
Программирование приложений
для мобильных устройств под
управлением Android
ANDROID

Евгений Владимирович Сенько

Программирование приложений для мобильных устройств под управлением Android. Часть 1

http://www.litres.ru/pages/biblio_book/?art=11643376

Аннотация

Книга посвящена разработке программ для мобильных устройств под управлением операционной системы Android. Рассматривается создание приложений с использованием системных компонентов и служб Android. Приведены базовые данные о структуре приложений, об основных классах и их методах, сопровождаемые примерами кода. Часть 1 содержит шесть глав, описывающих основные принципы создания приложений, пользовательский интерфейс, полномочия приложений, а так же базовые классы: Activity, Intent, Fragment. Книга предназначена для программистов, владеющих языком программирования Java и желающих освоить написание приложений, работающих под ОС Android. Книга является переводом общедоступных бесплатных англоязычных интернет ресурсов. Во второй части книги будут рассмотрены

Нотификации (Notifications), Broadcast Receivers, Потоки и асинхронное выполнение задач (Threads & AsyncTasks), Оповещения (Alarms), работа с сетью, графика и анимация, управление тачем и жестами, управление мультимедией, работа с датчиками, определение местоположения и привязка к картам, управление данными, а также классы ContentProvider и Service.

Содержание

Основные принципы создания приложения	5
Класс Activity	22
Класс Intent	40
Permissions	54
Класс Fragment	60
Классы пользовательского интерфейса	79

Евгений Сенько Программирование приложений для мобильных устройств под управлением Android. Часть 1

Основные принципы создания приложения

Сначала мы поговорим о четырех фундаментальных строительных блоках, из которых строятся все Android-приложения. Эти блоки реализованы в виде Java-классов. И первый из этих строительных блоков – класс Activity. Это основной класс, который видят пользователи при запуске приложения. Activity разработаны с целью обеспечения графического пользовательского интерфейса или GUI, и они позволяют пользователям передавать и получать информацию из приложения.

Остальные три компонента работают «за кулисами», по-

этому они не имеют пользовательских интерфейсов. Эти компоненты включают в себя:

Service для поддержки длительных или работающих в фоновом режиме операций;

Broadcast receiver – для прослушивания и реагирования на события, которые происходят на устройстве;

Content provider, который позволяет нескольким приложениям хранить и обмениваться данными.

Приложения обычно создаются из нескольких взаимодействующих компонентов, которые запускает Android, чтобы все работало как надо. И каждый из этих компонентов выполняет свои функции в экосистеме Android. И, следовательно, имеет свою точку входа и собственный API. Давайте взглянем на каждый из этих компонентов по одному.

Во-первых, давайте поговорим о классе Activity. Этот класс предоставляет графический интерфейс пользователя и обеспечивает взаимодействие с пользователем через этот интерфейс. Как правило, Activity должна поддерживать одно конкретное действие, которое пользователь может сделать. Такое действие, как набор телефонного номера или ввод контактной информации для одного человека, и так далее. Хотя сейчас уже появились устройства с большими экранами, особенно планшеты, и то, что мы называем одним конкретным действием, которое пользователь может сделать, конечно, может измениться. Теперь, в качестве примера Activity, давайте взглянем на приложение «Номерона-

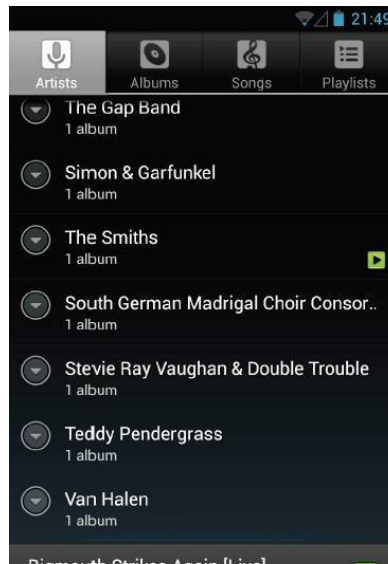
бираТель».

Вы наверняка знаете, что приложение «Телефон» может открыть пользовательский интерфейс с несколькими вкладками. Одна – для набора номера телефона, одна – для списка вызовов, одна – для контактов. В Android 4.2 исходный код этого приложения фактически является частью приложения «Контакты».

Следующий компонент – это класс Service (сервис). В от-

личие от Activity сервисы выполняются в фоновом режиме. Поэтому нет необходимости в пользовательских интерфейсах. Вместо этого сервисы имеют две главные цели. Первое – они могут выполнять длительные операции, как правило, отдельно от основного потока пользовательского интерфейса. И второе – они обеспечивают взаимодействие различных процессов для обмена данными.

Для примера давайте взглянем на приложение «Музыка». Приложение «Музыка» имеет несколько различных экранов пользовательского интерфейса, которые показывают, например, ваши музыкальные записи. Показаны песни в исполнении одного артиста, позволяя вам выбрать одну песню данного исполнителя и проиграть эту песню.



Теперь, если вы запустите воспроизведение песни, но затем решите вернуться и посмотреть, какие еще есть песни данного исполнителя или, если вы хотите сделать что-то совершенно другое, например проверить электронную почту, то вы, вероятно, не захотите чтобы при этом воспроизведение прекратилось. Android обеспечит это с помощью сервиса, чтобы музыка продолжала играть, пока вы проверяете почту.

Следующий компонент – это Broadcast receiver (приемник

сообщений и событий). Broadcast receiver «слушает» и реагирует на события. И, по сути, он играет роль подписчика в модели «публикация-подписка».

В Android события представлены классом Intent. О нем мы поговорим подробнее немного позже. «Издатели» создают этот Intent и затем рассылают его, используя специальные методы, как бы отправляя в эфир. Далее его принимает Broadcast receiver, который и подписан на этот конкретный Intent. И, приняв Intent, он может отреагировать на произошедшее событие.

Пример приложения, которое использует Broadcast receiver – приложение для обмена сообщениями. Давайте представим, что кто-то хочет отправить мне SMS-сообщение. SMS-сообщение будет создано и отправлено через телефонную сеть и в конечном счете достигнет моего телефона. И когда это произойдет, Android выставит значок уведомления в панели уведомлений, который даст мне знать, что для меня пришло сообщение. Но вы, конечно, никогда не можете точно знать, в какой момент вы получите такое сообщение. Поэтому Android имеет некоторое программное обеспечение, которое просто «сидит» и «ждет» когда придет SMS-сообщение. И когда оно приходит, это программное обеспечение передает в эфир Intent SMS_received. И есть еще один Broadcast receiver, который слушает этот Intent SMS_received, и он, получив этот Intent, запустит сервис, который загрузит и сохранит входящее SMS-сообщение.

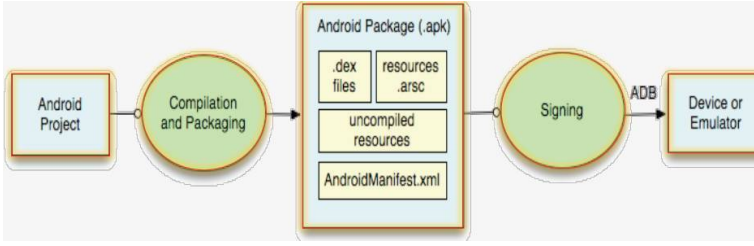
Последний компонент – это Content provider. Content provider позволяет приложениям хранить и обмениваться данными. Content provider использует интерфейс, подобный базам данных, но он больше, чем просто база данных. Например, Content provider сможет обрабатывать связи между процессами. Так что приложения, работающие в отдельных процессах, могут взаимодействовать и обмениваться данными безопасно и легко. Приложение «Браузер» является одним из примеров приложений, использующих Content provider. Если запустить «Браузер» и нажать на значок рядом с адресной строкой в браузере, откроется список закладок и сохраненных адресов веб-сайтов для быстрого доступа в будущем. Когда пользователь добавляет одну из этих закладок, браузер сохраняет ее в Content provider.

Если посмотреть на простейшее приложение «Hello Android», мы увидим всего одну Activity. Но мы будем рассматривать более сложные приложения, в нашем случае, включающее в себя две Activity. И, конечно, как вы понимаете, если можно использовать две Activity, то можно добавить и третью, и четвертую и так далее – просто будет больше одной.

Например, приложение «MapLocation» одним из видов Activity позволяет пользователю ввести почтовый адрес. Эта Activity имеет кнопку, которую пользователь может нажать, как только он ввёл адрес. И, нажав на эту кнопку, запустить вторую Activity – «Google Maps», которая представляет со-

бой карту, отцентрированную на адрес, который пользователь только что ввёл.

Следующая схема показывает процесс написания и построения приложений под Android.



Во-первых, вы создаете исходный код и код ресурсов, которые составляют приложение. Далее, вы компилируете исходный код и подготавливаете свои ресурсы. Результатом этого шага является пакет Android или APK, который является исполняемым файлом приложения. Далее APK подписывается цифровой подписью, чтобы идентифицировать вас как разработчика и, наконец, APK устанавливается на устройство или эмулятор и запускается.

Ваше участие в процессе сборки как разработчика, как правило, включает следующие четыре шага:

- определение ресурсов;
- реализация классов приложения;
- упаковка приложения;
- установка и запуск приложения, в частности для тести-

рования и отладки.

Шаг 1, определение ресурсов приложения. Приложения для Android больше, чем просто исходный код. Они включают в себя исходники, такие как файлы лейаута (layout – размещение наэкранных элементов: кнопок, полей ввода и т. д.), строковых констант, изображений, меню, анимации и многое другое. Управление ресурсами отдельно от приложения имеет несколько преимуществ, одно из которых заключается в том, что вы можете легко изменить эти ресурсы без изменения и перекомпиляции исходного кода приложения.

Один распространенный тип ресурсов – strings (строки). В Android существует три типа строковых ресурсов. Отдельные строки – strings, массивы строк – arrays и формы множественного числа – plurals. Строки и массивы строк довольно понятны, давайте поговорим о plurals. Plurals в основном – это массивы строк, которые могут использоваться, чтобы выбрать определенные строки, связанные с определенной величиной (количеством), например, одна книга или две книги.

Строки обычно хранятся в xml-файлах в каталоге приложения res/values. Строка может включать форматирование и информацию о стилях. Такие вещи, как HTML-теги, например. Основным преимуществом хранения строковых констант в файле ресурсов является простота интернационализации Android-приложений, т. е. их перевод на иностранные языки. Если вы хотите создать приложение, переведенное на несколько разных языков, то рекомендуется выбрать

язык по умолчанию – английский и поместить строки на английском языке в xml-файл strings в папку ресурсов, используемую по умолчанию – res/values. А файлы со строками на других языках – в соответствующие папки, например, для русского языка – в res/values-ru, для французского – в res/values-fr, немецкого – в res/values-de и так далее.

strings.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="show_map_string">Show Map</string>
    <string name="location_string">Enter Location</string>
</resources>
```

strings.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="show_map_string">Mostra la mappa</string>
    <string name="location_string">Digita l\'indirizzo</string>
</resources>
```

Во время компиляции приложения автоматически генерируется класс R.java. Java-код использует R-класс для доступа к ресурсам.

Наконец, другие файлы ресурсов могут ссылаться на строки, которые вы определили как @string/string_name. Вы также можете получить доступ к этим строкам из java-кода, но на этот раз вы делаете это так: R.string.string_name.

Другой вид ресурса – Layout или файл лейаута. Layout-

файл определяет на что будет похож пользовательский интерфейс для каждой части (каждого экрана) вашего приложения. Эти файлы так же записаны в xml, несмотря на то, что некоторые инструменты позволяют вам создавать лейаут визуально, вручную, и затем эти инструменты сгенерируют xml для вас.

К примеру, Eclipse создает Layout-файлы в каталоге res/layout вашего приложения. И вы можете получить доступ к Layout в Java как R.layout.layout_name. Так же вы можете получить доступ к Layout в других файлах ресурсов как @layout/layout_name.

Android позволяет вам создавать различные Layout-файлы для каждого разрешения экрана, ориентации экрана, размера экрана и т. д., и затем Android может выбрать из этих файлов нужный прямо во время выполнения приложения на основе конфигурации вашего устройства.

Давайте рассмотрим файл R.java. Как уже упоминалось, этот файл автоматически сгенерирован Android, таким образом вы не должны изменять его. Файл определяет класс R, и этот класс R содержит другой класс под названием Layout, у которого есть поле под названием Main. И это фактически дает вам ссылку или дескриптор к файлу main.xml.

Есть также класс ID, обеспечивающий дескрипторы для различных лейаутов, текстовых полей, полей редактирования и к кнопкам, определенным в файле main.xml. Если вы вернетесь и посмотрите main.xml-файл, вы увидите, что есть

строки, в которых написано «Android: ID». Это – то, откуда эти поля и ID появляются.

И наконец, есть класс String, который обеспечивает дескрипторы для всех строк, о которых мы говорили.

Следующий шаг в разработке приложений – это реализация ваших Java-классов. Это обычно включает в себя написание по крайней мере одной Activity. Точка входа для Activity – метод onCreate. Это – то, где вы будете обычно инициализировать свое приложение.

Давайте рассмотрим метод onCreate немного более детально. В onCreate вы обычно делаете следующие четыре вещи:

• вы восстанавливаете сохраненное состояние приложения;

• вы устанавливаете визуальное представление content view, которое сообщает андроиду, что необходимо вывести на экран в качестве пользовательского интерфейса Activity;

• вы инициализируете определенные элементы пользовательского интерфейса своего Activity;

• и последнее – вы присоединяете код к элементам пользовательского интерфейса так, чтобы определенные действия были выполнены, когда пользователи будут взаимодействовать с этими элементами Activity.

Давайте посмотрим, как эти шаги реализованы в «MapLocation». Исходный код «MapLocation» реализует класс под названием MapLocation, который является под-

классом Activity.

```
@Override
protected void onCreate(Bundle savedInstanceState) {

    // Required call through to Activity.onCreate()
    // Restore any saved instance state
    super.onCreate(savedInstanceState);

    // Set content view
    setContentView(R.layout.main);

    // Initialize UI elements
    final EditText addrText = (EditText) findViewById(R.id.location);
    final Button button = (Button) findViewById(R.id.mapButton);



   // Link UI elements to actions in code
   button.setOnClickListener(new OnClickListener() {

       // Called when user clicks the Show Map button
       public void onClick(View v) {
           try {

               // Process text for network transmission
               String address = addrText.getText().toString();
               address = address.replace(' ', '+');

               // Create Intent object for starting Google Maps application
               Intent geoIntent = new Intent(
                       android.content.Intent.ACTION_VIEW, Uri
                               .parse("geo:0,0?q=" + address));

               // Use the Intent to start Google Maps application using Activity.startActivity()
               startActivity(geoIntent);

           } catch (Exception e) {
               // Log any error messages to LogCat using Log.e()
               Log.e(TAG, e.toString());
           }
       }
   });
}
```

На первом шаге onCreate должен вызвать super.onCreate, передав сохраненное состояние в качестве параметра. Это

сохраненное состояние (savedInstanceState) – в основном структура данных, содержащая любую информацию, которую Android, возможно, сохранил с прошлого раза, когда эта Activity работала.

Мы будем говорить больше об этом далее, когда мы погрузимся глубже в класс Activity. Но на данный момент просто знайте, что onCreate должен вызвать super.onCreate, или вы получите ошибку.

Затем идет вызов setContentView. В нашем случае для вывода на экран содержимого файла лейаута этого приложения – R.layout.main. После этого идет некоторый код, который устанавливает соответствия отдельных элементов UI с лейаутом. Такие как текстовое поле редактирования, которое сохранено в вызываемой переменной addrText, и кнопка, которая сохранена в переменной button. Как видите, эти ссылки получены вызовом метода activity.findViewById, содержащем в параметре идентификатор желаемого элемента.

И наконец, есть некоторый код, который определяет что сделать, когда пользователь нажимает кнопку Show Map. Этот код реализует интерфейс onClickListener («слушатель» кликов), в котором есть метод onClick, вызывающийся каждый раз, когда пользователь кликает по кнопке. Код в методе onClick сначала считывает текст, который пользователь ввел в поле адреса. Затем он обрабатывает этот текст, чтобы удалить пробелы, и преобразовывает его в формат, который понимает «Google Maps». Затем он запускает приложение

«Google Maps» и передает ему этот преобразованный адрес.

Что этот код делает, более подробно мы рассмотрим позже, когда погрузимся в изучение классов Activity и Intent. Также для упрощения я не учел любую проверку на ошибки или проверку строки адреса, которая была введена. Но в конечном коде вы обязательно это сделаете.

