**КАЛУЖСКИЙ ФИЛИАЛ**
**ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО БЮДЖЕТНОГО**
**ОБРАЗОВАТЕЛЬНОГО УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ**
**«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**ИМЕНИ Н.Э. БАУМАНА (национальный исследовательский университет)»**

**Факультет** «Информатика и управление"

**Кафедра** "Программное обеспечение ЭВМ, информационные технологии"

# Введение в Android
# Первое Android приложение
# Компоновка элементов управления

**Калуга**

# ANDROID: ОТКРЫТАЯ ПЛАТФОРМА ДЛЯ РАЗРАБОТКИ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

Вот как описывает Android Энди Рубин из Google:

Первая действительно открытая и всеобъемлющая платформа для мобильных устройств и любого программного обеспечения, предназначенного для работы на мобильном телефоне, при этом без патентных ограничений, которые сдерживали развитие портативных устройств.

Упрощенно Android можно представить как комбинацию **трех компонентов**:
- свободной **операционной системы** с открытыми исходными кодами;
- **среды разработки** с открытыми исходными кодами для создания мобильных приложений;
- **устройств**, по большей части мобильных телефонов, на которых установлена операционная система Android вместе с разработанными для нее приложениями.

Главным преимуществом Android как среды разработки стал ее API.

Android как нейтральная к приложениям платформа предоставляет возможность создавать программы, которые станут такой же неотъемлемой частью телефона, как и компоненты, поставляемые в комплекте.

Следующий список иллюстрирует основные **характеристики Android**:
•отсутствие расходов на использование лицензии, распространение и разработку, а также каких-либо механизмов сертификации готовых программных продуктов;
•доступ к Wi-Fi-устройству;
•в сетях GSM, EDGE, 3G и LTE, предназначенных для телефонии и передачи данных, можно звонить или принимать звонки и SMS, отправлять и получать данные;
•комплексный API для работы с навигационными службами, например GPS;
•полный контроль над мультимедийными устройствами, включая проигрывание или запись информации с камеры и микрофона;
•API для работы с сенсорными устройствами, например акселерометром и компасом;
•библиотеки для работы с Bluetooth с возможностью передачи данных по протоколу p2p;

•хранилища для общих данных;
•фоновые приложения и процессы;
•виджеты для Рабочего стола, Живые каталоги (Live Folders) и Живые обои (Live Wallpaper);
•возможность интеграции результатов поиска приложения в системный поиск;
•встроенный браузер на базе WebKit с открытыми исходными кодами и поддержкой HTML5;
•полная поддержка приложений, которые используют функционал работы с картами в своем пользовательском интерфейсе;
•оптимизированная под мобильные устройства графическая система с аппаратным ускорением, включающая библиотеку для работы с векторной 2D-графикой и поддержку трехмерной графики с использованием OpenGL ES 2.0;
•мультимедийные библиотеки для проигрывания и записи аудио-, видеофайлов или изображений;
•локализация с помощью инструментов для работы с динамическими ресурсами;
•набор программных компонентов для повторного использования компонентов и замещения встроенных приложений.

# АЛЬЯНС ОТКРЫТЫХ МОБИЛЬНЫХ УСТРОЙСТВ (OPEN HANDSET ALLIANCE, OHA)

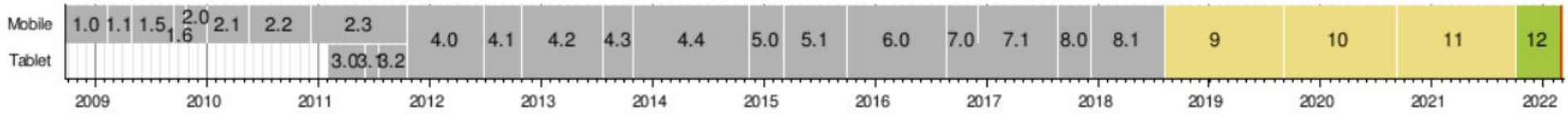
Альянс открытых мобильных устройств — это сообщество из более чем 50 компаний, включающее производителей аппаратного и программного обеспечения, а также мобильных операторов. Среди наиболее значительных членов Альянса можно назвать компании Motorola, HTC, T-Mobile и Qualcomm. Вот как формулируют основные идеи OHA участники этого сообщества:

«Приверженность открытости, общее видение будущего и конкретные задачи для воплощения мечты в реальность. Ускорение внедрения инноваций в сфере мобильных технологий и предоставление потребителям функциональных, менее дорогих и более продвинутых мобильных устройств».

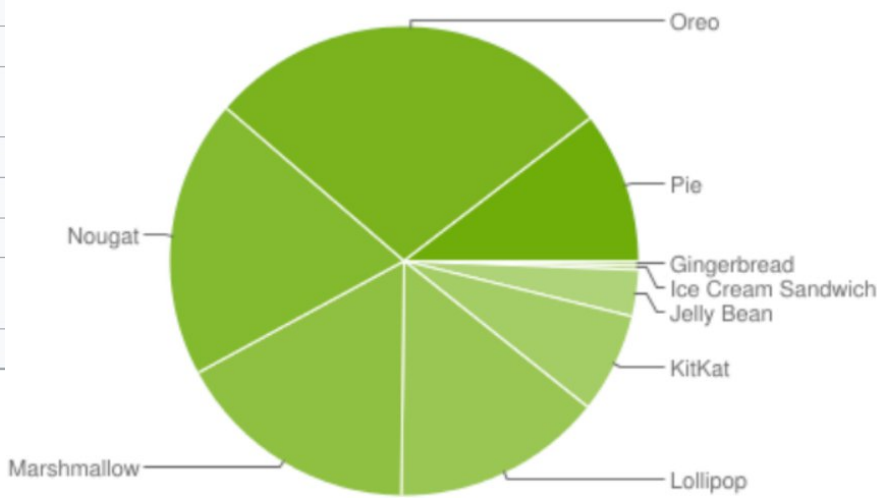
Было основана 5 ноября 2007 года под предводительством Google и 34 прочих членов, включающих производителей мобильных телефонов, разработчиков программного обеспечения, некоторых мобильных поставщиков и изготовителей чипов. Nokia, AT&T и Verizon Wireless не являются членами альянса, однако Verizon недавно выразил желание использовать Android в будущем, указывая на возможность скорого вступления в альянс. Android, основной программный пакет альянса, основан на открытом исходном коде и будет конкурировать с другими мобильными платформами от Apple Inc., Microsoft, Nokia, Palm, Research In Motion и Symbian.

Одновременно с объявлением о формировании Open Handset Alliance **5 ноября 2007**, ОНА также обнародовала информацию об Android, открытой платформе для мобильных телефонов, основанной на ядре Linux. Первая версия SDK для разработчиков была выпущена **12 ноября 2007**.

Первым коммерческим телефон, использующим Android, является T-Mobile G1 (также известный как HTC Dream). Он был одобрен FCC 18 августа 2008 и стал доступен 22 октября.




| Version | Marketing name | Release date | API level | Kernel | Runtime | Launched with |
|---|---|---|---|---|---|---|
| 12 | 12 | October 4, 2021 | 31 | 5.x | ART | Pixel 3, Pixel 3 XL, Pixel 3a, Pixel 3a XL, Pixel 4, Pixel 4 XL, Pixel 5, Pixel 6, Pixel 6 Pro |
| 11 | 11 | September 8, 2020 | 30 | 5.x | ART | Pixel 2, Pixel 2 XL, Pixel 3, Pixel 3 XL, Pixel 3a, Pixel 3a XL, Pixel 4, Pixel 4 XL[425] |
| 10 | 10 | September 3, 2019 | 29 | 5.x | ART | Asus ZenFone 5Z, Essential Phone, Pixel, Pixel XL, Pixel 2, Pixel 2 XL, Pixel 3, Pixel 3 XL, Pixel 3a, Pixel 3a XL, OnePlus 6, OnePlus 6T, OnePlus 7, OnePlus 7 Pro, Oppo Reno, Sony Xperia XZ3, Vivo X27, Vivo NEX S, Vivo NEX A, Xiaomi Mi MIX 3 5G, Xiaomi Mi 9, Tecno Spark 3 Pro, Huawei Mate 20 Pro, LG G8, Nokia 8.1, Realme 3 Pro[426] |
| 9 | Pie | August 6, 2018 | 28 | 4.x | ART | Essential Phone, Pixel, Pixel XL, Pixel 2, Pixel 2 XL, Nokia 7 Plus, OnePlus 6, Oppo R15 Pro, Sony Xperia XZ2, Vivo X21UD, Vivo X21, Xiaomi Mi Mix 2S[427] |
| 8.1 | Oreo | December 5, 2017 | 27 | 4.x | ART | Pixel, Pixel XL, Nexus 6P, Nexus 5X |
| 8.0 | | August 21, 2017 | 26 | 4.x | ART | N/A |
| 7.1 | Nougat | October 4, 2016 | 25 | 4.x | ART | Pixel, Pixel XL |
| 7.0 | | August 22, 2016 | 24 | 4.x | ART | Nexus 5X, Nexus 6P, LG V20 |
| 6.0 | Marshmallow | October 5, 2015 | 23 | 4.x | ART | Nexus 5X, Nexus 6P |
| 5.1 | Lollipop | March 9, 2015 | 22 | 3.x | ART | Android One |
| 5.0 | | November 3, 2014 | 21 | 3.x | ART 2.1.0 | Nexus 6, Nexus 9 |
| 4.4 | KitKat | October 31, 2013 | 19 | 3.x | Dalvik (and ART 1.6.0) | Nexus 5 |
| 4.3 | Jelly Bean | July 24, 2013 | 18 | 3.x | Dalvik | Nexus 7 2013 |
| 4.2 | | November 13, 2012 | 17 | 3.x | Dalvik | Nexus 4, Nexus 10 |
| 4.1 | | July 9, 2012 | 16 | 3.x | Dalvik | Nexus 7 |
| 4.0 | Ice Cream Sandwich | October 19, 2011 | 15 | 3.x | Dalvik | Galaxy Nexus |
| 2.3 | Gingerbread | February 9, 2011 | 10 | 2.6.32 | Dalvik 1.4.0 | Nexus S |



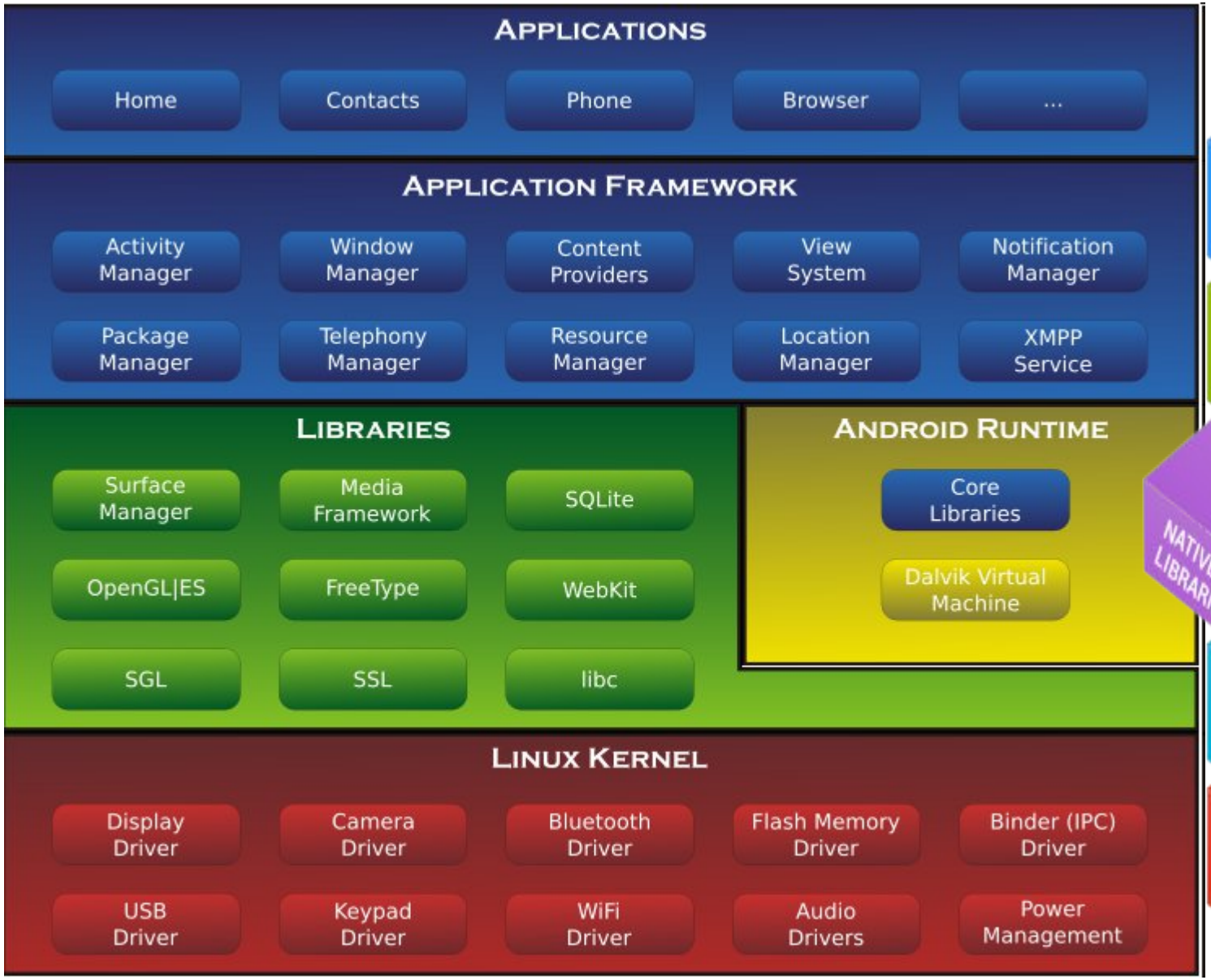
# Архитектура Android

Архитектура Android состоит из **четырех уровней**: уровень ядра, уровень библиотек и среды выполнения, уровень каркаса приложений (application framework) и уровень приложений.

Система Android основана на ядре Linux. Тем не менее Android не является Linux-системой в прямом смысле этого слова. У Android свои механизмы распределения памяти, другая система межпроцессного взаимодействия (Inter-Process Communication, IPC), специфические модули ядра и т. д. На уровне ядра происходит управление аппаратными средствами мобильного устройства. На этом уровне работают драйверы дисплея, камеры, клавиатуры, WiFi, аудиодрайверы. Особое место занимают драйверы управления питанием и драйвер межпроцессного взаимодействия (IPC). Linux обеспечивает уровень абстракций между оборудованием и остальными частями стека Android. С точки зрения внутренней архитектуры Android использует Linux для управления памятью, процессами, сетевым взаимодействием и другими возможностями операционной системы.

