

## **Вопросы**

### **«Программирование мобильных и встраиваемых систем» 2022-2023**

1. Встраиваемые системы: основные понятия, архитектуры, область применения и безопасность
2. Операционная система ОС Linux: история развития и основные характеристики.
3. История и тенденции развития мобильных операционных систем.
4. История и тенденции развития мобильной операционной системы Android.
5. Графический интерфейс в Unix-системах.
6. Уровни и основные подсистемы ядра Linux.
7. Виртуальная файловая система ОС Linux.
8. Архитектура и модули ядра ОС Linux.
9. Пакет BusyBox: состав, возможности и применение.
10. Установка программного обеспечения в ОС Linux.
11. Типы файлов и права доступа к файлам в ОС Linux.
12. Управление пользователями и правами доступа в ОС Linux.
13. Процессы в ОС Linux: создание, типы, атрибуты, просмотр информации о процессах.
14. Процессы в ОС Linux: планирование процессов, каналы и перенаправление ввода-вывода.
15. Командный интерпретатор Bash: возможности, режимы работы, файлы конфигурационные файлы.
16. Программирование в bash: исполнительная среда, переменные, исполнение команд и сценарии.
17. Программирование в bash: выражения, условное выполнение команд, циклы и функции.
18. Программирование в ОС Linux: инструментарий и менеджеры пакетов, сборка.
19. Архитектура ОС Android и безопасность Android-приложений.
20. Структура Android-приложения и основные компоненты.
21. Структура проекта Android-приложения: характеристика
22. Файл манифеста AndroidManifest.xml: элементы и структура.
23. Характеристика разметок (макетов) страниц, их типы и основные элементы UI.
24. Разметка (макеты) страниц на примере FrameLayout и LinearLayout.
25. Разметка (макеты) страниц на примере RelativeLayout и ConstraintLayout.
26. Разметка (макеты) страниц на примере TableLayout и GridLayout.
27. Разметка (макеты) страниц на примере ScrollView, ListView и GridView.
28. Ресурсы Android-приложения: строковые, булевые, числовые, меню, ресурсы разметки.
29. Ресурсы Android-приложения: цветовые, размеров, визуальных стилей и тем, drawable.
30. Ресурсы Android-приложения: отрисовываемых объектов, мipmap, анимации, необработанных объектов.
31. Ресурсы Android-приложения: создание псевдонимов ресурсов, компиляция ресурсов, доступ из кода.

32. Иерархия классов в Android SDK.
33. Задачи и стек переходов назад в мобильных приложениях для ОС Android.
34. Фрагменты и управление фрагментами.
35. Намерения в Android-приложении: класс Intent и методы класса.
36. Операции в Android-приложении: класс Activity и методы класса.
37. Сервисы в Android-приложении: класс Service и методы класса.
38. Контент-провайдеры. Класс ContentProvider и методы класса.
39. Приёмники широковещательных сообщений (Broadcast Receivers): основные классы и методы.
40. Рекомендации по разработке и проектированию интерфейсов мобильных приложений.
41. Основы разработки многооконных приложений для ОС Android.
42. Сенсорное управление в смартфонах.
43. Хранение данных в ОС Android на примере SharedPreferences и DataStore.
44. Хранение данных в ОС Android на примере внутреннего и внешнего хранилища: основные классы и методы.
45. Основы работы с базами данных SQLite. Запросы в sqlite.
46. Классы и основные методы для работы с SQLite.
47. 2D и 3D графика в мобильных приложениях для ОС Android.
48. Анимация в ОС Android и типы анимации.
49. Использование встроенной камеры и работа с мультимедиа в Android- приложениях.
50. Классификация сенсоров и датчиков для ОС Android и их характеристики.
51. Взаимодействие с системами позиционирования: права доступа, основные классы и методы.
52. Сетевые соединения: класс ConnectivityManager и класс NetworkInfo.
53. Использование Wi-Fi в Android: основные классы и их характеристики.
54. Использование Bluetooth в Android.

## 1. Встраиваемые системы: основные понятия, архитектуры, область применения и безопасность

Любая механическая или электрическая система, которая имеет в своем составе устройство управления, выполненное на основе вычислителя, называется встраиваемой системой (EmbeddedSystem).

Особенности встраиваемых систем

- Минимальное энергопотребление.
- Минимальные габариты и вес.
- Защита минимальна и обеспечивается прочностью и жесткостью конструкции.
- Обеспечение минимума требований тепловых режимов.
- Микропроцессор, системная логика, ключевые микросхемы на одном кристалле.
- Специальные требования с учетом области применения.

Основа встроенных систем:

- Одноплатные (однокристалльные) ЭВМ.
- Специализированные микропроцессоры. (ПЛИС)

**Система** защиты должна быть многоуровневая, где на каждом уровне используются свои методы защиты данных. Сочетание нескольких уровней защиты затрудняет взлом устройства злоумышленником.

Примеры ВcC: **DVD-проигрыватель, светофорный объект, банкомат, паркомат и т.д.** z Встраиваемой системой можно считать любую вычислительную систему, которая не является ПК, портативным компьютером или большим универсальным компьютером (mainframe computer). z Встроенная вычислительная система – устройство, которое включает в себя программируемый компьютер, но не является при этом компьютером.

## 2. Операционная система ОС Linux: история развития и основные характеристики.

Корни Linux уходят в два других проекта: **Unix** и **Multics**, которые ставили своей целью разработать многопользовательскую операционную систему. Unix – это собрание кроссплатформенных многопользовательских и многозадачных операционных систем. Первая официальная версия Linux 1.0 вышла в 1994 году, вторая – в 1996 году.

Linux – это только ядро операционной системы, и для разработки полноценной операционной системы используются различные инструменты и библиотеки GNU других ресурсов. Кроме того, все больше разработчиков используют Linux для разработки и запуска мобильных приложений. Кроме того, Linux играет ключевую роль в разработке таких устройств, как Хромбуки (портативные устройства под управлением операционной системы Chrome, которая в качестве ядра использует гибрид ядра Linux и сервисов, разработанных компанией Google)

Характеристики Линукса:

- Многозадачность
- Многопользовательский доступ
- Использование страниц памяти
- Использование модулей ядра по требованию
- Динамическое кэширование
- Работа с программами, разработанными под другие ОС
- Поддержка нескольких файловых систем
- Соответствие стандарту POSIX 1003.1

Дистрибутив Linux – это определение операционной системы, которая использует ядро Linux и которую можно установить на машину пользователя. В дистрибутивах обычно содержатся не только ядро и сама операционная система, но и полезные приложения: редакторы, проигрыватели, инструменты для работы с базами данных и другое программное обеспечение.

Ubuntu– один из самых распространенных дистрибутивов, легко устанавливается и интуитивно понятен в работе.

