# Методические указания на тему “Activity”

1. Activity
2. Android Manifest

## 1. Activity

Разработка приложений под Android предполагает использование некоторых базовых компонентов, из которых и состоит приложение. Ключевыми компонентами в данном случае являются классы android.content.Context и android.app.Activity.

Класс контекста используется для управления специфичными для приложения ресурсами и конфигурацией. Так, с помощью класса Conetxt мы можем получить различные строковые ресурсы, графические ресурсы, получить доступ к системным сервисам, управлять базами данных SQLite, управлять файлами и каталогами приложения

Для получения текущего контекста приложения мы можем воспользоваться методом **getApplicationContext()**:

```
Context context = getApplicationContext();
```

Для получения ресурсов в приложении в классе Context определен метод getResources(). Например, получим строковый ресурсhello_world, который определен в файле *res/values/strings.xml*:

```
String hello_world=
context.getResources().getString(R.string.hello_world);
```

Но в большинстве случае класс контекста не потребуется, поскольку в реальном приложении мы больше будет обращаться к функционалу класса Activity, который наследуется от класса Conetxt. Нередко activity

ассоциируется с отдельным экраном или окном приложения, а переключение между окнами будет происходить как перемещение от одной activity к другой. Приложение может иметь одну или несколько activity.

**Жизненный цикл приложения**

Все приложения Android имеют строго определенный системой жизненный цикл. При запуске пользователем приложения система дает этому приложению высокий приоритет. Каждое приложение запускается в виде отдельного процесса, что позволяет системе давать одним процессам более высокой приоритет, в отличие от других. Благодаря этому, например, при работе с одними приложениями не блокировать входящие звонки. После прекращения работы с приложением, система освобождает все связанные ресурсы и переводит приложение в разряд низкоприоритетного и закрывает его.

Все объекты activity, которые есть в приложении, управляются системой в виде стека activity, который называется **back stack**. При запуске новой activity она помещается поверх стека и выводится на экран устройства, пока не появится новая activity. Когда текущая activity заканчивает свою работу (например, пользователь уходит из приложения), то она удаляется из стека, и возобновляет работу та activity, которая ранее была второй в стеке.

Каждая activity имеет три состояния:

- **Активно (Resumed)**: activity отображается на экране и взаимодействует с пользователем
- **Приостановлено (paused)**: activity теряет фокус, однако по-прежнему остается видимой и продолжает работать. А в ранг активного переводится другая activity, которая может быть либо

невидимой, либо занимать весь экран. Приостановленная activity в то же время может быть завершена системой в случае, если другим activity и приложениям с более высоким приоритетом будет не хватать памяти.