На этом уровне также расположен набор драйверов для обеспечения работы с оборудованием мобильного устройства. Набор драйверов может отличаться в зависимости от производителя и модели устройства. Поскольку новое вспомогательное оборудование для мобильных устройств постоянно появляется на рынке, драйверы для них должны быть написаны на уровне ядра Linux для обеспечения поддержки оборудования, так же как и для настольных Linux-систем.

# Архитектура Android

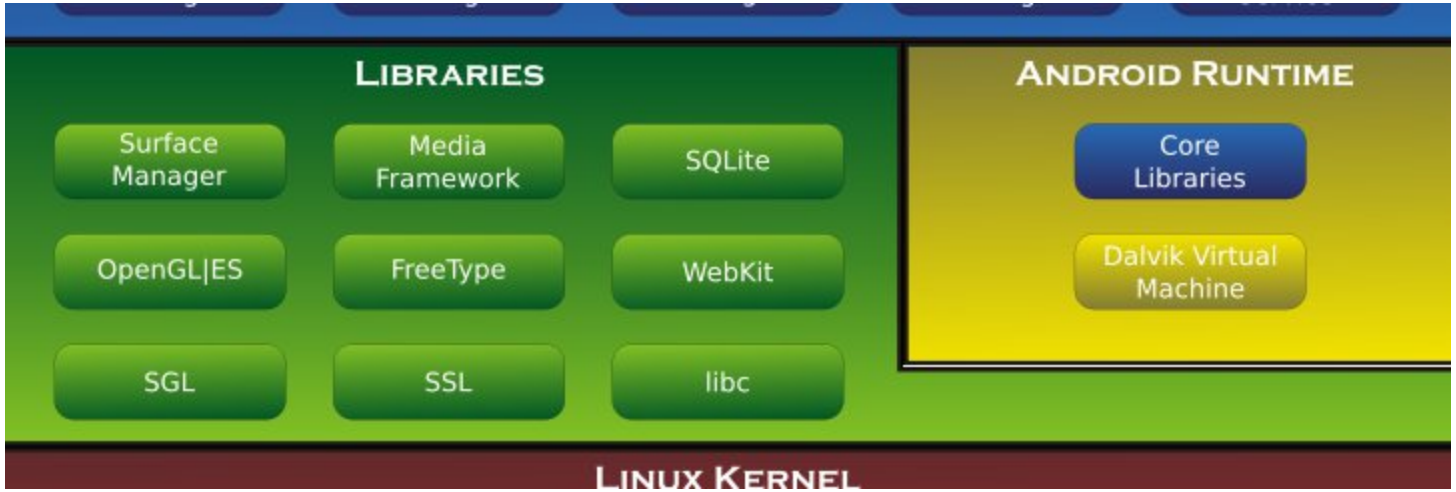



Уровень, следующий за уровнем ядра, содержит исходные библиотеки Android. Эти разделяемые библиотеки написаны на C или на C++, скомпилированы для конкретной аппаратной архитектуры и предустановлены на устройство разработчиком телефона.
Рассмотрим некоторые наиболее важные **исходные библиотеки**:

***Менеджер поверхностей.*** Android использует композитный менеджер окон, похожий на Vista или Compiz, но более простой. Вместо того чтобы выводить графические данные непосредственно в буфер экрана, команды отображения графики формируют закадровые битовые массивы, которые затем объединяются с другими массивами для того, чтобы сформировать изображение, которое видит пользователь. Это позволяет системе создавать различные интересные эффекты, например полупрозрачные окна и градиентные переходы.
***2D и 3D графика.*** В Android двух- и трехмерные графические элементы комбинируются в единый пользовательский интерфейс. Библиотека будет использовать возможности аппаратного 3D-ускорения, если устройство ими оснащено, или быстрый программный рендеринг, если нет.
***Медиа-кодеки.*** Android может проигрывать видеоролики и фильмы, записывать и воспроизводить аудио фрагменты в различных форматах, в том числе AAC, AVC (H.264), H.263, MP3 и MPEG4.

***База данных SQL.*** Android имеет «легковесную» встраиваемую реляционную базу данных SQLite, эта же база данных используется в Firefox и в Apple iPhone. Используйте этот механизм для постоянного хранения данных ваших приложений.
***Браузер.*** Для быстрого отображения HTML-контента Android использует библиотеку WebKit. Тот же механизм используется в браузере Google Chrome, браузере Apple Safari, в Apple iPhone и платформе Nokia S60.

Эти библиотеки не являются отдельными приложениями. Они существуют только для того, чтобы их могли вызывать высокоуровневые программы. Начиная с версии 1.5 Android позволяет писать и внедрять свои собственные библиотеки, используя Native Development Toolkit (NTK).



Следующий уровень — **уровень каркаса приложений.** Уровень каркаса приложений находится над библиотеками и средой выполнения.
Он обеспечивает высокоуровневые строительные блоки, которые можно использовать для создания приложений. Фреймворк поставляется предустановленным вместе с Android, но при необходимости можно расширять его с помощью собственных компонентов. На этом уровне работают **различные диспетчеры**:
Диспетчер активности (Activity Manager) — управляет жизненным циклом приложения;
Диспетчер пакетов (Package Manager) — управляет установкой пакетов прикладных программ;
Диспетчер окон (Window Manager) — управляет окнами приложений;
Диспетчер ресурсов (Resource Manager) — используется для доступа к строковым, графическим и другим типам ресурсов;
Контент-провайдеры (Content Providers) — службы, предоставляющие приложениям доступ к данным других приложений;
Диспетчер телефонии (Telephony Manager) — предоставляет API, с помощью которого можно контролировать основную телефонную информацию — статус подключения, тип сети и т. д.;
Диспетчер местоположения (Location Manager) — позволяет приложениям получать информацию о текущем местоположении устройства;
Диспетчер уведомлений (Notification Manager) — позволяет приложению отображать уведомления в строке состояния;

# СОСТАВ ANDROID SDK

Среда разработки приложений для Android включает все необходимое для создания, тестирования и отладки программ.

- **API-ПЛАТФОРМЫ ANDROID.** Ядро среды разработки — библиотеки API, которые обеспечивают программисту доступ к стеку платформы Android. Эти же самые библиотеки используются компанией Google для написания встроенных в Android приложений.
- **ИНСТРУМЕНТЫ РАЗРАБОТКИ.** Вы можете преобразовать исходный код в исполняемые приложения для Android. В состав среды разработки входят инструменты, которые позволяют компилировать и отлаживать приложения.
- **МЕНЕДЖЕР ВИРТУАЛЬНЫХ УСТРОЙСТВ И ЭМУЛЯТОР.** Эмулятор Android — это полностью интерактивный эмулятор устройств, включающий несколько альтернативных вариантов интерфейса. Эмулятор запускается внутри виртуального устройства Android, которое моделирует аппаратную конфигурацию определенной модели телефона. С помощью эмулятора вы можете увидеть, как приложения будут выглядеть и функционировать на реальном устройстве под управлением Android.

• **ПОЛНЫЙ НАБОР ДОКУМЕНТАЦИИ.** Среда разработки включает расширенную справочную информацию с примерами кода, в которой описывается, что входит в каждый программный пакет и класс и как их можно использовать. В дополнение к этому в справочной документации содержится стартовый курс для начинающих, а также подробное описание основ разработки программ для Android.

• **ПРИМЕРЫ КОДА.** В среде разработки вы найдете примеры программ, которые демонстрируют некоторые возможности Android, а также несколько простых приложений, посвященных определенным функциям API.

• **ОНЛАЙН-ПОДДЕРЖКА.** Группы Google по адресу http://developer.android.com/resources/community-groups.html — это форумы разработчиков, на которых часто появляются сообщения от команд инженеров и специалистов компании Google. Ресурс StackOverflow по адресу http://www.stackoverflow.com/questions/tagged/android также стал популярным — здесь публикуются вопросы, посвященные Android.

# СОСТОЯНИЯ АКТИВНОСТЕЙ

Создавая объекты Activity, система помещает их в стек. При уничтожении эти объекты оттуда убираются, проходя через **четыре возможных состояния**.

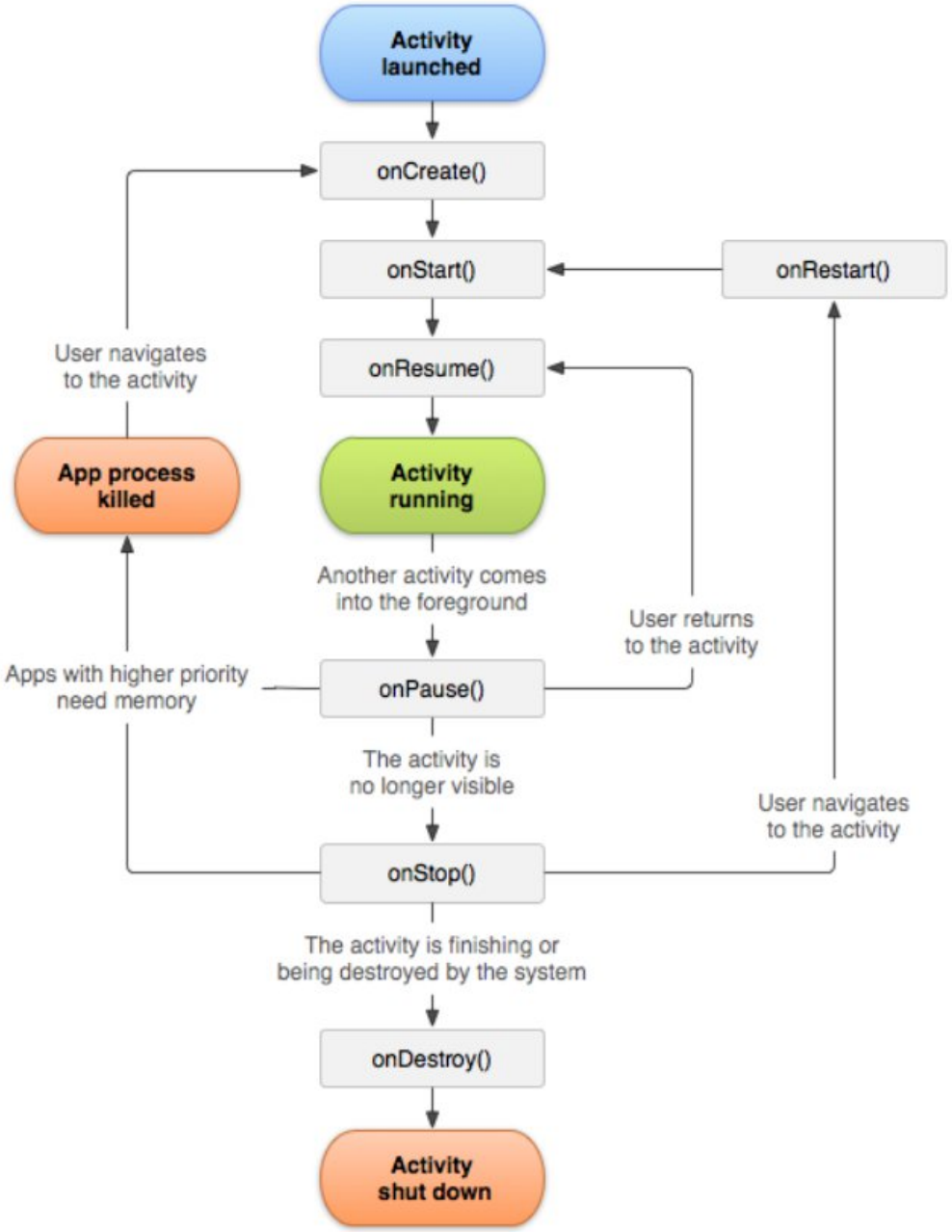
**АКТИВНОЕ.** Когда Активность на вершине стека, **она видна** и выходит на передний план, имея возможность принимать пользовательский ввод. Android будет пытаться любой ценой сохранить ее в рабочем состоянии, при необходимости прерывая работу других Активностей, которые находятся на более низких позициях в стеке, обеспечивая для нее все необходимые ресурсы. Ее работа будет приостановлена, если на передний план выйдет другая Активность.

**ПРИОСТАНОВЛЕННОЕ.** В некоторых ситуациях ваша Активность будет **видна на экране, но не сможет принимать пользовательский ввод:** в этот момент она приостановлена. Такое состояние наступает, когда полупрозрачные или плавающие диалоговые окна становятся активными и частично ее перекрывают. В приостановленном виде Активность рассматривается как **полноценно работающая**, однако она не может взаимодействовать с пользователем. Ее работа может преждевременно завершиться, если системе нужно выделить ресурсы для той Активности, что на переднем плане. При полном исчезновении с экрана Активность останавливается.

**ОСТАНОВЛЕННОЕ.** Когда Активность **перестает быть видимой**, она останавливается и остается в памяти, сохраняя всю информацию о своем состоянии. То есть становится кандидатом на преждевременное закрытие, если системе понадобится память для чего-нибудь другого. **При остановке Активности важно сохранить данные и текущее состояние пользовательского интерфейса.** Как только объект Activity завершает свою работу или закрывается, он становится неактивным.

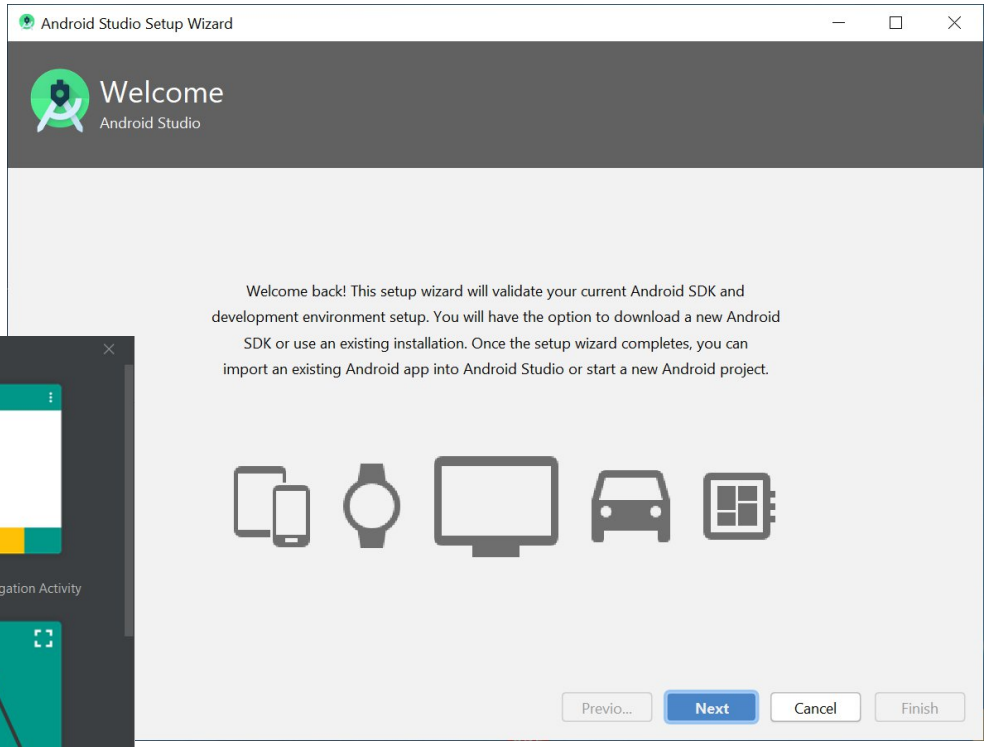
**НЕАКТИВНОЕ.** Это состояние наступает после того, как работа объекта Activity была завершена, и перед тем, как он будет запущен снова. Такая Активность удаляется из стека и должна запускаться повторно, чтобы ее можно было вывести на экран и снова использовать.

