# 1. ВВЕДЕНИЕ В РАЗРАБОТКУ ANDROID-ПРИЛОЖЕНИЙ.

## 1.1. Введение. История Android.

Android – открытая операционная система для мобильных телефонов, смартфонов, коммуникаторов, планшетных компьютеров, электронных книг, цифровых проигрывателей, наручных часов, нетбуков и смартбуков, основанная на ядре Linux и поддерживающая различные аппаратные платформы, такие как ARM, MIPS, POWER, x86.

Изначально разрабатывалась компанией Android Inc., которую затем (июль, 2005) купила Google. Впоследствии (ноябрь, 2007) Google инициировала создание бизнес-альянса Open Handset Alliance (в его состав вошли Google, HTC, Intel, Motorola, Nvidia и другие компании), который и занимается сейчас поддержкой и дальнейшим развитием платформы.

С момента выхода первой версии (сентябрь, 2008) произошло несколько обновлений системы. Эти обновления, как правило, касаются исправления обнаруженных ошибок и добавления нового функционала в систему. Каждая версия системы получает собственное кодовое имя на тему десерта.

## 1.2. Инструментарий разработчика.

Перед тем, как приступить к созданию Android-приложений, необходимо выбрать подходящий инструментарий разработки.

Как правило, разработка Android-приложений осуществляется на языке Java. Поэтому, в первую очередь, необходимо установить Java Development Kit (JDK). Но для начала следует разобраться, что представляет из себя Java.

**Java** – это объектно-ориентированный язык программирования. Программы на Java транслируются в байт-код, выполняемый виртуальной машиной Java, которая обрабатывает байтовый код и передает инструкции оборудованию как интерпретатор. Достоинство подобного способа выполнения программ заключается в полной независимости байт-кода от операционной системы и оборудования, что позволяет выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. Другой важной особенностью Java является гибкая система безопасности благодаря тому, что исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером) вызывают немедленное прерывание. Следует заметить, что фактически, большинство архитектурных решений, принятых при создании Java, было продиктовано желанием предоставить синтаксис, сходный с C/C++. В Java используются практически идентичные соглашения для объявления переменных, передачи параметров и операторов. Поэтому те, кто уже имеет опыт программирования на C/C++, смогут быстро освоиться и начать писать Java-приложения.

**JDK** – это бесплатно распространяемый комплект разработчика приложений на языке Java, включающий в себя компилятор Java, стандартные библиотеки классов Java, примеры, документацию, различные утилиты и исполнительную систему Java Runtime Environment (JRE). В состав JDK не входит интегрированная среда разработки (Integrated Development Environment). Поэтому после того, как будет установлен JDK, следует установить IDE.

Существует несколько популярных сред разработки, но в данном курсе мы остановим свой выбор на **Eclipse IDE** и соответствующем для нее плагине **Android Development Tools** (ADT).

**Android Software Development Kit** (SDK) содержит множество инструментов и утилит для создания и тестирования приложений. Например, с помощью SDK Manager можно установить Android API любой версии (Рис. 1.1), а также проверить репозиторий на наличие доступных, но еще не установленных пакетов и архивов.

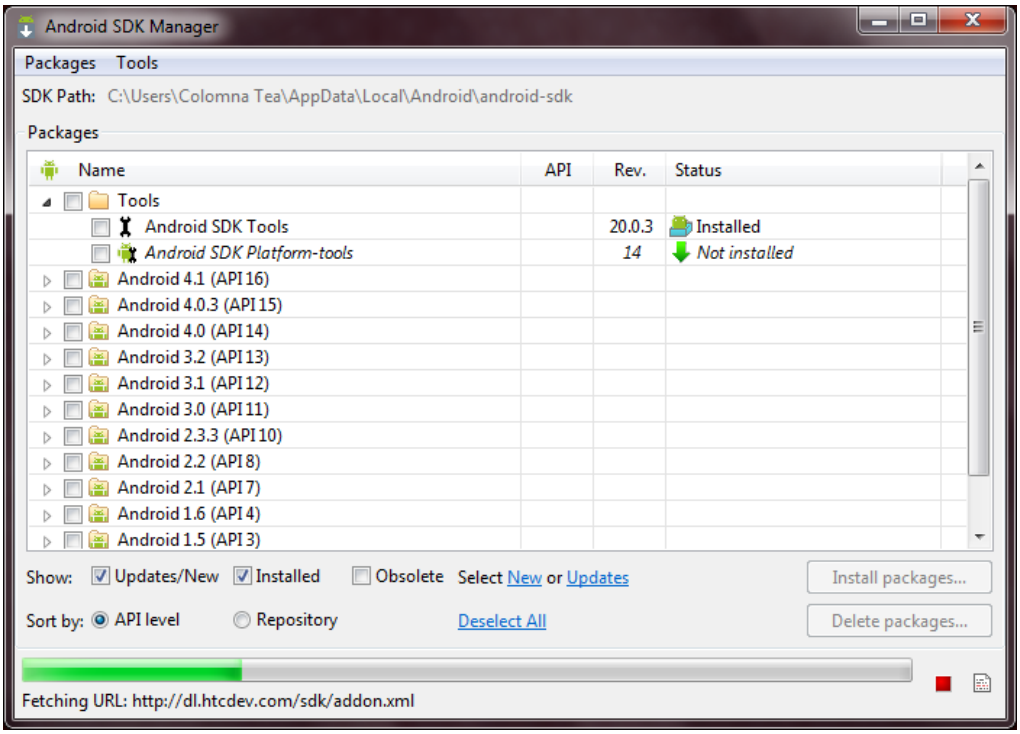


Рис. 1.1 Окно SDK Manager, в котором разработчику предоставляется возможность выбрать версии Android API для установки.