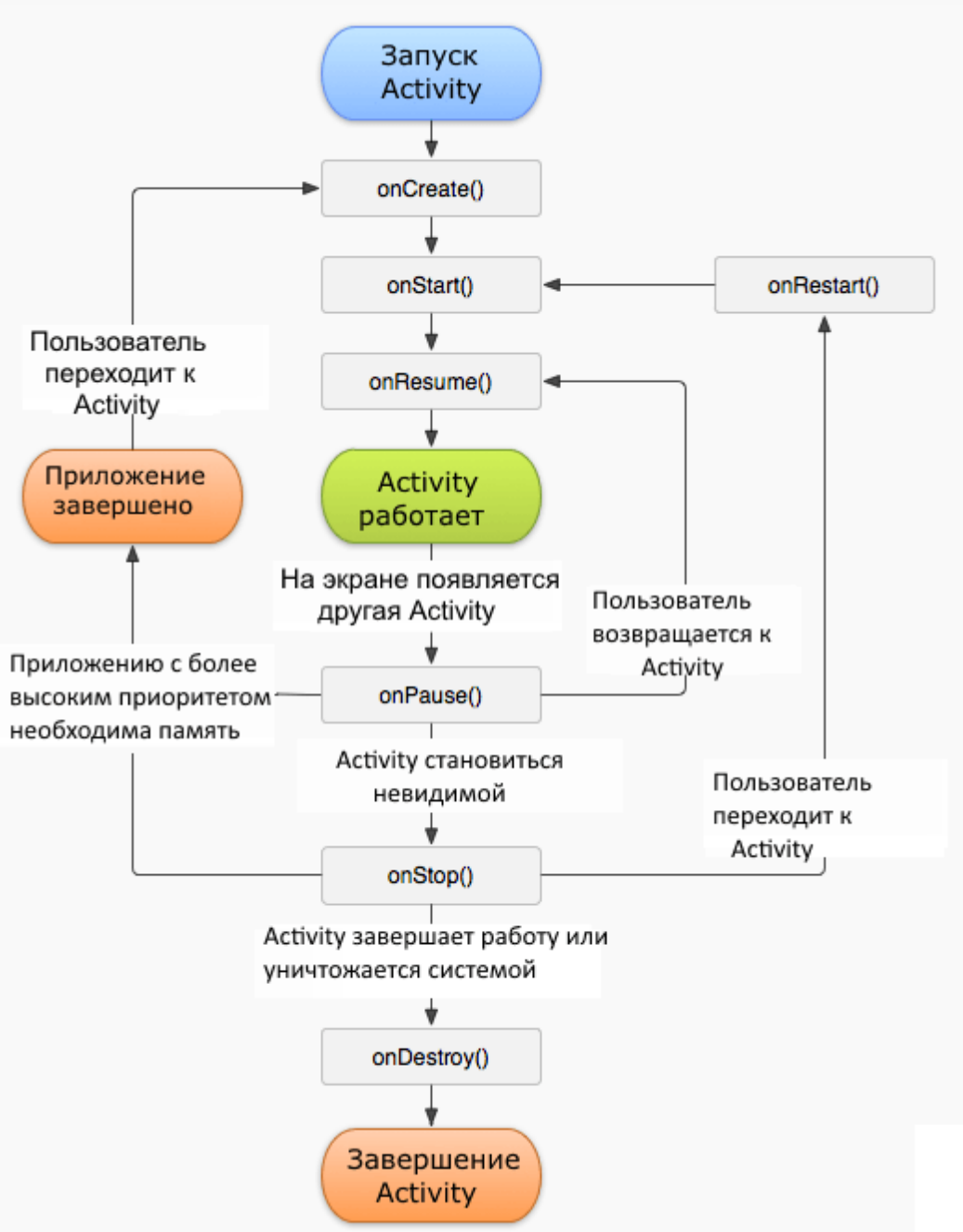
- **Остановлено (stopped)**: activity сохраняет свое состояние, однако она уже невидима пользователю, и если другим процессам потребуется память, то activity будет завершена.

После перевода activity в приостановленное или остановленное состояние система может удалить activity из памяти, либо уведомив о завершении процесса, либо просто убив соответствующий процесс. При последующем запуске и отображении activity полностью пересоздается и начинает свою работу заново.

После запуска activity проходит через ряд событий, которые обрабатываются системой и для обработки которых существует ряд обратных вызовов:

```
protected void OnCreate(Bundle saveInstanceState);
protected void OnStart();
protected void OnRestart();
protected void OnResume();
protected void OnPause();
protected void OnStop();
protected void OnDestroy();
```

Схематично взаимосвязь между всеми этими обратными вызовами можно представить следующим образом




При создании новой активности, например, при запуске приложения, Android вызывает метод OnCreate. В этом методе производится первоначальная настройка activity. В частности, создаются объекты визуального интерфейса. Этот метод получает объект Bundle, который содержит прежнее состояние activity, если оно было сохранено. Если activity заново создается, то данный объект имеет значение null. Если же

activity уже ранее была создана, но находилась в приостановленном состоянии, то bundle содержит связанную с activity информацию. Затем вызывается метод OnStart, а activity переходит в "видимое" состояние. А при вызове метода OnResume activity отображается на экране, и пользователь может с ней взаимодействовать.
Если пользователь решит перейти к другой активности, то система вызывает метод OnPause. После этого, если пользователь решит вернуться к прежней активности, то система вызовет снова метод OnResume, и activity снова появится на экране. Иначе, если activity больше невидима, то вызывается метод OnStop.
Если после вызова метода OnStop пользователь решит вернуться к прежней activity, тогда система вызовет метод OnRestart. Ну и завершается работа активности вызовом метода OnDestroy, который возникает либо, если система решит убить activity, либо при вызове метода finish().
Также следует отметить, что при изменении ориентации экрана система завершает activity и затем создает ее заново.