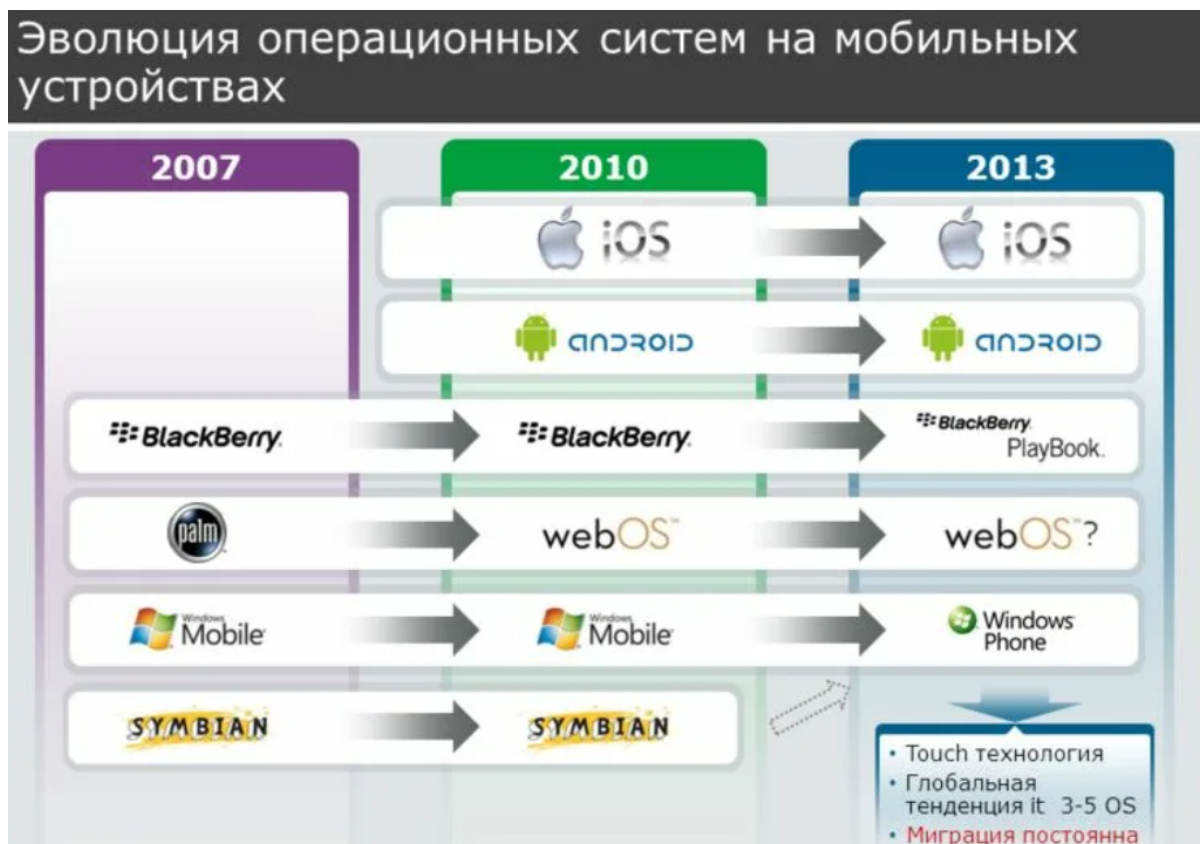
Debian– еще один популярный дистрибутив GNU/Linux, который оказал существенное влияние на развитие всех GNU/Linux-операционных систем в целом.

Linux Mint– дистрибутив, основанный на Ubuntu и Debian. Linux Mint обладает красивым и удобным дизайном и подойдет даже начинающим пользователям.

Manjaro– дистрибутив, основанный на Arch Linux.

Arch– мощный дистрибутив, базирующийся на принципах простоты, современности, прагматизма, гибкости и идеи, что в центре внимания должен быть пользователь.

### 3. История и тенденции развития мобильных операционных систем.



За 30 лет мобильные операционные системы (ОС) эволюционировали от скромной вычислительной среды для работы простых программ-помощников (контакты, записная книжка, калькулятор и др.) до мощного комплекса, обеспечивающего полноценную работу специализированных бизнес-приложений и развлекательных сервисов. Сегодня мобильные ОС дают возможность снимать, редактировать и просматривать видео в высоком разрешении, играть в 3D-игры, использовать искусственный интеллект и технологии дополненной реальности.

Первой полноценной мобильной операционной системой можно считать EPOC16 (кодовое название SIBO — Sixteen Bit Organizer, или Sixteen Bit Operating system), разработанную в 1988 году английским профессором Дэвидом Поттером, основателем компании Psion PLC. 16-разрядная система, написанная на языке C для семейства процессоров Intel 8086 (x86), использовалась в ноутбуках и карманных компьютерах Psion MC200, Psion 3 и в более поздних. EPOC16 поддерживала многооконность, размещение ярлыков на рабочем столе. В её состав входили календарь (встречи, задачи, голосовые заметки, напоминания о днях рождения), база данных для адресной книги, текстовый редактор, будильник, калькулятор, файловый менеджер и другие программы.

Phone OS, впоследствии переименованная в iOS, появилась в 2007 году как операционная система для смартфонов, планшетов и умных часов. Разработана на основе OS X.

В 2008-м — на рынок вышла ОС Android. Она обеспечивала работу смартфонов, планшетов, цифровых плееров, электронных книг, смарт-часов и браслетов, компьютеров, игровых приставок, телевизоров, роботов и других устройств.

Большую роль в распространении iOS и Android сыграло измеряемое миллионами количество приложений, легко доступных в онлайн-магазинах AppStore и Google Play. Установка приложений стала возможной нажатием одной кнопки.

Сегодня рынок мобильных операционных систем поделен между Android (87%) и iOS (12,3%). По количеству загрузок приложений безоговорочно лидирует Google Play. В 2016 году появилась первая российская мобильная операционная система — Sailfish Mobile OS RUS.

#### **4. История и тенденции развития мобильной операционной системы Android.**

Android — это операционная система с открытым исходным кодом, созданная для мобильных устройств на основе модифицированного ядра Linux. Выпущенный на рынок в 2007 году Android вскоре стал самой продаваемой операционной системой в истории, благодаря своей открытой модели разработки и удобному интерфейсу.

История развития Проект Android появился в 2003 году с целью разработки интеллектуальных мобильных устройств. Начинался он с разработки ОС для цифровых фотокамер, но вскоре акцент сместился на мобильные телефоны из-за их большой распространенности на рынке. В 2005 году проект приобрел Google и в качестве основы для этой ОС было выбрано ядро Linux за счет его гибкости и возможности обновления. Платформа Android была представлена в 2007 году и вышла на рынок на следующий год.

##### **Архитектура**

Первоначально Android разрабатывался для архитектуры ARM, а затем был расширен для поддержки архитектур x86 и x86–64.

Основой ОС Android является модифицированная версия ядра Linux LTS, которая непосредственно взаимодействует с оборудованием.

##### **Разработка приложения**

Основной принцип разработки в Android заключается в том, чтобы абстрагироваться от вариативности оборудования и предоставить унифицированный интерфейс для приложений. Это достигается запуском всех приложений на виртуальных машинах Java.

## 5. Графический интерфейс в Unix-системах.



Графический интерфейс в Linux строится на основе стандарта X Window System, разработка которого была начата в 1984 году. В настоящее время действует 6-ой релиз (выпуск) 11-ой версии стандарта на графическую подсистему для UNIX-систем, который кратко обозначается как X11R6.