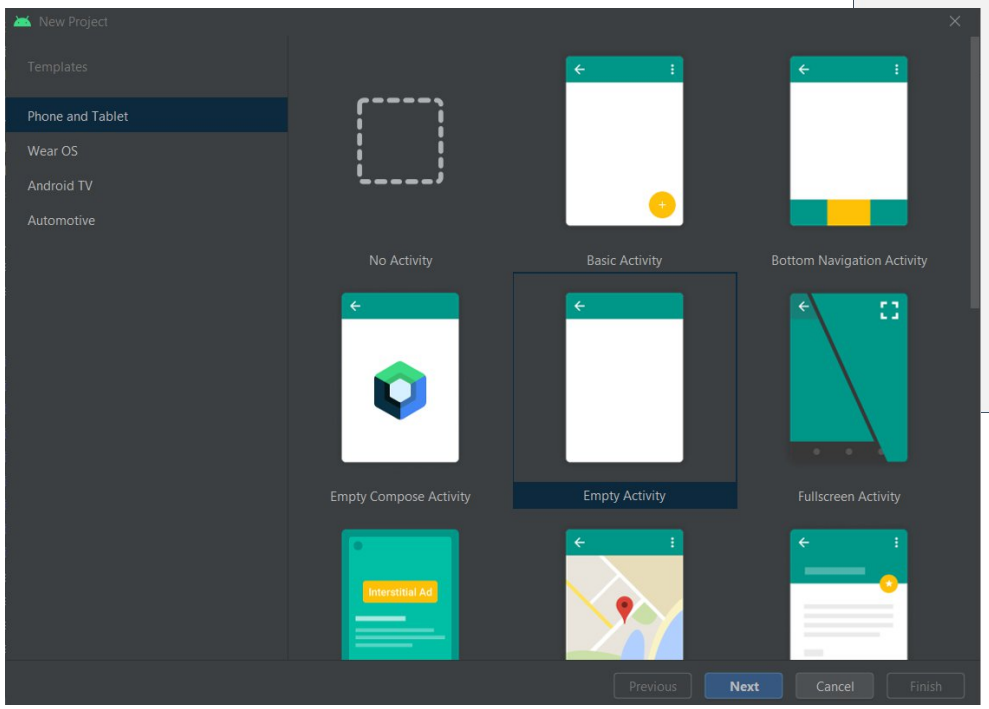
Смена состояний — недетерминированный процесс и полностью **управляется системным диспетчером памяти.** Сперва Android закроет приложения, содержащие объекты Activity в неактивном состоянии, потом перейдет к тем, чьи Активности остановлены или приостановлены (только в крайних случаях).

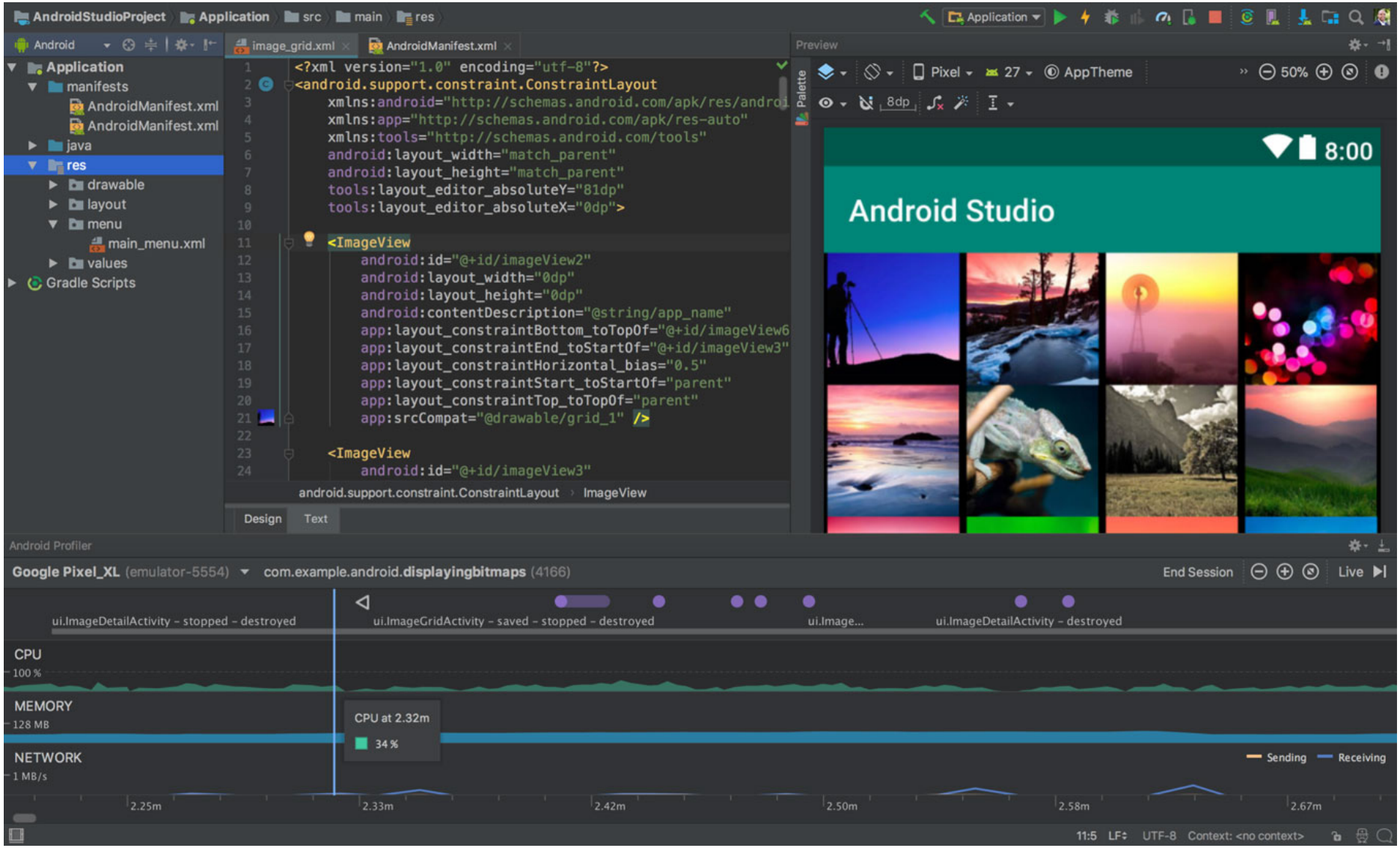

Activity launched
onCreate()
onStart()
onRestart()
onResume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

# Android Studio

android studio

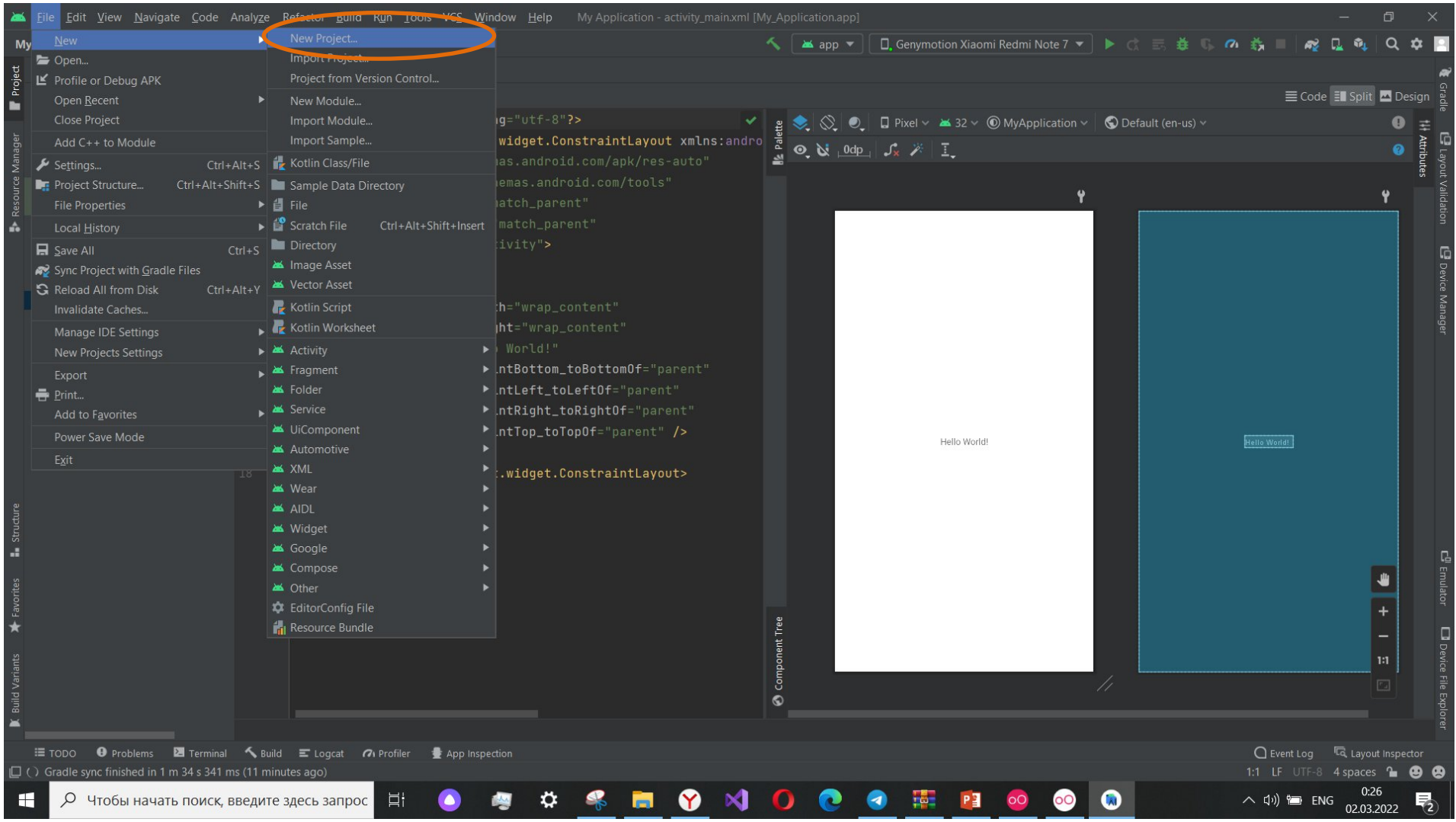


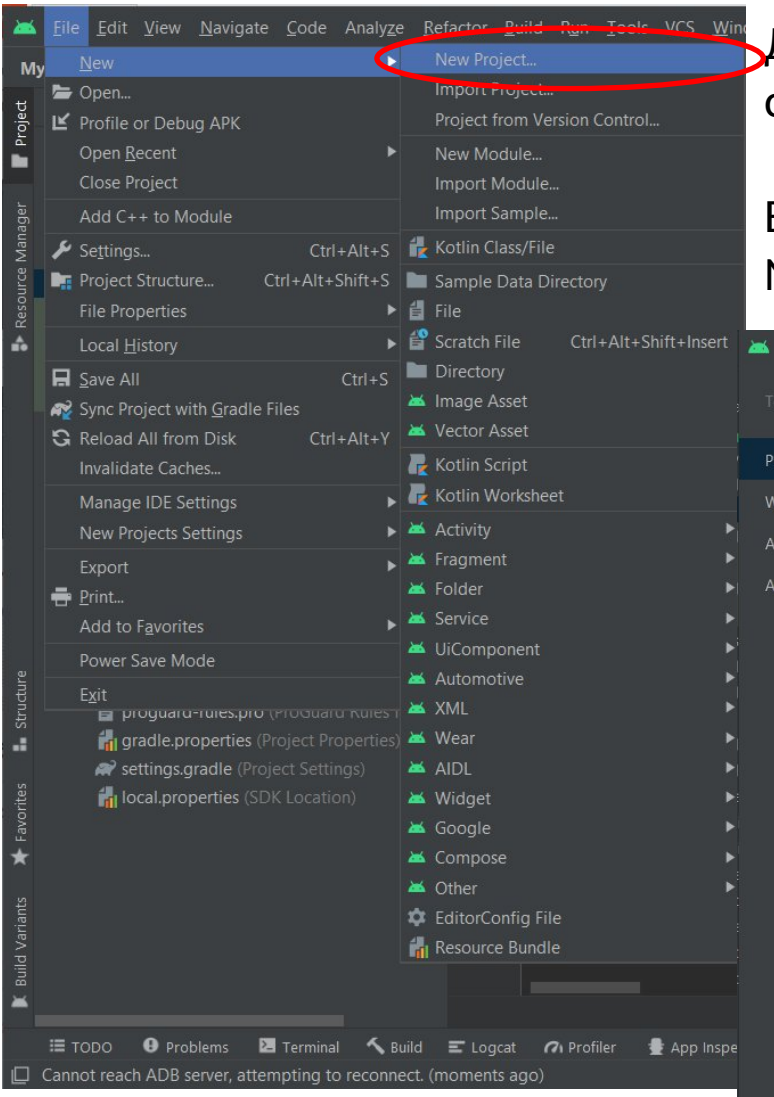

AndroidStudioProject
Application
src
main
res
Application
Android
image_grid.xml
AndroidManifest.xml
Preview
Application
manifests
AndroidManifest.xml
AndroidManifest.xml
java
res
drawable
layout
menu
main_menu.xml
values
Gradle Scripts
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/androi
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:layout_editor_absoluteY="81dp"
tools:layout_editor_absoluteX="0dp">
<ImageView
android:id="@+id/imageView2"
android:layout_width="0dp"
android:layout_height="0dp"
android:contentDescription="@string/app_name"
app:layout_constraintBottom_toTopOf="@+id/imageView6
app:layout_constraintEnd_toStartOf="@+id/imageView3"
app:layout_constraintHorizontal_bias="0.5"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:srcCompat="@drawable/grid_1" />
<ImageView
android:id="@+id/imageView3"
android.support.constraint.ConstraintLayout
ImageView
Design
Text
Palette
Pixel
27
AppTheme
50%
8dp
8:00
Android Studio
Android Profiler
Google Pixel_XL (emulator-5554)
com.example.android.displayingbitmaps (4166)
End Session
Live
ui.ImageDetailActivity - stopped - destroyed
ui.ImageGridActivity - saved - stopped - destroyed
ui.Image...
ui.ImageDetailActivity - destroyed
CPU
100 %
MEMORY
128 MB
CPU at 2.32m
34 %
NETWORK
1 MB/s
Sending
Receiving
2.25m
2.33m
2.42m
2.50m
2.58m
2.67m
11:5
LF
UTF-8
Context: <no context>