**Android Native Development Kit** (NDK) позволяет осуществлять разработку Android-приложений на языке C/C++. Зачем это может потребоваться? Есть несколько причин. Например, необходимость использовать код, который уже написан для нативной платформы, или ускорение выполнения критических кусков кода.

### 1.3. Архитектура Android

Рассмотрим основные компоненты операционной системы Android (Рис. 1.2).

**Applications.** Android поставляется с набором основных приложений, включающий календарь, карты, браузер, менеджер контактов и другие. Все перечисленные приложения написаны на Java.

**Application Framework.** Предоставляя открытую платформу разработки, Android дает разработчикам возможность создавать гибкие и инновационные приложения. Разработчики могут использовать аппаратные возможности устройства, получать информацию о местоположении, выполнять задачи в фоновом режиме, устанавливать оповещения и многое другое. Разработчики имеют полный доступ к тем же API, что используются в основных приложениях.

Архитектура приложений разработана с целью упрощения повторного использования компонентов; любое приложение может "публиковать" свои возможности и любое другое

приложение может затем использовать эти возможности (с учетом ограничений безопасности). Этот же механизм позволяет заменять стандартные компоненты на пользовательские.

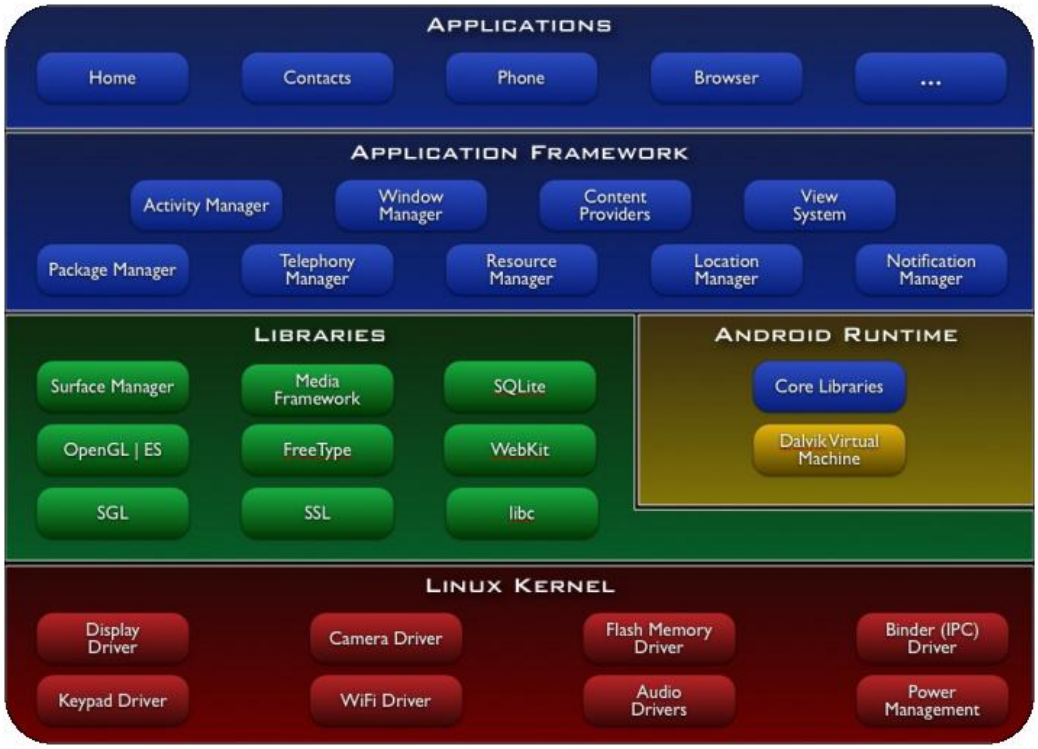


Рис. 1.2 Архитектура операционной системы Android.

**Libraries.** Android включает в себя набор C/C++ библиотек, используемых различными компонентами системы. Эти возможности доступны разработчикам в контексте применения Android Aplication Framework. Некоторые основные библиотеки, перечислены ниже:

- Медиа библиотеки – эти библиотеки предоставляют поддержку воспроизведения и записи многих популярных аудио, видео форматов и форматов изображений, в том числе MPEG4, MP3, AAC, AMR, JPG, PNG и дргих;
- Surface Manager – управляет доступом к подсистеме отображения 2D и 3D графических слоев;
- LibWebCore – современный веб-движок, на котором построен браузер Android;
- SGL – основной графический движок 2D;
- 3D библиотеки – реализованы на основе OpenGL; библиотеки используют либо аппаратное 3D-ускорение (при его наличии), либо включены программно;
- FreeType – поддержка растровых и векторных шрифтов
- SQLite – механизм базы данных, доступной для всех приложений.

**Android Runtime.** Android включает в себя набор основных библиотек, которые обеспечивают большинство функций, доступных в библиотеках Java. Каждое приложение Android работает в своем собственном процессе, со своим собственным экземпляром виртуальной машины Dalvik. Dalvik была написана так, что устройство может работать эффективно с несколькими виртуальными машинами одновременно.

Dalvik проектировалась специально под платформу Android. Виртуальная машина оптимизирована для низкого потребления памяти и работы на мобильном аппаратном обеспечении. Dalvik использует собственный байт-код. Android-приложения переводятся компилятором в специальный машинно-независимый низкоуровневый код. И именно Dalvik

интерпретирует и выполняет такую программу при выполнении на платформе. Кроме того, с помощью специальной утилиты, входящей в состав Android SDK, Dalvik способна переводить байт-коды Java в коды собственного формата и также исполнять их в своей виртуальной среде.

**Linux Kernel.** Android основан на Linux версии 2.6 с основными системными службами – безопасность, управление памятью, управление процессами и модель драйверов. Разработчики Android модифицировали ядро Linux, добавив поддержку аппаратного обеспечения, используемого в мобильных устройствах и, чаще всего, недоступного на компьютерах.