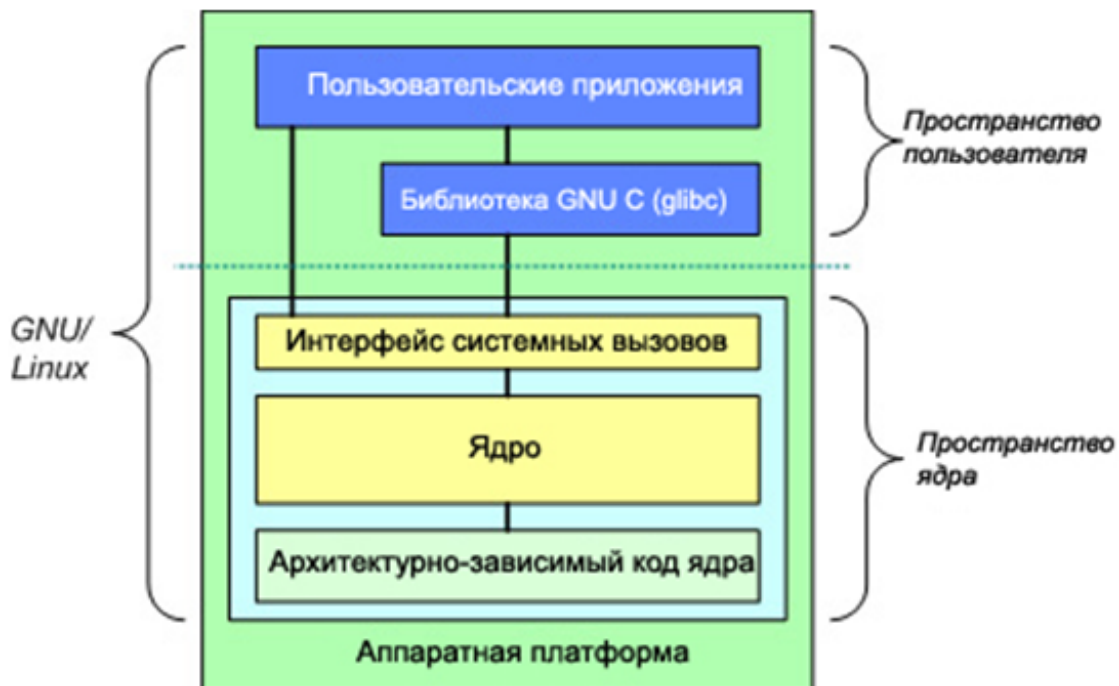
Операционная система UNIX с самого начала была многопользовательской, многозадачной системой, работавшей в режиме разделения времени. При этом она позволяла пользователям работать в удаленном режиме, либо через терминалы, либо с использованием сетевых технологий. Эти основные концепции были учтены при создании графического интерфейса для UNIX и поэтому система X Window построена на основе модели "клиент/сервер".

Элементы графического интерфейса (иконки, кнопки, диалоговые окна, линейки прокрутки, различные рамки и оконные меню) и прорисовываются на экране с помощью низкоуровневых функций из библиотеки X-Lib

Существует два основных варианта запуска графического интерфейса пользователя в системе Red Hat Linux. В первом варианте X-сессия запускается менеджером дисплея xdm, после чего пользователь получает возможность войти в систему (логироваться) непосредственно в графическом режиме. Во втором варианте пользователь вначале входит в систему в текстовом режиме, а потом запускает X-сессию с помощью программы xinit (чаще всего для этого используется скрипт startx, который является просто оболочкой для программы запуска графического режима xinit). В любом случае система X Window запускается с правами суперпользователя, поскольку ей требуется доступ к аппаратным устройствам.



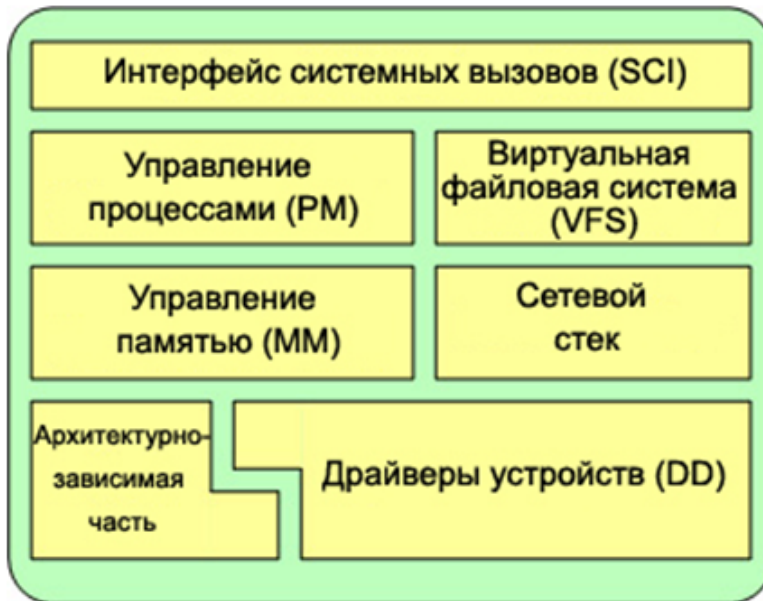
## 6. Уровни и основные подсистемы ядра Linux.



На верхнем уровне находится пользовательское пространство (пространство приложений). Здесь исполняются приложения пользователя. Под пользовательским пространством располагается пространство ядра. Здесь функционирует ядро Linux.

Ядро Linux можно, в свою очередь, разделить на три больших уровня. Наверху располагается интерфейс системных вызовов, который реализует базовые функции, например, чтение и запись. Ниже интерфейса системных вызовов располагается код ядра, точнее говоря, архитектурно-независимый код ядра.

Основные подсистемы ядра Linux:



SCI - это тонкий уровень, предоставляющий средства для вызова функций ядра из пространства пользователя.

#### Управление процессами

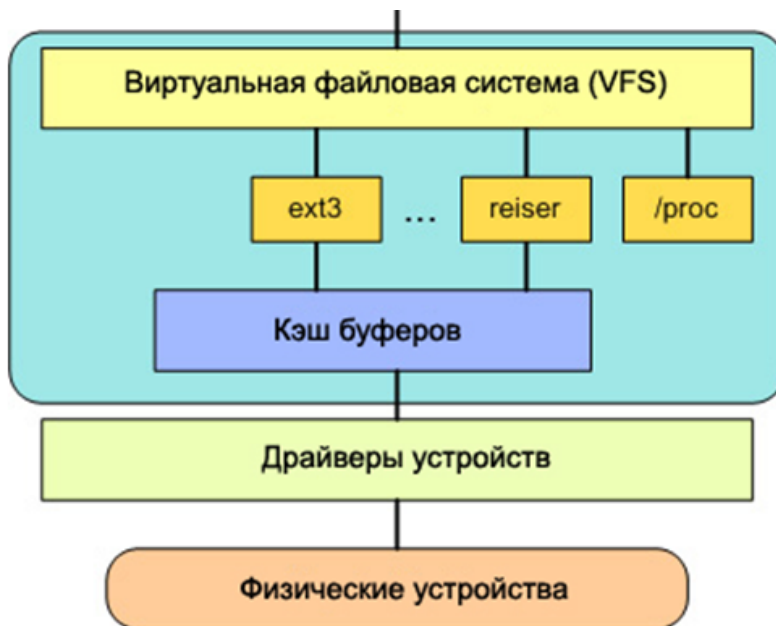
Управление процессами сконцентрировано на исполнении процессов. В ядре эти процессы называются потоками (threads)

#### Управление памятью

Для повышения эффективности, учитывая механизм работы аппаратных средств с виртуальной памятью, память организуется в виде т.н. страниц В Linux имеются средства для управления имеющейся памятью, а также аппаратными механизмами для установления соответствия между физической и виртуальной памятью.

## 7. Виртуальная файловая система ОС Linux.

### Виртуальная файловая система



Δ<sub>1</sub>

Виртуальная файловая система (VFS) предоставляет общую абстракцию интерфейса к файловым системам. VFS предоставляет уровень коммутации между SCI и файловыми системами, поддерживаемыми ядром.

На верхнем уровне VFS располагается единая API-абстракция таких функций, как открытие, закрытие, чтение и запись файлов. На нижнем уровне VFS находятся абстракции файловых систем, которые определяют, как реализуются функции верхнего уровня. Они представляют собой подключаемые модули для конкретных файловых систем (которых существует более 50).

Ниже уровня файловой системы находится кэш буферов, предоставляющий общий набор функций к уровню файловой системы (независимый от конкретной файловой системы). Этот уровень кэширования оптимизирует доступ к физическим устройствам за счет краткосрочного хранения данных (или упреждающего чтения, обеспечивающего готовность данных к тому моменту, когда они понадобятся). Ниже кэша буферов находятся драйверы устройств, реализующие интерфейсы для конкретных физических устройств.