Febr 2020

# Создание проекта

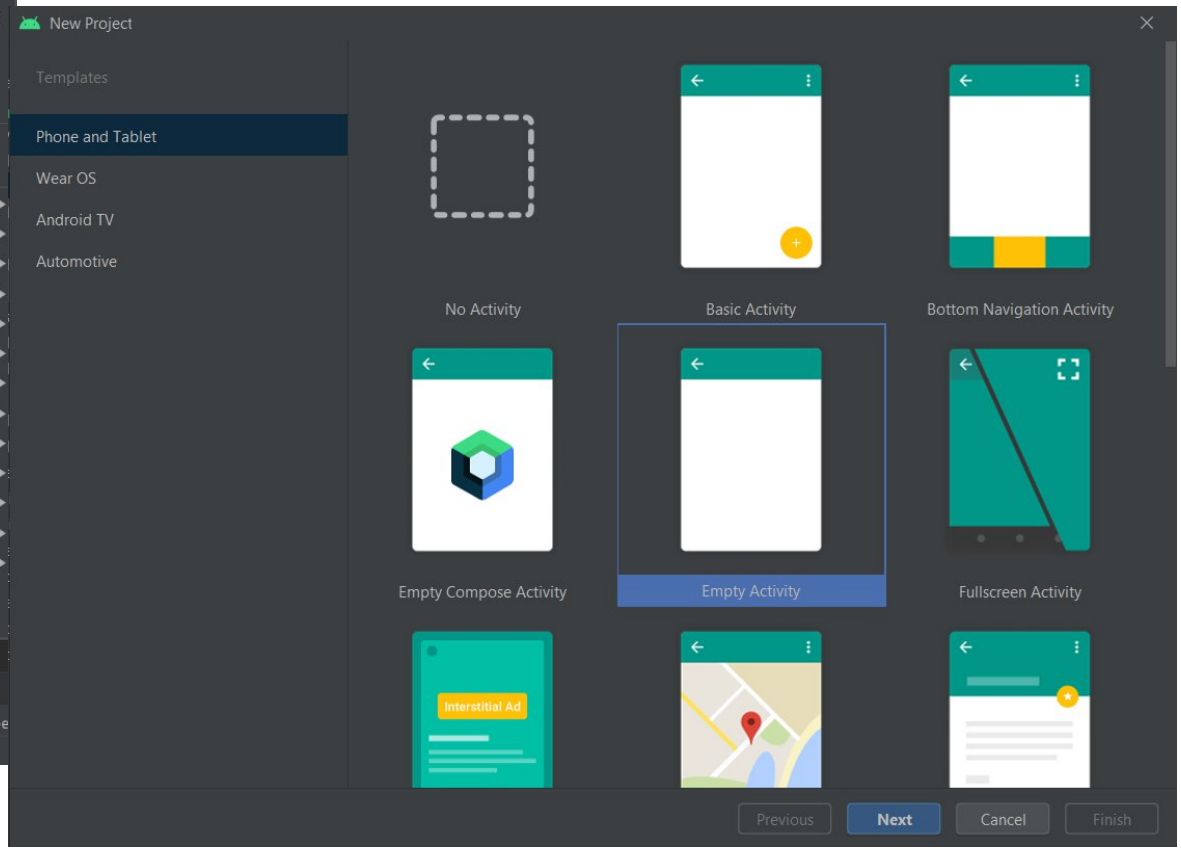


# Создание проекта на Kotlin

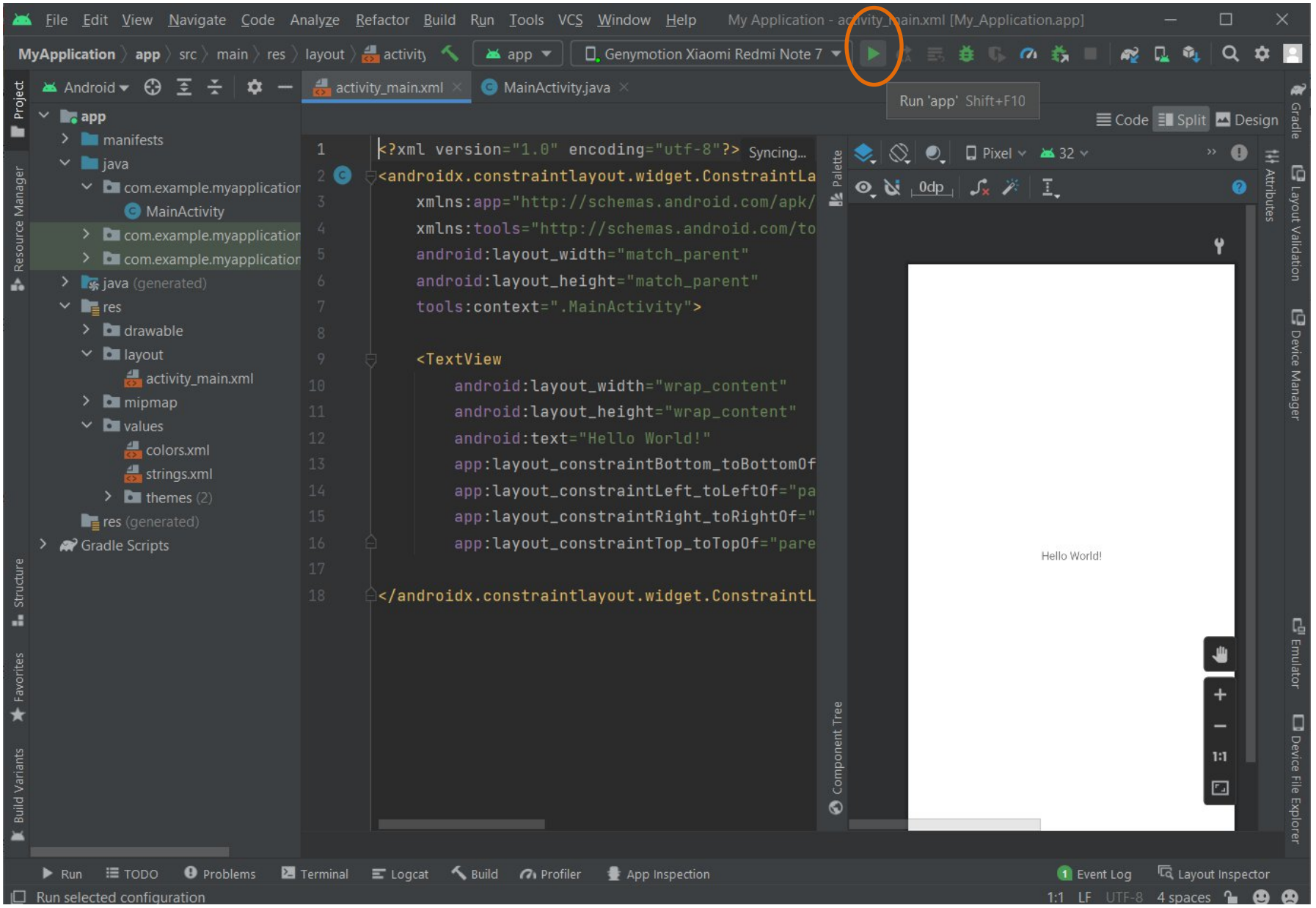



Для создания проекта на языке Kotlin аналогично сначала нажимаем File->New->New Project

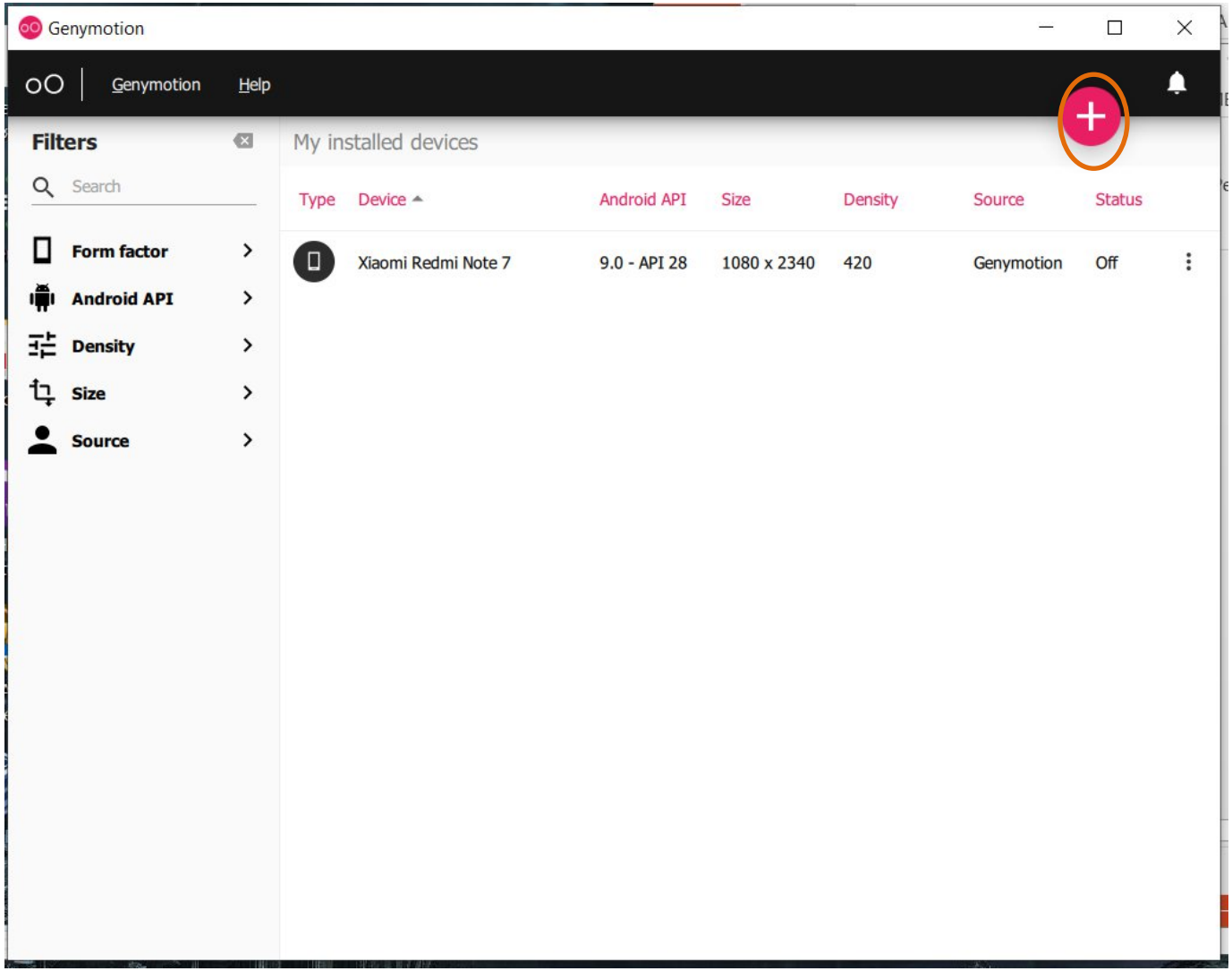
В появившемся окне выбираем Empty Activity и жмём Next. Выбираем язык Kotlin и версию SDK.