Следующим шагом в создании Android-приложений является предоставление информации, которая позволяет Андроиду создать пакет приложения или APK, и эта информация записывается в манифесте – xml-файле androidmanifest.xml. Файл androidmanifest.xml содержит большое разнообразие упаковочной информации, включая имя приложения и список компонентов, которые составляют это приложение. Он также включает другую информацию, которую мы обсудим позже. Это такие вещи как: Permissions (полномочия), которые необходимы, чтобы запустить это приложение, аппаратные функции, такие как камера, если она нужна для работы этого приложения, и самая ранняя версия платформы, на которой это приложение сможет работать. Давайте рассмотрим файл androidmanifest.xml для «MapLocation».

Файл androidmanifest.xml находится в корневом каталоге приложения. Один из элементов, который мы видим здесь, является элементом uses-SDK. Этот элемент включает атрибут minSdkVersion, который определяет минимальный уровень API для этого приложения. И в нашем случае этот уро-

вень равняется 13, что соответствуют Android 3.2.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="course.examples.MapLocation"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="13"
        android:targetSdkVersion="19" />

    <application
        android:allowBackup="false"
        android:icon="@drawable/ic_launcher"
        android:label="MapLocation" >
        <activity android:name="course.examples.MapLocation.MapLocation" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Этот элемент также включает другой атрибут – targetSdkVersion, который определяет последний уровень API, для которого было протестировано это приложение. В данном случае это уровень 19, что соответствует версии Android 4.4.2.

Следующий элемент манифеста – application, который содержит различные атрибуты: иконку для этого приложения, метку, которая показана в строке заголовка приложения, элемент Activity, который перечисляет в нашем случае всего одну Activity – MapLocation, которое включает это приложе-

ние.

И последним шагом мы устанавливаем приложение на устройство или эмулятор, чтобы протестировать и отладить его.

Класс Activity

Activity – основной класс для взаимодействия с пользователем. Activity разработаны, чтобы обеспечить визуальный интерфейс, через который пользователь может взаимодействовать с приложением. И условно Activity должны быть модулями приложения в том смысле, что каждая Activity, как отдельный модуль, должна поддерживать в фокусе единственную вещь, которую пользователь может сделать с вашим приложением, например просмотр электронного письма или показ экрана входа в систему.

Когда пользователь выполняет свою задачу, он перемещается по цепочке из Activity, и Android помогает пользователям в навигации несколькими способами, поддерживая концепцию приложения, обеспечивая временное хранение Activity в стеке (backstack), так что они должным образом могут быть приостановлены и затем возобновлены при возврате из стека.

Таким образом задача (Task) в Android – это ряд связанных Activity. Эти связанные Activity могут, но не должны быть частью одного приложения. Одна задача (Task) может охватить несколько приложений.

Большинство задач запускается с домашнего экрана. Таким образом, когда пользователь запускает приложение с домашнего экрана, обычно запускается новая задача (Task). И

когда пользователь нажимает кнопку «Домой», чтобы возвратиться назад к домашнему экрану, текущая задача, по крайней мере временно, приостанавливается.

Стек задачи (Task backstack) работает следующим образом. Когда Activity запущена, она заносится в верхнюю ячейку стека как часть текущей задачи. А когда эта Activity позже будет уничтожена, например, потому что пользователь нажал кнопку «Назад», или потому что Activity сама завершила свою работу программно, или даже потому что сам Android решил уничтожить эту Activity, чтобы освободить занимаемые ей ресурсы, тогда Activity удаляется из верхней ячейки стека задач.

Если же вместо уничтожения Activity будет запущена вторая Activity, то первая Activity продвинется вглубь стека, а ее место в верхней ячейке стека займет вторая Activity. Тогда после окончания работы второй Activity по кнопке «Назад» она будет уничтожена и удалена из стека, а в верхнюю ячейку стека и, соответственно на экран пользователя, вновь вернется первая Activity.

Activity имеют жизненный цикл. И что важно для вас как для разработчика, приложения не контролируют свой жизненный цикл. Некоторые изменения жизненного цикла зависят от выбора, который делает пользователь, например нажатие кнопки «Назад» или кнопки «Домой». Другие изменения жизненного цикла зависят от самого Android. Например, если ваше устройство испытывает нехватку памяти, Android может уничтожить activity, которые в настоящее время приостановлены.

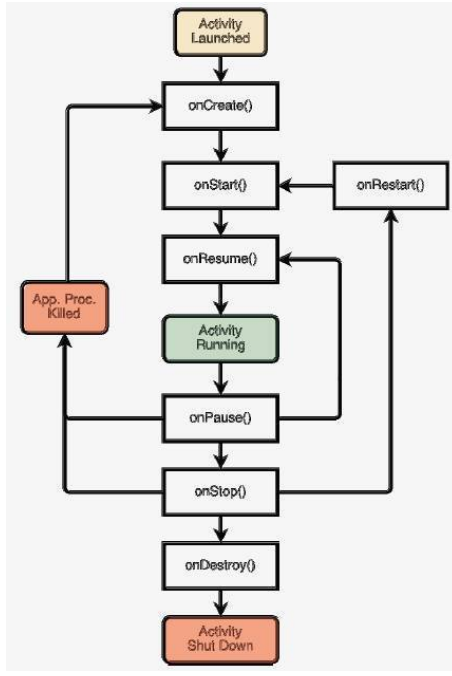
Как только Activity запущена, она может быть в состоянии resumed (возобновленном) или running (выполнения). И в то время как она находится в этом состоянии, Activity видима (на экране) и пользователь может взаимодействовать с ней.

Activity может также быть paused (приостановлена). Например, когда новая Activity начинает раскрываться перед

ней. В этой ситуации Activity может все еще быть частично видима, но пользователь не может взаимодействовать с ней, потому что он будет взаимодействовать с новой Activity, которая только что запущена (до версии 3.0 Android мог завершить Activity, как только они вошли в приостановленное состояние).

И, наконец, Activity может быть stopped (остановлена). И когда она остановлена, эта Activity больше не видима, и Android может ее уничтожить. Уничтоженная Activity может быть воссоздана позже, если пользователь перемещается к ней по задаче.

Ваши Activity будут часто вести себя по-разному во время различных частей их жизненного цикла. Например, если Activity показывает анимацию, но в это время раскрывается частично прозрачное Activity диалога перед ней, то вы захотите приостановить анимацию, пока пользователь отвечает на диалог, а затем перезапустить анимацию, как только Activity диалога закроется.



Чтобы поддерживать сценарии как этот, Android сообщает об изменениях жизненного цикла в Activity, вызывая предопределенные методы (call back) жизненного цикла. И вот некоторые из этих методов:

• onCreate, вызывается перед тем как Activity будет создано;

• onStart, вызывается перед тем как Activity станет видимым;

• onDestroy, вызывается перед тем как Activity будет уничтожена.

И, если вы хотите выполнить некоторые действия, когда ваша Activity изменяет состояние, то вы должны переопределить (override) эти методы в вашей Activity.

Давайте рассмотрим как эти различные методы взаимосвязаны друг с другом. Эта диаграмма изображает последовательность, в которой могут быть вызваны методы жизненного цикла Activity. И важно помнить, что приложения Android не работают полностью сами по себе. Вместо этого есть четко определенное направление взаимодействия между вашим приложением и Android. И вы должны понять правила этого взаимодействия, если хотите, чтобы ваши приложения функционировали должным образом.

Давайте представим простое приложение с одной Activity. Оно запускается, ожидает мгновение, и затем закрывается. В этом простом случае как только приложение будет запущено, Android вызовет (call back) метод onCreate.

Затем Android вызовет свой метод onStart, и затем onResume, после которого пользовательский интерфейс Activity появится на экране устройства, и пользователь сможет взаимодействовать с ним.

Приблизительно после одной минуты наша Activity начнет закрываться. И в этот момент Android вызовет метод onPause. Затем onStop. И, наконец, onDestroy. Теперь Activity абсолютно уничтожена.

Итак, как видите, все время жизни Activity заключено между onCreate при запуске и onDestroy в конце.

Когда эта простая Activity запускалась, сперва она не была видима на экране. В некоторый момент она стала видимой, позже она стала невидимой и затем была удалена с экрана.

```
@Override
protected void onStart() {
    super.onStart();
    Log.i(TAG, "The activity is visible and about to be started.");
}


@Override
protected void onRestart() {
    super.onRestart();
    Log.i(TAG, "The activity is visible and about to be restarted.");
}

@Override
protected void onResume() {
    super.onResume();
    Log.i(TAG, "The activity is and has focus (it is now \"resumed\")");
}
```

```
@Override
protected void onPause() {
    super.onPause();
    Log.i(TAG,
            "Another activity is taking focus (this activity is about to be \"paused\")");
}

@Override
protected void onStop() {
    super.onStop();
    Log.i(TAG, "The activity is no longer visible (it is now \"stopped\")");
}

@Override
protected void onDestroy() {
    super.onDestroy();
    Log.i(TAG, "The activity is about to be destroyed.");
}
```

Когда Activity становятся видимыми, Android иногда вызывает метод onStart, а иногда – onRestart. Перед тем, как Activity станут невидимыми, Android вызывает метод onStop. Таким образом, время жизни Activity в видимом состоянии длится между запуском вызовов onStart и onStop. И, наконец, в то время как Activity видима на экране, есть отрезки времени, когда пользователь может взаимодействовать с ней, и есть моменты когда не может. Например, это может произойти, когда устройство «засыпает». В этом случае пользователь не может взаимодействовать с Activity даже при том, что эта Activity все еще запущена.

Когда Activity оказывается готова к взаимодействию с пользователем, Android вызывает метод onResume. Когда Activity собирается прекратить взаимодействие с пользователем, Android вызывает метод onPause.

Таким образом время жизни Activity, когда она видима и

способна взаимодействовать с пользователем, длится между запуском вызовов onResume и onPause.

Теперь давайте более глубоко рассмотрим эти методы жизненного цикла. Первый метод – onCreate вызывается, когда Activity создается. Обычно onCreate используется, чтобы инициализировать Activity и выполнить следующие четыре функции:

• вызов super.onCreate, который сделает часть собственной инициализации Android;

• установка content view Activity, которая сообщит Android, каким должен быть пользовательский интерфейс Activity;

• получение и сохранение всех необходимых ссылок и соответствий между различными элементами (view) лейаута пользовательского интерфейса и кодом Activity;

• конфигурирование элементов пользовательского интерфейса и view.

Давайте рассмотрим метод onCreate приложения «MapLocation». @Override – переопределяет activity.onCreate. В методе onCreate вы видите четыре функции, о которых мы говорили.

```
@Override
protected void onCreate(Bundle savedInstanceState) {

    // Required call through to Activity.onCreate()
    // Restore any saved instance state
    super.onCreate(savedInstanceState);

    // Set content view
    setContentView(R.layout.main);

    // Initialize UI elements
    final EditText addrText = (EditText) findViewById(R.id.location);
    final Button button = (Button) findViewById(R.id.mapButton);
```

Во-первых, есть вызов super.onCreate. Затем есть вызов setContentView с ID файла лейаута R.layout.main в качестве параметра. После этого Activity сохраняет ссылки на view редактирования текста и на кнопку, которые появляются в пользовательском интерфейсе. И, наконец, onCreate присоединяет «слушателя» кнопки button.setOnClickListener. Android вызывает этот код «слушателя» каждый раз, когда пользователь нажимает на кнопку.

```
// Initialize UI elements
final EditText addrText = (EditText) findViewById(R.id.location);
final Button button = (Button) findViewById(R.id.mapButton);

// Link UI elements to actions in code
button.setOnClickListener(new OnClickListener() {

    // Called when user clicks the Show Map button
    public void onClick(View v) {
        try {

            // Process text for network transmission
            String address = addrText.getText().toString();
            address = address.replace(' ', '+');

            // Create Intent object for starting Google Maps application
            Intent geoIntent = new Intent(
                    android.content.Intent.ACTION_VIEW, Uri
                            .parse("geo:0,0?q=" + address));

            // Use the Intent to start Google Maps application using Activity.startActivity()
            startActivity(geoIntent);

        } catch (Exception e) {
            // Log any error messages to LogCat using Log.e()
            Log.e(TAG, e.toString());
        }
    }
});
```

onRestart – следующий метод жизненного цикла. Он вызывается, если Activity, которая была остановлена, будет запущена снова. И вы можете использовать onRestart, чтобы выполнить некоторые действия, которые необходимы, перед тем, как Activity будет запущена снова.

onStart вызывается перед тем, как Activity станет видимой. Некоторые типичные применения для этого метода – это запрос обновлений датчика положения или загрузка или сброс персистентного состояния приложения.

onResume вызывается перед тем, как Activity начнёт взаимодействовать с пользователем, и уже видима. Действия, которые вы совершаете в этом методе, должны быть запущены

на переднем плане приложения. Например, стартовые анимации или воспроизведение фоновой звуковой дорожки.

onPause вызывается пред тем, как Activity потеряет фокус. В этом методе вы можете завершить действия, совершающиеся на переднем плане, например, уничтожить анимации. Кроме этого вы должны сохранить любые изменения, которые совершил пользователь только что.

onStop вызывается, когда Activity больше не видима пользователю. В этой точке предполагается, что Activity может быть опять запущена позже. Так, некоторые типичные действия, которые надо сделать здесь – это кэширование состояния Activity, которое вы захотите восстановить, когда Activity позже перезапустится по вызову onStart. И помните: onStop не всегда вызывается, когда Activity завершается. Например, его нельзя вызвать, если Android уничтожает процесс приложения из-за нехватки памяти. Поэтому вместо сохранения персистентных данных в этом методе, лучше сделайте это в onPause.

onDestroy вызывается перед тем, как Activity будет уничтожена. Некоторые типичные действия, которые надо сделать в этом методе – высвободить ресурсы, занимаемых этой Activity. Например, закрыть частные потоки, которые были запущены этой Activity. И снова помните, что onDestroy не будет вызван, например, если Android уничтожит приложение из-за нехватки памяти на устройстве.

Теперь, когда мы увидели жизненный цикл Activity в

действии, давайте рассмотрим как одна Activity может программно запускать другую Activity. Чтобы запустить Activity программно, вы сначала создаете объект Intent (намерение), который определяет Activity, которую вы хотите запустить. И затем вы передаете этот созданный Intent тем методам, которые запускают Activity – startActivity или startActivityForResult. startActivity запустит желаемую Activity, скрывая текущую Activity с переднего плана. startActivityForResult, тоже запустит желаемую Activity, но сделает это с ожиданием того, что запущенная Activity вернет результат для вызывающей Activity.

Давайте рассмотрим как приложение «Map Location» запускает Activity «Google Maps». Каждый раз, когда пользователь кликает кнопкой, вызывается метод onClick. В строковую переменную address помещается географический адрес, считанный из текстового поля addrText. Далее создается Intent geoIntent, содержащий в себе заданный географический адрес, который затем и передается методу startActivity для запуска приложения «Карты» с географической картой, центрированной на указанном адресе.

Но вместо того, чтобы запросить у пользователя ввод адреса через клавиатуру, мы можем позволить ему выбрать контакт из приложения «Контакты» и использовать адресные данные этого контакта как географический адрес, который открыть на карте. Для этого рассмотрим приложение «Map location from contacts».

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

    final Button button = (Button) findViewById(R.id.mapButton);
    button.setOnClickListener(new Button.OnClickListener() {

        // Called when user clicks the Show Map button
        @Override
        public void onClick(View v) {
            try {
                // Create Intent object for picking data from Contacts database
                Intent intent = new Intent(Intent.ACTION_PICK,
                        CONTACTS_CONTENT_URI);

                // Use intent to start Contacts application
                // Variable PICK_CONTACT_REQUEST identifies this operation
                startActivityForResult(intent, PICK_CONTACT_REQUEST);

            } catch (Exception e) {
                // Log any error messages to LogCat using Log.e()
                Log.e(TAG, e.toString());
            }
        }
    });
}
```

Отличие от «Map Location» находится в коде OnClickListener, в частности, этот код создает несколько иной Intent, а затем передает его в качестве параметра методу StartActivityForResult. Вторым параметром является requestCode (код запроса), который пригодится позже, когда результат вернется обратно в запускающую Activity.

Как только новая Activity запущена, она ответственна за результат, который должен быть возвращен назад вызывающей Activity. Это делается вызовом Activity.SetResult, передвая в нем код результата (resultCode). Эти коды ре-

зультата включают некоторые встроенные коды, такие как Result_Canceled (отмененный результат), который указывает, что пользователь принял решение не завершать действие, а отменил его, например, нажав кнопку «Назад», и Result_OK, указывающий, что Activity фактически завершена штатно. Разработчики могут также добавить пользовательские коды результата после встроенного кода результата (Result_First_User).

Перед тем, как Activity завершится, она должна выдать свой результат, вызвав SetResult, и этот результат будет в конечном счете передан назад в вызвавшую Activity и использован в методе onActivityResult.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {

    // Ensure that this call is the result of a successful PICK_CONTACT_REQUEST request
    if (resultCode == Activity.RESULT_OK
            && requestCode == PICK_CONTACT_REQUEST) {
```

Как вы видите, его параметры включают начальный код запроса (requestCode), который был передан в StartActivityForResult, код результата (resultCode), используемый запущенной Activity, когда она вызвала SetResult, и Intent, если он был передан в SetResult. onActivityResult обычно начинается с проверки кода результата и кода запроса, чтобы определить, что сделать с полученным результатом.