## 8. Архитектура и модули ядра ОС Linux.

Архитектура любой ОС может быть разделена структурно на 2 части это ядро (Kernel) и программы (Applications). Applications+Kernel это ОС. Ядро (Kernel) – это центр ОС. Оно обеспечивает доступ программы к оборудованию компьютера, такого как оперативная память, процессорное время, жесткие диски, видеокарта и т.д.

Ядро любой ОС можно классифицировать как: монолитное ядро, модульное ядро, микроядро, экзо-ядро, нано-ядро, гибридное ядро. В ОС GNU/Linux используется монолитное ядро.

Автор статьи: admin

Метки: [Linux](#) / [Обзор](#)

В этой статье вы прочтаете про архитектура ОС Linux, думаю будет очень интересно и полезно.

Также посмотрите статью «[Лучшие книги для изучения Linux](#)», там вы найдёте ещё книги для изучения Linux.

### Архитектура ОС GNU/Linux:

Для того что бы понять GNU/Linux нужно разобраться с его архитектурой для этого вспомним что же такое операционная система (сокращенно ОС). ОС – это системная программа, которая выступает в качестве посредника между пользователем и аппаратурой компьютера. ОС – это системная программа, которая выступает в качестве посредника между прикладными программами и аппаратурой компьютера.

Для простоты условно разделим архитектуру на 3 части: двух уровневая (простая), структурная (схематическая, с основными компонентами системы) и полная схема ядра (объединение всех частей ядра в единую схему). Постепенно разберемся с каждой из них и будем углублять знания о GNU/Linux. И в конце сравним архитектуры Windows и Linux.

- Двухуровневая архитектура;
- Структурная архитектура;
- Сравнение Windows и GNU/Linux;

- Полная схема ядра Linux;

### **Двух уровневая архитектура:**

Архитектура любой ОС может быть разделена структурно на 2 части это ядро (Kernel) и программы (Applications). Applications+Kernel это ОС. Ядро (Kernel) – это центр ОС.

Оно обеспечивает доступ программы к оборудованию компьютера, такого как оперативная память, процессорное время, жесткие диски, видеокарта и т.д.

Программы (Applications) – это разные утилиты, сервисы, прикладные программы и т.д. Из 2-х уровневой архитектуры, можно сделать вывод: ничего особенного.

Схематически так можно изобразить любую операционную систему. Windows, Dos, Unix, MAC OS и другие. Попробуем разобраться детальней.

Программное обеспечение в GNU/Linux взято из проекта Ричарда Столлмэна GNU — то самое свободное программное обеспечение (open source). Ядро, которое использует GNU/Linux, то самое ядро Linux, написанное Линусом Торвальдсом.

Ядро любой ОС можно классифицировать как: монолитное ядро, модульное ядро, микроядро, экзо-ядро, нано-ядро, гибридное ядро. В ОС GNU/Linux используется монолитное ядро.

Монолитное ядро – изображено на рисунке и состоит из ядра (kernel) и модулей (modules). Части ядра – называют модулями.

При этом модуль ядра — это часть ядра, то есть модуль не является полноценной, независимой программой, а является частью одной большой программы, которая называется ядро ОС.

Все модули используют единое адресное пространство оперативной памяти, одни и те же данные. Иными словами, любой модуль может обратиться к данным которые использует другой модуль, так и к ОП другого модуля.

Как вывод крах одного модуля может повлечь за собой крах другого модуля или всей системы. К примеру, в случае если модуль некорректно изменит общие данные, то это может привести к ошибке в другом модуле и возможно к ошибке во всем ядре.

В архитектуре монолитного ядра есть свои плюсы и минусы. Плюсы заключаются в том, что разрабатывать отдельные модули ядра очень просто, и работает такое ядро очень быстро.

Минусом является то, что ошибка в работе любого модуля, может привести к краху всей системы.

Старые ядра Linux требовали перекомпиляции ядра (то есть созданию нового ядра) при использовании нового оборудования.

То есть если возникала необходимость добавить новый модуль в ядро, то приходилось целиком пересобрать ядро. Новые ядра Linux могут на ходу подгружать модули. Такие модули часто называют динамическими.

### **Структурная схема ядра:**

Посмотрим на архитектуру GNU/Linux немного детальней. Архитектура разбита на 3 функциональных уровня: уровень пользователя, уровень ядра и уровень аппаратуры.

Уровень пользователя — это то программы (Applications). На этом уровне работают разные программы, службы, системные утилиты и т.д. В уровне пользователя есть программное обеспечение (пользовательские приложения), которое может работать напрямую с ядром, либо посредством специальных системных библиотек (к примеру: glibc).

Уровень ядра – тут находится само ядро ОС.

Уровень аппаратуры — тут находятся разные устройства такие как оперативная память, процессор, жесткие диски, видеокарта и т.д.

В пространстве ядра есть “обращение к операционной системе”, это посредник между программами и ядром. (Executive Services), который принимает данные от программ и передает их ядру системы, а ядро непосредственно работает с оборудованием.

Именно “Подсистема управления файлами” ( файловая система) и “подсистема управления процессами”( управления процессами) — две основных компоненты ядра, именно поэтому Подсистема управления файлами связана с драйверами напрямую не битовыми потоками, а символьными.

## 9. Пакет BusyBox: состав, возможности и применение.

**BusyBox** — набор UNIX-утилит командной строки, используется в качестве основного интерфейса во встраиваемых операционных системах. Преимуществами этого приложения являются малый размер и низкие требования к аппаратуре. Оно представляет собой единый файл (это позволяет сэкономить дисковое пространство)

`busybox` — это набор консольных `unix` утилит, ориентированный на малый размер и производительность, что так актуально для мобильных систем. Вместе с системой *android* поставляется свой набор утилит — `toolbox`, который предоставляет минимально необходимых функционал для системы, и как следствие более простой в количественном и функциональном плане. Наличие *busybox* в системе, с одной стороны, позволит нам, как разработчикам, чувствовать себя более комфортно при удаленной работе на устройстве, с другой, позволит писать сложные скрипты, и, например, реализовать механизм запуска собственных скриптов при загрузке, используя *run-parts*. Также стоит учитывать, что для некоторых *android* приложений (особенно те, которые используют *root*) наличие `busybox` — обязательно

```
# busybox --list
```

Встроенная справка вызывается ключом «--help»:

```
# busybox --help
```

BusyBox используется несколькими операционными системами, работающими на встраиваемых системах, и является важным компонентом таких дистрибутивов, как OpenWRT, OpenEmbedded (включая проект Yocto) и Buildroot. Sharp Zaurus широко использует BusyBox для обычных Unix-подобных задач, выполняемых в системной оболочке.<sup>[43]</sup>

BusyBox также является важным компонентом VMware ESXi и Alpine Linux, которые не являются встроенными дистрибутивами.

## 10. Установка программного обеспечения в ОС Linux.

В Linux каждый компонент системы или прикладной программы представлен в виде пакета.

Все файлы, необходимые для работы программы, объединяются в архивы – пакеты. Специальная программа - менеджер пакетов занимается установкой, удалением, обновлением и проверкой пакетов. Менеджер пакетов определяет, какие пакеты нужны для установки программы, проверяет, какие пакеты уже были установлены в системе другими программами, отслеживает чтобы в разных пакетах не оказалось файлов с одинаковым именем и путем, то есть чтобы файл одного пакета не был заменен файлом другого пакета при установке. Менеджер пакетов скачивает их из специальных хранилищ – репозиториев.