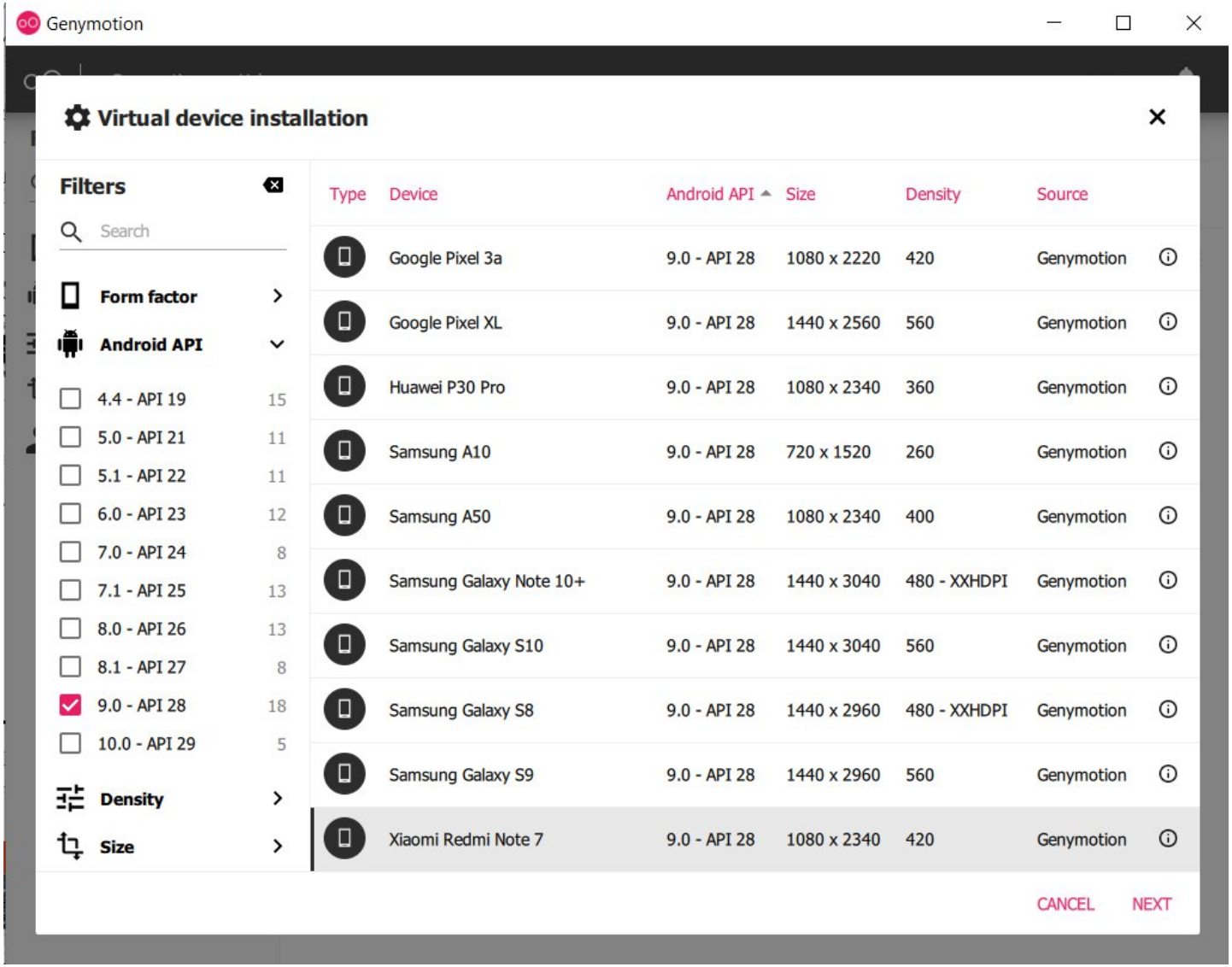
# Запуск Activity



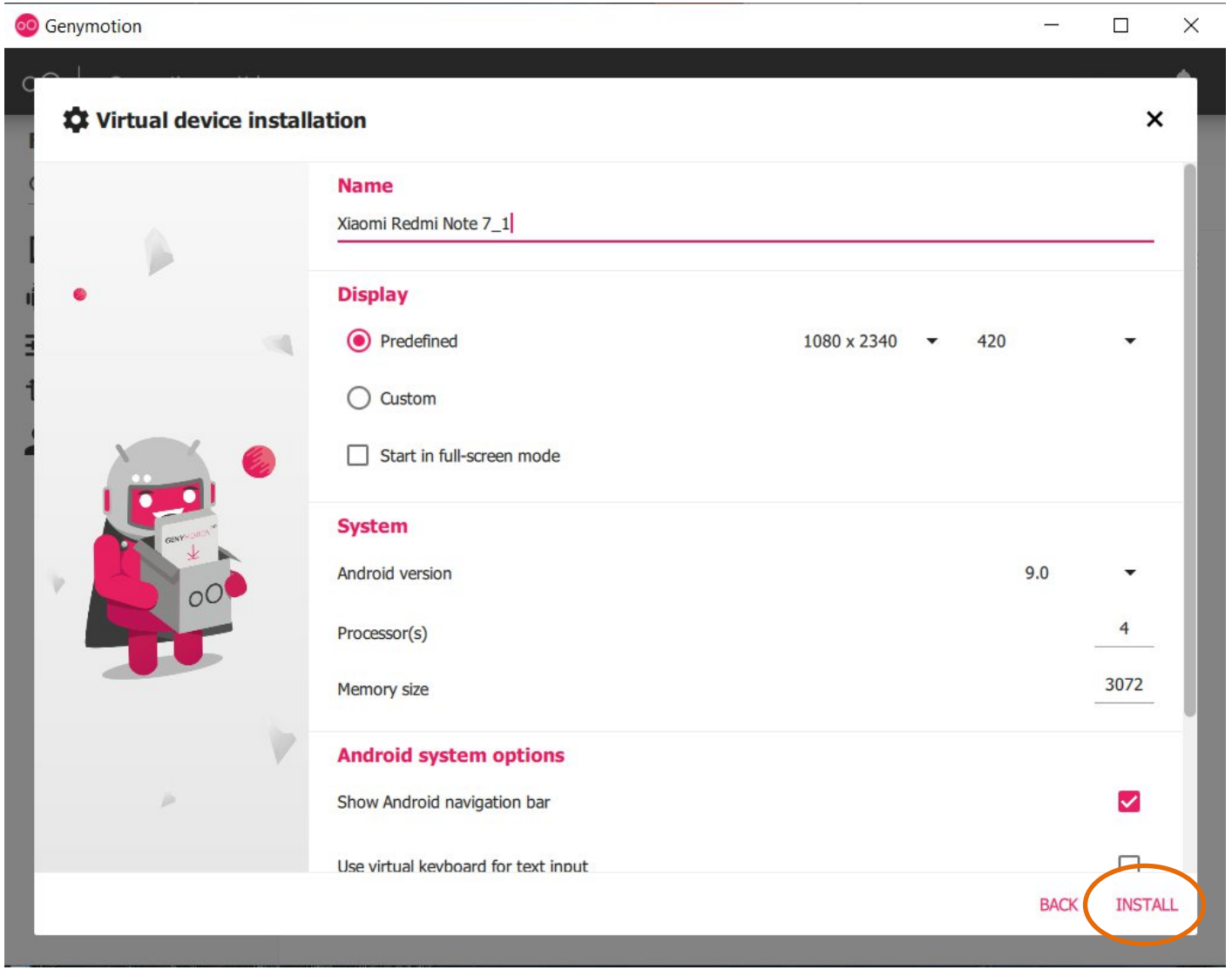
# Создание устройства с использованием Genymotion



# Создание устройства с использованием Genymotion



# Создание устройства с использованием Genymotion

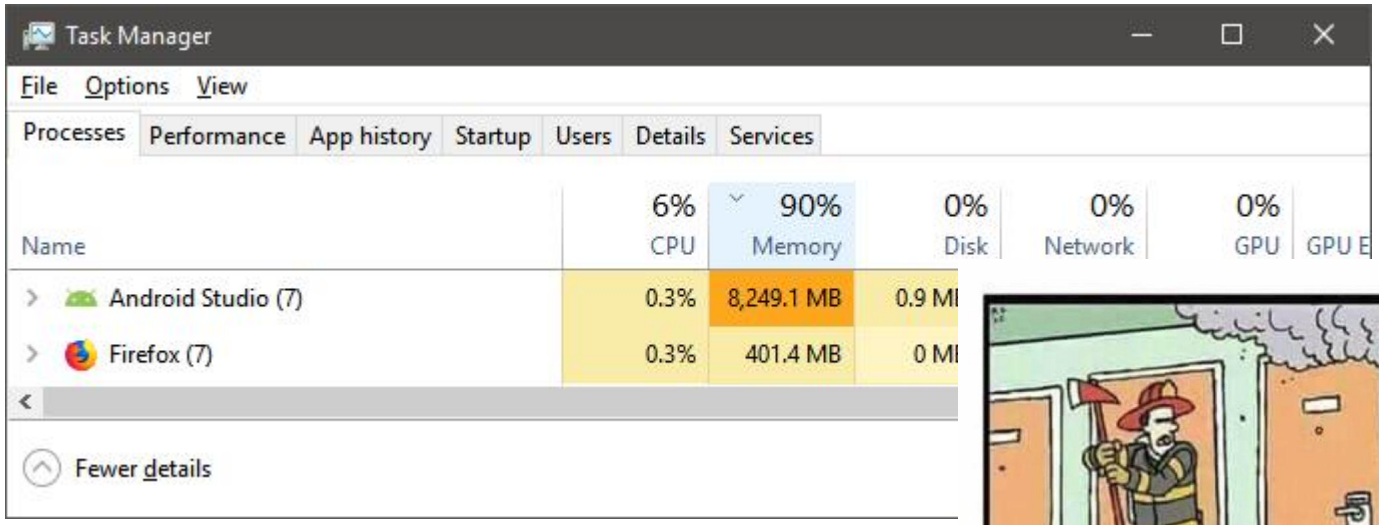


# Запуск AVD


Xiaomi Redmi Note 7 (1080x2340, 420dpi)...
3:04
Google
Email
Gallery
Free for personal use


Xiaomi Redmi Note 7 (1080x2340, 420dpi)...
3:12
My Application
Hello World!
Free for personal use

# Загрузка ПК


Task Manager
File Options View
Processes Performance App history Startup Users Details Services
6% CPU
90% Memory
0% Disk
0% Network
0% GPU
GPU E
Name
Android Studio (7)
0.3%
8,249.1 MB
0.9 MB
Firefox (7)
0.3%
401.4 MB
0 MB
Fewer details


Just
Running
Android studio

# *Компоненты Android-приложения*

У Android-приложения нет единственной точки входа. Android-приложение состоит из компонентов, которые система может запускать, когда ей это необходимо. Одно приложение может также вызывать компоненты другого приложения, если, конечно, приложение разрешило использование своих компонентов.

Всего существует **четыре типа** компонентов:

Activity — деятельность;
Service — служба;
Content Provider — контент-провайдер;
Broadcast Receiver — приемник широковещательных намерений.

Первый компонент — **Activity**. Деятельность представляет собой **визуальный интерфейс пользователя**, т.е., попросту говоря, окно. Окно обычно заполняет весь экран мобильного устройства, но может быть меньших размеров, чем экран устройства. Если у приложения **несколько окон**, то у него будет **несколько деятельностей**. Каждая деятельность независима от других, при открытии новой деятельности (нового окна) работа предыдущей деятельности приостанавливается. Каждый экран приложения будет наследником класса Activity. **Активности используют Представления для формирования графического пользовательского интерфейса**, отображающего информацию и взаимодействующего с пользователем. С точки зрения разработки под настольные платформы Активность — эквивалент Формы (Form).

У следующего компонента приложения — **службы (Service)** — нет графического интерфейса пользователя. Служба выполняется **в фоновом режиме** до тех пор, пока не завершит свою работу.
Другие **приложения могут подключаться к службе**. После подключения к службе вы можете **использовать функции, предоставляемые службой**. Приложение может также запустить или остановить службу. Сервисные компоненты работают в фоновом режиме, запуская уведомления, обновляя Источники данных и видимые Активности. **Используются для** регулярных операций, которые должны продолжаться даже тогда, когда Активности вашего приложения не на переднем плане.

**ИСТОЧНИКИ ДАННЫХ.** Хранилища информации. Данные компоненты нужны для **управления базами данных** в пределах одного приложения и предоставления к ним доступа извне. Источники данных задействуются при **обмене информацией** между разными программами. Это значит, что вы можете настраивать собственные объекты ContentProvider, **открывая к ним доступ** из других приложений, а также **использовать чужие источники**, чтобы работать с данными, которые открыли для вас внешние программы. Устройства под управлением Android содержат несколько стандартных Источников, которые предоставляют доступ к полезным базам данных, включая хранилища мультимедийных файлов и контактной информации. Используется для предоставления доступа к данным программы другим приложениям. Данные программы могут храниться как в файловой системе, так и в базе данных SQLite (могут быть и другие способы хранения данных).

**НАМЕРЕНИЯ. Система передачи сообщений между приложениями**. Используя Намерения, вы можете транслировать сообщения на системном уровне или для конкретных Активностей или Сервисов. Тем самым диктуется необходимость выполнения заданных действий. После этого Android сам определит компоненты, которые должны обработать поступивший запрос.

Для получения **информации о внешних событиях и реакции на них** используется компонент **Broadcast Receiver**. **Источником событий могут быть службы и другие приложения.** Приложение может реагировать на те любые события, которые посчитает важным.

У приемников широковещательных намерений нет пользовательского интерфейса, но в ответ на событие они могут запустить деятельность, т.е. отобразить окно. Если вы создадите и зарегистрируете объект BroadcastReceiver, ваше приложение сможет отслеживать трансляцию Намерений, которые соответствуют заданным критериям. Широковещательные приемники автоматически запустят программу, чтобы она могла ответить на принятое Намерение. Благодаря этому данный механизм идеально подходит для создания приложений, использующих событийную модель.

**ВИДЖЕТЫ.** Визуальные программные компоненты, которые можно добавлять на домашний экран. Этот особый вид Широковещательных приемников позволяет создавать динамические, интерактивные компоненты, которые пользователи могут встраивать в свои домашние экраны.

**УВЕДОМЛЕНИЯ.** Система пользовательских уведомлений. Позволяет сигнализировать о чем-либо, не обращая на себя внимание или не прерывая работу текущей Активности. Механизм уведомлений лучше всего подходит для Сервисов и Широковещательных приемников, когда необходимо привлечь внимание пользователя. Например, принимая текстовое сообщение или входящий звонок, устройство оповещает вас, мигая светодиодами, воспроизводя звуки, отображая значки или показывая сообщения. Вы можете инициировать все эти события из собственного приложения, используя уведомления.

# *Файл AndroidManifest.xml*

Перед запуском компонента операционная система должна убедиться, что компонент существует. Для описания компонентов используется файл AndroidManifest.xml. У каждого Android-приложения есть свой файл манифеста — даже у самых простых приложений.

**В файле манифеста указывается** имя Java-пакета приложения (имя пакета используется как уникальный идентификатор приложения), разрешения, которыми должно обладать приложение для обращения к защищенным частям API (нужно для взаимодействия с другими приложениями), библиотеки, необходимые для выполнения этого приложения и т.д.

Но самое главное — это **описание компонентов приложения**, а именно деятельностей, служб, контент-провайдеров, намерений и т.д. Объявления в файле манифеста позволяют операционной системе узнать, чем является тот или иной компонент и при каких условиях он должен быть запущен.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapplication">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.MyApplication">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

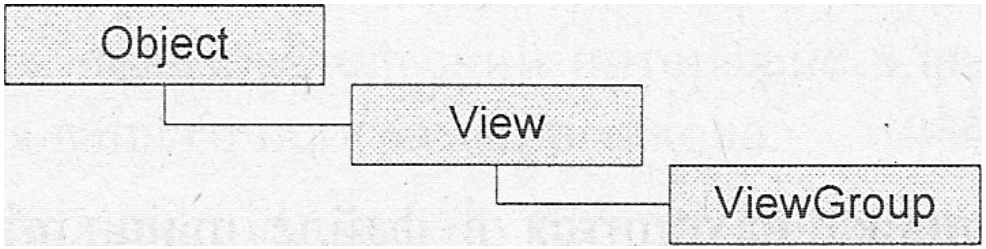
Компоновка— это **архитектура расположения элементов интерфейса** пользователя для конкретного окна, представляющего Activity. Компоновка определяет структуру расположения элементов в окне и содержит все элементы, которые предоставляются пользователю программы.

Проектирование пользовательского интерфейса для мобильных телефонов **сложнее**, чем для настольных систем или для Web-страниц. Экраны мобильных телефонов имеют гораздо меньшее разрешение, чем обычные мониторы. Кроме того, существует **много разновидностей дисплеев** для мобильных телефонов, отличающихся размерами, разрешением и плотностью пикселов.

Необходимо также учесть, что большинство экранов для мобильных телефонов **сенсорные**, причем могут быть разного типа. Например, емкостный экран реагирует на касание пальцем, а для взаимодействия с резистивным экраном используется стилус. Поэтому важно правильно задавать компоновку и размеры элементов управления, чтобы пользователю было удобно управлять вашим приложением независимо от типа экрана.