### 1.4. Обзор Java-интерфейсов.

Для прикладного программиста Andoid – это набор интерфейсов на языке Java. Рассмотрим, как он организован. В основе набора – пакеты, входящие в стандарт языка Java, такие как java.util, java.lang, java.io. Они есть на любой платформе, где могут быть запущены java-приложения, и неспецифичны для Andoid. К ним примыкают расширения, которые не входят в стандарт языка – пакеты javax.net, javax.xml и другие. Но самым большим и интересным является набор интерфейсов, созданных специально для Android. Рассмотрим некоторые из его пакетов.

Пакеты **android.view** и **android.widget** отвечают за графический интерфейс пользователя (Graphical User Interface). Они содержат набор встроенных виджетов, таких как кнопки и поля ввода, разметки для расположения виджетов на экране. С их помощью можно создать простейшее Android-приложение.

Для работы с примитивами рисования и графическими файлами предназначен пакет **android.graphics**, а с помощью **android.animation** можно создавать несложную анимацию.

Пакет **android.opengl** предоставляет движок OpenGL, **android.gesture** осуществляет поддержку управления жестами на сенсорном экране, позволяет распознавать жесты и создавать новые.

Большое количество интерфейсов предназначено для коммуникации. Пакет **android.net** включает стеки сетевых протоколов высокого уровня, таких как HTTP и SIP, поддержку Wi-Fi. Пакет **android.webkit** – популярный движок веб-браузера, который позволяет легко отображать веб-страницы в приложении. Пакеты **android.bluetooth** и **android.nfc** предоставляют стеки протоколов связи на коротких расстояниях BlueTooth и Near Field Communication соответственно. Пакет **android.telephony** дает доступ к телефонной функциональности – например, отправка SMS.

Для управления прикладными приложениями предназначен пакет **android.app**. Пакет **android.hardware** позволяет обращаться к камере и датчикам, а пакет **android.location** предоставляет информацию о географических координатах устройства, в том числе с помощью датчика GPS.

Пакет **android.media** отвечает за кодирование звуковых и видео потоков, для мобильных устройств это до сих пор вычислительно сложная задача, требующая качественной оптимизации. Пакет **android.database** предоставляет доступ к базам данных.

### 1.5. Структура Android-приложения.

Android-приложения могут быть простыми и сложными, но строение приложений всегда будет одинаковым. Есть обязательные элементы приложений, а есть опциональные, которые используются по мере необходимости. Android-приложение состоит из нескольких основных компонентов: манифест приложения, набор различных ресурсов и исходный код программы.

Следующая таблица демонстрирует обязательные и возможные составляющие структуры Android-приложения:

| Название | Описание | Необходимость |
|---|---|---|
| gen | Файлы, сгенерированные самой Java. Здесь находится такой важный файл как R.java | Да |
| AndroidManifest.xml | Файл манифеста AndroidManifest.xml предоставляет системе основную информацию о программе. Каждое приложение должно иметь свой файл манифеста | Да |
| src | Каталог, в котором содержится исходный код приложения | Да |
| assets | Произвольное собрание каталогов и файлов | Нет |
| res | Каталог, содержащий ресурсы приложения. В данном каталоге могут находиться подпапки drawable, anim, layout, menu, values, xml и raw (см. ниже) | Да |

### 1.5.1. Файл манифеста AndroidManifest.xml.

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

Назначение файла:

- описывает компоненты приложения – Activities, Services, Broadcast receivers и Content providers;
- 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.

Корневым элементом манифеста является <manifest>. Помимо данного элемента обязательными элементами являются 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. Перечислим некоторые из них:

- **<manifest>** является корневым элементом манифеста.
  По умолчанию Eclipse создает элемент с четырьмя атрибутами:
  **xmlns:android** определяет пространство имен Android.
  **package** определяет уникальное имя пакета приложения.
  **android:versionCode** указывает на внутренний номер версии.
  **android:versionName** указывает номер пользовательской версии.
- **<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

приложению. Приложение может также защитить свои собственные компоненты (Activities, Services, Broadcast receivers и Content providers) разрешениями. Оно может использовать любое из системных разрешений, определенных Android или объявленных другими приложениями, а также может определить свои собственные разрешения.

- **<uses-permission>** запрашивает разрешения, которые приложению должны быть предоставлены системой для его нормального функционирования. Разрешения предоставляются во время установки приложения, а не во время его работы.
  Наиболее распространненные 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_FINE_LOCATION** – точное определение местонахождения при помощи GPS
- **<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.
  **android:maxSDKVersion** позволяет определить самую позднюю версию, которую готова поддерживать программа.
  **targetSDKVersion** позволяет указать платформу, для которой разрабатывалось и тестировалось приложение.
- **<uses-configuration>** указывает требуемую для приложения аппаратную и программную конфигурацию мобильного устройства. Спецификация используется, чтобы избежать инсталляции 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.
- **<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 камеры с автофокусом. Если устройство не имеет встроенную камеру с автофокусом, приложение не будет установлено.
  Возможные атрибуты:
  **android.hardware.camera** – требуется аппаратная камера.
  **android.hardware.camera.autofocus** – требуется камера с автоматической фокусировкой.
- **<supports-screens>** определяет разрешение экрана, требуемое для функционирования устройства. По умолчанию современное приложение с уровнем API 4 или выше поддерживает все размеры экрана и должно игнорировать этот элемент.
- **<application>** один из основных элементов манифеста, содержащий описание компонентов приложения. Содержит дочерние элементы (**<activity>**, **<service>**, **<receiver>**, **<provider>** и другие), которые объявляют каждый из компонентов, входящих в состав приложения. В манифесте может быть только один элемент <application>.

### 1.5.2. Ресурсы.

В Android принято хранить такие объекты, как изображения, строковые константы, цвета, анимацию, стили и тому подобное, за пределами исходного кода. Система поддерживает хранение ресурсов во внешних файлах. Внешние ресурсы легче поддерживать, обновлять и редактировать.