Следующим шагом необходимо получить данные контакта и проанализировать их, чтобы извлечь почтовый адрес контакта. Как только почтовый адрес был извлечен, «Map location from contacts» обрабатывает данные так, чтобы они были в формате, который ожидают карты Google (строковая переменная formattedAddress), помещает эти данные в Intent и затем вызывает startActivity, передавая ей этот Intent.

```
// Create Intent object for starting Google Maps application
Intent geoIntent = new Intent(
        android.content.Intent.ACTION_VIEW,
        Uri.parse("geo:0,0?q=" + formattedAddress));

// Use the Intent to start Google Maps application using Activity.startActivity()
startActivity(geoIntent);
```

Далее рассмотрим обработку изменения конфигурации. Конфигурация устройства относится к характеристикам устройства и приложения, связанным с ресурсами приложения, такими как: используемые языки, размер экрана, доступность клавиатуры и ориентация экрана устройства. Эти характеристики могут измениться во время выполнения приложения и, когда это происходит, Android уничтожает текущую Activity и затем перезапускает ее с ресурсами, соответствующими измененной конфигурации.

Предположим, например, что изменяется ориентация устройства. Если вы поворачиваете устройство из портретного режима в альбомный и назад в портретный, это заставит систему уничтожить текущую Activity и перезапустить

ее дважды. Чтобы оптимизировать скорость изменения конфигурации, Android позволяет вам сделать две вещи. Первое, когда происходит изменение конфигурации, вы можете создать и сохранить произвольный объект Java, тем самым сохранив в кэше важную информацию о состоянии. И второе, вы можете вручную обработать изменение конфигурации, избежав завершения и перезапуска Activity в целом.

Итак, чтобы ускорить реконфигурирование, вы можете сохранить данные в объекте Java. И один способ сделать это – переопределить метод onRetainNonConfigurationInstance и вызвать его между onStop и onDestroy. Объект, который вы создаете и возвращаете в onRetainNonConfigurationInstance, может быть получен путём вызова экземпляра getLastNonConfigurationInstance, когда Activity воссоздана. Это обычно делается во время вызова onCreate, когда вы воссоздаете Activity.

И вторая вещь, которую вы можете сделать, чтобы ускорить реконфигурирование, это избежать уничтожения и перезапуска Activity для определенных изменений конфигурации. Чтобы сделать это, вы объявляете в файле androidmanifest.xml те определенные изменения, которые ваша Activity обработает. Например, этот отрывок xml-файла указывает, что MyActivity вручную обработает изменения в ориентации устройства, размере экрана и доступности клавиатуры.

```
<activity android:name=".MyActivity“
  android:configChanges=
      "orientation|screensize|keyboardHidden”...>
```

Если вы обрабатываете изменения конфигурации вручную, тогда, как только произойдут изменения конфигурации во время работы приложения, ваша Activity примет вызов onConfigurationChanged, и в качестве параметра этот метод получит объект конфигурации, который детализирует новую конфигурацию. Затем ваш код сможет считать этот объект и внести любые изменения, если необходимо.

Класс Intent

Класс Intent (намерение) – структура данных, которая служит двум целям. Первая – это определить операцию, которую вы хотите выполнить, и вторая – это представить событие, произошедшее в системе, о котором вы хотите уведомить другие компоненты. Мы поговорим о второй из них позже, когда будем рассматривать Broadcast receivers.

Вы можете представить себе Intent как предоставление своего рода языка для определения операций, которые вы хотите выполнить (ваше намерение). В сущности Intent даёт вам простой способ сказать такие вещи как: «я хочу выбрать контакт», «я хочу сделать фотографию», «я хочу набрать телефонный номер» или «я хочу вывести на экран карту». Intent обычно создаётся одной Activity, которой необходимо, чтобы некоторая операция была выполнена, и затем Android использует этот Intent чтобы запустить другую Activity, которая может выполнить желаемую работу.

Теперь о видах информации, которая может быть определена в Intent. Например: action (действие), data (данные), category (категория) и прочие. Action – строковая последовательность, которая представляет или называет операцию, которая будет выполнена. Например, action_dial означает, что я хочу набрать телефонный номер. Action_edit – я хочу вывести на экран некоторые данные для пользователя, чтобы от-

редактировать. Action_sync – синхронизировать некоторые данные на устройстве с данными на сервере. Action_main – запустить Activity как начальную Activity приложения.

Вы можете установить Action несколькими способами. Например, вы можете передать Action-строку в качестве параметра конструктору Intent. Как альтернатива, вы можете создать пустой Intent и затем вызвать метод setAction в нем, передав Action-строку в качестве параметра.

```
Intent newInt = new
            Intent(Intent.ACTION_DIAL);
OR
Intent newInt = new Intent();
newInt.setAction(Intent.ACTION_DIAL);
```

У Intent также есть поле данных (data), которое представляет данные, связанные с намерением. Эти данные отформатированы как Универсальный идентификатор ресурса или URI. Один пример данных Intent – geo URI, который означает данные географической карты.

```
Uri.parse("geo:0,0?
        q=1600+Pennsylvania+
        Ave+Washington+DC")
```

Другой пример – URI, указывающий телефонный номер, который вы хотите набрать. Обратите внимание на то, что последовательности, которые представляют базовые данные, сначала передаются методу uri.parse, который получает эти последовательности и затем возвращает объект URI.

```
Intent newInt = new Intent (
                Intent.ACTION_DIAL,
                Uri.parse("tel:+15555555555"));
Or
Intent newInt =
                new Intent(Intent.ACTION_DIAL);
newInt.setData(
                Uri.parse("tel:+15555555555"));
```

Вы можете установить данные Intent различными способами. Например, вы можете передать его непосредственно конструктору Intent, или вы можете создать пустой Intent и затем использовать метод setData, чтобы установить данные для этого Intent.

Категория (category) представляет дополнительную информацию о видах компонентов, которые может или должен обработать этот Intent. Некоторые примеры: category_browsable означает, что Activity может быть вызвана браузером через ссылку URI. Другой пример – category_launcher означает, что целевая Activity может быть начальной Activity задачи (task) и описана в средстве запуска

приложения верхнего уровня.

У Intent также есть поле Type (тип), которое определяет MIME-тип данных. Некоторые примеры MIME-типов включают различные форматы изображений, такие как png или jpg. Есть также различные текстовые форматы, такие как HTML или txt. Если вы не определите MIME-тип, Android попытается указать его для вас. Вы можете установить MIME-тип при помощи метода setType, передав его в строковой последовательности, которая представляет желаемый MIME-тип. Вы можете также установить и данные, и тип вместе, вызвав метод setDataAndType.

```
Intent.setType(String type)

Or

Intent.setDataAndType(Uri data,
                      String type)
```

Поле Component идентифицирует целевую Activity. Вы можете установить это поле, если знаете точно ту Activity, которая должна всегда получать этот Intent. Вы можете установить Component, используя один из конструкторов Intent, передав его в объекте контекста и объекте класса, который представляет Activity, которая должна выполнить желаемую операцию.

```
Intent newInt = Intent(
        Context packageContext, Class<?> cls);
```

Вы можете также создать пустой Intent и затем использовать один из методов setComponent(), setClass() или setClassName(), чтобы установить целевую Activity.

```
Intent newInt = new Intent ();

and one of:

setComponent(), setClass(), or setClassName()
```

Поле Extras. Extras хранят дополнительную информацию, связанную с Intent. Extras – это набор пар ключ-значение. Таким образом, целевая Activity должна знать и имя, и тип любых дополнительных данных (extras), которые она намеревается использовать. Например, класс Intent определяет дополнительные данные (extras) EXTRA_EMAIL, которые используются, чтобы передать список получателей при отправке электронного письма.

```
Intent newInt = new Intent(Intent.ACTION_SEND);
newInt.putExtra(android.content.Intent.EXTRA_EMAIL,
 new String[]{
```

```
            "potus@whitehouse.gov","mozart@musician.org"
        }
);
```

Как здесь видно, этот код создает Intent с действием (action) ACTION_SEND. Таким образом, мы хотим послать некоторое электронное письмо. А также добавляет Extras к этому Intent (EXTRA_EMAIL).

Есть несколько различных методов для установки Extras, и форма этих методов будет зависеть от типа данных, которые вы хотите там хранить. Так, например, есть один метод для хранения строковой последовательности, ещё один метод – для хранения массива чисел, и т. д.

```
putExtra(String name, String value);
putExtra(String name, float[] value);
```

Поле флагов (flags). Флаги представляют информацию о том, как Intent должен быть обработан. Некоторые встроенные примеры включают flag_activity_no_history. Что означает, что, когда Activity запущено на основе этого Intent, оно не должно быть помещено в стек.

```
Intent newInt =
            new Intent(Intent.ACTION_SEND);
newInt.setFlags(
        Intent.FLAG_ACTIVITY_NO_HISTORY);
```

Другой флаг – FLAG_DEBUG_LOG_RESOLUTION, который говорит Android распечатывать дополнительную информацию о журналировании (log), когда этот Intent обрабатывается, и это – важный инструмент, чтобы его использовать, если ваш Intent не запускает ту Activity, которую вы хотите.

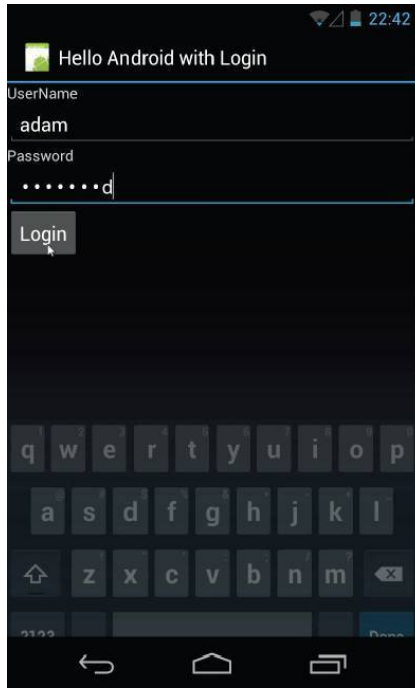
Давайте поговорим о том, как использовать Intent, чтобы запустить Activity. Когда вы используете один из методов startActivity или startActivityForResult, Android должен выяснить, какую конкретно Activity он собирается запустить. И есть два пути, которыми Android это делает.

Первый путь состоит в том, что, если вы явно указали целевую Activity, когда создали Intent (explicit intent), то Android может просто найти и запустить эту определенную Activity.

Второй путь используется, только тогда, когда вы явно не указали целевую Activity, и состоит в том, что Android может неявно определить ее на основе Intent (implicit intent), который использовался, и на основе свойств Activity (или приложений), которые установлены на устройстве.

Давайте рассмотрим приложение, которое явно запуска-

ет другую Activity. Это приложение называется «Hello World With Login». Оно состоит из двух Activity. Одна Activity называется LoginActivity, а другая – HelloAndroid. Когда это приложение запущено, первым запускается LoginActivity, оно предоставляет текстовое поле для ввода имени пользователя и поле для ввода пароля. Есть также кнопка Login, которую пользователь нажимает после того как введет имя пользователя и пароль. И затем LoginActivity проверяет введенные данные – пароль и имя пользователя, и если примет их, запустит Activity HelloAndroid, которая выведет на экран слова “HelloAndroid”.



Давайте рассмотрим код, чтобы увидеть как это реализовано. Это класс LoginActivity. Здесь реализован onClickListener («слушатель» кнопки), он присоединен к кнопке Login. Код захватывает имя пользователя и пароль, которое пользователь только что ввел, и затем передает их методу checkPassword. Если checkPassword возвратит true, то код будет выполняться дальше.

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

    final EditText uname = (EditText) findViewById(R.id.username_edittext);
    final EditText passwd = (EditText) findViewById(R.id.password_edittext);

    final Button loginButton = (Button) findViewById(R.id.login_button);
    loginButton.setOnClickListener(new OnClickListener() {

        public void onClick(View v) {

            if (checkPassword(uname.getText(), passwd.getText())) {

                // Create an explicit Intent for starting the HelloAndroid Activity
                Intent helloAndroidIntent = new Intent(LoginScreen.this,
                        HelloAndroid.class);

                // Use the Intent to start the HelloAndroid Activity
                startActivity(helloAndroidIntent);

            } else {
                uname.setText("");
                passwd.setText("");
            }
        }
    });
}
```

Сначала создается Intent (helloAndroidIntent). Вызов конструктора получает два параметра – контекст и объект класса Activity, которая будет запущена. В этом случае LoginScreen.this используется в качестве контекста.

Контекст (Context) – это интерфейс, который предоставляет доступ к базовым функциям приложения. Так как Activity – это подкласс контекста, то можно использовать. this для этого параметра. Второй параметр – объект класса, в этом случае – HelloAndroid.class. Этот параметр указывает Activity, которая будет запущена вызовом startActivity.

Второй способ запустить Activity – неявная активация. В этой ситуации вы не сообщаете Android, какую Activity запустить, система должна решить это самостоятельно. Android пытается найти соответствие между Intent, который был передан в startActivity или startActivityForResult с возможностями других Activity (или приложений), которые находятся на устройстве. И этот процесс называют intent resolution.

Intent resolution полагается на два вида информации. Во-первых, Activity создает Intent, описывающий действие, которое должно быть выполнено. Во-вторых, у некоторых Activity в Android определены Intent-фильтры (IntentFilters), определяющие виды операций, которые они могут обработать. Эта информация обычно помещается в файл androidmanifest.xml или в приложения, которым эти Activity принадлежат.

Во время выполнения, когда вызван метод startActivity, Android попытается найти соответствие между Intent и известными Intent-фильтрами. Но он будет использовать только часть информации, содержащейся в Intent. В частности он будет смотреть на поле Action (действие), на Data (данные), включая URI и mime-тип и на Категорию (Category).

Рассмотрим то, как приложения определяют Intent-фильтры в своих файлах androidmanifest.xml. Вот отрывок xml, показывающий тег Activity, этот тег содержит в себе тег Intent-фильтра с соответствующим действием, которое он поддерживает. Например, если Activity может набирать но-

мер телефона, то она должна содержать Intent-фильтр со стандартным intent.action.dial.

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

Если ваша Activity обрабатывает определенный тип данных, она может добавить эту информацию к ее Intent-фильтру, например, Activity может определить mime-тип данных, которые она обрабатывает. Вы можете также определить схему, хост или номер порта данных UI, которые она обрабатывает. В дальнейшем можно будет ограничить формат данных URI, определив путь, шаблон или префикс пути.

Если Activity хочет объявить, что она может, например, показывать карты, тогда она могла бы содержать Intent-фильтр, который говорит Android, что она может обработать данные URI со схемой geo.

```
<intent-filter ...>
  ...
  <data android:scheme=”geo" />
  ...
</intent-filter>
```

Точно так же Activity могут определить категории для Intent, которые они обрабатывают.

Теперь мы можем предположить или представить на основе тех приложений «Map location», которые мы видели прежде, что «Google Maps» может обработать Intent, которые содержат action_view и которые содержат поле данных со схемой geo. Файл androidmanifest.xml для приложения карт Google выглядит примерно следующим образом. Имеется Activity с Intent-фильтром. И этот фильтр перечислен в Action intent.action.view. И у него есть данные со схемой geo. Фактически это то, что приложение «Map location» передали в стартовую Activity Google-карт.

```
<intent-filter ...>
  <action android:name =
            "android.intent.action.VIEW" />
  <category android:name =
      "android.intent.category.DEFAULT" />
  <category android:name=
      "android.intent.category.BROWSABLE”/>
  <data android:scheme = "geo”/>
</intent-filter>
```

Activity также перечисляет две категории. Одна – category.browsable означает, что Activity может реагировать на ссылки браузера. Другая – cateogry.default. Оказывается, что в большинстве случаев Activity, которые хотят принимать implicit intents, должны включать category.defualt в свои Intent-фильтры.

Наконец, когда больше чем одна Activity может принять определенный Intent, Android должен будет приостановить исполнение для принятия решения. Один вариант – спросить пользователя, какую Activity (или приложение) он хочет использовать. И второй вариант – это использовать возможность того, что Activity могут определить приоритет, который Android примет во внимание при выборе между двумя различными Activity, которые могут обработать данный Intent. Значения приоритета должны быть между минус 1000 и плюс 1000, более высокие значения имеют более высокий приоритет.

Permissions

Android использует Permissions (полномочия, разрешения), чтобы защитить ресурсы, данные и операции. Приложения могут определять и устанавливать свои собственные полномочия, чтобы ограничить доступ к их ресурсам и информации о пользователе. Например, чтобы ограничить, кто может получить доступ к базе данных приложения, платным услугам, ограничить какие компоненты могут вызвать дорогостоящие операции, такие как исходящие сообщения SMS или MMS, ограничить доступ к системным ресурсам, например, чтобы определить какие компоненты могут использовать камеру устройства для съемки фото или записи видео.

В Android полномочия (Permissions) представлены как строковые последовательности. Приложения включают эти строки в файл androidmanifest.xml. Они могут сделать это, чтобы объявить полномочия, которые они используют сами, и полномочия, которые они требуют от других компонентов, которые хотят использовать их.

Сначала, если приложение должно использовать разрешение (permission), оно объявляет это разрешение тегом uses-permission в его файле androidmanifest.xml. И когда это приложение будет устанавливаться на устройство, пользователь должен будет согласиться с этим разрешением, иначе произойдут ошибки или отказы доступа.

Рассмотрим кусок файла androidmanifest.xml для гипотетического приложения.