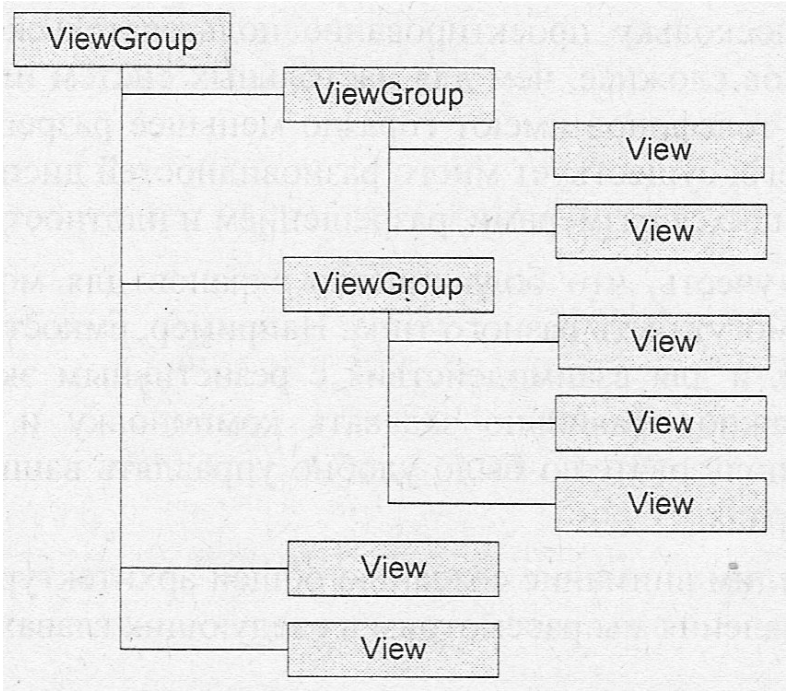
Разметка интерфейса пользователя — это процесс размещения элементов интерфейса (виджетов) для конкретного окна приложения (для конкретной деятельности). Разметку можно выполнить **двумя способами**. Первый заключается в редактировании XML-файла окна приложения. Для главного окна приложения этот файл называется **res/layout/main.xml**. Начинающим программистам больше нравится второй способ — **графический**. Плагин ADT предлагает очень удобный редактор разметки интерфейса пользователя

В Android-приложении графический интерфейс пользователя формируется с использованием объектов **view** и **viewGroup**. Класс view является базовым классом для viewGroup и состоит из коллекции объектов view. Есть множество типов view и viewGroup, каждый из которых является потомком класса view.




**Объекты view** — это основные модули для создания графического интерфейса пользователя на платформе Android. Класс view служит базовым для классов элементов управления, называемых виджетами, — текстовых полей, кнопок и т.д.
**Объект view** — структура, свойства которой сохраняют параметры компоновки и содержание для определенной прямоугольной области экрана. Как объект в интерфейсе пользователя объект view является точкой взаимодействия пользователя и программы.
Класс **viewGroup** представляет собой контейнер, который служит ядром для подклассов, называемых ***компоновками*** (layouts). Эти классы формируют расположение элементов пользовательского интерфейса на форме и содержат дочерние элементы View или ViewGroup.

На платформе Android необходимо определить **пользовательский интерфейс для каждого Activity**, используя иерархии узлов view и viewGroup, как показано на рис. Это **дерево иерархии** может быть и простым, и сложным — в зависимости от требований к графическому интерфейсу приложения.




Каждый элемент файла разметки является объектом класса View или ViewGroup. Если представить все элементы интерфейса пользователя в виде иерархии, то объекты класса ViewGroup будут ветвями дерева, а объекты класса View — листьями. Иерархия созданного интерфейса отображается на вкладке **Outline редактора интерфейса.**

**При запуске программы** система Android получает ссылку на корневой узел дерева и использует ее для прорисовки графического интерфейса на экране мобильного устройства. Система также анализирует элементы дерева от вершины дерева иерархии, прорисовывая дочерние объекты view и viewGroup и добавляя их родительским элементам. Для этого в методе **onCreate()** необходимо вызвать метод **SetContentView()**, передав ему в качестве параметра ссылку на ресурс компоновки в следующем виде:

**R.layout.layout_file_name**

Например, если компоновка находится в файле **activity_main.xml**, ее загрузка в методе onCreate() происходит так, как представлено в листинге.

```
//Java
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}
//Kotlin
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
}
```

Прорисовка начинается с корневого узла дерева компоновки. Затем последовательно прорисовываются дочерние объекты дерева компоновки. Это означает, что родители будут прорисовываться раньше, чем их дочерние объекты, — т.е. по окончании процесса прорисовки родители будут находиться на заднем плане по отношению к дочерним узлам.

# Создание компоновки

Компоновку можно объявлять **двумя способами**:

объявить элементы пользовательского интерфейса в XML-файле. Android обеспечивает прямой **XML-словарь**, который соответствует классам view и viewGroup;

создать компоновку для окна **в коде программы** во время выполнения — инициализировать объекты Layout и дочерние объекты View, ViewGroup и управлять их свойствами программно.

При создании пользовательского интерфейса можно использовать каждый из этих методов в отдельности или оба сразу для объявления и управления пользовательским интерфейсом в приложении. Например, можно объявить заданную по умолчанию компоновку окна вашего приложения в XML-файле, включая экранные элементы, которые появятся в них, и их свойства, а затем добавить код в приложение, который во время выполнения изменит состояние объектов на экране, включая объявленные в XML-файле.

# XML-файл компоновки

Самый общий способ определять компоновку и создавать иерархию элементов интерфейса — в **XML-файле компоновки**. XML предлагает удобную структуру для компоновки, похожую на HTML-компоновку Web-страницы.

Преимущество объявления пользовательского интерфейса в XML-файле состоит в том, что это дает возможность **отделить дизайн приложения от программного кода**, который управляет поведением приложения. Ваше описание пользовательского интерфейса является внешним по отношению к программному коду, это означает, что вы можете изменять пользовательский интерфейс в файле компоновки без необходимости изменения вашего программного кода.

Используя **XML-словарь Android**, можно быстро проектировать пользовательский интерфейс компоновки и экранные элементы, которые он содержит, тем же самым способом, которым вы создаете Web-страницы в HTML — с рядом вложенных элементов.

**Каждый файл** компоновки должен содержать **только один корневой элемент**, который должен быть объектом view или viewGroup. Как только вы определили корневой элемент, вы можете **добавить дополнительные объекты компоновки** или виджеты как дочерние элементы, чтобы постепенно формировать иерархию элементов, которую определяет создаваемая компоновка.

# XML-файл компоновки приложения "Hello, Android!"

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android
            ="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, Android!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

В каждом файле разметки должен быть только один корневой элемент. В нашем случае таким элементом является **LinearLayout** (линейная разметка). После определения корневого элемента вы можете добавить в него дополнительные объекты разметки или виджеты в качестве дочерних элементов.

Рассмотрим **атрибуты** элемента LinearLayout:

**`xmlns:android`** — объявление пространства имен Android. Стандартный атрибут и стандартное значение для Android-приложения. Декларация пространства имен XML, которая сообщает среде Android, что вы ссылаетесь на общие атрибуты, определенные в пространстве имен Android. В каждом файле компоновки у корневого элемента должен быть атрибут со значением http://schemas.android.com/apk/res/android

**`android:layout_width`** — атрибут определяет, сколько из доступной ширины на экране должен использовать этот объект view (или viewGroup). В нашем случае он — единственный объект, таким образом, можно растянуть его на весь экран, которому в данном случае соответствует значение **fill_parent**;

**`android:layout_height`** — аналогичен атрибуту android:layout_width за исключением того, что он ссылается на доступную высоту экрана;

**`android:text`** — текст, который должен отобразить объект TextView. В нашем примере вместо строковой константы используется значение из файла strings.xml — строка hello. Благодаря такому приему очень легко выполнить локализацию приложения (перевод на другой язык).

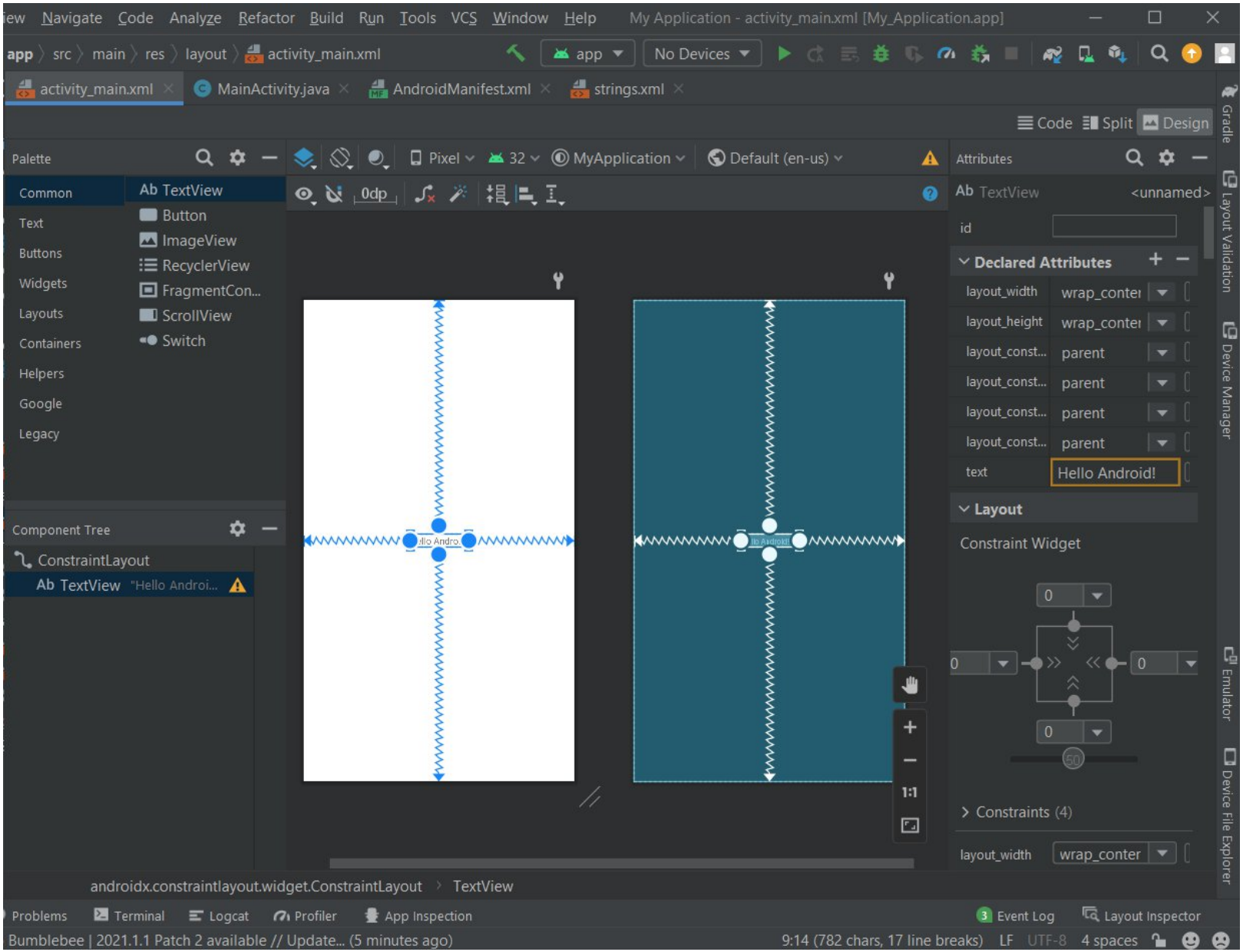
У каждого объекта View или ViewGroup свой набор атрибутов.

# Создание компоновки в Layout Editor

**ADT-плагин** для Eclipse предлагает удобный инструмент — **визуальный редактор компоновки Layout Editor**, который применяется для создания и предварительного просмотра создаваемых файлов компоновки, которые находятся в каталоге res/layout/ проекта.

Например, можно создавать XML-компоновки для различных ориентаций экрана мобильного устройства (portrait, landscape), размеров экрана и языков интерфейса. Дополнительно, объявление компоновки в XML-файле облегчает визуализацию структуры вашего пользовательского интерфейса, что упрощает отладку приложения. На вкладке **Outline** отображается компоновка в виде дерева. Каждый элемент в XML является объектом view или viewGroup (или его потомком). Объекты view — листья дерева, объекты viewGroup — ветви. Вы можете также создавать объекты view и viewGroup в Java-коде, используя метод addview(view), чтобы динамически вставлять новые объекты view и viewGroup в существующую компоновку.

# Layout Editor



# Типы компоновок

Используя различные виды viewGroup, можно структурировать дочерние объекты view и viewGroup многими способами в зависимости от требований к графическому интерфейсу приложения.

Для создания окон существует несколько стандартных типов компоновок, которые вы можете использовать в создаваемых приложениях:

**FrameLayout;**

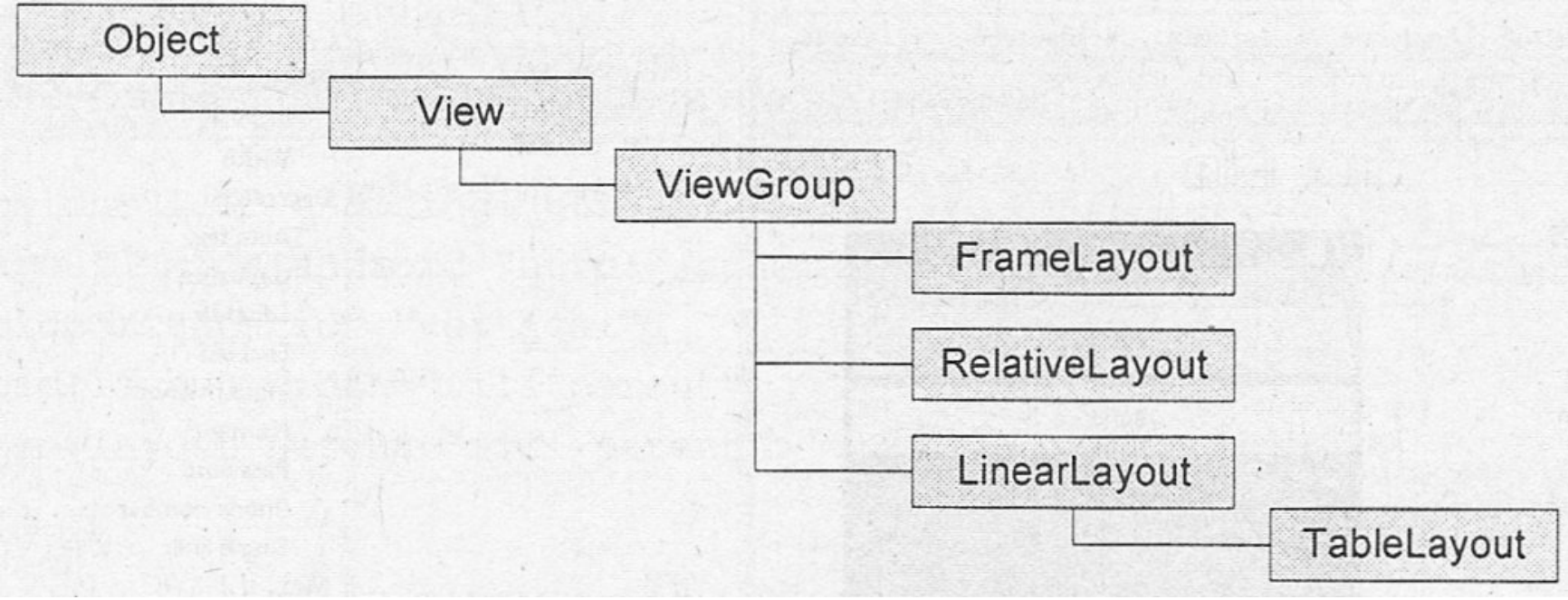
**LinearLayout;**

**TableLayout;**

**RelativeLayout.**

Все эти компоновки являются подклассами viewGroup и наследуют свойства, определенные в классе view.