Наиболее известный и популярный менеджер пакетов называется APT (Advanced Package Tool).

Для операционной системы (ОС) Linux существует два основных типа пакетов: RPM (Red Hat Package Manager) и Debian.

Для управления пакетами формата RPM в Linux используется соответствующая команда — `rpm`. Она производит установку, удаление, а также опрос состояния программ.

Аналогом команды `rpm` для работы с Debian-пакетами является команда `dpkg`. Режимы её работы указываются аналогично команде `rpm` – с помощью соответствующих опций.

Для облегчения поиска, загрузки программ, отслеживания зависимостей, а также для автоматизации обновлений приложений существуют соответствующие системы управления пакетами (СУП). Самыми распространёнными являются APT и `yum`.



## 11. Типы файлов и права доступа к файлам в ОС Linux.

Все типы файлов, которые встречаются в Linux:

обычные файлы (-);  
каталоги (d);  
символьные устройства (c);  
блочные устройства (b);  
сокеты (s);  
символьные ссылки (l).

Устройства в Linux тоже представлены файлами, для них даже выделен каталог /dev который хранит виртуальную файловую систему devfs. Эта файловая система хранит список всех устройств компьютера. Такие устройства разделяются на символьные и блочные.

Блочные устройства это диски и их разделы, raid-массивы, и тому подобное. Эти устройства могут хранить файловую систему и файлы на ней. Они умеют обрабатывать операции ввода-вывода, то-есть умеют записывать или считывать блоки данных. И обычно поддерживают произвольный доступ к данным.

Символьные устройства это COM-порты, LPT-порты, PS/2-мышки и клавиатуры, USB-мышки и клавиатуры. Такие устройства обычно поддерживают операции (read, write, open, close). И поддерживают посимвольный, то-есть последовательный доступ к данным.

Для взаимодействия программ друг с другом часто используются сокеты. На сокеты в выводе ls -l указывает символ "s".

Чтобы получить доступ к файлам в Linux, используются разрешения. Эти разрешения назначаются трем объектам: файлу, группе и другому объекту (то есть всем остальным).

Отображение владельца файла или каталога

В Linux у каждого файла и каждого каталога есть два владельца: пользователь и группа.

Эти владельцы устанавливаются при создании файла или каталога. Пользователь, который создаёт файл становится владельцем этого файла, а первичная группа, в которую входит этот же пользователь, так же становится владельцем этого файла. Чтобы определить, есть ли у вас как у пользователя права доступа к файлу или каталогу, оболочка проверяет владение.

Для управления правами используется команда chmod. При использовании chmod вы можете устанавливать разрешения для пользователя (user), группы (group) и других (other).

## 12. Управление пользователями и правами доступа в ОС Linux.

Linux - многопользовательская операционная система. Это означает, что несколько пользователей могут работать одновременно, решая различные задачи и совершенно не мешая друг другу.

Многопользовательская среда предполагает наличие механизма регулирования прав доступа к любому ресурсу в системе. Существует три типа прав доступа: на чтение, запись и исполнение. Каждый файл имеет определенного владельца и группу, увидеть это можно с помощью команды `ls -l`:

У каждого объекта в Linux есть свой идентификатор, а также права доступа, применяемые к данному идентификатору. Идентификатор есть у пользователя - *UID*, у группы - *GID*, у файла - *inode*.

Собственно **inode** является, как идентификатором файла/каталога, так и сущностью, которая содержит в себе информацию о файле/каталоге. Например такую, как: принадлежность к владельцу/группе, тип файла и права доступа к файлу.

Для каждого объекта файловой системы в модели полномочий Linux есть три типа полномочий:

- Полномочия чтения (r от read).
- Записи (w от write).
- Выполнения (x от execution).

Управление правами доступа происходит с помощью команды **chmod**, управление владельцем файла происходит с помощью команды **chown**. Синтаксис команд следующий:

```
chmod[к_какой_группе_прав][что_сделать_с_правами][какие_права]  
<над_каким_объектом>
```

```
chmod [права] <над_чем>
```

### **13. Процессы в ОС Linux: создание, типы, атрибуты, просмотр информации о процессах.**

Процесс в ядре представляется просто как структура с множеством полей.

Процессы переднего плана (также известны как интерактивные процессы) - они инициализируются и контролируются в терминальной сессии. Другими словами, для запуска таких процессов в системе должен находиться пользователь, они не запускаются автоматически как часть системных служб.

Фоновые процессы (также известны как неинтерактивные/автоматические процессы) - не подключены к терминалу. Они не ждут ввода от пользователя.

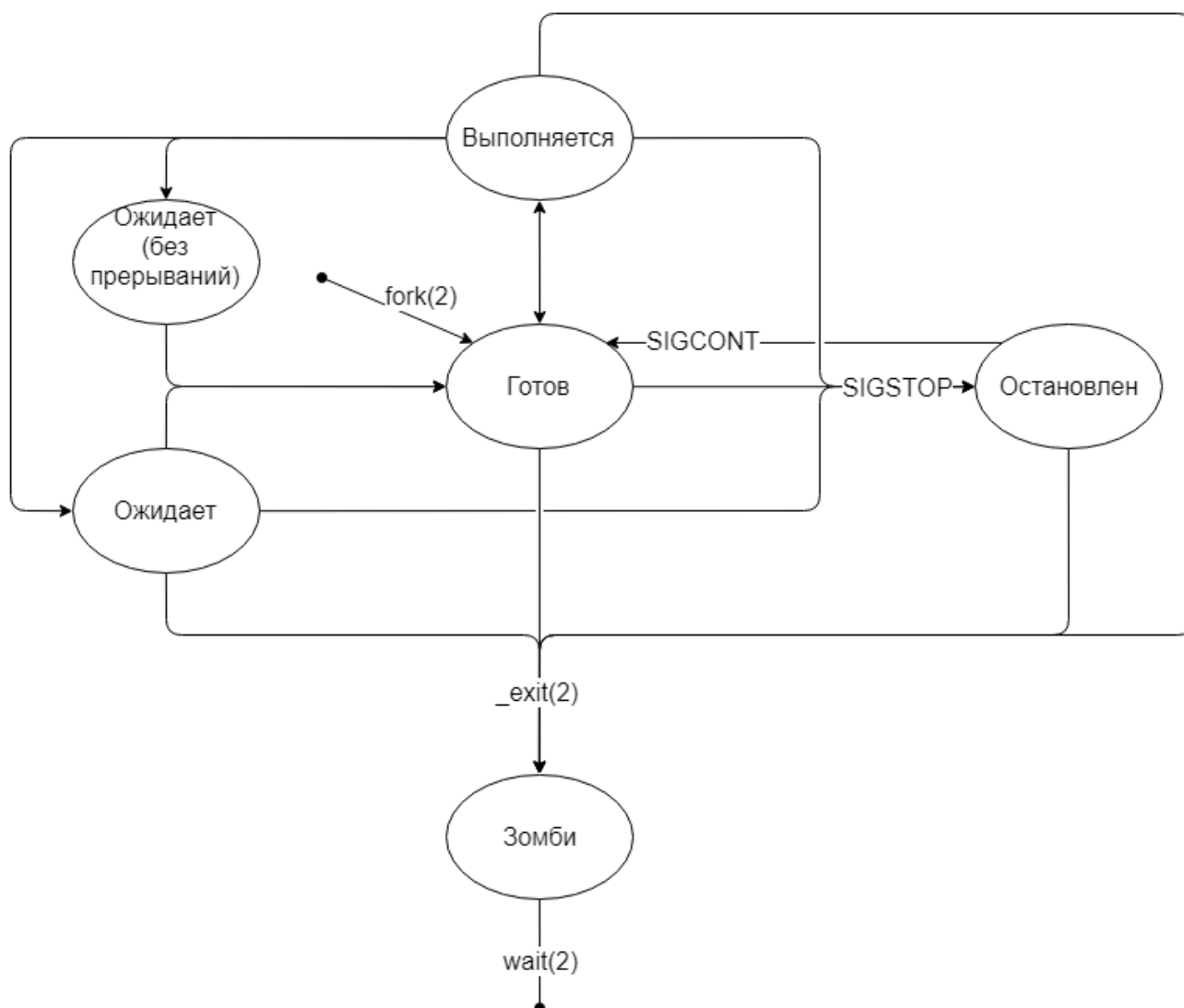
#### **Создание процессов в Linux**

Обычно новый процесс создается уже существующим процессом, который делает в памяти свою точную копию. Дочерний процесс получает то же окружение, что и его родительский процесс, отличается только номер ID.