```
<manifest ... >
...
<uses-permission  android:name=
                      "android.permission.CAMERA"/>
<uses-permission  android:name=
                    "android.permission.INTERNET"/>
<uses-permission  android:name=
  "android.permission.ACCESS_FINE_LOCATION"/>
...
</manifest >
```

Как видите, для этого приложения нужны полномочия получить доступ к камере устройства, доступ к Интернету и доступ к информации о точном местоположении с приемника GPS.

Вспомним приложение «Map location from contacts». Это приложение позволяет пользователю выбирать контакт из телефонной книги (приложение Contacts) и затем отображать карту, центрируемую на почтовом адресе этого контакта. Если запустить это приложение, я увижу кнопку, которая позволит мне выбрать контакт. Когда я щелкаю по этой кнопке, приложение контактов показывает мне мои контакты и позволяет мне выбрать один из них. Как только я сделаю это, почтовый адрес и другая информация о контакте передается в «Google Maps», которая выводит на экран карту, центрируемую на почтовом адресе выбранного контакта.

Но мой список контактов – частная информация. Android не позволяет любому приложению на моем устройстве просто иметь доступ к списку контактов. Так как «Map location from contacts» получает доступ к нему? Приложение, должно быть, объявило, что использует надлежащее разрешение (permission). И мы можем увидеть подтверждение этому в файле androidmanifest.xml.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="course.examples.MapLocationFromContacts"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-permission android:name="android.permission.READ_CONTACTS" >
    </uses-permission>
```

В дополнение к объявленным разрешениям, приложения Android могут также определить и установить свои собственные полномочия (permissions).

Предположим, вы написали приложение, которое выполняет некоторую привилегированную или опасную операцию, и вы бы не хотели, чтобы любое приложение имело возможность выполнить эти операции, просто вызвав ваше приложение. Поэтому Android позволяет нам определить и объявить свое собственное специализированное разрешение (permission). И тогда другие приложения должны будут получить ваше разрешение, иначе Android не позволит им использовать ваше приложение.

Рассмотрим это более подробно с простым примером.

Представим простое приложение под названием «Бум!» Притворимся, что оно выполняет некоторое опасное действие, которое в реальной жизни могло бы быть, например, операцией форматирования карты памяти. Мы не хотим, чтобы любой был в состоянии просто запустить это приложение. Итак, мы собираемся определить и осуществить наше собственное специализированное разрешение (permission).

Давайте откроем файл androidmanifest.xml, чтобы видеть, как это реализуется. Вы видите тег permission, который определяет новое разрешение – *course.examples.permissionexample.boom_perm*.

```
<!-- Defines a custom permission -->
<permission
   android:name="course.examples.permissionexample.BOOM_PERM"
   android:description="@string/boom_perm_string"
   android:label="@string/boom_permission_label_string" >
</permission>

<!-- Enforces the BOOM_PERM permission on users of this application -->
<application
   android:allowBackup="true"
   android:icon="@drawable/ic_launcher"
   android:label="@string/app_name"
   android:permission="course.examples.permissionexample.BOOM_PERM" >
   <activity
      android:name=".BoomActivity"
      android:label="@string/app_name" >
      <intent-filter>
         <action android:name="android.intent.action.MAIN" />
```

В этом примере тег permission включает два других атрибута: label (метка) и description (описание), которое может быть показано пользователю, когда он устанавливает прило-

жение. Значения для этих строковых последовательностей находятся в файле strings.xml. Далее немного ниже вы видите, что приложение также включает это разрешение как атрибут тега application. Благодаря этому Android удостоверится, что компонентам, которые пытаются запустить это приложение, уже предоставлено разрешение *boom_perm*. Так как приложение «Бум!» теперь требует наличие разрешения *boom_perm*, то любое другое приложение должно будет сначала получить это разрешение, если захочет использовать его.

Android также позволяет вам устанавливать полномочия для отдельных компонентов (component permissions). Эти параметры будут иметь приоритет над любыми уровнями разрешений приложения. Давайте рассмотрим некоторые из этих полномочий компонентов.

Activity permissions (полномочия Activity) ограничивают, какие компоненты могут запустить связанную Activity. Эти полномочия проверяются в процессе выполнения методов, таких как startActivity и startActivityForResult, которые вызываются, когда одна Activity пытается запустить другую. И, если требуемое разрешение будет отсутствовать, Android выдаст ошибку безопасности (security exception).

Service permissions (полномочия сервисов или служб) ограничивают, какие компоненты могут запустить связанный Service. Эти полномочия проверяются, когда выполнены запросы на запуск, остановку или подключение к

службам, используя такие методы как Context.startService(), Context.stopService() или Context.bindService(). Так же, как и в случае с Activity, Android выдаст security exception (ошибку безопасности), если необходимое полномочие отсутствует.

Broadcast receiver permission ограничивает, какие компоненты могут отправлять и получать широковещательные сообщения. Эти полномочия проверяются различными путями в различное время. Эту тему мы обсудим более подробно позже.

Content provider permissions ограничивает, какие компоненты могут читать и записывать данные, содержавшиеся в Content provider. И это тоже подробнее будет обсуждаться позже.

Класс Fragment

Фрагменты (Fragments) были добавлены в Android версии 3.0, чтобы лучше поддерживать пользовательские интерфейсы для устройств с большими экранами, таких как планшеты. За последние несколько лет популярность планшетов выросла невероятно. На большом дисплее планшета мы можем сделать намного больше, чем мы могли сделать на маленьком телефонном дисплее.

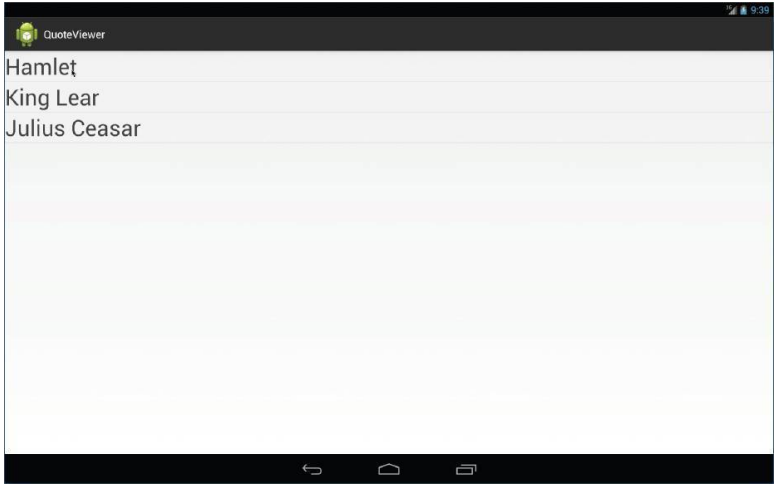
Чтобы быть конкретнее, давайте рассмотрим простое приложение под названием «QuoteViewer». Это приложение состоит из двух Activity. Первая показывает заголовки нескольких пьес Шекспира и позволяет пользователю выбрать один из заголовков. И, когда пользователь выбирает заголовок, запускается вторая Activity, которая выводит на экран одну цитату из выбранной пьесы.

Если я запущу приложение «QuoteViewer», первая Activity покажет нам заголовки трех пьес Шекспира: «Гамлет», «Король Лир» и «Юлий Цезарь». Далее я выберу Hamlet. Запустится вторая Activity, которая покажет мне цитату Горацио: «Now cracks a noble heart. Good night sweet prince, and flights of angels sing thee to thy rest».

Теперь я нажму кнопку «Назад», чтобы возвратиться к заголовкам. То же самое произойдет, если выбрать два других

заголовка – будет запущено вторая Activity с цитатой из соответствующей пьесы.

Это прекрасный интерфейс для телефона. Фактически было бы трудно сделать больше и все еще иметь читаемый и простой в использовании интерфейс. На планшете, однако, этот же лейаут (layout) будет выглядеть довольно непрезентабельно. Здесь я покажу эмулированный планшет.

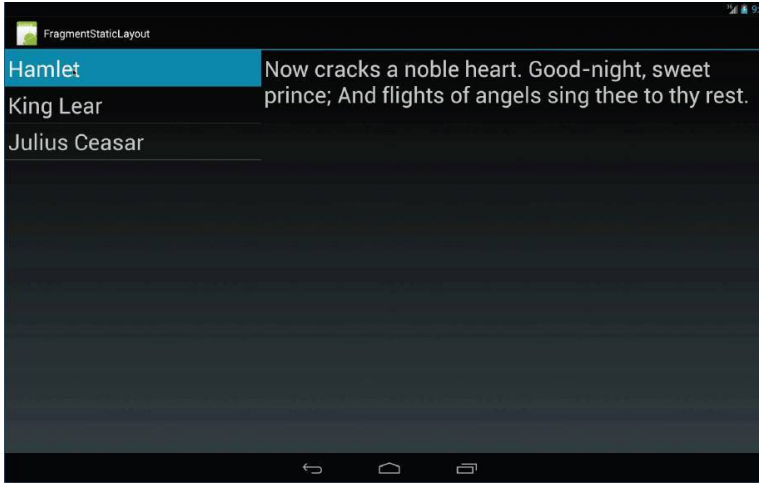


Первое, что вы замечаете, – это неиспользованное пустое белое поле. Заголовки занимают просто немного места на левой стороне дисплея, а остальное пространство в значительной степени потрачено впустую.

Теперь, если кликнуть по любому заголовку, вы увидите,

что цитата простирается полностью через весь десятидюймовый экран слева направо, но использует ничтожно мало вертикального пространства.

Давайте рассмотрим более удачный лейаут (layout), который показывает и заголовки, и цитаты одновременно. Когда я кликаю по заголовку, вы видите, что пользовательский интерфейс по существу разделен на две части. Заголовки остаются там, где они были слева, но цитата теперь показана одновременно на правой стороне экрана. Теперь, если я захочу увидеть другую цитату, я просто выбираю новый заголовок пьесы слева. И тогда новая цитата раскрывается справа взамен предыдущей.



Этот лейаут исключает потраченное впустую пространство, и это делает навигацию немного проще, потому что я не должен перемещать палец вниз к нижней части дисплея, чтобы нажать кнопку «Назад», и затем обратно до заголовков, и т. д.

Приложение «QuoteViewer» было составлено из двух Activity. Приложение, которое мы сейчас увидели, составлено из единственной Activity, но с применением двух фрагментов. Фрагменты представляют часть пользовательского интерфейса Activity. У этого приложения есть две таких интерфейсных части. Одна – для заголовков – расположена слева, и одна – для цитат, которая расположена справа. И каждая из этих частей пользовательского интерфейса реализована как фрагмент.

Фрагменты размещены в Activity. Одна Activity может содержать любое количество фрагментов. И один фрагмент может быть использован в любом количестве Activity. Таким образом, фрагменты должны быть загружены в Activity, затем выведены на экран, удалены, и т. д., поскольку Activity изменяет свое состояние. И в результате жизненный цикл фрагмента связан и синхронизирован с жизненным циклом Activity, которая его содержит.

Вместе с тем у фрагментов есть некоторые их собственные call back-методы жизненного цикла. На данный момент будем говорить о жизненном цикле фрагмента, предполагая, что фрагмент статически связан с Activity. О динамическом

добавлении и удалении фрагментов в Activity мы будем говорить немного позже.

Фрагмент может быть в состоянии:

• running (выполнение) или resumed (возобновленный) – в этих состояниях фрагмент видим (is visible) в работающей Activity;

• paused (приостановлен) – фрагменты будут приостановлены, когда их Activity будет видима, но какая-либо другая Activity (или другой фрагмент) находится на переднем плане и имеет фокус;

• stopped (остановлен) – в этом состоянии фрагмент невидим.

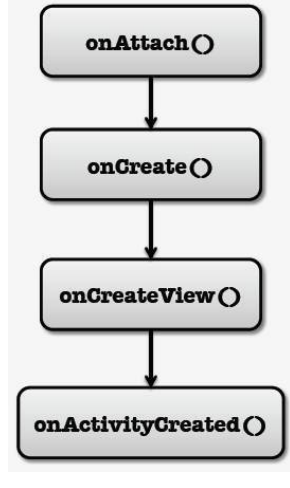
Теперь поговорим о методах обратного вызова (call back) жизненного цикла, которые может получить фрагмент. Мы будем говорить о том, когда эти методы могут быть вызваны относительно состояний жизненного цикла Activity, которая размещает (содержит) фрагмент.

Когда содержащая фрагмент Activity создается, фрагмент может получить несколько вызовов метода жизненного цикла. Во-первых, когда фрагмент присоединяется к своей Activity, он получает вызов onAttach. Затем фрагмент принимает вызов своего метода onCreate. И отметьте здесь для себя, что речь идет о методе fragment.onCreate, а не activity.onCreate, который вызывается при создании Activity. Вы инициализируете фрагмент в методе фрагмента onCreate, различие между двумя методами в том, что,

в отличие от activity.onCreate, пользовательский интерфейс в fragment.onCreate вы не устанавливаете. Это фактически происходит в следующем методе.

Теперь, после onCreate, Android вызывает метод фрагмента onCreateView. Этот метод устанавливает и возвращает лейаут, содержащий пользовательский интерфейс фрагмента. И этот лейаут передается в Activity, чтобы установить его в layout-иерархии Activity.

И, наконец, после того, как Activity была создана, и пользовательский интерфейс фрагмента был установлен, фрагмент принимает вызов метода onActivityCreated.



Теперь, пока фрагмент присоединен к Activity, его жиз-

ненный цикл зависит от жизненного цикла Activity. Когда лейаут пользовательского интерфейса фрагмента, который ранее создавался при вызове onCreateView, отсоединен от Activity, фрагмент принимает вызов onDestroyView. Здесь необходимо очистить ресурсы, связанные с лейаутом интерфейса. Затем, когда фрагмент больше не используется, он принимает вызов метода onDestroy. Здесь необходимо высвободить ресурсы фрагмента. Затем, когда фрагмент больше не присоединен к своей Activity, он примет вызов onDetach. Здесь необходимо обнулить ссылки на Activity фрагмента.

Есть два основных способа, которыми фрагменты добавляются к Activity.

Во-первых, они могут быть статически добавлены к Activity. Например, помещая фрагмент в файл лейаута Activity, который затем используется при вызове setContentView. Во-вторых, можно добавить фрагменты динамически, используя FragmentManager.

В любом случае, как только они будут добавлены, Android вызовет onCreateView. В onCreateView фрагмент может использовать свой собственный лейаут xml, подобно тому, как делают Activity, когда они вызывают setContentView, или фрагменты могут программно создать свои пользовательские интерфейсы.

Как только представление пользовательского интерфейса создано, onCreateView должен возвратить лейаут, располо-

женный в корне его пользовательского интерфейса. И этот лейаут будет в конечном счете передан в Activity и добавлен к ее пользовательскому интерфейсу.

Давайте рассмотрим приложение «FragmentStaticLayout» и посмотрим, как оно определяет свой лейаут. На левой панели показаны заголовки пьес Шекспира, а на правой панели показаны цитаты из этих пьес. И каждая из этих панелей реализована отдельным фрагментом, один называется TitleFragment, а другой QuoteFragment. И основная Activity этого приложения называется QuoteViewerActivity.

