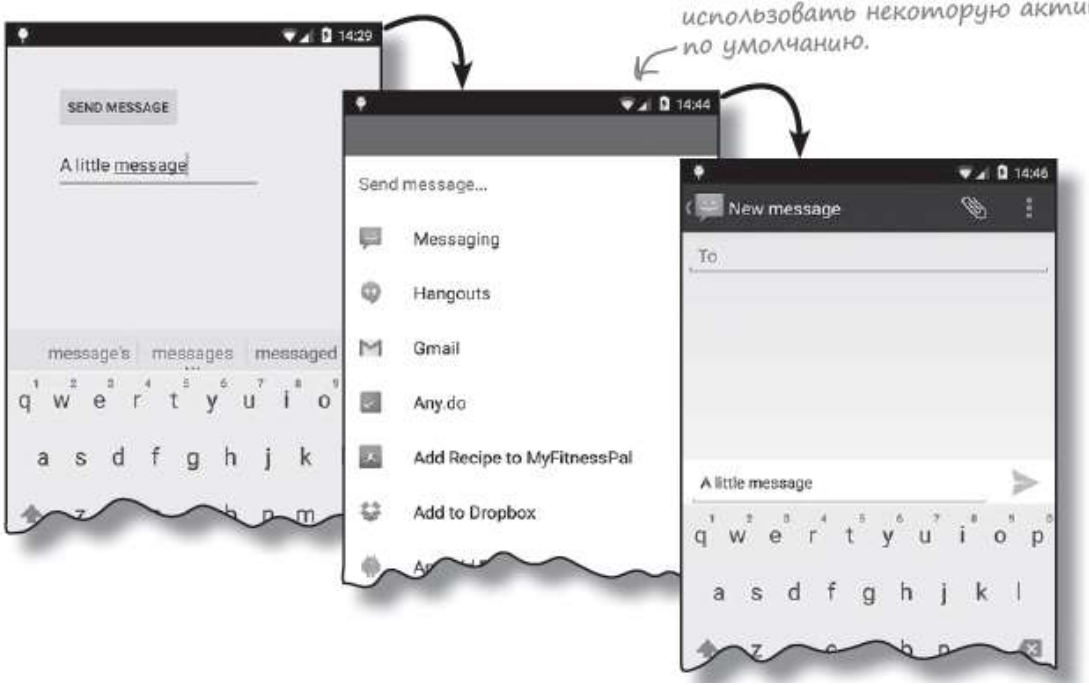
Здесь ничего не изменилось — Android, как и прежде, сразу запускает активность.



Если найдено несколько активностей

Android выводит окно выбора, но на этот раз не спрашивает, нужно ли всегда использовать некоторую активность. Также в заголовке выводится значение строкового ресурса.

Окно выбора, созданное вызовом createChooser(). Оно уже не предлагает использовать некоторую активность по умолчанию.



Если подходящих активностей НЕТ

Если на устройстве не обнаружено ни одной активности, способной отправлять сообщения, метод `createChooser()` выводит соответствующее сообщение.

Это еще одно из преимуществ метода `createChooser()`. Метод `createChooser()` корректно справляется с ситуацией, в которой указанное действие не может быть выполнено ни одной активностью.

Если вы захотите воспроизвести это сообщение, попробуйте запустить приложение в эмуляторе с отключением приложения Сообщения.

ВИСНОВКИ

- Задачей называются две и более активности, объединенные в цепочку.
- Элемент `<EditText>` определяет текстовое поле с возможностью редактирования и ввода текста. Класс текстового поля наследует от класса Android `View`.
- Новая активность в Android Studio создается командой File → New... → Activity.
- Для каждой создаваемой активности в файле *AndroidManifest.xml* должна быть создана запись.
- **Интент** представляет собой разновидность сообщений, используемых для организации взаимодействия между компонентами Android.
- Явный интент предназначен для конкретного компонента. Явный интент создается командой
  `Intent intent = new Intent(this, Target.class);`
- Активности запускаются вызовом `startActivity(intent)`. Если ни одна подходящая активность не найдена, метод инициирует исключение `ActivityNotFoundException`.
- Используйте метод `putExtra()` для включения дополнительной информации в интент.
- Используйте метод `getIntent()` для получения интента, запустившего активность.
- Используйте методы `get*Extra()` для чтения дополнительной информации, связанной с интентом. Метод `getStringExtra()` читает String, `getIntExtra()` читает int, и т. д.
- Действие описывает стандартную операцию, которую может выполнять активность. Так, для отправки сообщений используется обозначение `Intent.ACTION_SEND`.
- Чтобы создать неявный интент с указанием действия, используйте запись
  `Intent intent = new Intent(action);`
- Для описания типа данных в интенте используется метод `setType()`.
- Android производит разрешение интентов на основании имени компонента, действия, типа данных и категорий, указанных в интенте. Содержимое интента сравнивается с фильтрами интентов из файла *AndroidManifest.xml* каждого приложения. Чтобы активность получала неявные интенты, она должна включать категорию `DEFAULT`.
- Метод `createChooser()` позволяет переопределить стандартное диалоговое окно выбора активности в Android. При использовании этого метода можно указать текст заголовка, а у пользователя нет возможности назначить активность по умолчанию. Если метод не находит ни одной активности, способной получить переданный интент, он выводит сообщение. Метод `createChooser()` возвращает объект `Intent`.
- Для чтения значений строковых ресурсов используется синтаксис `getString(R.string.stringname);`.