В основном, ресурсы хранятся в виде XML-файлов в каталоге res с подкаталогами values, drawable-ldpi, drawable-mdpi, drawable-hdpi, layout. Но также бывают еще два типа ресурсов: raw и assets.

Для удобства система создает идентификаторы ресурсов и использует их в файле **R.java** (класс R, который содержит ссылки на все ресурсы проекта), что позволяет ссылаться на ресурсы внутри кода программы. Статический класс R генерируется на основе заданных ресурсов и создается во время компиляции проекта. Так как файл R генерируется автоматически, то нет смысла его редактировать вручную, потому что все изменения будут утеряны при повторной генерации.

В общем виде ресурсы представляют собой файл (например, изображение) или значение (например, заголовок программы), связанные с создаваемым приложением. Удобство использования ресурсов заключается в том, что их можно изменять без повторной компиляции или новой разработки приложения.

Самыми распространенными ресурсами являются, пожалуй, строки (string), цвета (color) и графические рисунки (bitmap).

В следующей таблице перечислены основные ресурсы Android-приложения:

| Тип ресурса | Размещение | Описание |
|---|---|---|
| Цвета | /res/values/ | Идентификатор цвета, указывающий на цветовой код. |
| Строки | /res/values/ | Строковые ресурсы. В их число также входят строки в формате java и html. |
| Меню | /res/values/ | Меню в приложении можно задать как XML-ресурсы. |
| Параметры | /res/values/ | Представляет собой параметры или размеры различных элементов. |
| Изображения | /res/drawable/ | Ресурсы-изображения. Поддерживает форматы JPG, GIF, PNG (самый предпочтительный) и другие. Каждое изображение является отдельным файлом. Система также поддерживает stretchable images, в которых можно менять масштаб отдельных элементов, а другие элементы оставлять без изменений. |
| Отрисовываемые цвета | /res/values/ или /res/drawable/ | Представляет цветные прямоугольники, которые используются в качетве фона основных отрисовываемых объектов, например точечных рисунков. |
| Анимация | /res/anim/ | Android может выполнить простую анимацию на графике или на серии графических изображений. |
| Произвольные XML-файлы | /res/xml/ | В Android в качестве ресурсов могут использоваться произвольные XML-файлы. |
| Произвольные необработанные ресурсы | /res/raw/ | Любые нескомпилированные двоичные или текстовые файлы, например, видео. |

Помимо изображений в каталоге res/drawable могут храниться ресурсы простых геометрических фигур. Вот лишь некоторые из возможных атрибутов:

- **android:shape** задает тип фигуры: rectangle (прямоугольник), oval (овал), line (линия), ring (окружность);
- **<corners>** создает закругленные углы для прямоугольника;
- **<gradient>** задает градиентную заливку для фигуры; в Android можно создавать три типа градиентов: Linear (линейный), Radial (радиальный) и Sweep (разверточный);
- **<size>** задает размеры фигуры;
- **<solid>** задает сплошной цвет для фигуры.

Анимация в Android бывает двух видов:

- **Frame Animation** – кадровая анимация, традиционная анимация при помощи быстрой смены последовательных изображений, как на кинопленке.
- **Tween Animation** – анимация преобразований может выполняться в виде ряда простых преобразований: изменение позиции (класс TranslateAnimation), размера (ScaleAnimation), угла вращения (RotateAnimation) и уровня прозрачности (AlphaAnimation). Команды анимации определяют преобразования, которые необходимо произвести над объектом. Преобразования могут быть последовательными или одновременными. Последовательность команд анимации определяется в XML-файле (предпочтительно) или в программном коде.

В Android имеется еще один каталог, в котором могут храниться файлы, предназначенные для включения в пакет – **/assets**. Это не ресурсы, а просто необработанные файлы. Этот каталог находится на том же уровне, что и /res. Для файлов, располагающихся в /assets, в R.java не генерируются идентификаторы ресурсов. Для их считывания необходимо указать путь к файлу. Путь к файлу является относительным и начинается с /assets. Этот каталог, в отличие от подкаталога res/, позволяет задавать произвольную глубину подкаталогов и произвольные имена файлов.

### 1.5.3. Разметка.

В Android-приложениях, пользовательский интерфейс построен на View и ViewGroup объектах. Класс ViewGroup является основой для подкласса Layout (разметка).

Разметка (также используются термины компоновка или макет) хранится в виде XML-файла в папке /res/layout. Это сделано для того, чтобы отделить код от дизайна, как это принято во многих технологиях (HTML и CSS, Visual Studio и Expression Blend). Кроме основной компоновки для всего экрана, существуют дочерние компоновки для группы элементов. По сути, компоновка – это некий визуальный шаблон для пользовательского интерфейса приложения, который позволяет управлять элементами, их свойствами и расположением. В своей практике вам придется познакомиться со всеми способами размещения.

Android-плагин для Eclipse включает в себя специальный редактор для создания разметки двумя способами. Редактор имеет две вкладки: одна позволяет увидеть, как будут отображаться элементы управления, а вторая – создавать XML-разметку вручную.

Создавая пользовательский интерфейс в XML-файле, можно отделить представление приложения от программного кода. Можно изменять пользовательский интерфейс в файле разметки без необходимости изменения программного кода. Например, можно создавать XML-

разметки для различных ориентаций экрана мобильного устройства (portrait, landscape), размеров экрана и языков интерфейса. Впрочем, элементы интерфейса можно создавать и программно, когда это необходимо.

Каждый файл разметки должен содержать только один корневой элемент компоновки, который должен быть объектом View или ViewGroup. Внутри корневого элемента можно добавлять дополнительные объекты разметки или дочерние элементы интерфейса, чтобы постепенно формировать иерархию элементов, которую определяет создаваемая разметка.