# Android Manifest

Файл манифеста **AndroidManifest.xml** предоставляет основную информацию о программе системе. Каждое приложение должно иметь свой файл **AndroidManifest.xml**. Редактировать файл манифеста можно вручную, изменяя XML-код или через визуальный редактор Manifest Editor (Редактор файла манифеста), который позволяет осуществлять визуальное и текстовое редактирование файла манифеста приложения.

## Назначение файла

- объявляет имя Java-пакета приложения, который служит уникальным идентификатором;
- описывает компоненты приложения — деятельности, службы, приемники широковещательных намерений и контент-провайдеры, что позволяет вызывать классы, которые реализуют каждый из компонентов, и объявляет их намерения;
- содержит список необходимых разрешений для обращения к защищенным частям API и взаимодействия с другими приложениями;
- объявляет разрешения, которые сторонние приложения обязаны иметь для взаимодействия с компонентами данного приложения;
- объявляет минимальный уровень API Android, необходимый для работы приложения;
- перечисляет связанные библиотеки;

## Общая структура манифеста

Файл манифеста инкапсулирует всю архитектуру Android-приложения, его функциональные возможности и конфигурацию. В процессе разработки приложения вам придется постоянно редактировать данный файл, изменяя его структуру и дополняя новыми элементами и атрибутами.

Корневым элементом манифеста является **<manifest>**. Помимо данного элемента обязательными элементами является теги**<application>** и **<uses-sdk>**. Элемент **<application>** является основным элементом манифеста и содержит множество дочерних элементов, определяющих структуру и работу приложения. Порядок расположения элементов, находящихся на одном уровне, произвольный. Все значения устанавливаются через атрибуты элементов. Кроме обязательных

элементов, упомянутых выше, в манифесте по мере необходимости используются другие элементы.

## Описание

```
<?xml version="1.0" encoding="utf-8"?>
<manifest />

 <uses-permission />
 <permission />
 <permission-tree />
 <permission-group />
 <instrumentation />
 <uses-sdk />
 <uses-configuration />
 <uses-feature />
 <supports-screens />

 <application>

   <activity>
     <intent-filter>
       <action />
       <category />
       <data />
     </intent-filter>
     <meta-data />
   </activity>
```

```
<activity-alias>
  <intent-filter>
    <action />
    <category />
    <data />
  </intent-filter>
  <meta-data />
</activity-alias>

<service>
  <intent-filter>
    <action />
    <category />
    <data />
  </intent-filter>
  <meta-data />
</service>

<receiver>
  <intent-filter>
    <action />
    <category />
    <data />
  </intent-filter>
  <meta-data />
</receiver>
```

```
   <provider>
     <grant-uri-permission />
          <path-permission />
     <meta-data />
   </provider>

   <uses-library />

 </application>

</manifest>
```