Есть два распространенных способа создания нового процесса в Linux:

1. С помощью функции `System()`. Этот способ сравнительно прост, однако неэффективен и создает определенные риски с точки зрения безопасности.
2. С помощью функций `fork()` и `exec()` - более продвинутая техника с точки зрения гибкости, скорости и безопасности.

#### **Жизненный цикл процесса**



В зависимости от различных обстоятельств состояние процесса во время работы может меняться. В Linux процесс может находиться в следующих состояниях:

Running (работа) - процесс работает (он является текущим процессом в системе) или готов к работе (ждет выделения ресурсов процессора).

Waiting (ожидание) - в этом состоянии процесс ждет события, которое должно запустить его, или выделения системных ресурсов.

Кроме того, ядро системы делит процессы в состоянии ожидания на два типа: прерываемые процессы, состояние ожидания которых может быть прервано сигналом, и непрерываемые, состояние ожидания которых может быть прервано только аппаратным способом.

Stopped (остановка) - в этом состоянии процесс останавливает работу, обычно после получения соответствующего сигнала. Например, процесс может быть остановлен для отладки.

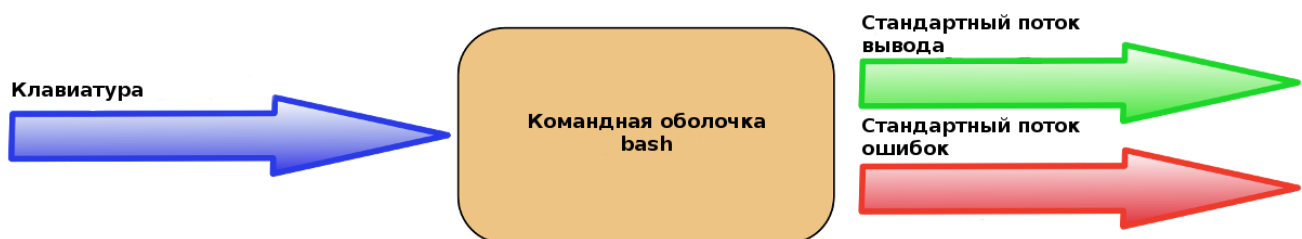
Zombie (зомби) - процесс мертв, то есть он был остановлен, но в системе осталась выполняемая им задача.

## 14. Процессы в ОС Linux: планирование процессов, каналы и перенаправление ввода-вывода.

Процессом в Linux называется выполняющаяся программа. Поскольку Linux – многозадачная система, то в одно и то же время могут выполняться несколько процессов. Чтобы их различать, Linux присваивает каждому процессу уникальный номер, называемый идентификатором процесса (process ID).

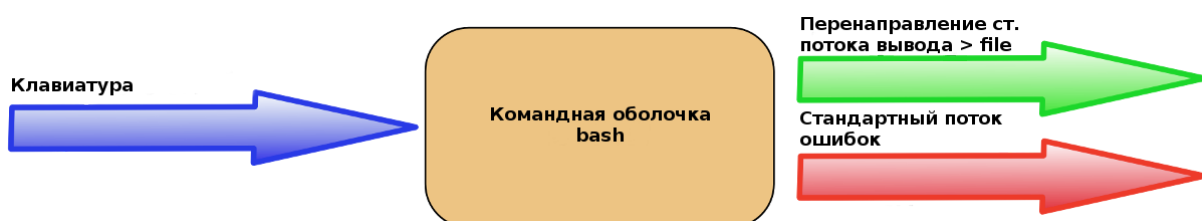
Идентификатор процесса – это число, идентифицирующее каждый выполняющийся процесс. Увидеть идентификаторы процессов можно, например, с помощью команды ps.

Командная оболочка bash поддерживает три типа базовых потоков данных; она принимает данные из стандартного потока ввода stdin (поток 0), отправляет данные в стандартный поток вывода stdout (поток 1), а также отправляет сообщения об ошибках в стандартный поток ошибок stderr (поток 2).



### Операция перенаправления потока данных stdout (>)

Перенаправление стандартного потока вывода stdout может быть осуществлено с помощью символа знака "больше". В том случае, если при разборе строки команды командная оболочка обнаруживает символ знака >, она удаляет данные из файла и перенаправляет данные из стандартного потока вывода в него.



### Операция перенаправления потока данных stderr (2>)

Перенаправление стандартного потока ошибок осуществляется с помощью оператора 2>. Такое перенаправление может оказаться очень полезным для предотвращения заполнения вашего экрана сообщениями об ошибках.

## 15. Командный интерпретатор Bash: возможности, режимы работы, файлы конфигурационные файлы.

+

## 16. Программирование в bash: исполнительная среда, переменные, исполнение команд и сценарии.

командный интерпретатор - это программа, имеющая свои собственные встроенные команды (built-in commands), свое собственное переменное окружение (environment), а также позволяющая выполнять внешние команды, которые присутствуют в системе.

### Навигация

В ОС «Линукс» каталоги и файлы имеют иерархическую организацию, т. е. существует начальный каталог, который называется корневым. В этом каталоге содержатся подкаталоги и файлы, в свою очередь содержащие свои подкаталоги и файлы.

#### **pwd**

Команда pwd (print working directory) служит для отображения текущего местоположения в структуре каталогов.

#### **cd**

Даёт возможность перейти в новый каталог.

**Bash** — популярный командный интерпретатор, используемый в юниксоподобных системах, например, в GNU/Linux.

#### **mkdir**

Служит для создания нового каталога в текущем каталоге.

Пользовательская оболочка bash может работать в двух режимах - интерактивном и, соответственно, не интерактивном. Открыть оболочку в Ubuntu можно комбинацией клавиш **Ctrl + Alt + F1**, привычный графический интерфейс исчезнет, а перед вами откроется один из семи виртуальных терминалов, доступных в дистрибутиве Ubuntu.

конфигурационные файлы - <https://losst.pro/osnovnye-konfiguratsionnye-fajly-linux>

| Обычные флаги | Описание                               |
|---------------|--|
| -i            | Отключение чувствительности к регистру |
| -r            | Рекурсивный поиск по директориям       |
| -w            | Поиск только целых слов                |
| -c            | Вывод количества найденных элементов   |
| -n            | Вывод всей строки, содержащей запрос   |
| -v            | Вывод инвертированного совпадения      |

Сценарии командной строки — это наборы тех же самых команд, которые можно вводить с клавиатуры, собранные в файлы и объединённые некоей общей целью. При этом результаты работы команд могут представлять либо самостоятельную ценность, либо служить входными данными для других команд. Сценарии — это мощный способ автоматизации часто выполняемых действий.