Давайте откроем файл QuoteViewerActivity и посмотрим, как он обрабатывает и создает свой лейаут.

```
public class QuoteViewerActivity extends Activity implements
        ListSelectionListener {

    public static String[] mTitleArray;
    public static String[] mQuoteArray;
    private QuotesFragment mDetailsFragment;

    private static final String TAG = "QuoteViewerActivity";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Get the string arrays with the titles and qutoes
        mTitleArray = getResources().getStringArray(R.array.Titles);
        mQuoteArray = getResources().getStringArray(R.array.Quotes);

        setContentView(R.layout.main);

        // Get a reference to the QuotesFragment
        mDetailsFragment = (QuotesFragment) getFragmentManager()
                .findFragmentById(R.id.details);
    }
```

Сначала вы видите, что в методе onCreate есть вы-

зов setContentView со значением параметра R.layout.main. Поэтому давайте посмотрим в каталоге res/layout файл main.xml.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:baselineAligned="false"
    android:orientation="horizontal" >

    <fragment
        android:id="@+id/titles"
        android:layout_width="0px"
        android:layout_height="match_parent"
        android:layout_weight="1"
        class="course.examples.Fragments.StaticLayout.TitlesFragment" />

    <fragment
        android:id="@+id/details"
        android:layout_width="0px"
        android:layout_height="match_parent"
        android:layout_weight="2"
        class="course.examples.Fragments.StaticLayout.QuotesFragment" />

</LinearLayout>
```

Как видите, весь лейаут состоит из LinearLayout, и что LinearLayout содержит два фрагмента. И в тегах <fragment…/>, есть атрибут, названный class. Значение этого атрибута – это имя класса, который реализует этот фрагмент. В этом случае один фрагмент реализован классом TitleFragment, а другой – классом QuotesFragment.

Когда этот xml-файл считан, Android поймет, что надо создать эти два фрагмента и поместить их в QuoteViewerActivity. Это запустит цепочку вызовов жизненного цикла, о которых мы говорили ранее. Один из этих вы-

зовов будет вызовом onCreateView, в котором фрагмент ответственен за создание его лейаута пользовательского интерфейса.

Давайте рассмотрим один из лейаутов в QuoteFragment, то, как он создает свой пользовательский интерфейс. Имеется класс QuoteFragment и его метод onCreateView.

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

Этот метод вызывает метод inflate класса LayoutInflater, передавая файл лейаута (quote_fragment.xml) в качестве параметра. Это работает подобно тому, что происходит в setContentView.

Вы можете также добавить фрагменты в Activity без хардкодинга фрагментов в файле лейаута Activity. Для этого, в то время как Activity работает, надо сделать четыре вещи. Первое – получить ссылку на FragmentManager. Второе – начать FragmentTransaction. Третье – добавить фрагмент в Activity. И четвертое – зафиксировать (commit) FragmentTransaction.

Давайте откроем QuoteViewerActivity и посмотрим, как она обрабатывает свой лейаут (layout).

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Get the string arrays with the titles and qutoes
    mTitleArray = getResources().getStringArray(R.array.Titles);
    mQuoteArray = getResources().getStringArray(R.array.Quotes);

    setContentView(R.layout.main);

    // Get a reference to the FragmentManager
    FragmentManager fragmentManager = getFragmentManager();

    // Begin a new FragmentTransaction
    FragmentTransaction fragmentTransaction = fragmentManager
            .beginTransaction();

    // Add the TitleFragment
    fragmentTransaction.add(R.id.title_frame, new TitlesFragment());

    // Add the QuoteFragment
    fragmentTransaction.add(R.id.quote_frame, mQuoteFragment);

    // Commit the FragmentTransaction
    fragmentTransaction.commit();
}
```

В onCreate есть вызов setContentView со значением параметра R.layout.main. Поэтому давайте посмотрим в каталоге res/layout файл main.xml.

Этот лейаут состоит из LinearLayout с двумя подразделами. Но вместо фрагментов на сей раз подразделы – это FrameLayout. FrameLayout должен зарезервировать некоторое пространство в пользовательском интерфейсе, и мы заполним это пространство позже, когда будем добавлять фрагменты в эту Activity.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activityFrame"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:baselineAligned="false">

    <FrameLayout
        android:id="@+id/title_frame"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" >
    </FrameLayout>

    <FrameLayout
        android:id="@+id/quote_frame"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="2" >
    </FrameLayout>

</LinearLayout>
```

Теперь давайте вернемся к методу onCreate в QuoteViewerActivity и пройдем по тем четырем шагам, добавляя фрагменты в Activity. Во-первых, получаем ссылку на FragmentManager. Затем вызываем метод beginTransaction. В результате получим FragmentTransaction. Затем вызываем метод add, передавая ID FrameLayout во фрагмент, который будет содержать этот FrameLayout. Это надо сделать и для TitleFragment, и для QuoteFragment. И, наконец, вызовем метод commit, чтобы зафиксировать все изменения.

```
@Override
protected void onCreate(Bundle savedInstanceState) {

    Log.i(TAG, getClass().getSimpleName() + ":entered onCreate()");

    super.onCreate(savedInstanceState);

    // Get the string arrays with the titles and qutoes
    mTitleArray = getResources().getStringArray(R.array.Titles);
    mQuoteArray = getResources().getStringArray(R.array.Quotes);

    setContentView(R.layout.main);

    // Get references to the TitleFragment and to the QuotesFragment
    mTitleFrameLayout = (FrameLayout) findViewById(R.id.title_fragment_container);
    mQuotesFrameLayout = (FrameLayout) findViewById(R.id.quote_fragment_container);


    // Get a reference to the FragmentManager
    mFragmentManager = getFragmentManager();
```

```
    // Start a new FragmentTransaction
    FragmentTransaction fragmentTransaction = mFragmentManager
            .beginTransaction();

    // Add the TitleFragment to the layout
    fragmentTransaction.add(R.id.title_fragment_container,
            new TitlesFragment());

    // Commit the FragmentTransaction
    fragmentTransaction.commit();

    // Add a OnBackStackChangedListener to reset the layout when the back stack changes
    mFragmentManager
            .addOnBackStackChangedListener(new FragmentManager.OnBackStackChangedListener() {
                public void onBackStackChanged() {
                    setLayout();
                }
            });
}
```

Теперь мы знаем как добавить фрагменты в Activity программно. Но в нашем примере это не имело такого большого значения в том смысле, что даже при том, что мы добавили фрагменты программно, их расположение (лейаут) было статично. Всегда есть только две панели, и это никогда не изменяется.

И один из плюсов способности добавлять фрагменты на лету – то, что она позволяет динамически изменять пользовательский интерфейс во время работы программы. И если делать это правильно, это позволит рационально использовать драгоценное экранное пространство.

Рассмотрим приложение «FragmentProgrammaticLayout», которое демонстрирует некоторые разновидности динамических пользовательских интерфейсов. Приложение иногда будет показывать единственный фрагмент, а иногда – несколько фрагментов.

Сразу после запуска мы видим только TitleFragment с заголовками пьес. Если кликнуть по заголовку, вы увидите справа уже знакомую цитату из Гамлета. В этой точке приложение выводит на экран два фрагмента. Если теперь нажать кнопку «Назад», вы увидите, что мы вернулись к отображению единственного фрагмента.

Теперь рассмотрим код. На сей раз в начале мы добавим только TitleFragment, а фрагмент с цитатами будем добавлять только, если пользователь кликнет по заголовку. Предположим, пользователь действительно кликает по заголовку. Тогда будет вызван метод onListSelection. Рассмотрим этот метод.

```
// Called when the user selects an item in the TitlesFragment
@Override
public void onListSelection(int index) {

    // If the QuoteFragment has not been added, add it now
    if (!mQuoteFragment.isAdded()) {

        // Start a new FragmentTransaction
        FragmentTransaction fragmentTransaction = mFragmentManager
                .beginTransaction();

        // Add the QuoteFragment to the layout
        fragmentTransaction.add(R.id.quote_fragment_container,
                mQuoteFragment);

        // Add this FragmentTransaction to the backstack
        fragmentTransaction.addToBackStack(null);

        // Commit the FragmentTransaction
        fragmentTransaction.commit();

        // Force Android to execute the committed FragmentTransaction
        mFragmentManager.executePendingTransactions();
    }

    if (mQuoteFragment.getShownIndex() != index) {

        // Tell the QuoteFragment to show the quote string at position index
        mQuoteFragment.showQuoteAtIndex(index);

    }
}
```

Во-первых, мы проверяем, был ли фрагмент с цитатами уже добавлен в layout. И, если нет, то мы добавим его, запуская другой FragmentTransaction.

Далее мы собираемся добавить эту транзакцию в стек задачи (Task backstack) для того, чтобы пользователь вернулся к тому состоянию экрана, который был прежде, до добавления нового фрагмента, когда он нажмет кнопку «Назад». И это надо сделать, потому что по умолчанию в стеке задач изменения фрагментов не отслеживаются.

В конце добавляем вызов FragmentManager.executePendingTransactions. Это вынужда-

ет транзакцию выполниться сразу, а не тогда, когда система посчитает удобным это сделать, и нам пришлось бы ждать, пока обновится экран.

В главе об Activity мы говорили о том, как Activity могут обрабатывать изменения конфигурации вручную используя методы, такие как onRetainNonConfigurationInstance и getLastNonConfigurationInstance.

При изменении конфигурации устройства Android по умолчанию уничтожит Activity и воссоздаст ее. Однако, если вызвать метод setRetainInstance во фрагменте, передав в качестве параметра true, то когда произойдут изменения конфигурации, Android уничтожит Activity, но не уничтожит фрагменты, которые она содержит. Android сохранит состояние фрагмента, отсоединив фрагмент от Activity. И, кроме того, Android не уничтожит этот фрагмент и не воссоздаст его позже, когда Activity будет воссоздана. В результате фрагмент не получит вызова своих методов onDestroy и onCreate.

Давайте рассмотрим другой пример приложения, который демонстрирует это поведение. Это приложение называется «FragmentStaticConfigLayout», и его функциональность совпадает с предыдущими примерами. Различие здесь в том, что добавлен некоторый код, обрабатывающий изменения конфигурации. Когда устройство находится в альбомном режиме, лейаут выглядит так же, как и в предыдущем примере. Оба фрагмента используют большой шрифт или шрифт,

независимый от масштаба. TitleFragment занимает приблизительно одну треть горизонтального пространства, в то время как QuoteFragment берет остающиеся две трети пространства. И, если заголовок будет слишком длинным, чтобы поместиться на одной строке в TitleFragment, то текст просто будет перенесен на вторую строку.

Теперь давайте повернем устройство. Когда устройство окажется в портретном режиме, лейаут немного изменится. Оба фрагмента используют меньший шрифт, TitleFragment занимает только одну четверть горизонтального пространства, в то время как QuoteFragment занимает остающиеся три четверти рабочего пространства. И, если заголовок будет слишком длинным, чтобы поместиться на одной строке в TitleFragment, то он будет все же ограничен одной строкой, и мы просто заменим часть текста замещающими знаками.

Заметьте, что даже при том, что Android уничтожил и перезапустил QuoteViewerActivity, заголовок, который был выбран во фрагменте слева, все еще остаётся выбранным, потому что был вызван setRetainInstance с параметром true в обоих фрагментах. В итоге информация о том, что некоторый элемент был выбран, сохранилась.

Давайте рассмотрим, как это работает. Код файла TitleFragment.java приложения «FragmentStaticConfigLayout» главным образом такой же, как тот, что мы видели в других примерах приложений, но есть по крайней мере два различия. Первое различие нахо-

дится в методе onCreate – добавлен вызов setRetainInstance, true. Еще раз – это означает, что, когда происходят изменения конфигурации, Android не уничтожит этот фрагмент. Второе различие находится в методе onActivityCreated. В конце добавлен некоторый код, который проверяет индекс, означающий, что пользователь ранее выбрал заголовок, и что этот вызов onActivityCreated, вероятно, происходит из-за изменения конфигурации. В этом случае мы хотим удостовериться, что тот заголовок остается выбранным. Такие же изменения внесены и в класс QuoteFragment.

Классы пользовательского интерфейса

В этой главе будет рассказано о классах, с помощью которых Android позволяет создавать пользовательский интерфейс приложений:

• класс View (вью, на жаргоне – вьюшка) – основа для всего, что видно на экране устройства, а также различные события, которые вьюшки могут принимать;

• высокоуровневые составные представления View Groups (группы вью), которые комбинируют несколько вьюшек, чтобы создать определенный вид или поведение;

• некоторые определенные группы вью, в частности Adapter view (адаптер) и лейауты (layout);

• Menu (меню) и Action Bar (панель действий), которые предоставляют пользователю легкий доступ к часто используемым функциям;

• Dialogs (диалоговые окна), которые раскрываются на переднем плане приложения, чтобы информировать пользователя или получить от пользователя информацию.

В первую очередь пользовательский интерфейс (UI) – это место и средства, с помощью которых пользователь и приложение обмениваются информацией. Это визуальные элементы, которые пользователи видят и которых касаются на

экране.

Android обеспечивает богатый набор классов, из которых разработчики могут создавать пользовательские интерфейсы. Начнем с классов View. Класс view – ключевой стандартный блок для компонентов UI. View занимают прямоугольное место на экране.

В Andriod имеется много предопределенных или предустановленных вьюшек. Давайте поговорим о некоторых из них. В частности о кнопках (button), переключателях (toggle button), чекбоксах (checkbox) и других.

Вы видели уже много различных кнопок. Кнопка – это просто вьюшка, по которой пользователь может кликнуть, чтобы запустить или выполнить некоторое действие.

Давайте рассмотрим пример кнопки в приложении «UIButton». У него есть единственная кнопка внизу экрана, и эта кнопка подписана «Press me». И, если поддаться искушению и нажать на кнопку, то надпись на ней изменится. И теперь она гласит, что была нажата и сколько раз. И к настоящему времени вы, вероятно, уже можете предположить то, на что похож код для этого действия.

```
// Get a reference to the Press Me Button
final Button button = (Button) findViewById(R.id.button);

// Set an OnClickListener on this Button
// Called each time the user clicks the Button
button.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {

        // Maintain a count of user presses
        // Display count as text on the Button
        button.setText("Got Pressed:" + ++count);

    }
});
```

Код этой Activity создал объект-кнопку (button) и присоединил (setOnClickListener) «слушателя» этой кнопки. И каждый раз при нажатии на эту кнопку Android вызывает метод onClick «слушателя» кнопки (OnClickListener).

Следующая вьюшка называется ToggleButton (переключатель). Переключатель – это другой вид кнопки, однако имеет дополнительное свойство – когда вы нажимаете его, он остается нажатым, пока вы не нажмете его снова. Таким образом, переключатель всегда находится в одном из двух состояний – включено или выключено. Переключатели также обычно выводят на экран некоторый индикатор, чтобы сообщить пользователю, в каком положении находится кнопка в настоящее время.

```
// Get a reference to the ToggleButton
final ToggleButton button = (ToggleButton) findViewById(R.id.togglebutton);

// Set an OnClickListener on the ToggleButton
button.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {

        // Toggle the Background color between a light and dark color
        if (button.isChecked()) {
            bg.setBackgroundColor(0xFFF3F3F3);
        } else {
            bg.setBackgroundColor(0xFF000000);
        }
    }
});
```

Следующей view, о которой я буду говорить, является Checkbox (чекбокс, флажок, галочка). Используется, например, в анкете, где вы можете отметить и выбрать нужный пункт. Флажок – фактически просто другой вид кнопки с двумя состояниями, как переключатель. Основное различие в том, как он выглядит для пользователя. Флажки обычно изображаются в виде пустого квадратика, когда флажок не отмечен. И показывают галочку или символ «x», когда флажок находится в отмеченном состоянии. Код приложения для использования чекбокса аналогичен двум предыдущим.

```
// Get a reference to the CheckBox
final CheckBox checkbox = (CheckBox) findViewById(R.id.checkbox);

// Set an OnClickListener on the CheckBox
checkbox.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {

        // Check whether CheckBox is currently checked
        // Set CheckBox text accordingly
        if (checkbox.isChecked()) {
            checkbox.setText("I'm checked");
        } else {
            checkbox.setText("I'm not checked");
        }
    }
});
```

И последняя view, о которой мы поговорим – AutoCompleteTextView. Это поле для ввода и редактирования текста с функцией автозавершения является немного более усовершенствованной разновидностью TextView (надпись или нередактируемый текст) и EditText (поле для ввода и редактирования текста). Автозавершение показывает пользователю список вариантов для ввода текста. И мы будем фильтровать этот список в зависимости от того, что вы вводите. И как только вы сузите список, вы сможете коснуться единственной записи, которая будет тогда помещена в текстовое поле.