Существует несколько стандартных типов разметок:

- **FrameLayout** является самым простым типом разметки. Обычно это пустое пространство на экране, которое можно заполнить только дочерним объектом View или ViewGroup. Все дочерние элементы FrameLayout прикрепляются к верхнему левому углу экрана. В разметке FrameLayout нельзя определить различное местоположение для дочернего объекта View. Последующие дочерние объекты View будут просто рисоваться поверх предыдущих представлений, частично или полностью затеняя их, если находящийся сверху объект непрозрачен
- **LinearLayout** выравнивает все дочерние объекты в одном направлении – вертикально или горизонтально. Направление задается при помощи атрибута ориентации android:orientation. Все дочерние элементы помещаются в стек один за другим, так что вертикальный список представлений будет иметь только один дочерний элемент в строке независимо от того, насколько широким он является. Горизонтальное расположение списка будет размещать элементы в одну строку с высотой, равной высоте самого высокого дочернего элемента списка.
- **TableLayout** позиционирует свои дочерние элементы в строки и столбцы. TableLayout не отображает линии обрамления для строк, столбцов или ячеек. TableLayout может иметьстроки с разным количеством ячеек. При формировании разметки таблицы некоторыеячейки при необходимости можно оставлять пустыми. TableLayout удобно использовать, например, при создании логических игр типа Судоку, Крестики-Нолики и тому подобных.
- **RelativeLayout** позволяет дочерним элементам определять свою позицию относительно родительского представления или относительно соседних дочерних элементов.

Все описываемые разметки являются подклассами ViewGroup и наследуют свойства, определенные в классе View.

Разметки ведут себя как элементы управления, и их можно группировать. Расположение элементов управления может быть вложенным. Например, можно использовать RelativeLayout в LinearLayout и так далее. Однако, слишком большая вложенность элементов управления вызывает проблемы с производительностью.

### 1.6. Компоненты Android-приложения.

Каждое Android-приложение запускается в своем собственном процессе. Поэтому приложение изолировано от других запущенных приложений, и неправильно работающее приложение не может беспрепятственно навредить другим Android-приложениям.

Тем не менее, главным параметром Android-приложения является возможность использовать компоненты других приложений, если они дают на это соответствующие права. Допустим, нам нужен некий компонент с прокруткой для отображения текста, и похожий компонент уже реализован в другом приложении. Тогда у нас есть возможность использовать реализованный

компонент. В этом случае наше приложение не копирует необходимый код к себе и не создает ссылку на него. Вместо этого приложение делает запрос на исполнение части кода другого приложения, где есть нужный нам компонент.

В Android существует четыре типа компонентов: Activities, Services, Broadcast receivers и Content providers.

Также важно отметить объекты Intents, в Android-приложениях почти все работает благодаря им. Intent – это механизм для описания одной операции (выбрать фотографию, отправить письмо, сделать звонок, запустить браузер и перейти по указанному адресу и другие). Наиболее распространенный сценарий использования Intent – запуск другой Activity в своём приложении.

### 1.6.1. Activities.

Activity представляет собой пользовательский интерфейс для одного действия, которое пользователь может совершить. Например, приложение для обмена текстовыми сообщениями может иметь одно Activity для отображения списка контактов, другое – для написания сообщения выбранному контакту, третье – для просмотра сообщений и ещё одно для изменения настроек. Все эти Activities формируют единый пользовательский интерфейс, но не зависят друг от друга.

Приложение может состоять из одного Activity или из нескольких. Это зависит от типа приложения и его дизайна. Одно Activity может вызвать другое. Каждое activity задаёт окно для отображения, которое, обычно, занимает весь экран, но может быть меньше и плавать поверх других окон. Activity может использовать дополнительные окна, например, всплывающий диалог, который требует промежуточного ответа пользователя, или окно, которое отображает пользователям важную информацию при выборе элемента, заслуживающего особого внимания.

Визуальный интерфейс строится на основе иерархии визуальных компонентов — объектов, производных от базового класса View. Android имеет ряд готовых к использованию компонентов, включая кнопки, текстовые поля, полосы прокрутки, меню, флажки и многие другие.

Activity может находиться в одном из трех состояний:

- **Active** или **Running** – находится на переднем плане и имеет фокус для взаимодействия с пользователем.
- **Paused** – потеряло фокус, но всё ещё видно пользователю. Сверху находится другое Activity, которое или прозрачно или закрывает не весь экран. Приостановленное Activity полностью «живое» (его состояние сохранено), но может быть уничтожено системой в случае нехватки памяти.
- **Stopped** – полностью перекрыто другим Activity. Оно больше не видно пользователю и будет уничтожено системой, когда понадобится память.

Если Activity приостановлено или остановлено, система может удалить его из памяти, либо послать запрос на его завершение, или просто уничтожить его процесс. Когда Activity снова отображается пользователю, его состояние полностью восстанавливается.

Переходя от состояния к состоянию, Activity уведомляет об этом, вызывая следующие методы:

- void onCreate()
- void onStart()
- void onRestart()
- void onResume()

- void onPause()
- void onStop()
- void onDestroy()

Жизненный цикл Activity состоит из трёх вложенных циклов (Рис. 1.3):

- Жизненный цикл activity начинается с вызова метода onCreate(), в котором производится первоначальная настройка глобального состояния, и завершается вызовом метода onDestroy(), в котором оно освобождает занятые ресурсы.
- Видимая часть жизненного цикла происходит между вызовами onStart() и onStop(). В течение этого времени пользователь может видеть Activity на экране, хотя оно может быть не на переднем плане и не взаимодействовать с пользователем. Методы onStart() и onStop() могут вызываться столько раз, сколько Activity становится видимым или скрытым для пользователя.
- На переднем плане Activity находится между вызовами onResume() и onPause(). В течение этого времени Activity находится поверх других и взаимодействует с пользователем. Activity может часто переходить в состояние паузы и выходить из него. Например, метод onPause() может быть вызван, когда устройство переходит в спящий режим или когда запускается другое Activity, а метод onResume() – при получении результата от закрывающегося Activity.