### Вывод сообщений

Для вывода текста в консоль Linux применяется команда `echo`. Воспользуемся знанием этого факта и отредактируем наш скрипт, добавив пояснения к данным, которые выводят уже имеющиеся в нём команды:

```
#!/bin/bash
# our comment is here
echo "The current directory is:"
pwd
echo "The user logged in is:"
whoami
```

### Переменные среды

Иногда в командах оболочки нужно работать с некими системными данными. Вот, например, как вывести домашнюю директорию текущего пользователя:

```
#!/bin/bash
# display user home
```

```
echo "Home for the current user is: $HOME"
```

## Пользовательские переменные

В дополнение к переменным среды, bash-скрипты позволяют задавать и использовать в сценарии собственные переменные. Подобные переменные хранят значение до тех пор, пока не завершится выполнение сценария.

Как и в случае с системными переменными, к пользовательским переменным можно обращаться, используя знак доллара:

```
#!/bin/bash
# testing variables
grade=5
person="Adam"
echo "$person is a good boy, he is in grade $grade"
```

## Проверки файлов

Пожалуй, нижеприведённые команды используются в bash-скриптах чаще всего. Они позволяют проверять различные условия, касающиеся файлов. Вот список этих команд.

```
-d file Проверяет, существует ли файл, и является ли он директорией.
-e file Проверяет, существует ли файл.
-f file Проверяет, существует ли файл, и является ли он файлом.
-r file Проверяет, существует ли файл, и доступен ли он для чтения.
-s file Проверяет, существует ли файл, и не является ли он пустым.
-w file Проверяет, существует ли файл, и доступен ли он для записи.
-x file Проверяет, существует ли файл, и является ли он исполняемым.
file1 -nt file2 Проверяет, новее ли file1 , чем file2 .
file1 -ot file2 Проверяет, старше ли file1 , чем file2 .
-O file Проверяет, существует ли файл, и является ли его владельцем текущий
пользователь.
-G file Проверяет, существует ли файл, и соответствует ли его идентификатор группы
идентификатору группы текущего пользователя.
```



## 17. Программирование в `bash`: выражения, условное выполнение команд, циклы и функции.

Регулярные выражения — это специальным образом записанные строки, используемые для поиска символьных шаблонов в тексте.

### Утилита `grep`

Программа `grep` — это основной инструмент для работы с регулярными выражениями. `Grep` анализирует данные со стандартного ввода, ищет совпадения с указанным шаблоном и выводит все подходящие строки.

```
grep [параметры] регулярное_выражение [файл...]
```

Basic Regular Expressions (BRE). Ему соответствуют практически все POSIX-совместимые программы. Второй — Extended Regular Expression (ERE). Этот вид позволяет создавать более сложные регулярные выражения, но поддерживается не всеми утилитами.

### Extended Regular Expressions

Стандарт POSIX ERE позволяет создавать более выразительные регулярные выражения, однако не все программы умеют с ним работать. Диалект ERE поддерживался программой `egrep`, но GNU-версия `grep` также поддерживает расширенные регулярные выражения при вызове с ключом `-E`.

### Группировка

Элементы регулярных выражений можно объединять и ссылаться на них как на один элемент. Делается это с помощью круглых скобок.

### Квантификаторы

Квантификаторы позволяют определить число совпадений с элементом. BRE поддерживают несколько способов.

Квантификатор `?` означает совпадение с элементом ноль или один раз. Иными словами совпадение с предыдущим элементом необязательно:

```
echo "tet" | grep -E 'tes?t'
```

Output:

```
tet
```

```
echo "test" | grep -E 'tes?t'
```

Output:

```
test
```

Основной синтаксис оператора *if ... then* выглядит следующим образом:

```
if <condition>; then
```

<commands>

fi

Условие, в зависимости от его типа, окружено определенными скобками, например. []. Вы можете прочитать о различных типах дальше в учебнике. Вы можете добавить команды, которые будут выполняться, когда условие ложно, с помощью ключевого слова *else* и использовать ключевое слово *elif* (*elseif*) для выполнения команд с другим условием, если основное условие ложно. Ключевое слово *else* всегда стоит последним. Пример:

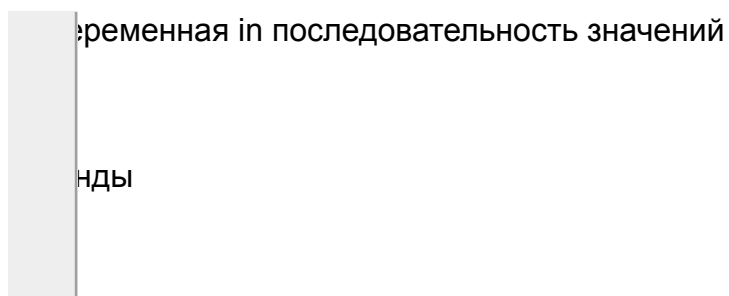
```
if [ -r somefile ]; then
    content=$(cat somefile)
elif [ -f somefile ]; then
    echo "The file 'somefile' exists but is not readable to the script."
else
    echo "The file 'somefile' does not exist."
fi
```

Циклы в Bash:

- **for** - позволяет перебрать все элементы из массива или использует переменную-счетчик для определения количества повторений;
- **while** - цикл выполняется пока условие истинно;
- **until** - цикл выполняется пока условие ложно.

### Цикл for

Цикл **for** в **bash** имеет два вида. Рассмотрим сначала классический вариант **for**. Общий вид следующий:



переменная in последовательность значений

команды

## 18. Программирование в ОС Linux: инструментарий и менеджеры пакетов, сборка.

Система управления **пакетами** — набор программного обеспечения, позволяющего управлять процессом установки, удаления, настройки и обновления различных компонентов программного обеспечения. Системы управления **пакетами** активно используются в различных дистрибутивах операционной системы **Linux** и других UNIX-подобных операционных системах.

### Категории пакетных менеджеров

- Высокоуровневые менеджеры. Применяются для поиска и скачивания пакетов из репозитория. В процессе работы могут задействовать низкоуровневые менеджеры для установки загруженных программ.
- Низкоуровневые менеджеры. Используются для установки локальных пакетов, загруженных вручную пользователем, или высокоуровневым пакетным менеджером.

### Распространенные форматы пакетов

- DEB (.deb). Самый популярный формат пакетов дистрибутива Debian и его ближайших родственников — Ubuntu, MX Linux, Pop!\_OS, elementary OS и других.
- RPM (.rpm). Разработан компанией Red Hat и внедрен в дистрибутив RHEL. Также применяется в таких системах как Fedora и CentOS.
- TAR.XZ. Стандартный тип пакетов для дистрибутива ArchLinux и его производных — Manjaro, ARCOLINUX и других.
- Ebuild (.ebuild). Скрипт bash-сценария для компиляции программ в дистрибутивах Gentoo и Calculate Linux.

DPKG (Debian Package) – система управления пакетами в Debian и дистрибутивах на его основе, например Ubuntu.

Утилита DPKG появилась в дистрибутиве Debian в 1995 году. Низкоуровневый пакетный менеджер создан только для работы с локальными DEB пакетами и не может самостоятельно разрешать зависимости, а также скачивать пакеты из репозитория.

### Особенности

- Поддерживает добавление архитектур из других дистрибутивов Linux.
- DPKG выполняет работу только с локальными пакетами.
- Под архитектуру DEB выпущено более 55000 пакетов.

APT (Advanced Packaging Tool) – консольная утилита, выполняющая роль «поисковика» и загрузчика пакетов из репозитория. Установка скачанных пакетов производится утилитой DPKG. Благодаря эффективному разрешению зависимостей, пакетный менеджер APT используется по умолчанию в дистрибутивах с архитектурой Debian и поддерживает систему в актуальном состоянии.

RPM (Red Hat Package Manager) – формат пакетов и низкоуровневый пакетный менеджер систем RED HAT (RHEL, CentOS, Fedora и др.) Как и DPKG, способен работать только с локальными файлами.