```
// Get a reference to the AutoCompleteTextView
AutoCompleteTextView textView = (AutoCompleteTextView) findViewById(R.id.autocomplete_country);

// Create an ArrayAdapter containing country names
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
        R.layout.list_item, COUNTRIES);

// Set the adapter for the AutoCompleteTextView
textView.setAdapter(adapter);
```

Каждая вьюшка имеет множество свойств, которые разработчик может менять на свое усмотрение по своему вкусу. Например, вы можете установить непрозрачность или прозрачность, вы можете установить цвет фона, ориентацию на дисплее и многое другое.

Вьюшки обрабатывают события. И эти события могут происходить из различных источников, включая пользователя, когда он касается вьюшки или использует физические устройства ввода данных, такие как физическая клавиатура или геймпад. Android также может быть источником событий. Например, вьюшки получают вызовы различных методов, когда Android должен изменить местоположение или перерисовать вьюшку.

Самый распространенный способ обработать событие – это присоединить «слушателя» (listener) к вьюшке. Android определяет много различных видов интерфейсов «слушателя». И методы, определенные этими интерфейсами, будут вызваны каждый раз, когда произойдут определенные события с вьюшкой.

Например, класс View определяет интерфейс «слушателя» onClickListener, который имеет метод onClick. Этот метод вызывается каждый раз, когда по вьюшке кликнули.

Класс View также определяет «слушателя» длинного клика – onLongClickListener. Он имеет метод onLongClick, и этот метод вызывается каждый раз, когда вьюшку нажимают

и удерживают нажатой в течение определенного промежутка времени.

Класс View также определяет «слушателя» изменения фокуса onFocusChangeListener. Он имеет метод onFocusChange, и этот метод вызывается, когда вьюшка получила или потеряла фокус. Также есть много других событий, которые вы также можете прослушивать.

До сих пор мы говорили главным образом об отдельных вью. Но в действительности, нам часто будут нужны составные вью, в которых соединены несколько отдельных вьюшек, чтобы обеспечить некоторую сложную функциональность. Простым примером является RadioGroup – по существу это ряд связанных флажков.

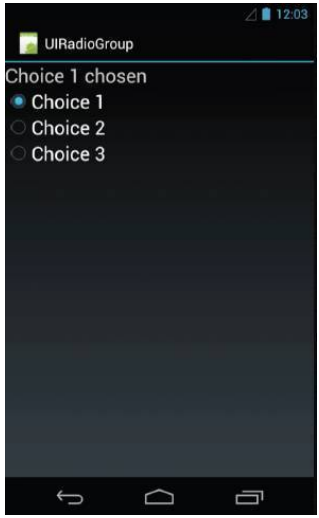
Например, у вас могло бы быть приложение, которое спрашивает пользователя, какого он возраста, и позволяет выбрать из ряда возрастных диапазонов: до 20, 20–34, 35–49, и более чем 50. Чтобы это реализовать, у вас был бы набор текстовых вью для всех различных возрастных диапазонов, и затем рядом с каждым текстовым вью вы поместили бы флажок. Но вы хотели бы удостовериться, что только один из флажков отмечается, потому что возрастные диапазоны, очевидно, являются взаимоисключающими.

Чтобы поддерживать такие сложные представления как это, у Android есть класс под названием ViewGroup – это невидимые группы, которые содержат другие вью. И таким образом, вы можете использовать их, чтобы сгруппировать

и организовать по нескольку вьюшек. ViewGroup – базовый класс для контейнеров вью и лейаутов. Точно так же, как с простыми вью, Android обеспечивает много предопределенных групп вью. Это такие, как: RadioGroup, TimePicker, DatePicker, WebView, MapView, Gallery и Spinner. Давайте рассмотрим каждый из них по отдельности.

Давайте начнем с RadioGroup. Это ViewGroup, содержащий ряд взаимоисключающих флажков или переключателей. Таким образом, в любой момент может быть установлен только один из переключателей. Давайте посмотрим на пример приложения.

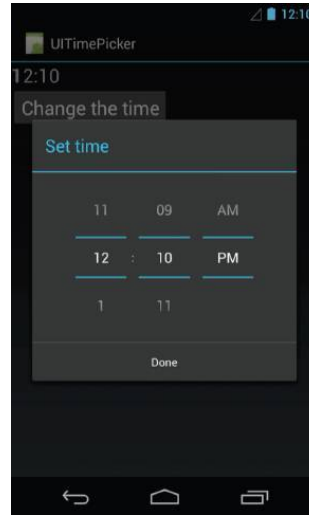
Приложение выводит на экран текстовое вью и RadioGroup. Текстовое вью выводит на экран текст «No Choice Made», поскольку прямо сейчас ни один из переключателей не установлен. Теперь выберем «Choice 1». Как видите, текст изменяется, чтобы отразить выбор, который был сделан. Теперь, если выбрать «Choice 2», «Choice 1» автоматически отменится, и будет выбран «Choice 2». И то же произойдет, если выбрать «Choice 3».




```
<RadioGroup
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    >

    <RadioButton
        android:id="@+id/choice1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/choice_1_string"
        android:textSize="24sp"/>

    <RadioButton
        android:id="@+id/choice2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/choice_2_string"
        android:textSize="24sp"/>

    <RadioButton
        android:id="@+id/choice3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/choice_3_string"
        android:textSize="24sp" />
</RadioGroup>
```

```java
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    final TextView tv = (TextView) findViewById(R.id.textView);

    // Define a generic listener for all three RadioButtons in the RadioGroup
    final OnClickListener radioListener = new OnClickListener() {
        @Override
        public void onClick(View v) {
            RadioButton rb = (RadioButton) v;
            tv.setText(rb.getText() + " chosen");
        }
    };

    final RadioButton choice1 = (RadioButton) findViewById(R.id.choice1);
    // Called when RadioButton choice1 is clicked
    choice1.setOnClickListener(radioListener);

    final RadioButton choice2 = (RadioButton) findViewById(R.id.choice2);
    // Called when RadioButton choice2 is clicked
    choice2.setOnClickListener(radioListener);

    final RadioButton choice3 = (RadioButton) findViewById(R.id.choice3);
    // Called when RadioButton choice3 is clicked
    choice3.setOnClickListener(radioListener);

}
```

Следующий ViewGroup – TimePicker – позволяет пользователю выбирать и устанавливать определенное значение времени. Рассмотрим приложение «UITimePicker», которое выводит на экран текстовую вью, показывающую текущее время и кнопку, подписанную «Change the time». Если кликнуть по кнопке, появится средство выбора времени – TimePicker.



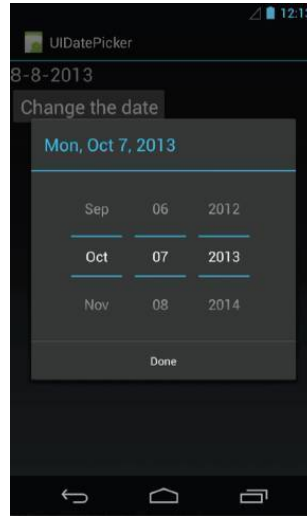
TimePicker составлен из многих различных вью, но вместе они позволяют пользователю независимо устанавливать час, минуты и a.m. или p.m. Есть также кнопка в нижней части, чтобы указать, что выбор сделан. Как только вы нажмете эту кнопку, текстовая вью изменится, чтобы показать время, которое только что выбрали.

```
<TextView
    android:id="@+id/timeDisplay"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text=""
    android:textSize="24sp" />

<Button
    android:id="@+id/pickTime"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/change_the_time_string"
    android:textSize="24sp" />
```

```
// The callback received when the user "sets" the time in the dialog
private TimePickerDialog.OnTimeSetListener mTimeSetListener = new TimePickerDialog.OnTimeSetListener() {
    public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
        mHour = hourOfDay;
        mMinute = minute;
        updateDisplay();
    }
};
```

```
@Override
protected Dialog onCreateDialog(int id) {
    switch (id) {
    case TIME_DIALOG_ID:
        return new TimePickerDialog(this, mTimeSetListener, mHour, mMinute,
                false);
    }
    return null;
}
```

Подобно TimePicker, есть также группа вью DatePicker. Эта ViewGroup позволяет пользователю выбирать определенную дату. Рассмотрим приложение «UIDatePicker». Приложение выводит на экран текстовую вью, показывающую текущую дату и кнопку, подписанную «Change the date». Таким образом, если кликнуть по кнопке, появится средство выбора даты DatePicker.



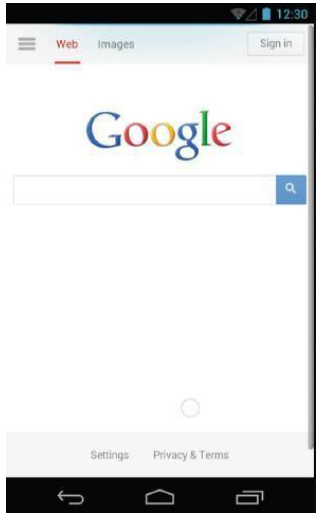
Средство выбора даты составлено из многих различных вью, которые вместе позволяют пользователю независимо устанавливать месяц, день и год. Также есть кнопка в нижней части, чтобы указать, что выбор сделан. Когда вы кликнете по этой кнопке, текстовая вью изменится, чтобы показать выбранную дату.

```
<TextView
    android:id="@+id/dateDisplay"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text=""
    android:textSize="24sp" />

<Button
    android:id="@+id/pickDate"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/change_the_date_string"
    android:textSize="24sp" />
```

```
// The callback received when the user "sets" the date in the Dialog
private DatePickerDialog.OnDateSetListener mDateSetListener = new DatePickerDialog.OnDateSetListener() {

    public void onDateSet(DatePicker view, int year, int monthOfYear,
            int dayOfMonth) {
        mYear = year;
        mMonth = monthOfYear;
        mDay = dayOfMonth;
        updateDisplay();
    }
};
```

```
    // Create and return DatePickerDialog
    @Override
    protected Dialog onCreateDialog(int id) {
        switch (id) {
        case DATE_DIALOG_ID:
            return new DatePickerDialog(this, mDateSetListener, mYear, mMonth,
                    mDay);
        }
        return null;
    }
```

Следующая ViewGroup – это веб-представление WebView, которое выводит на экран веб-страницы. Вот приложение, которое загрузит и выведет на экран знакомую веб-страницу www.google.com.



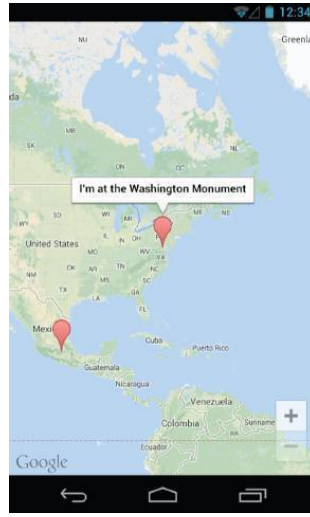

```
<WebView  xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/webview"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
/>
```

```
mWebView = (WebView) findViewById(R.id.webview);

// Set a kind of listener on the WebView so the WebView can intercept
// URL loading requests if it wants to

mWebView.setWebViewClient(new HelloWebViewClient());

mWebView.getSettings().setJavaScriptEnabled(true);
mWebView.loadUrl("http://www.google.com");
```

```
private class HelloWebViewClient extends WebViewClient {
    private static final String TAG = "HelloWebViewClient";;

    // Give application a chance to catch additional URL loading requests
    @Override
    public boolean shouldOverrideUrlLoading(WebView view, String url) {
        Log.i(TAG, "About to load:" + url);
        view.loadUrl(url);
        return true;
    }
}
```

Следующая ViewGroup – MapView. Как предполагает ее имя, MapView выводит на экран карту и позволяет пользователю взаимодействовать с ней. Давайте рассмотрим пример приложения, который фактически использует класс MapFragment, но использует его, чтобы вывести на экран базовую MapView.

Это приложение выводит на экран карту, центрируемую на некоторой части Америки. Карта также выводит на экран два красных маркера – один около Вашингтона в США и другой – в Мексике.



Если кликнуть по верхнему маркеру, появляется надпись, указывающая, что пользователь находится у памятника Вашингтону. А если кликнуть по другому маркеру, то появится другая надпись о том, что пользователь в Мексике.

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
          android:id="@+id/map"
          android:layout_width="match_parent"
          android:layout_height="match_parent"
          android:name="com.google.android.gms.maps.MapFragment"/>
```

```
// The GoogleMap instance underlying the GoogleMapFragment defined in main.xml
mMap = ((MapFragment) getFragmentManager().findFragmentById(R.id.map))
        .getMap();

if (mMap != null) {

    // Set the map position
    mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(new LatLng(29,
            -88), 0));

    // Add a marker on Washington, DC, USA
    mMap.addMarker(new MarkerOptions().position(
            new LatLng(38.8895, -77.0352)).title(
            getString(R.string.in_washington_string)));

    // Add a marker on Mexico City, Mexico
    mMap.addMarker(new MarkerOptions().position(
            new LatLng(19.13, -99.4)).title(
            getString(R.string.in_mexico_string)));
}
```

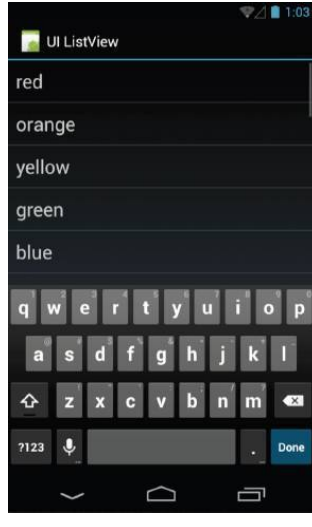
ViewGroups, о которых я только что говорил, каждая имеет довольно четкую цель и по большей части работает с фиксированным видом входных данных. Следующий набор групп вью, который я хочу обсудить, разработан для ситуаций, когда разработчики смогут вывести на экран различные виды данных.

Рассмотрим, например, List view (список), которая может использоваться, чтобы показать список телефонных номеров, список песен, изображений и т. д. Для таких групп вью как List view, чтобы работать со всеми этими различными типами данных, Android обеспечивает подкласс под названием AdapterView. AdapterView – это ViewGroup, дочерние вью которой и базовые данные управляются не самой ViewGroup, а другим классом, называемым адаптером (Adapter). Класс Adapter ответственен за управление данными и за создание

и обеспечение вьюшек данными по запросу AdapterView.

Давайте бросим более глубокий взгляд на класс ListView. ListView – это AdapterView, который выводит на экран прокручиваемый список выбираемых элементов. Этими элементами управляет адаптер, названный ListAdapter. ListView может также дополнительно фильтровать этот список элементов по введенному тексту точно так же, как AutocompleteTextView.

Рассмотрим приложение «UIListView». Вы видите, что в основе этого ListView лежат данные, которые являются длинным перечнем цветов. Красный, оранжевый, желтый, и так далее.



Этот ListView также вызывает клавиатуру в рабочее состояние. Мы будем использовать ее. Если ввести букву «O», ListView отфильтрует все цвета, которые не начинаются с буквы «O», останутся только оранжевый и оливковый. Если теперь ввести букву «l», то в списке останется только оливковый, поскольку только он начинается с введенных букв «Ol». И если кликнуть по оставшемуся слову «olive», то мы увидим, что «слушатель», который присоединён к ListView, выведет на экран наш выбор.

Давайте рассмотрим исходный код этого приложения. Откроем файл ListViewActivity. И давайте перейдем к методу onCreate. Здесь мы вызываем setListAdapter, чтобы установить ListAdapter для ListView. Фактически адаптер – ArrayAdapter, который реализует интерфейс адаптера. Конструктор для ArrayAdapter получает несколько параметров. Два, которые мы рассмотрим, являются идентификатором ресурса, который сообщает адаптеру как создать вью, содержащую каждую часть данных (R.layout.list_item). Второй параметр – сам массив данных (R.array.colors).

```
// Create a new Adapter containing a list of colors
// Set the adapter on this ListActivity's built-in ListView
setListAdapter(new ArrayAdapter<String>(this, R.layout.list_item,
        getResources().getStringArray(R.array.colors)));

ListView lv = getListView();

// Enable filtering when the user types in the virtual keyboard
lv.setTextFilterEnabled(true);

// Set an setOnItemClickListener on the ListView
lv.setOnItemClickListener(new OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View view,
            int position, long id) {

        // Display a Toast message indicting the selected item
        Toast.makeText(getApplicationContext(),
                ((TextView) view).getText(), Toast.LENGTH_SHORT).show();
    }
});
```

Давайте посмотрим на эти данные. Они определены как строковый массив в файле res/values/strings.xml. Он содержит набор цветов, точно такой же, как мы видели – красный, оранжевый, желтый, и т. д.

Теперь вернемся к Activity ListView и получим имя файла лейаута, который будет использоваться, чтобы создать вьюшку для каждого из этих цветов. Этот ресурс находится в res/layout/list_item.xml. Каждая часть данных вставлена в TextView с определенными отступами и размером шрифта. Далее код связывает ListView с ListActivity. И затем устанавливает setTextFilterEnabled (true). Это заставляет клавиатуру раскрываться, и включается фильтрация. И затем мы устанавливаем onItemClickListener, у которого есть метод

OnItemClick. Этот метод выведет на экран цвет, который выбирает пользователь, когда он кликает по нему в ListView.

Следующий AdapterView – класс Spinner. Это AdapterView, который обеспечивает прокручиваемый выпадающий список элементов. Пользователь может кликнуть по этой вьюшке, которая заставляет появляться выпадающий список, и затем позволяет пользователю выбрать элемент из этого выпадающего списка. Данными для spinner управляет SpinnerAdapter. Давайте рассмотрим этот класс на примере приложения «UISpinner».