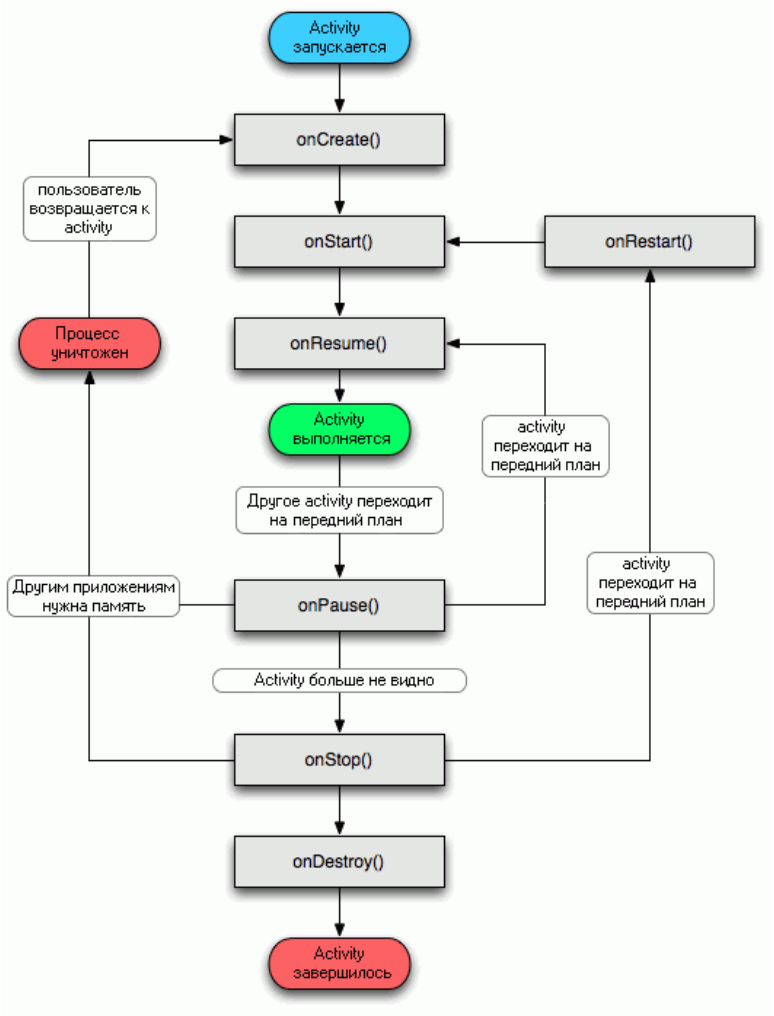


Рис. 1.3 Жизненный цикл Activity.

Следующая таблица более детально описывает каждый из уже перечисленных методов и его место в жизненном цикле Activity.

| Метод | Описание | Может ли быть уничтожен? | Следующий метод |
|---|---|---|---|
| onCreate() | Вызывается один раз, при создании Activity. Здесь должна производится первоначальная настройка – создание экземпляров класса View (пользовательский интерфейс), запись данных в списки и так далее.<br>За ним всегда следует вызов метода onStart(). | нет | onStart() |
| onStart() | Вызывается после того как Activity было остановлено и теперь снова запускается.<br>За ним всегда следует вызов onStart() | нет | onStart() |
| onRestart() | Вызывается перед тем как Activity становится видимым для пользователя.<br>За ним следует вызов onResume(), если Activity переходит на передний план, или onStop(), если оно скрывается. | нет | onResume() или onStop() |
| onResume() | Вызывается перед тем, как Activity начинается взаимодействовать с пользователем. С этого момента Activity находится на вершине стека и принимает весь пользовательский ввод.<br>За ним следует вызов onPause(). | нет | onPause() |
| onPause() | Вызывается системой перед возобновлением другого Activity. Этот метод обычно используется для сохранения изменённых данных, остановки анимации и других вещей, которые могут потреблять процессорное время. Другое Activity не будет возобновлено, пока он не завершится.<br>За ним может быть вызван метод onResume() если Activity возвращается на передний план, или метод onStop(), если оно становится невидимым для пользователя. | да | onResume() или onStop() |
| onStop() | Вызывается когда Activity больше не видно пользователю. Это может происходить потому, что оно уничтожается или потому, что другое Activity было возобновлено и перекрыло его.<br>Затем может быть вызван либо метод onRestart(), если Activity возвращается для взаимодействия с пользователем, либо метод onDestroy(), если оно завершается. | да | onRestart() или onDestroy() |
| onDestroy() | Вызывается перед уничтожением Activity. Это последний вызов, который оно получит. Может быть вызван либо потому, что Activity завершается (кто-то вызвал метод finish()), либо потому, что система временно уничтожает этот экземпляр для экономии памяти. | да | нет |

Обратите внимание на колонку «Может ли быть уничтожен?». Она указывает на то, может ли система уничтожить процесс, в котором запущено Activity, в любой момент после возврата из этого метода, не выполняя больше ни одной строчки кода Activity. Три метода: OnPause(), OnStop() и OnDestroy() – отмечены «Да». Но только метод OnPause() будет гарантированно вызван перед уничтожением процесса, поскольку он первый в этом списке, а OnStop() и OnDestroy() могут не вызываться. Поэтому для сохранения изменённых данных нужно использовать метод OnPause().

Методы, отмеченные «Нет», защищают процесс, в котором запущено Activity, от уничтожения с момента их вызова.