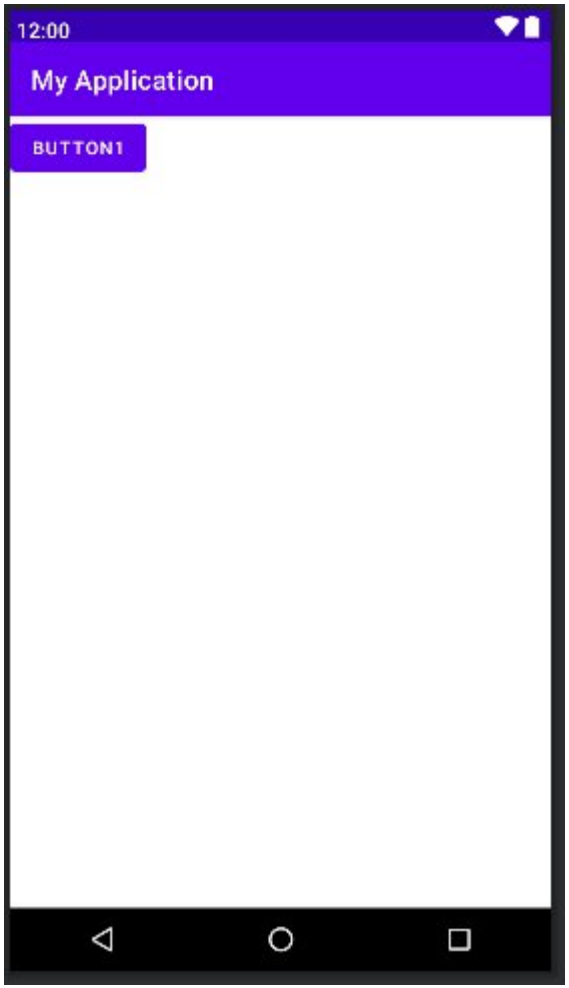
# Иерархия классов компоновок




Каждый, из этих типов компоновки предлагает уникальный набор параметров, которые используются, чтобы определить позиции дочерних элементов view и структуру компоновки на экране. В зависимости от требований, предъявляемых к пользовательскому интерфейсу, выбирается наиболее подходящий тип компоновки. Далее мы рассмотрим все варианты компоновок и их использование.

# FrameLayout

FrameLayout является **самым простым типом компоновки**. Это в основном пустое пространство на экране, которое можно позже **заполнить только единственным дочерним объектом** view или ViewGroup. Все дочерние элементы FrameLayout прикрепляются к **верхнему левому углу экрана**.




Файл компоновки main.xml для `FrameLayout`

```
<FrameLayout
xmlns:android="http://schemas.android.com
/apk/res/android"
    android:id="@+id/FrameLayout01"
    android:layout_height="wrap_content"
    android:1ayout_width="fill_parent" >

  <Button
    android:id="@+id/Button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button1" />
</FrameLayout>
```

В компоновке FrameLayout нельзя определить различное местоположение для дочернего объекта view. Последующие дочерние объекты view будут просто **рисоваться поверх предыдущих**, частично или полностью затеняя их, если находящийся сверху объект непрозрачен, поэтому единственный дочерний элемент для FrameLayout обычно растянут до размеров родительского контейнера и имеет атрибуты `layout_width="fill_parent"` и `layout_height="fill_parent"`.

```
<FrameLayout
    android:id="@+id/FrameLayout01"
    android:layout_height="fill_parent"
    android:layout_width="fill_parent">
<Button
    android:id="@+id/Button1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:text="Button1" />
<Button
    android:id="@+id/Button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="But2" />
</FrameLayout>
```

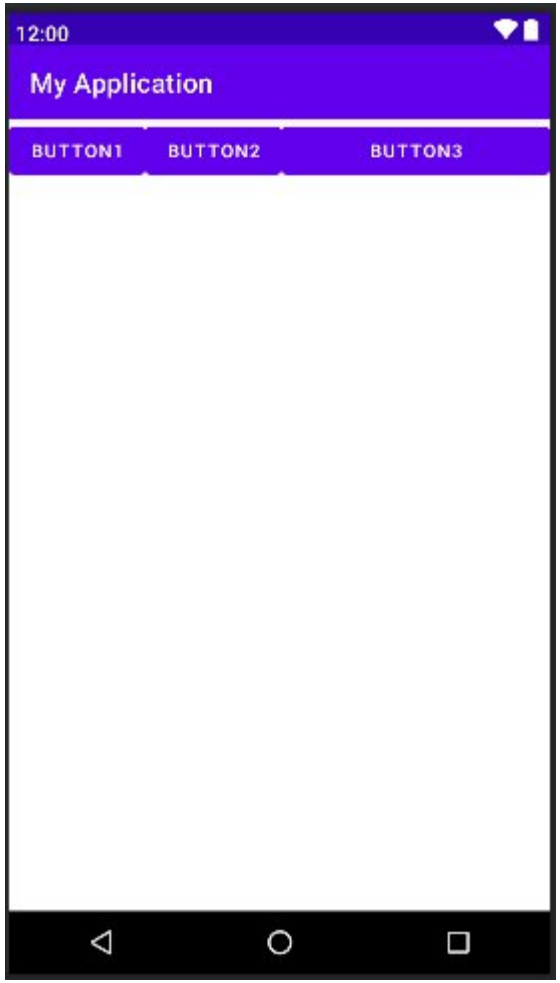
# LinearLayout

Компоновка LinearLayout выравнивает все дочерние объекты view в одном направлении — вертикально или горизонтально, в зависимости от того, как определен атрибут ориентации **`android:orientation:`**

```
android:orientation="horizontal"
```
или
```
android:orientation="vertical"
```

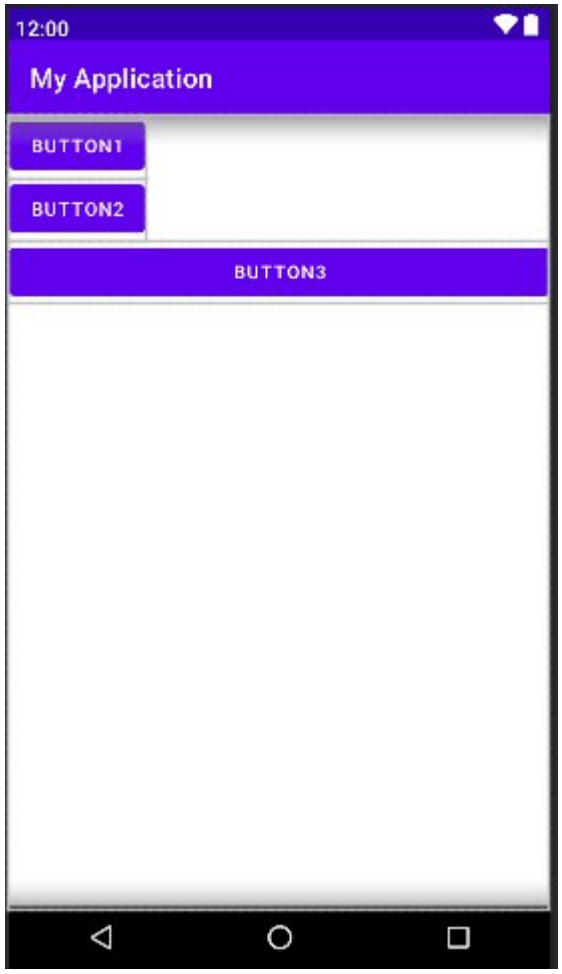
Все дочерние элементы помещаются в стек один за другим, так что вертикальный список объектов view будет иметь только один дочерний элемент в строке независимо оттого, насколько широким он является. Горизонтальное расположение списка будет размещать элементы в одну строку с высотой, равной высоте самого высокого дочернего элемента списка.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal" >

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button1" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button2" />

    <Button
        android:id="@+id/button3"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Button3" />

</LinearLayout>
```



Обратите внимание, что у первых двух кнопок атрибуту `android:layout_width` присвоено значение `wrap_content`, а у третьей кнопки — `fill_parent`, т.е. последняя кнопка заполнит оставшееся свободное пространство в компоновке.
В результате получится линейное горизонтальное размещение дочерних элементов. Если изменить в корневом элементе значение атрибута `android:orientation=”vertical”`, элементы в контейнере расположатся вертикально.

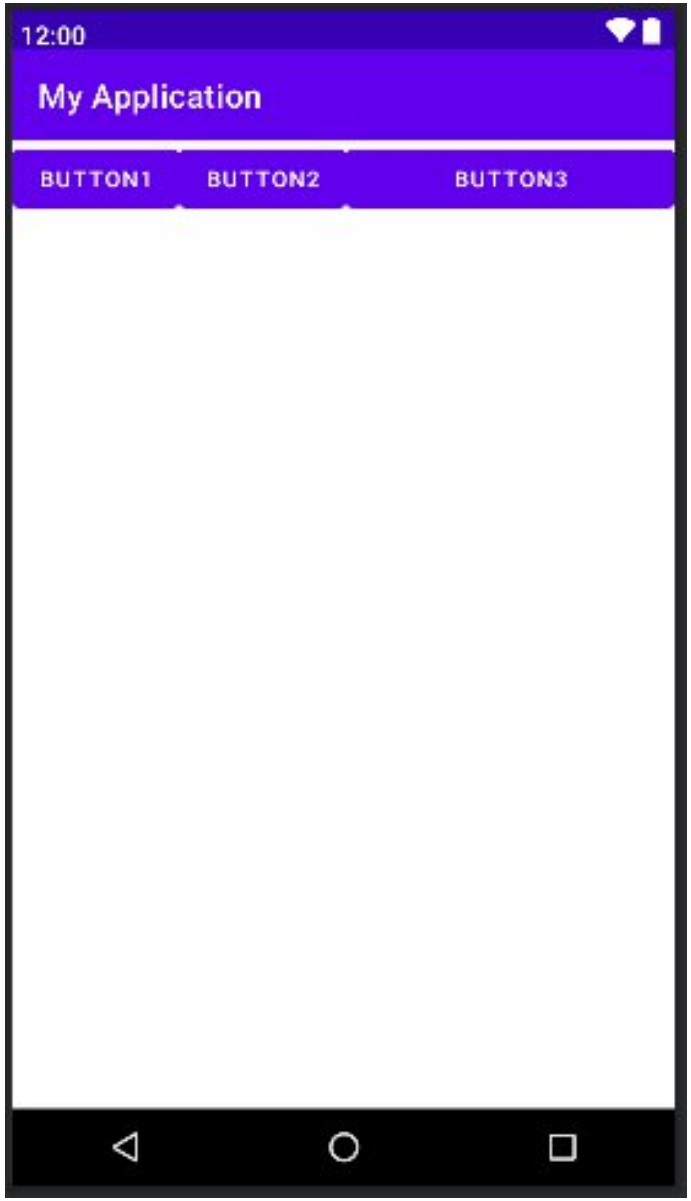
```
---
android:layout_height="fill_parent"
android:orientation="vertical">

<Button
android:id="@+id/button1"
---
```



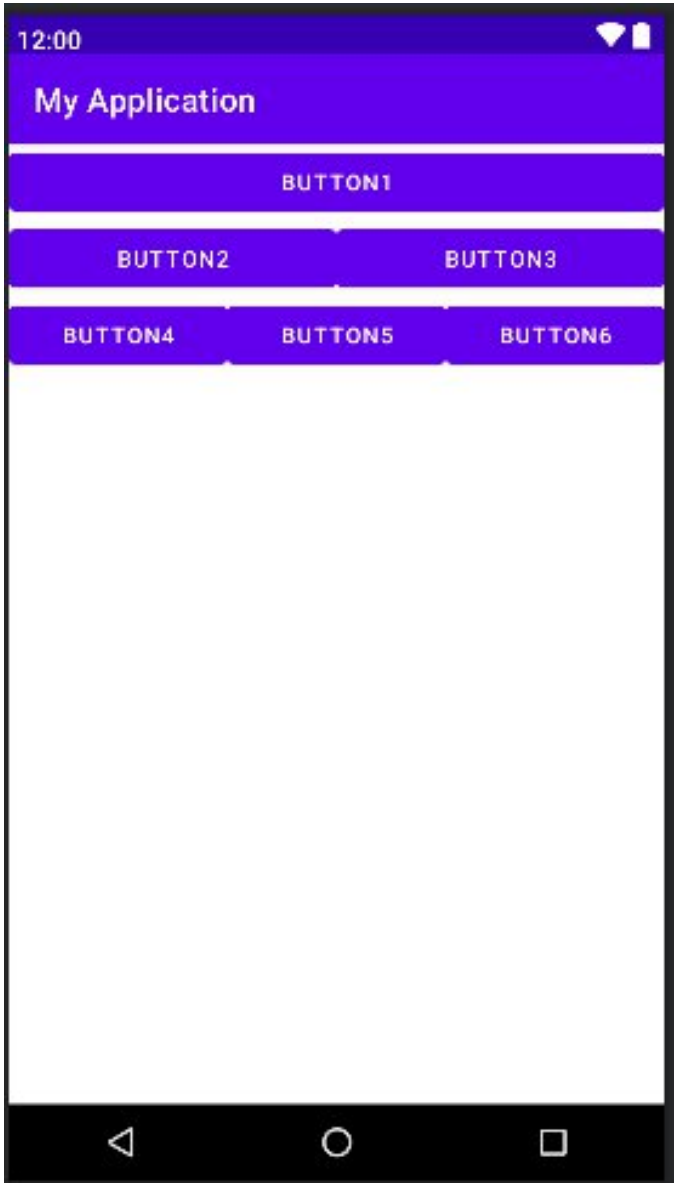
Компоновка LinearLayout также поддерживает атрибут `android: layout_weight`, который назначает **индивидуальный вес для дочернего элемента**. Этот атрибут определяет "важность" объекта view и позволяет этому элементу расширяться, чтобы заполнить любое оставшееся пространство в родительском объекте view. **Заданный по умолчанию вес является нулевым**.

Например, если есть три текстовых поля, и двум из них объявлен вес со значением 1, в то время как другому не дается никакого веса (0), третье текстовое поле без веса не будет расширяться и займет область, определяемую размером текста, отображаемого этим полем. Другие два расширятся одинаково, чтобы заполнить остаток пространства, не занятого третьим полем. Если третьему полю присвоить вес 2 (вместо 0), это поле будет объявлено как "более важное", чем два других, так что третье поле получит 50% общего пространства, в то время как первые два получат по 25% общего пространства.