## <manifest>

Элемент **<manifest>** является корневым элементом манифеста. По умолчанию Eclipse создает элемент с четырьмя атрибутами:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="ru.alexanderklimov.helloandroid"
android:versionCode="1"
android:versionName="1.0">
```

## Атрибуты

**xmlns:android**

определяет пространство имен Android. Оно всегда одно и то же

**package**

определяет уникальное имя пакета приложения, которое вы задали при создании проекта. Android Marketplace проверяет уникальность при приеме приложения, поэтому рекомендуется использовать свое имя для избежания конфликтов с другими разработчиками. Например, я использую имя своего сайта в обратном порядке: ru.alexanderklimov.appname

**android:versionCode**

указывает на внутренний номер версии, используемый для сравнения версий программы. «versionCode» должен быть целым, и Android Market использует это для определения, предоставили ли вы новую версию, передавая триггеру обновления на устройствах, на которых установлено ваше приложение. Как правило. начинается с 1 и увеличивается на единицу, если вы выпускаете новую версию приложения.

**android:versionName**

указывает номер пользовательской версии. Можно использовать строку или строковый ресурс. Этот номер видит пользователь.

## <permission>

Элемент **<permission>** объявляет разрешение, которое используется для ограничения доступа к определенным компонентам или функциональности данного приложения. В этой секции описываются права, которые должны запросить другие приложения для получения доступа к вашему приложению. Приложение может также защитить свои

собственные компоненты (деятельности, службы, приемники широковещательных намерений и контент-провайдеры) разрешениями. Оно может использовать любое из системных разрешений, определенных Android или объявленных другими приложениями, а также может определить свои собственные разрешения.

**android:name**

название разрешения

**android:label**

имя разрешения, отображаемое пользователю

**android:description**

описание разрешения

**android:icon**

значок разрешения

**android:permissionGroup**

определяет принадлежность к группе разрешений

**android:protectionLevel**

уровень защиты

## <uses-permission>

Элемент **<uses-permission>** запрашивает разрешение, которые приложению должны быть предоставлены системой для его нормального

функционирования. Разрешения предоставляются во время установки приложения, а не во время его работы.

**android:name**

<uses-permission> имеет единственный атрибут с именем разрешения android:name. Это может быть разрешение, определенное в элементе <permission> данного приложения, разрешение, определенное в другом приложении или одно из стандартных системных разрешений, например: **android:name="android.permission.CAMERA"** или**android:name=""android.permission.READ_CONTACTS"**

Наиболее распространенные разрешения

- **INTERNET** - доступ к интернету
- **READ_CONTACTS** - чтение (но не запись) данных из адресной книги пользователя
- **WRITE_CONTACTS** - запись (но не чтение) данных из адресной книги пользователя
- **RECEIVE_SMS** - обработка входящих SMS
- **ACCESS_COARSE_LOCATION** - использование приблизительного определения местонахождения при помощи вышек сотовой связи или точек доступа Wi-Fi
- **ACCESS_FINE_LOCATION** - точное определение местонахождения при помощи GPS

## <permission-tree>

Элемент <permission-tree> объявляет базовое имя для дерева разрешений. Этот элемент объявляет не само разрешение, а только пространство имен, в которое могут быть помещены дальнейшие разрешения.

## <permission-group>

Элемент **<permission-group>** определяет имя для набора логически связанных разрешений. Это могут быть как объявленные в этом же манифесте с элементом <permission> разрешения, так и объявленные в другом месте. Этот элемент не объявляет разрешение непосредственно, только категорию, в которую могут быть помещены разрешения. Разрешение можно поместить в группу, назначив имя группы в атрибуте **permissionGroup** элемента <permission>.