### 1.6.2. Типы процессов в Android-приложении.

Жизненный цикл приложения тесно связан с жизненным циклом его процесса. Также он зависит от текущего состояния системы. В случае нехватки памяти, Android убивает наименее значимые процессы. Значимость процесса зависит от его типа. Типы процессов, в зависимости от важности, выглядят следующим образом (от наиболее до наименее важных):

- **Процесс переднего плана** – процесс приложения, с которым пользователь взаимодействует в данный момент. Процесс считается таковым, если его Activity находится на вершине Activity-стека (была вызвана функция onResume()), или его Broadcast Receiver работает в настоящее время (в данный момент исполняется приложением onReceive()), или же его Service выполняет callback-методы, такие как onCreate(), onStart() или onDestroy(). Как правило, таких процессов очень мало и они закрываются в самую последнюю очередь.
- **Видимый процесс** — процесс, который имеет Activity, видимый конечному пользователю в данный момент времени. Процессов, которые выводятся на экран, очень мало, поэтому их работа прерывается только в крайнем случае, если не хватает ресурсов для активных приложений.
- **Служебный процесс** – процесс, содержащий Service, для которого была вызвана функция startService(), при условии, что данный Service сейчас работает.
- **Процесс заднего фона**. Данный процесс не имеет видимых пользователю Activity (была вызвана функция onStop()). Как правило, существует множество фоновых процессов, работа которых завершается по принципу «последний запущенный закрывается первым», чтобы освободить ресурсы для приложений, работающих на переднем плане.

### 1.6.3. Services.

Service – это некий процесс, который запускается в фоновом режиме. Как пример, Service может получать данные по сети, выполнять какие-либо длительные вычисления. Хорошим примером Service служит проигрыватель музыки. Пользователь может выбрать любую песню в проигрывателе, включить ее и закрыть плеер занявшись чем-нибудь другим. Музыка будет проигрываться в фоновом процессе. Service проигрывания музыки будет работать, даже если Activity плеера закрыта.

Подобно Activity, Service имеет свои методы жизненного цикла:

- void onCreate()
- void onStart(Intent intent)
- void onDestroy()

В полном жизненном цикле Service существует два вложенных цикла:

- полная целая жизнь Service – промежуток между временем вызова метода onCreate() и временем возвращения onDestroy(). Подобно Activity, для Services производят начальную инициализацию в onCreate() и освобождают все остающиеся ресурсы в onDestroy();
- активная целая жизнь Service – начинается с вызова метода onStart(). Этому методу передается объект Intent, который передавался в startService().

Как и Activities, Services запускаются в главном потоке процесса приложения. По этой причине их следует запускать в отдельном потоке, чтобы они не блокировали другие компоненты или пользовательский интерфейс.

### 1.6.4. Broadcast receivers.

Broadcast receiver – это компонент, который ничего не делает, кроме того, что рассылает и реагирует на широковещательные сообщения. Примером широковещательных компонентов могут быть: сообщения об переходе на летнее/зимнее время, сообщения об минимальном заряде батареи и так далее.

Broadcast receiver не отображает пользовательский интерфейс, но может запустить Activity на полученное сообщение или использовать NotificationManager для привлечения внимания пользователя. Привлечь внимание пользователя можно, например, вибрацией устройства, проигрыванием звука или миганием вспышки.

Приемник широковещательных сообщений имеет единственный метод жизненного цикла: onReceive(). Когда широковещательное сообщение прибывает для получателя, Android вызывает его методом onReceive() и передает в него объект Intent, содержащий сообщение. Приемник широковещательных сообщений является активным только во время выполнения этого метода. Процесс, который в настоящее время выполняет Broadcast receiver, является приоритетным процессом и будет сохранен, кроме случаев критического недостатка памяти в системе.

Когда программа возвращается из onReceive(), приемник становится неактивным и система полагает, что работа объекта Broadcast receiver закончена. Процесс с активным широковещательным получателем защищен от уничтожения системой. Однако процесс, содержащий неактивные компоненты, может быть уничтожен системой в любое время, когда память, которую он потребляет, будет необходима другим процессам.

### 1.6.5. Content providers.

Content providers предоставляют доступ к данным (чтение, добавление, обновление). Content provider может предоставлять доступ к данным не только своему приложению, но и другим. Данные могут размещаться в файловой системе, в базе данных.

## 1.7. Виджеты.

Виджет – это объект View, который служит интерфейсом для взаимодействия с пользователем. Иначе, виджеты – это обычные элементы управления: кнопки, текстовые поля, флажки, переключатели, списки.

Стандартные элементы имеют привычные свойства: ширина, высота, цвет и тому подобные. Еще два важных свойства, которые могут влиять на размер и положение дочерних элементов - важность (weight) и выравнивание (gravity). Weight используется для присвоения элементу показателя важности, отличающего его от других элементов, находящихся в контейнере. Предположим, в контейнере находится три элемента управления: первый имеет важность 1 (максимальное возможное значение), а два других имеют значение 0. В этом случае элемент управления, который имеет значение важности 1, займет в контейнере все свободное пространство. Gravity – это ориентация в контейнере. Например, необходимо выровнять текст

надписи по правому краю, тогда свойство gravity будет иметь значение right. Набор значений для gravity ограничен: left, center, right, top, bottom, center_vertical, clip_horizontal и еще некоторые.

### 1.7.1. TextView.

Виджет TextView предназначен для отображения текста без возможности редактирования его пользователем. TextView один из самых используемых виджетов. С его помощью пользователю удобнее ориентироваться в программе. По сути, TextView служит для представления пользователю описательного текста.

Атрибуты TextView:

- **android:textsize** – размер текста. При установке размера текста используются несколько единиц измерения: px (pixels), dp (density-independent pixels), sp (scale-independent pixels), in (inches), pt (points), mm (millimeters). Чтобы текст мог меняться в зависимости от выбора пользователя, используют единицы измерения sp.
- **android:textstyle** – стиль текста. Используются константы: normal, bold, italic.
- **android:textcolor** – цвет текста. Используются четыре формата в шестнадцатеричной кодировке: #RGB; #ARGB; #RRGGBB; #AARRGGBB, где R, G, B – соответствующий цвет, A – прозрачность (alpha-channel). Значение A, установленное в 0, означает прозрачность 100%.