```
---
android:orientation="horizontal">
<Button
android:id="@+id/button1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Button1"
android:layout_weight="0"/>
<Button
android:id="@+id/button2"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Button2"
android:layout_weight="0"/>
<Button
android:id="@+id/button3"
android:layout_height="wrap_content"
android:layout_width="wrap_content"
android:text="Button3"
android:layout_weight="2"/>

</LinearLayout>
```



**Компоновки могут быть и вложенными.** При проектировании окон с многочисленными элементами управления для заданного расположения элементов часто задаются вложенные компоновки, которые являются контейнерами для элементов управления. Например, для корневой LinearLayout с атрибутом `orientation="vertical"` мы можем задать простой дочерний элемент Button и еще два контейнера LinearLayout с атрибутом `orientation="horizontal"`, которые, в свою очередь, содержат дочерние элементы управления.



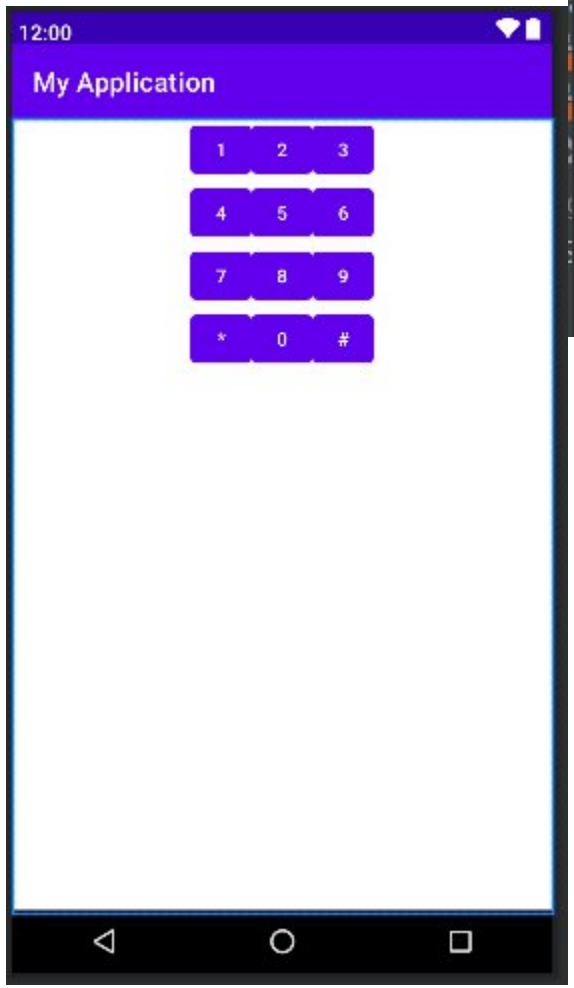
# TableLayout

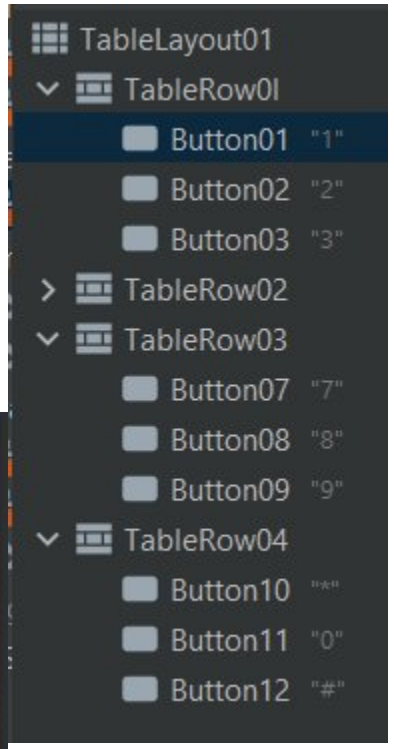
Компоновка TableLayout позиционирует свои дочерние элементы в строки и столбцы. TableLayout не отображает линии обрамления для их строк, столбцов или ячеек. TableLayout может иметь строки с разным количеством ячеек. При формировании компоновки таблицы некоторые ячейки при необходимости можно оставлять пустыми.

При создании компоновки для строк используются объекты **TableRow**, которые являются дочерними классами TabieLayout (каждый TableRow определяет единственную строку в таблице). Строка может не иметь ячеек или иметь одну и более ячеек, которые являются контейнерами для других объектов view или viewGroup. Ячейка может также быть объектом viewGroup (например, допускается вложить другой TabieLayout или LinearLayout как ячейку).

Для примера с использованием компоновки TableLayout можно создать окно, похожее на наборную панель телефона с 12 кнопками. Рассмотрим пример создания TableLayout с четырьмя дочерними TableRow и двенадцатью кнопками, по три кнопки в каждой строке.

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
xmlns:android="http://schemas.android.com/apk/r
es/android"
    android:id="@+id/TableLayout01"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">

<TableRow
android:id="@+id/TableRow01"
android:layout_height="wrap_content"
android:layout_width="fill_parent"
android:gravity="center">
<Button
android:id="@+id/Button01"
android:layout_height="wrap_content"
android:text="1"
android:layout_width="20pt"/>
<Button
android:id="@+id/Button02"
android:layout_height="wrap_content"
android:text="2"
android:layout_width="20pt"/>
<Button
android:id="@+id/Button03"
android:layout_height="wrap_content"
android:text="3"
android:layout_width="20pt"/>
</TableRow>
```



```
<TableRow
android:id="@+id/TableRow02"
android:layout_height="wrap_content"
android:layout_width="fill_parent"
android:gravity="center">
<Button
android:id="@+id/Button04"
android:layout_height="wrap_content"
android:layout_width="20pt"
android:text="4"/>
<Button
android:id="@+id/Button05"
android:layout_height="wrap_content"
android:layout_width="20pt"
android:text="5"/>
<Button
android:id="@+id/Button06"
android:layout_height="wrap_content"
android:layout_width="20pt"
android:text="6"/>
</TableRow>
<TableRow
android:id="@+id/TableRow03"
android:layout_height="wrap_content"
android:layout_width="fill_parent"
android:gravity="center">
<Button
android:id="@+id/Button07"
android:layout_height="wrap_content"
android:layout_width="20pt"
android:text="7"/>
<Button
android:id="@+id/Button08"
android:layout_height="wrap_content"
android:layout_width="20pt"
android:text="8"/>
<Button
android:id="@+id/Button09"
android:layout_height="wrap_content"
android:layout_width="20pt"
android:text="9"/>
</TableRow>
<TableRow
android:id="@+id/TableRow04"
android:layout_height="wrap_content"
android:layout_width="fill_parent"
android:gravity="center">
<Button
android:id="@+id/Button10"
android:layout_height="wrap_content"
android:layout_width="20pt"
android:text="*"/>
<Button
android:id="@+id/Button11"
android:layout_height="wrap_content"
android:layout_width="20pt"
android:text="0"/>
<Button
android:id="@+id/Button12"
android:layout_height="wrap_content"
android:layout_width="20pt"
android:text="#"/>
</TableRow>
</TableLayout>
```

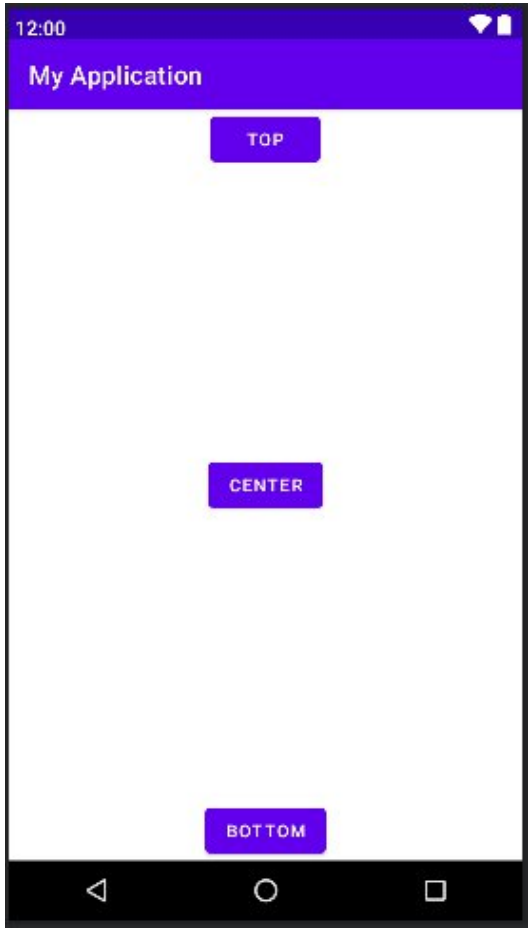
Компоновка TableLayout на практике применяется довольно редко, обычно вместо нее используют сочетание компоновок LinearLayout. TableLayout удобно использовать, если расположение элементов представлено в виде "таблицы“.

# RelativeLayout

**RelativeLayout** (относительная компоновка) позволяет дочерним объектам **определять свою позицию относительно родительского объекта, или относительно соседних дочерних элементов** (по идентификатору элемента).
В RelativeLayout дочерние элементы расположены так, что если первый элемент расположен по центру экрана, другие элементы, выровненные относительно первого элемента, будут выровнены относительно центра экрана. При таком расположении, при объявлении компоновки в XML-файле, элемент, на который будут **ссылаться для позиционирования другие объекты**, должен быть объявлен раньше, чем другие элементы, которые **обращаются к нему по его идентификатору**.
Если в программном коде мы не работаем с некоторыми элементами пользовательского интерфейса, создавать идентификаторы для них необязательно, однако определение идентификаторов для объектов важно при создании RelativeLayout. В компоновке RelativeLayout расположение элемента может определяться относительно другого элемента, на который ссылаются через его уникальный идентификатор:

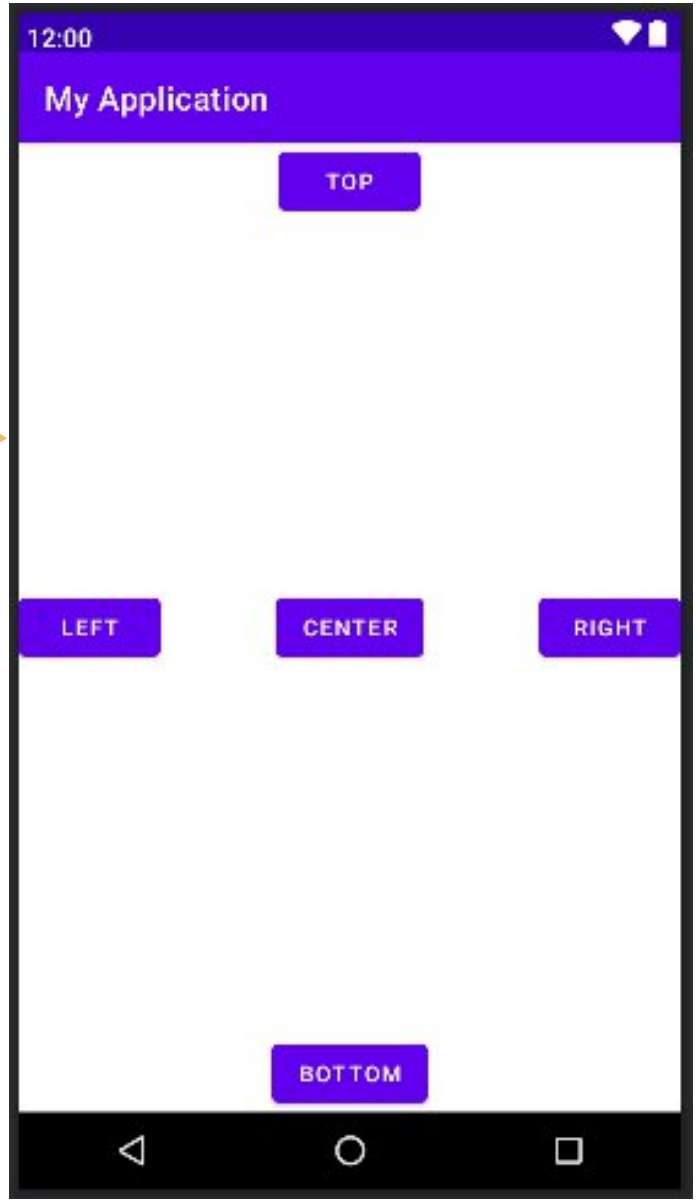
```
android:layout_toLeftOf="@id/TextView1"
```

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com
/apk/res/android"
android:layout_height="fill_parent"
android:layout_width="fill_parent">
<Button
android:id="@+id/button_center"
android:text="Center"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerVertical="true"
android:layout_centerInParent="true"/>
<Button
android:id="@+id/button_bottom"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Bottom"
android:layout_centerHorizontal="true"
android:layout_alignParentBottom="true"/>
<Button
android:id="@+id/button_top"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Top"
android:layout_alignParentTop="true"
android:layout_centerHorizontal="true"/>
```




Тип компоновки RelativeLayout применяется довольно редко. Тем более что такое задание расположения элементов зависит от разрешения и ориентации экрана мобильного устройства. Если вы будете использовать RelativeLayout в собственных приложениях, всегда проверяйте внешний вид окна для различных разрешений и ориентации экрана, поскольку возможно наложение элементов друг на друга.

```
<Button
  android:id="@+id/button_right"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_centerVertical="true"
  android:text="Right"
  android:layout_alignParentRight="true"/>
<Button
  android:id="@+id/button_left"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_centerVertical="true"
  android:text="Left"
  android:layout_alignParentLeft="true"/>
```



# Единицы измерения расстояния в Android

**pt** - 1/72 дюйма, определяется по физическому размеру экрана
**px** – пиксел, не рекомендуется использовать т.к. на разных экранах приложение будет выглядеть по-разному.
**mm** – миллиметр, определяется по физическому размеру экрана
**in** – дюйм, определяется по физическому размеру экрана

**dp** или **dip** - Density-independent Pixels
**sp** - Scale-independent Pixels