Это приложение демонстрирует текстовую вью, которая говорит: «Pick a Color, Any Color» («Выберите цвет, любой цвет»). В настоящее время красный используется в качестве выбора по умолчанию. Теперь, предположим, я хочу выбрать другой цвет. Чтобы сделать это, я кликну spinner, который сейчас показывает красный. И тогда появится выпадающий список со списком цветов.

Теперь я выберу желтый. Выпадающий список исчезнет. Желтый цвет теперь появляется как выбранный цвет.

Далее откроем файл SpinnerActivity и перейдем к методу onCreate. Во-первых, здесь присутствует вызов setContentView, использующий main.xml в качестве файла лейаута. У пользовательского интерфейса есть элемент spinner.

```
<Spinner
    android:id="@+id/spinner"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="24sp" />
```

И, возвращаясь в SpinnerActivity, устанавливаем адаптер для spinner. Мы создаем этот адаптер, вызывая метод ArrayAdapter.createFromResource. Параметры этого метода включают список цветов и лейауты для каждой вью, которая появится в выпадающем списке.

```
// Get a reference to the Spinner
Spinner spinner = (Spinner) findViewById(R.id.spinner);

// Create an Adapter that holds a list of colors
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(
        this, R.array.colors, R.layout.dropdown_item);

// Set the Adapter for the spinner
spinner.setAdapter(adapter);

// Set an setOnItemSelectedListener on the spinner
spinner.setOnItemSelectedListener(new OnItemSelectedListener() {
    public void onItemSelected(AdapterView<?> parent, View view,
            int pos, long id) {
```

Файл strings.xml содержит массив цветов. Откроем файл drop-down_item.xml. В нем содержится лейаут для выпадающих вьюшек, так что каждый цвет появится как TextView с определенным отступом и размером шрифта.

Возвращаясь назад в SpinnerActivity, в следующих нескольких строках мы устанавливаем адаптер, затем устанавливаем OnItemSelectedListener, который вызывается, когда пользователь выбирает цвет.

Следующий ViewGroup – класс Gallery (Галерея), он выводит на экран ряд данных с горизонтально прокручивающимся списком. Как у spinner, данными для объектов Галереи управляет SpinnerAdapter.

Пример – приложение «UIGallery». Данные для этого приложения – ряд изображений, которые можно смахнуть в сторону по дисплею, чтобы прокрутить вперед и назад список изображений. Когда пользователь выбирает определенное изображение, вызывается «слушатель», чтобы вывести на экран индекс выбранного изображения.

```
<Gallery xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gallery"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
/>
```

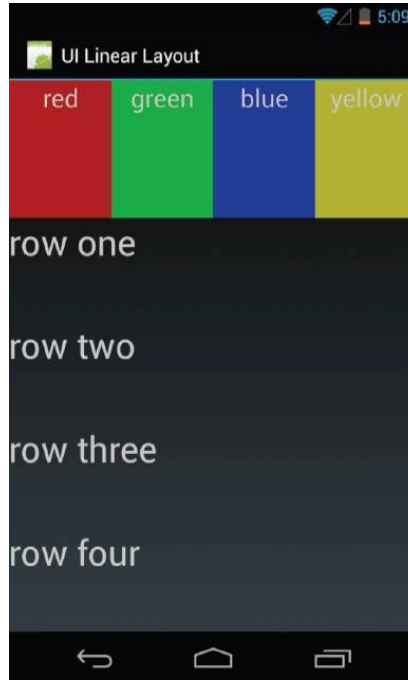
```
Gallery g = (Gallery) findViewById(R.id.gallery);

// Create a new ImageAdapter and set in as the Adapter for the Gallery
g.setAdapter(new ImageAdapter(this));

// Set an setOnItemClickListener on the Gallery
g.setOnItemClickListener(new OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View v,
            int position, long id) {
        // Display a Toast message indicate the selected item
        Toast.makeText(GalleryActivity.this, "" + position,
                Toast.LENGTH_SHORT).show();
    }
});
```

```
// The Adapter class used with the Gallery
public class ImageAdapter extends BaseAdapter {

    private static final int IMAGE_DIM = 800;

    private int mGalleryItemBackground;
    private Context mContext;

    // List of IDs corresponding to the images
    private Integer[] mImageIds = { R.drawable.sample_1,
            R.drawable.sample_2, R.drawable.sample_3, R.drawable.sample_4,
            R.drawable.sample_5, R.drawable.sample_6, R.drawable.sample_7 };

    public ImageAdapter(Context c) {
        mContext = c;
        TypedArray a = obtainStyledAttributes(R.styleable.GalleryActivity);
        mGalleryItemBackground = a.getResourceId(
                R.styleable.GalleryActivity_android_galleryItemBackground,
                0);
        a.recycle();
    }

    public int getCount() {
        return mImageIds.length;
    }

    public Object getItem(int position) {
        return mImageIds[position];
    }

    public long getItemId(int position) {
        return position;
    }
```

```
public View getView(int position, View convertView, ViewGroup parent) {

    ImageView imageView = (ImageView) convertView;

    // If convertView is not recycled set it up now
    if (null == imageView) {
        imageView = new ImageView(mContext);

        imageView.setLayoutParams(new Gallery.LayoutParams(IMAGE_DIM,
                IMAGE_DIM));
        imageView.setScaleType(ImageView.ScaleType.FIT_XY);
        imageView.setBackgroundResource(mGalleryItemBackground);

    }

    // Set the image for the imageView
    imageView.setImageResource(mImageIds[position]);

    return imageView;
}
```

Layout – своего рода разметка экрана или иначе – подобие группы вью, которая используется, чтобы организовать и структурировать другие вьюшки и группы вью. Например, LinearLayout содержит набор дочерних вью или групп вью, располагая дочерние элементы в один ряд горизонтально или вертикально.

Давайте рассмотрим пример – приложение под названием «UILinearLayout».



Вы видите ряд цветных полей, маркированных красным, зеленым, синим и желтым цветом, и все они размещены в горизонтальной строке. Под этим есть другой набор полей, размещенных вертикально, они маркированы: «строка один», «строка два», «строка три» и «строка четыре».

Рассмотрим исходный код, чтобы увидеть, как этот лейаут фактически создавался. В файле main.xml вся разметка – это LinearLayout, и у него есть два дочерних элемента,

каждый из которых является также LinearLayout. У внешнего LinearLayout есть параметры layout_width (ширина) и layout_height (высота), равные match_parent (совпадает с родителем). Это означает, что он занимает все пространство его родителя, в этом случае – все окно приложения. Ориентация – вертикальная, это означает, что дочерние элементы будут размещены один под другим.

Далее идет первый дочерний LinearLayout. Мы видим, что его layout_width установлен match_parent, таким образом его ширина будет такой же, как у родителя – внешнего LinearLayout.

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"
    android:orientation="horizontal" >
```

Его layout_height установлена в 0. И у него также есть layout_weight = 1, об этом чуть позже. И последнее, ориентация – горизонтальная. Таким образом, дочерние элементы этого LinearLayout будут размещены друг рядом с другом горизонтально.

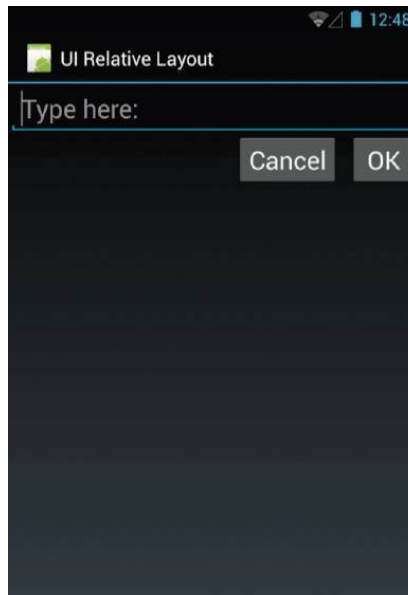
Теперь давайте перейдем ко второму дочернему элементу внешнего LinearLayout. Это тоже LinearLayout.

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="3"
    android:orientation="vertical" >
```

И у этого элемента layout_width установлен match_parent. И layout_height = 0. Но его layout_weight, однако, равен 3. Layout weight – это вес, означающий, какую часть пространства внутри родителя может занять этот элемент по отношению к своим соседям (сестрам). Принимая во внимание, что у первого дочернего элемента был вес 1, эти веса говорят, что первый дочерний LinearLayout должен получить одну четверть пространства, в то время как второй дочерний элемент получает оставшиеся 3 четверти. И у второго дочернего элемента ориентация вертикальная, а не горизонтальная.

Следующий лейаут – RelativeLayout. В RelativeLayout дочерние элементы позиционируются друг относительно друга и относительно их родителей, а не в фиксированном порядке, как в LinearLayout.

Рассмотрим приложение «UIRelativeLayout», которое содержит вью редактирования текста и две кнопки.



Здесь внешней группой вью является RelativeLayout. В нем размещены элементы, которые мы видели на экране – EditTextView и две кнопки, подписанные «OK» и «Cancel». Если посмотреть ближе, мы увидим, что кнопка «OK» должна быть выровнена по правому краю родителя (это – относительное расположение) и ниже поля редактирования текста, который определяется его ID. А кнопка «Cancel» должна быть выровнена слева от кнопки «OK», и ее верх должен

быть выровнен по верху кнопки «ОК».

Следующий лейаут – TableLayout. В TableLayout дочерние элементы расположены в строки и столбцы. Рассмотрим приложение «UITableLayout», в котором данные расположены в строках, а каждая строка содержит элементы, составляющие столбцы.

Если открыть файл лейаута, мы увидим TableLayout, у ко-

торого есть несколько строк таблицы (TableRow). В каждой строке таблицы есть несколько вью, которые, как предполагается, находятся в столбцах.

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:stretchColumns="1" >

    <!-- First row -->
    <TableRow>

        <!-- start in column 1 -->
        <TextView
            android:layout_column="1"
            android:padding="3dip"
            android:text="@string/open_string"
            android:textSize="24sp" />

        <TextView
            android:gravity="right"
            android:padding="3dip"
            android:text="@string/ctrl_o_string"
            android:textSize="24sp" />
    </TableRow>
```

Но, если необходимо, можно определить столбец layout_column. Например, в этом примере ничего нет в нулевом столбце. Таким образом, мы можем сказать Android, что в первой строке текстовая вью должна быть размещена в столбец 1, а не в нулевом столбце. Также вы видите, что эта текстовая вью определяет gravity (притягивать) вправо, что означает, что текст вьюшки должен быть сдвинут вправо.

Следующий лейаут – GridView, располагает свои дочерние элементы в двумерной сетке с возможностью прокрутки.

Рассмотрим приложение «UIGridView», которое считы-

вает набор изображений и затем автоматически расставляет их в прямоугольной сетке. Если кликнуть по любому из этих изображений, то запустится другая Activity, которая выводит на экран целиком это изображение.

Давайте рассмотрим исходный код. Откроем лейаут-файл main.xml, где вы видите элемент GridView. Здесь определены такие элементы, как ширина столбцов (columnWidth) и интервал (horizontalSpacing), чтобы разделить изображения. Также установлено, что GridView сам определяет число используемых столбцов (numColumns).

```
<GridView
    android:id="@+id/gridview"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnWidth="90dp"
    android:gravity="center"
    android:horizontalSpacing="10dp"
    android:numColumns="auto_fit"
    android:stretchMode="columnWidth"
    android:verticalSpacing="10dp" />
```

Теперь откроем файл GridLayoutActivity. Там вы видите, что определен список ресурсов изображения, которые должны быть выведены на экран GridView.

```
GridView gridview = (GridView) findViewById(R.id.gridview);

// Create a new ImageAdapter and set it as the Adapter for this GridView
gridview.setAdapter(new ImageAdapter(this, mThumbIdsFlowers));

// Set an setOnItemClickListener on the GridView
gridview.setOnItemClickListener(new OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View v,
            int position, long id) {
```

Ниже в onCreate установлен contentView и затем установлен адаптер, который является экземпляром класса адаптера изображения (ImageAdapter). Давайте рассмотрим класс ImageAdapter. Первый адаптер изображения – это подкласс основного адаптера, который реализует интерфейс адаптера. У этого класса есть несколько методов, которые используются, когда GridView запрашивает данные.

Пройдемся через несколько методов этого класса. Во-первых, есть метод GetCount. Этот метод должен вернуть число элементов данных, которыми управляет адаптер. Другой метод – GetItemID, который возвращает ID для элемента данных в указанной позиции. Он используется, например, когда пользователь кликает по изображению в GridView, чтобы указать, какое изображение развернуть во весь эран. Последним методом, о котором упомянем, является GetView. Этот метод вызывают, когда GridView запрашивает у адаптера вью, которая войдет в сетку. Один из параметров этого метода – ConvertView может быть равен «no». Если так, тогда необходимо создать новую вью и сконфигурировать ее

так, как вы хотите. Но по прошествии времени ConvertView станет неизвестен, фактически он сошлется на вью, которая была уже возвращена этим методом в прошлом.

Например, если у вас есть много вьюшек в сетке, то только несколько из них могут быть видимы одновременно. Тогда GridView запросит только те вьюшки, которые он собирается вывести на экран. Но, если пользователь позже прокрутит GridView, некоторые вьюшки, которые были видимы, уйдут из видимости. Тогда Android попытается повторно использовать ранее загруженные вьюшки и выдаст одну из них адаптеру, реализуя метод GetView.

Затем вы будете просто использовать эту вью и переустанавливать любые нужные вам поля для своего текущего элемента данных. И это хорошо, потому что экономит время, выделяемое на новые вью, и делает прокрутку более плавной.

Следующее, о чем мы поговорим, – меню и Action Bar (панель выбора действий). Activity могут поддерживать меню, которые могут быть представлены пользователю по-разному, но основная идея состоит в том, что меню дают пользователям быстрый способ получить доступ к важным функциям. Так, Activity могут добавить элементы к меню, и они могут реагировать, когда пользователь выбирает пункт меню, нажав на него. Вызов меню изменялся в Android в течение долгого времени. Сперва поговорим об основных видах меню, а затем о более новом классе – Action Bar.

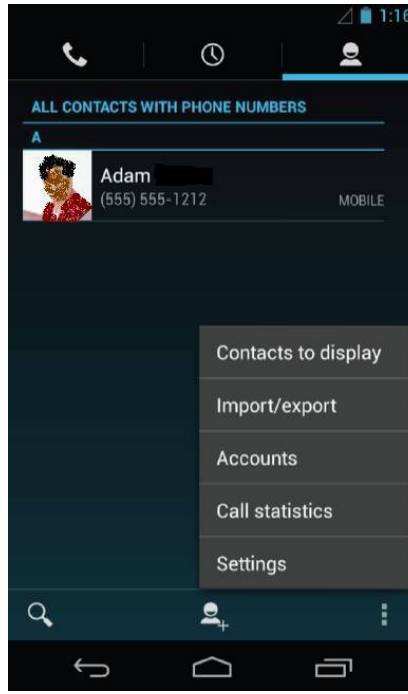
Есть три вида меню Android:

Меню опций (options menu), которое пользователь получает, когда нажимает кнопку меню. У более старых устройств на базе Android обычно была специальная физическая кнопка меню, которую уже не делают на более новых телефонах.

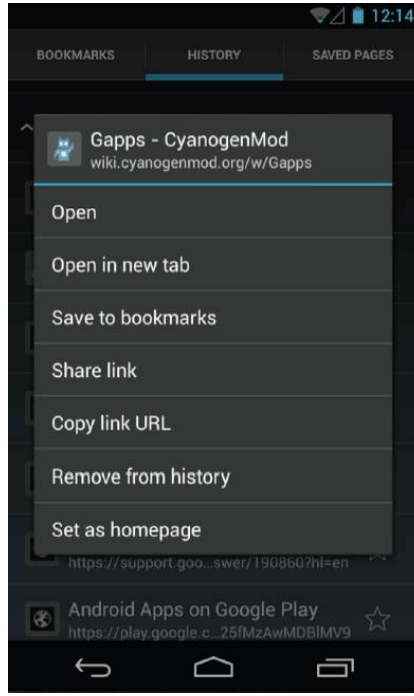
Контекстные меню (context menu). Эти меню присоединены к определенным вью, и появляются только тогда, когда пользователь длительно нажимает эту вьюшку. Контекстные меню обычно используются для работы с определенными данными, связанными с вьюшками, в то время как меню опций используются для глобальных операций и влияют на целое приложение.

Подменю. Это вторичные меню, которые активируются только тогда, когда пользователь выбирает пункт уже видимого меню.

Давайте рассмотрим некоторые примеры. Пример меню опций. В этом приложении есть значок меню (три вертикальные точки) в нижней части. Если по нему кликнуть, раскрывается меню, позволяя сделать такие вещи, как: определить, какие контакты вывести на экран, импортировать/экспортировать контакты, а также произвести настройку приложения.



Затем давайте посмотрим пример контекстного меню. Откроем браузер. Далее через меню опций переходим до закладок. И далее выбираем вкладку «History», которая показывает список недавно посещённых веб-страниц. Теперь, если нажать и удерживать одну из этих записей истории, будет вызвано новое меню.



Это контекстное меню поддерживает действия, которые могут быть применены к выбранной ссылке веб-страницы, такие как: открыть, добавить в закладки, поделиться и другие.

Чтобы создать меню, сначала необходимо определить содержание меню в xml-файле в каталоге res/menu. Когда пользователь открывает меню, Android вызывает определенный метод – onCreateOptionsMenu для меню опций и их подме-

ню или onCreateContextMenu для контекстных меню. В этих методах мы будем использовать Menu Inflater, чтобы создавать лейаут меню.

Когда пользователь выберет один из пунктов меню, Android вызовет метод onOptionsItemSelected для меню опций и подменю или onContextItemSelected для контекстных меню. Давайте рассмотрим простой пример со всеми этими различными видами меню.

Это приложение «HelloAndroidWithMenus». При его запуске вы увидите текстовое вью со словами «Hello Android». Если нажать и подержать это текстовое поле, то раскроется контекстное меню.

В верхнем правом углу есть значок (три вертикальные точки), который предоставляет доступ к пунктам меню. Назовем этот значок кнопкой «Menu», а место, где он находится – зоной переполнения (overflow area).

Давайте посмотрим на то, как это реализовано в исходном коде. Сначала мы посмотрим на метод onCreateOptionsMenu. В этом методе мы получаем Menu Inflater, и затем вызываем его метод Inflate, передавая ссылку на лейаут меню.