## <instrumentation>

Элемент **<instrumentation>** объявляет объект *instrumentation*, который дает возможность контролировать взаимодействие приложения с системой. Обычно используется при отладке и тестировании приложения и удаляется из release-версии приложения.

## <uses-sdk>

Элемент **<uses-sdk>** позволяет объявлять совместимость приложения с указанной версией (или более новыми версиями API) платформы Android. Уровень API, объявленный приложением, сравнивается с уровнем API системы мобильного устройства, на который инсталлируется данное приложение.

## Атрибуты

**android:minSdkVersion**

определяет минимальный уровень API, требуемый для работы приложения. Система Android будет препятствовать тому, чтобы пользователь установил приложение, если уровень API системы будет ниже, чем значение, определенное в этом атрибуте. Вы должны всегда объявлять этот атрибут, например: *android:minSdkVersion="11"*. Вы можете ради интереса установить значение 7, а потом 11 и сравнить

внешний вид приложения. Например, у младшей версии не будет отображаться ActionBar.

**android:maxSDKVersion**

позволяет определить самую позднюю версию, которую готова поддерживать ваша программа. Ваше приложение будет невидимым в Google Play для устройств с более свежей версией. Рекомендуется устанавливать в том случае, когда вы точно уверены, что приложение не будет корректно работать на новой платформе.

**targetSDKVersion**

позволяет указать платформу, для которой вы разрабатывали и тестировали приложение. Устанавливая значение для этого атрибута, вы сообщаете системе, что для поддержки этой конкретной версии не требуется никаких изменений.

## <uses-configuration>

Элемент **<uses-configuration>** указывает требуемую для приложения аппаратную и программную конфигурацию мобильного устройства. Например, приложение могло бы определить требования обязательного наличия на устройстве физической клавиатуры или USB-nopTa. Спецификация используется, чтобы избежать установки приложения на устройствах, которые не поддерживают требуемую конфигурацию. Если приложение может работать с различными конфигурациями устройства, необходимо включить в манифест отдельные элементы <uses-configuration> для каждой конфигурации. Вы можете задать любую комбинацию, содержащие следующие устройства

- **reqFiveWayNav** - используйте значение *true*, если приложению требуется устройство ввода, поддерживающее навигацию вверх,

вниз, влево, вправо, а также нажатие выделенного элемента. К таким устройствам относятся трекболы и D-pad. В принципе устарело

- **reqHardKeyboard** - используйте значение *true*, если приложению нужна аппаратная клавиатура.
- **reqKeyboardType** - позволяет задать тип клавиатуры: nokeys, qwerty, twelvekey, undefined
- **reqNavigation** - укажите одно из значений: nonav, dpad, trackball, wheel или undefined, если требуется устройство для навигации
- **reqTouchScreen** - если требуется сенсорный экран, то используйте нужное значение из возможных вариантов: notouch, stylus, finger, undefined. Сейчас практически все устройства содержат сенсорный экран, поэтому тоже устарело

Приложение не будет устанавливаться на устройстве, которое не соответствует заданной вами конфигурации. В идеале, вы должны разработать такое приложение, которое будет работать с любым сочетанием устройств ввода. В этом случае **<uses-configuration>** не нужен.

## <uses-feature>

Элемент **<uses-feature>** объявляет определенную функциональность, требующуюся для работы приложения. Таким образом, приложение не будет установлено на устройствах, которые не имеют требуемую функциональность. Например, приложение могло бы определить, что оно требует камеру с автофокусом. Если устройство не имеет встроенную камеру с автофокусом, приложения не будет инсталлировано.

Пример

**android.hardware.camera**

требуется аппаратная камера

**android.hardware.camera.autofocus**

требуется камера с автоматической фокусировкой

Можно переопределить требование по умолчанию, добавив атрибут **required** со значением **false**. Например, если вашей программе не требуется, чтобы камера поддерживала автофокус, то используйте вариант:

```
<uses-feature android:name="android.hardware.camera.autofocus"
android:required="false" />
```

## <supports-screens>

Элемент **<supports-screens>** определяет разрешение экрана, требуемое для функционирования устройства. Данный тег позволяет указать размеры экран, для которого был спроектировано приложение. Система будет масштабировать ваше приложение на основе ваших макетов на тех устройствах, которые поддерживают указанные вами разрешения экран. Для других случаев система будет растягивать макет по мере возможности.

Возможные значения

**smallScreen**

как правило экраны QVGA

**normalScreen**

стандартные экраны HVGA и WQVGA

**largeScreen**

большие экраны

**xlargeScreen**

очень большие экраны, которые превосходят размеры планшетов

**anyDensity**

установите значение *true*, если ваше приложение способно масштабироваться для отображения на экране с любым разрешением.

По умолчанию, для каждого атрибута установлено значение *true*. Вы можете указать, какие размеры экранов ваше приложение не поддерживает.

```
<supports-screens
   android:smallScreen=["false"]
        android:normalScreen=["true"]
        android:largeScreen=["true"]
        android:anyDensity=["false"] />
```

Начиная с API 13 (Android 3), у тега появились новые атрибуты:

- **requiresSmallestWidthDp** - указываем минимальную поддерживаемую ширину экрана (наименьшая сторона устройства) в аппаратно-независимых пикселях. С его помощью

можно отфильтровать устройства при размещении приложения в Google Play

- **compatibleWidthLimitDp** - задаёт верхнюю границу масштабирования для вашего приложения. Если экран устройства выходит за указанную границу, система включит режим совместимости.
- **largestWidthLimitDp** - задаёт абсолютную верхнюю границу, за пределами которой ваше приложение точно не может быть смаштабировано. В этом случае приложение запускается в режиме совместимости, которую нельзя отключить. Следует избегать подобных ситуаций и разрабатывать макеты для любых экранов.

```
<supports-screens android:smallScreens="false"
    android:normalScreens="true"
    android:largeScreens"="true"
    android:requiresSmallestWidthDp="480"
    android:compatibleWidthLimitDp="600"
    android:largestWidthLimitDp="720" />
```

## <application>

Элемент **<application>** один из основных элементов манифеста, содержащий описание компонентов приложения, доступных в пакете: стили, значок и др. Содержит дочерние элементы, которые объявляют каждый из компонентов, входящих в состав приложения. В манифесте может быть только один элемент **<application>**.

## <activity>

Элемент **<activity>** объявляет активность. Если приложение содержит несколько активностей, не забывайте объявлять их в манифесте, создавая для каждой из них свой элемент <activity>. Если активность не объявлена в манифесте, она не будет видна системе и не будет запущена при выполнении приложения или будет выводиться сообщение об ошибке.

Для этого класса зарегистрирован фильтр вызовов, определяющий, что это действие запущено в приложении (действие android:name=«android.intent.action.MAIN»). Определение категории (категория android:name=«android.intent.category.LAUNCHER» ) определяет, что это приложение добавлено в директорию приложений на Android-устройстве. Значения @ направляют файлы ресурсов, которые содержат актуальные значения. Это упрощает работу с разными ресурсами, такими как строки, цвета, значки.

Пример:

```
<activity android:name="ru.alexanderklimov.HelloWorld.AboutActivity"
android:label="@string/app_name">
```

## Атрибуты

**android:name**

имя класса. Имя должно включать полное обозначение пакета, но т. к. имя пакета уже определено в корневом элементе <manifest>, имя класса, реализующего деятельность, можно записывать в сокращенном виде, опуская имя пакета

**android:label**

текстовая метка, отображаемая пользователю

Элемент <activity> содержит множество других атрибутов, определяющих разрешения, ориентацию экрана и т. д.

## Изменение конфигурации во время выполнения программы

При изменении языка, региона или аппаратной конфигурации Android прерывает работу всех приложений и затем запускает их повторно, перезагружая значения из ресурсов. Подобное поведение не всегда уместно и желательно. Например, некоторые изменения конфигурации (ориентация экрана в пространстве, доступность клавиатуры) могут произойти только лишь из-за того, что пользователь повернул устройство или выдвинул клавиатуру. Вы можете настраивать, каким образом ваше приложение будет реагировать на подобные изменения, обнаруживая их и выполняя собственные действия. Чтобы заставить Активность отслеживать изменения конфигурации при выполнении программы, добавьте в ее узел в манифесте атрибут**android:configChanges**, указав, какие именно события хотите обрабатывать.

Перечислим некоторые значения, с помощью которых можно описать изменения конфигурации:

- **orientation** — положение экрана изменено с портретного на альбомное (или наоборот);
- **keyboardHidden** — клавиатура (или D-pad и другое устройство) выдвинута или спрятана;
- **fontScale** — пользователь изменил предпочтительный размер шрифта;
- **locale** — пользователь выбрал новые языковые настройки;

- **keyboard** — изменился тип клавиатуры; например, телефон может иметь 12-клавишную панель, при повороте которой появляется полноценная клавиатура. Или была подключена внешняя клавиатура.
- **touchscreen** или **navigation** — изменился тип клавиатуры или способ навигации. Как правило, такие события не встречаются.
- **mcc** или **mnc** - обнаружена новая SIM-карта, при этом изменились страна и код сотовой сети соответственно.
- **uiMode** - изменился режим пользовательского интерфейса, например, при переключении между автомобильным, дневным и ночным режимами.
- **screenLayout** - изменились характеристики экрана, например, при активации другого дисплея.
- **screenSize** - изменлись размеры экрана, например, при смене ориентации. Появилось в Android 3 (API 12)
- **smallestScreenSize** - изменился физический размер экрана, например, при подключении внешнего дисплея. Появилось в Android 3 (API 12)

В некоторых случаях одновременно будут срабатывать несколько событий. Например, когда пользователь выдвигает клавиатуру, большинство устройств генерируют события **keyboardHidden** и **orientation**. Вы можете выбирать несколько событий, которые хотите обрабатывать самостоятельно, разделяя их символом |.

Наличие атрибута android:configChanges отменяет перезапуск приложения при заданных изменениях конфигурации. Вместо этого внутри активности срабатывает метод **onConfigurationChanged()**. Переопределите его, чтобы появилась возможность обрабатывать изменения в конфигурации. Используйте переданный объект **Configuration**, чтобы получить новые значения. Не забудьте вызвать

одноименный метод из родительского класса и перезагрузить измененные значения со всех ресурсов, которые используются внутри активности.