Чтобы оживить текст, можно дополнительно задействовать атрибуты для создания эффектов тени: shadowColor (цвет тени), shadowDx (смещение тени по горизонтали), shadowDy (смещение по вертикали) и shadowRadius (ширина тени). Во время установки значений изменения не видны, необходимо запустить пример в эмуляторе или на устройстве.

Рассмотрим частую ошибку при попытке изменить фон элемента программным способом.

Предположим, в ресурсах определен зелёный цвет:

```
<color name="tvBackground">#337700</color>
```

Следующий код будет ошибочным:

```
tv.setBackgroundColor(R.color.tvBackground);
```

Нужно так (два варианта):

```
tv.setBackgroundResource(R.color.tvBackground);
tv.setBackgroundColor(getResources().getColor(R.color.tvBackground));
```

### 1.7.2. Button.

Кнопка – один из самых распространенных элементов управления в программировании. Наследуется от Textview и является базовым классом для класса CompoundButton. От класса CompoundButton, в свою очередь, наследуются такие элементы как CheckBox, ToggleButton и RadioButton. На кнопке располагается текст и на кнопку нужно нажать, чтобы получить результат.

Если вы разместили на экране кнопку и будете нажимать на неё, то ничего не произойдёт. Необходимо написать код, который будет выполняться при нажатии. Существует несколько способов обработки нажатий на кнопку. Относительно новый и простой для начинающего программиста способ – использовать атрибут onClick.

Иногда нужно сделать кнопку недоступной. Через XML нельзя сделать кнопку недоступной (нет подходящего атрибута). Это можно сделать программно через метод setEnable(). Как альтернативу можно рассмотреть атрибут android:clickable, который позволит кнопке не реагировать на касания, но при этом вид кнопки останется обычным.

### 1.7.3. Другие типы виджетов.

- **CheckBox** является флажком, с помощью которого пользователь может отметить (поставить галочку) определенную опцию. Очень часто флажки используются в настройках, когда нужно выборочно выбрать определенные пункты, необходимые для комфортной работы пользователю.
- **RadioButton.** Главная особенность элемента RadioButton состоит в том, что он не используется в одиночестве. Всегда должно быть два и более переключателя и только один из них может быть выбранным.
- **ToggleButton** по своей функциональности похож на флажок (checkbox) или переключатель (radiobutton). Это кнопка, которая может находиться в одном из двух состояний: активна (On) или неактивна (Off).
- **Switch** – ещё один вид переключателей, представляет собой полоску с двумя состояниями, переключиться между которыми можно сдвиганием ползунка.
- **Spinner** похож на выпадающий список. В закрытом состоянии элемент показывает одну строчку, при раскрытии выводит список в виде диалогового окна с переключателями.
- **ProgressBar** (индикатор прогресса) применяется в тех случаях, когда пользователю нужно показать, что программа не зависла, а выполняет продолжительную работу.
- **SeekBar** – обычный слайдер, чтобы пользователь мог передвигать ползунок пальцем на экране. Также можно передвигать ползунок с помощью клавиш-стрелок.
- **RatingBar** показывает значение рейтинга в виде звездочек. Можно установить рейтинг касанием пальца или с помощью клавиш курсора, используя заранее заданное количество звездочек.
  Для виджета RatingBar используются следующие методы:
  setNumStart(int) – устанавливает число звездочек
  getRating() – возвращает значение рейтинга
  setRating(float) – устанавливает значение рейтинга
  setStepSize(float) – устанавливает шаг приращения рейтинга

### 1.7.4. Адаптеры

В Android часто используются адаптеры. Если говорить в общих чертах, то адаптеры упрощают связывание данных с элементом управления. Адаптеры используются при работе с виджетами: ListView, Spinner и другими.

**ListAdapter** наследует базовый класс Adapter и служит мостом между данными и ListView. Часто данные могут быть представлены курсорами, но необязательно. Удобство в том, что ListView может отображать любые данные, лишь бы они были завернуты в ListAdapter. ListAdapter имеет несколько подклассов (ArrayAdapter, BaseAdapter, CursorAdapter и другие), которые предназначены для различных целей.

- **ArrayAdapter** специально предназначен для работы с элементами списка. Он представляет данные в виде массива и добавляет удобный функционал для работы с ними (добавление, удаление, поиск).
- **SimpleAdapter** – очень простой адаптер, обычно используется для заполнения списка статическими данными (которые могут быть взяты из ресурсов).
- **CursorAdapter** предоставляет данные для списка через курсор.

### 1.8. Эмулятор.

Эмулятор Android – это важный инструмент разработчика. Необходимо изучить его особенности и использовать его на начальном этапе разработки. Однако, следует помнить, что эмулятор лишь моделирует общее поведение реального устройства. Поэтому окончательное тестирование необходимо проводить на настоящем телефоне.

Эмулятор создается при помощи Android Virtual Device Manager (AVD Manager). Создавая новое виртуальное устройство, в окне свойств можно задать произвольное название для эмулятора, указать версию API и установить остальные параметры (наприме, разрешение, плотность пикселей на экране, емкость SD-карты и другие).

С помощью эмулятора можно иметь полноценный доступ к интернету, настраивать скорость и латентность соединения. Также можно имитировать входящие и исходящие телефонные звонки и SMS-сообщения. Но в то же время эмулятор не поддерживает виброзвонок, светодиоды, камеру, акселерометр и работу с компасом.

Теперь, когда известны основные принципы программирования для Android и создан эмулятор, можно приступать к написанию приложений.