```
// Create Options Menu
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.top_menu, menu);
    return true;
}
```

Теперь откроем файл меню Top_menu.xml. Этот файл содержит тег меню, и в нем есть несколько тегов элементов. В каждом теге есть атрибуты, такие как: ID, иконка, которая выводится на экран для этого элемента и заголовок этого элемента.

В Activity в последней строке onCreateOptionsMenu возвращаем значение true, указав, что мы хотим вывести на экран этот пункт меню. Теперь, когда пользователь выбирает один из этих пунктов меню, Android вызовет onOptionsItemSelected, передав ему выбранный пункт.

```
// Process clicks on Options Menu items
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    case R.id.help:
        Toast.makeText(getApplicationContext(), "you've been helped",
                Toast.LENGTH_SHORT).show();
        return true;
    case R.id.more_help:
        Toast.makeText(getApplicationContext(), "you've been helped more",
                Toast.LENGTH_SHORT).show();
        return true;
    case R.id.even_more_help:
        return true;
    default:
        return false;
    }
}
```

Здесь мы проверяем ID элементов, и затем совершаем соответствующие действия для этого элемента.

Теперь посмотрим на то, как устанавливается контекстное меню. Сначала, когда пользователь впервые вызывает контекстное меню, Android вызывает onCreateContextMenu.

```
// Create Context Menu
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
        ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.context_menu, menu);
}
```

Код подобен тому, что мы видели в меню опций. Мы получаем Menu Inflater и передаем ему xml-файл лейаута. У этого меню есть один-единственный элемент с ID help_guide.

```
// Process clicks on Context Menu Items
@Override
public boolean onContextItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    case R.id.help_guide:
        Toast.makeText(getApplicationContext(), "ContextMenu Shown",
                Toast.LENGTH_SHORT).show();
        return true;
    default:
        return false;
    }
}
```

Когда пользователь выбирает один из пунктов контекстного меню, Android вызывает onContextItemSelected, передав ему выбранный пункт.

В дополнение к вышесказанному, меню могут также поддерживать еще много расширенных функций. Например, вы можете поместить связанные пункты меню в группу – таким образом, вы сможете обработать и управлять ими как одним

пунктом. Вы можете также связать сочетания клавиш с определенными пунктами меню (shortcut) – таким образом, вы сможете получить доступ к ним более быстро. И вы можете связать Intent с пунктами меню. Так, например, можно запустить какую-нибудь Activity, когда пользователь кликает по определенному пункту меню.

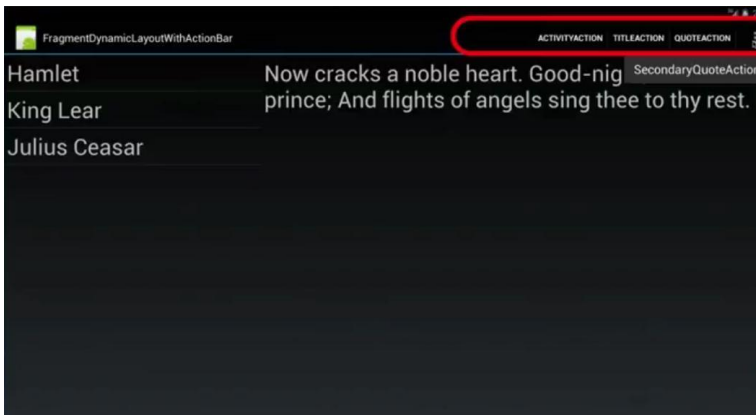
Панель Action Bar была добавлена в Android 3.0 и представляет собой подобие панели управления приложением, которую вы часто видите в десктопных приложениях. Это та панель наверху окна приложения, которая показывает такие пункты как: «Файл», «Редактирование», «Справка» и т. д. Основная идея Action Bar состоит в том, что вместо того, чтобы скрывать действия внутри традиционного всплывающего меню, вы можете предоставить пользователям быстрый доступ к действиям, которые они будут использовать. Давайте рассмотрим некоторые варианты использования Action Bar.

Первый пример возвращает нас к уроку о фрагментах, к приложению с просмотром цитат из пьес Шекспира. Здесь добавлено несколько элементов к Action Bar, эти элементы описаны тремя различными классами. Некоторые элементы пришли из основной Activity. Некоторые пришли из фрагмента, который выводит на экран заголовки. И некоторые пришли из фрагмента, который выводит на экран цитаты.

Как прежде, когда запускается приложение, есть основная Activity, которая размещает единственный фрагмент за-

головков. Теперь, если мы посмотрим на Action Bar наверху экрана, то увидим две текстовых кнопки в правом верхнем углу – «ActivityAction» и «TitleAction». Первая была помещена туда кодом Activity, а вторая – фрагментом заголовков. И, если кликнуть по ним, то появится всплывающее сообщение, показывая какой объект фактически выполняет действия, связанные с этой кнопкой в Action Bar.

Теперь, если кликнуть по одному из заголовков, то в лейаут будет динамически добавлен фрагмент цитат. А фрагмент цитат тоже добавляет динамически некоторые элементы к Action Bar. В этом случае добавились два действия – QuoteAction и второе действие, которое помещено в зону переполнения (доступ к зоне переполнения обеспечен через кнопку «Меню» – три вертикальные точки). И, если кликнуть по QuoteAction, появится всплывающее сообщение, где говорится, что это действие было вызвано фрагментом цитат. Если кликнуть по значку переполнения (он же – кнопка «Меню»), это вызовет второй элемент – SecondaryQuoteAction, созданный фрагментом цитат. И если кликнуть по нему, то раскрывается связанный с ним текст.



Теперь давайте посмотрим на то, как это реализовано в исходном коде.

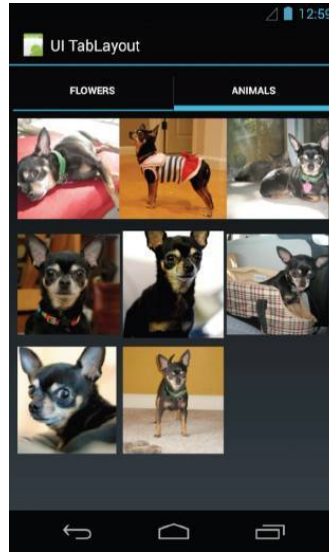
```
<item
    android:id="@+id/detail_menu_item_main"
    android:showAsAction="ifRoom|withText"
    android:title="QuoteAction">
</item>
<item
    android:id="@+id/detail_menu_item_secondary"
    android:showAsAction="never"
    android:title="SecondaryQuoteAction">
</item>
```

В файле menu.xml есть новый параметр – ShowAsAction, его значение – ifRoom/withText. Это означает, что Android должен показать этот элемент в Action Bar, если для него есть свободное место, иначе он будет скрыт в зоне переполнения.

Это также означает, что элемент должен быть показан как текст, а не значок. Если же этот параметр будет равен “never”, то он всегда будет скрыт в зоне переполнения.

Action Bar также обеспечивает удобный способ переключения между различными окнами приложения. При таком использовании Action Bar экран разделен на две части – область вкладок и область содержимого. Класс ActionBar.Tab позволяет связать каждый фрагмент с отдельной вкладкой в области вкладок, и только одна вкладка может быть выбрана в любой момент. Таким образом, когда пользователь выбирает определенную вкладку, ее фрагмент может быть показан в области содержимого. Если пользователь выберет другую вкладку, то в области содержимого будет показан другой фрагмент.

Вот пример приложения под названием «UITabLayout», который использует класс ActionBar.Tab, чтобы переключаться между двумя фрагментами, которые используют лейаут gridView.



Давайте рассмотрим исходный код этого приложения. Откроем файл tab_layout_activity и перейдем к методу onCreate. Во-первых, код получает Action Bar и конфигурирует его, чтобы он работал в режиме вкладок.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.main);

    // Put ActionBar in Tab Mode
    final ActionBar tabBar = getActionBar();
    tabBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);

    // Create a GridFragment for the Flower thumbnails
    GridFragment flowerFrag = new GridFragment();

    // Store the list of thumbnails as an argument to the GridFragment
    Bundle args = new Bundle();
    args.putIntegerArrayList(THUMBNAIL_IDS, mThumbIdsFlowers);
    flowerFrag.setArguments(args);

    // Configure a tab for the Flower thumbnail GridFragment
    tabBar.addTab(tabBar.newTab().setText(FLOWERS_TABSTRING)
            .setTabListener(new TabListener(flowerFrag)));

    // Create a GridFragment for the Animal thumbnails
    GridFragment animalFrag = new GridFragment();

    // Store the list of thumbnails as an argument to the GridFragment
    args = new Bundle();
    args.putIntegerArrayList(THUMBNAIL_IDS, mThumbIdsAnimals);
    animalFrag.setArguments(args);

    // Configure a tab for the Animal thumbnail GridFragment
    tabBar.addTab(tabBar.newTab().setText(ANIMALS_TABSTRING)
            .setTabListener(new TabListener(animalFrag)));

}
```

Затем создаем фрагмент с gridView и передаем ему список изображений, которые он должен вывести на экран. В первом случае это изображения цветов. Наконец, создаем и конфигурируем новую вкладку и присоединяем ее к Action Bar. И то же самое сделаем с другой вкладкой. Теперь, когда вкладки добавлены, создаем экземпляры «слушателей» вкладки (TabListener). Это объекты, которые будут вызваны, когда пользователь выберет какую-либо вкладку.

```
// This class handles user interaction with the tabs
public static class TabListener implements ActionBar.TabListener {
    private static final String TAG = "TabListener";
    private final Fragment mFragment;

    public TabListener(Fragment fragment) {
        mFragment = fragment;
    }
```

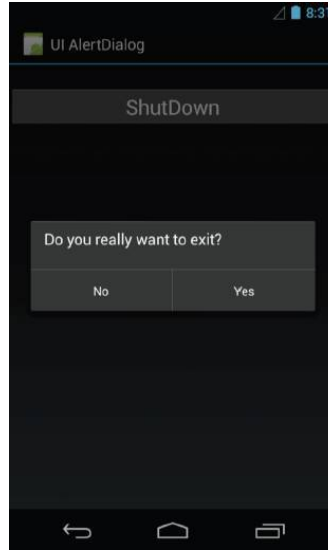
```
    @Override
    public void onTabReselected(Tab tab, FragmentTransaction ft) {
    }

    // When a tab is selected, change the currently visible Fragment
    @Override
    public void onTabSelected(Tab tab, FragmentTransaction ft) {
        Log.i(TAG, "onTabSelected called");

        if (null != mFragment) {
            ft.replace(R.id.fragment_container, mFragment);
        }
    }

    // When a tab is unselected, remove the currently visible Fragment
    @Override
    public void onTabUnselected(Tab tab, FragmentTransaction ft) {
        Log.i(TAG, "onTabUnselected called");

        if (null != mFragment)
            ft.remove(mFragment);
    }
}
```

Метод onTabSelected. Когда вкладка выбрана, этот код добавляет соответствующий фрагмент в Activity. Далее – метод onTabUnselected. Когда вкладка отменяется (при выборе другой вкладки), этот код удаляет текущий фрагмент из Activity.

Диалог – своего рода независимое окно, которое Activity использует для коротких связей с пользователем. Некоторые из классов диалогов, предоставляемые Android, это диалоги оповещения (alert), прогресса (ProgressDialog) и диалоговые

окна выбора даты (DatePicker) и времени (TimePicker).

Давайте рассмотрим пример приложения, которое использует и AlertDialog, и ProgressDialog. Приложение «UIAlertDialog». Когда оно запущено, на экран выведена одна кнопка, которую пользователь может нажать, чтобы инициировать закрытие приложения. Если сделать это, то приложение раскрывает диалоговое окно оповещения – alert dialog, которое показывает сообщение «Вы действительно хотите выйти?» и предлагает пользователю ответить «Да» или «Нет».



Выберем сейчас «No» – это просто закроет диалоговое окно и возвратит назад в приложение. Скажем, через некоторое время я действительно захочу выйти. Тогда я нажму кнопку завершения работы снова, которая выведет то же самое диалоговое окно на экран. На сей раз, однако, я выберу кнопку «Yes». В этой точке диалоговое окно alert dialog закроется, а новое диалоговое окно – на сей раз диалоговое окно прогресса (progress dialog) – будет выведено на экран. Оно сообщает мне, что процесс завершения работы продолжается. В конечном счете процесс завершается и приложение закрывает себя.

Давайте рассмотрим как это реализовано в исходном коде. Откроем файл alert_dialog_activity и перейдем к методу onCreate. Когда пользователь нажимает кнопку ShutDown, вызывается метод showDialogFragment, передавая ID желаемого диалогового окна. Метод showDialogFragment создает экземпляр AlertDialogFragment и затем вызывает его методом show.

AlertDialogFragment – это подкласс DialogFragment. И у него есть метод onCreateDialog. Этот метод будет вызван в ответ на вызов метода show. Он создает новый экземпляр диалогового окна (alert dialog).

```
// Class that creates the AlertDialog
public static class AlertDialogFragment extends DialogFragment {

    public static AlertDialogFragment newInstance() {
        return new AlertDialogFragment();
    }

    // Build AlertDialog using AlertDialog.Builder
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        return new AlertDialog.Builder(getActivity())
                .setMessage("Do you really want to exit?")

                // User cannot dismiss dialog by hitting back button
                .setCancelable(false)

                // Set up No Button
                .setNegativeButton("No",
                        new DialogInterface.OnClickListener() {
                            public void onClick(DialogInterface dialog,
                                    int id) {
                                ((AlertDialogActivity) getActivity())
                                        .continueShutdown(false);
                            }
                        })

                // Set up Yes Button
                .setPositiveButton("Yes",
                        new DialogInterface.OnClickListener() {
                            public void onClick(
                                    final DialogInterface dialog, int id) {
                                ((AlertDialogActivity) getActivity())
                                        .continueShutdown(true);
                            }
                        }).create();
    }
}
```

Фактически здесь вы видите вызов setMessage. И сразу после этого – вызов setCancelable, затем – вызов setNegativeButton, и т. д. Как только мы сделали все вызовы, какие хотели, мы заканчиваем эту часть кода вызовом метода create, который эффективно соединяет все предыдущие вызовы и возвращает конечный сконфигурированный объект. Когда это диалоговое окно выведено на экран, если пользователь выбирает «No», то происходит вызов continueShutdown с параметром shouldContinue, равным false. А если пользователь выбирает «Yes», то происходит

вызов continueShutdown с параметром true. Таким образом, если shouldContinue равен false, то мы закрываем диалоговое окно, и все продолжается, как будто ничего не произошло. А если shouldContinue – true, мы закрываем диалог и затем вызываем showDialogue, передавая ему тег прогресса ID. Это создает объект progressDialogueFragment и затем вызывает его методом show. В конечном счете мы заканчиваем методом onCreateDialogue в классе progressDialogueFragment, где мы создаем новый диалог прогресса (ProgressDialog), устанавливаем ему сообщение «Activity Shutting Down», и затем вызываем setIndetermitate с параметром true, который позволит диалоговому окну оставаться видимым, пока оно не будет закрыто.