```
@Override
public void onConfigurationChanged(Configuration _newConfig) {
  super.onConfigurationChanged(_newConfig);
  [ ... Обновите пользовательский интерфейс, используя данные из ресурсов ... ]
  if (_newConfig.orientation == Configuration.ORIENTATION_LANDSCAPE) {
    [ ... Реакция на измененную ориентацию экрана ... ]
  }
  if (_newConfig.keyboardHidden == Configuration.KEYBOARDHIDDEN_NO) {
    [ ... Реакция на выдвигание/задвигание клавиатуры ... ]
  }
}
```

На момент вызова метода, все данные из ресурсов будут обновлены, поэтому применять метод можно без опаски.

Любые изменения конфигурации, которые не были явно помечены для обработки внутри вашего приложения, приведут к перезапуску активности, минуя вызов метода **onConfigurationChanged()**.

<intent-filter>

Каждый тег **<activity>** поддерживает вложенные узлы **<intent-filter>**. Элемент **<intent-filter>** определяет типы намерений, на которые могут ответить деятельность, сервис или приемник намерений. Фильтр намерений объявляет возможности его родительского компонента — что могут сделать деятельность или служба и какие типы рассылок получатель может обработать. Фильтр намерений предоставляет для компонентов-клиентов возможность получения намерений объявляемого типа, отфильтровывая те, которые не значимы для компонента, и содержит дочерние элементы <action>, <category>, <data>.

## <action>

Элемент **<action>** добавляет действие к фильтру намерений. Элемент <intent-filter> должен содержать один или более элементов <action>. Если в элементе <intent-fiiter> не будет этих элементов, то объекты намерений не пройдут через фильтр. Пример объявления действия:

```
<action android:name="android.intent.action.MAIN">
```

## <category>

Элемент **<category>** определяет категорию компонента, которую должно обработать намерение. Это строковые константы, определенные в классе intent, например:

```
<category android:name="android.intent.category.LAUNCHER">
```

## <data>

Элемент **<data>** добавляет спецификацию данных к фильтру намерений. Спецификация может быть только типом данных (атрибут mimeType), URI или ТИПОМ данных вместе с URI. Значение URI определяется отдельными атрибутами для каждой из его частей, т. е. URI делитСЯ на части: android:scheme, android:host, android:port, android:path или android:pathPrefix, android:pathPattern.

## <meta-data>

Элемент **<meta-data>** определяет пару "имя-значение" для элемента дополнительных произвольных данных, которыми можно снабдить родительский компонент. Составляющий элемент может содержать любое число элементов <meta-data>.

## <activity-alias>

Элемент **<activity-alias>** — это псевдоним для Activity, определенной в атрибуте **targetActivity**. Целевая деятельность должна быть в том же самом приложении, что и псевдоним, и должна быть объявлена перед псевдонимом деятельности в манифесте. Псевдоним представляет целевую деятельность как независимый объект. У псевдонима может быть свой собственный набор фильтров намерений, определяющий, какие намерения могут активизировать целевую деятельность и как система будет обрабатывать эту деятельность. Например, фильтры намерений на псевдониме деятельности могут определить флаги**android:name="android.intent.action.MAIN"** и **android:name="android.intent.category.LAUNCHER"**, заставляя целевую деятельность загружаться при запуске приложения даже в том случае, когда в фильтрах намерений на целевой деятельности эти флаги не установлены.

## <service>

Элемент **<service>** объявляет службу как один из компонентов приложения. Все службы должны быть представлены элементом <service> в файле манифеста. Службы, которые не были объявлены, не будут обнаружены системой и никогда не будут запущены. Этот элемент имеет много атрибутов, определяющих имя, доступность, разрешения, процесс и т. д. Поддерживает вложенные узлы <intent-fiiter>

## <receiver>

Элемент **<receiver>** объявляет приемник широковещательных намерений как один из компонентов приложения. Приемники широковещательных намерений дают возможность приложениям получить намерения, которые переданы системой или другими приложениями, даже когда другие компоненты приложения не работают.

## <provider>

Элемент **<provider>** объявляет контент-провайдера (источник данных) для управления доступом к базам данных. Все контент-провайдеры, которые являются частью приложения, должны быть представлены в элементах <provider> в файле манифеста. Если они не объявлены, они не будут работать, т. к. система их не сможет увидеть. Элемент <provider> содержит свой набор дочерних элементов для установления разрешений доступа к данным:

- <grant-uri-permission>;
- <path-permission>;
- <meta-data>

Этот элемент имеет много атрибутов, определяющих имя, доступность, разрешения, процесс и т. д.

## <grant-uri-permission>

Элемент **<grant-uri-permission>** является дочерним элементом для <provider>. Он определяет, для кого можно предоставить разрешения на подмножества данных контент-провайдера. Предоставление разрешения является способом допустить к подмножеству данных, предоставляемым контент-провайдером, клиента, у которого нет разрешения для доступа к полным данным. Если атрибут granturiPermissions контент-провайдера имеет значение true, то разрешение предоставляется для любых данных, поставляемых контент-провайдером. Однако, если атрибут поставлен в false, разрешение можно предоставить только подмножествам данных, которые определены этим элементом. Контент-провайдер может содержать любое число элементов <grant-uri-permission>.

## <path-permission>

Элемент **<path-permission>** — дочерний элемент для <provider>. Определяет путь и требуемые разрешения для определенного подмножества данных в пределах поставщика оперативной информации. Этот элемент может быть определен многократно, чтобы поставлять множественные пути.

## <uses-library>

Элемент **<uses-library>** определяет общедоступную библиотеку, с которой должно быть скомпоновано приложение. Этот элемент указывает системе на необходимость включения кода библиотеки в загрузчик классов для пакета приложения. Каждый проект связан по умолчанию с библиотеками Android, в которые включены основные пакеты для сборки приложений (с классами общего назначения типа Activity, Service, Intent, View, Button, Application, ContentProvider и т. д.). Однако некоторые пакеты находятся в отдельных библиотеках, которые автоматически не компонуются с приложением. Если же приложение использует пакеты из этих библиотек или других, от сторонних

разработчиков, необходимо сделать явное связывание с этими библиотеками и манифест обязательно должен содержать отдельный элемент <uses-library>.