#### Особенности

- Обновление программ производится в ускоренном режиме, благодаря замене только отредактированных разработчиком элементов пакета.
- Для скачивания, обновления пакетов, а также разрешения зависимостей придётся использовать пакетные менеджеры более высокого уровня (YUM, DNF).
- Начиная с 2010 года, пакеты подписываются с хешем MD5. Это исключает вероятность изменения файла RPM злоумышленником для внедрения вирусного кода.

## 19. Архитектура ОС Android и безопасность Android-приложений.

Если представить компонентную модель Android в виде иерархии (рис.1.5), то в самом низу, в самой основе будет располагаться ядро операционной системы. Оно обеспечивает функционирование системы и отвечает за безопасность, управление памятью, энергосистемой и процессами, а также предоставляет сетевой стек и модель драйверов. Ядро также действует как уровень абстракции между аппаратным и программным обеспечением.

"Выше" ядра, как программное обеспечение промежуточного слоя, лежит набор библиотек (Libraries), предназначенный для решения типовых задач, требующих высокой эффективности. То есть, именно этот уровень отвечает за предоставление реализованных алгоритмов для вышележащих уровней, поддержку файловых форматов, осуществление кодирования и декодирования информации (в пример можно привести мультимедийные кодеки), отрисовку графики и многое другое. Библиотеки реализованы на C/C++ и скомпилированы под конкретное аппаратное обеспечение устройства, вместе с которым они и поставляются производителем в предустановленном виде.

В ядре ОС Android было решено отказаться от средств межпроцессного взаимодействия ОС Linux и вместо них создать единый механизм, названный Binder. Binder позволяет вызывать методы одного процесса из другого процесса, передавая им аргументы и получая результат



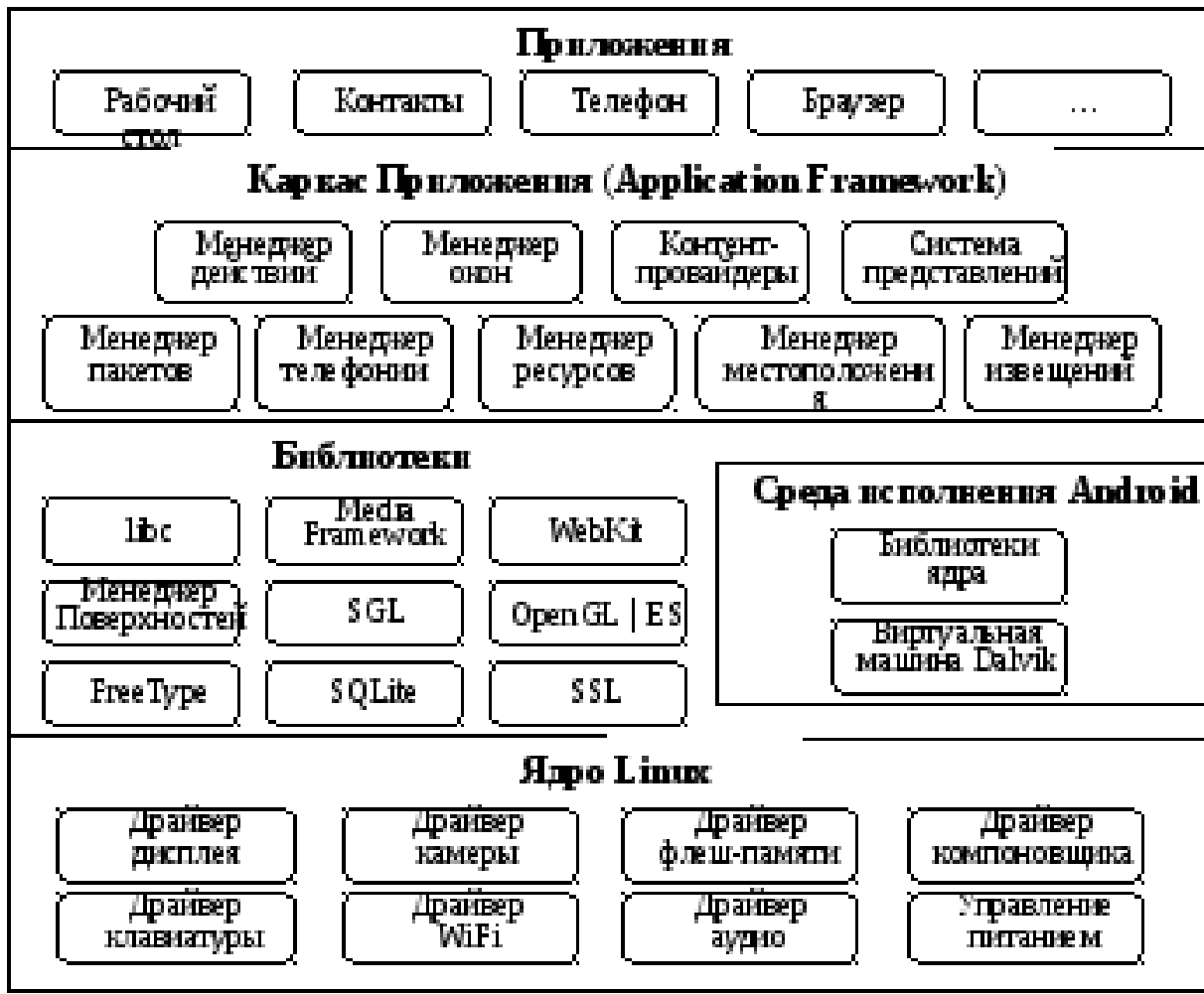
Устройство ОС Android

В основе ОС Android лежит ядро Linux с некоторыми архитектурными изменениями, которые были сделаны инженерами Google. Приложения для операционной системы Android разрабатываются на языке Java. Начиная с версии Android 1.5 был представлен набор инструментов Android NDK, который позволяет разрабатывать модули приложений на языках C и C++ и компилировать их в машинный код [4]. Приложения поставляются в виде файлов специального формата APK, который является ZIP-архивом с определенной структурой каталогов и файлов. APK-файл приложения содержит:

- манифест;
- модули, скомпилированные в машинный код (разделяемые библиотеки .so);
- различные ресурсы приложения;
- DEX-файл;
- прочие необходимые файлы.

Начиная с Android 10 появились некоторые особенности механизма добавления сертификатов в системные сертификаты. До Android 4.0 существовали только системные сертификаты, которые были встроены в систему, и пользователи не имели никакой возможности контролировать их. Единственным решением было рутануть устройство и изменить системные папки с этими сертификатами. Затем появилась возможность добавлять пользовательские сертификаты. В Android для управления доверенными данными используется TrustManager, в данном случае именно он будет проверять надежность сертификатов. То есть теперь в Android есть **системные** сертификаты (/system/etc/security/cacerts/), **пользовательские** CA сертификаты сохраняются в специализированной директории (/data/misc/user/0/cacerts-added).

20. Структура Android-приложения и основные компоненты.



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.

— **Основными компонентами** любого **Android-приложения** являются **компоненты**, управляемые средой выполнения: Активности, Сервисы, Broadcast Receiver и Content Provider. Конфигурация и взаимодействие этих **компонентов** определяют поведение приложения. Перечисленные **компоненты** имеют разные зоны ответственности и разные жизненные циклы, но все они являются точками входа в приложение.

Существуют следующие основные компоненты приложения для Android:

Действия:

```
public class MainActivity extends Activity {}
```

Сервисы:

```
public class ServiceName extends Service { }
```

Поставщики контента:

```
public class contentProviderName extends ContentProvider { public void onCreate(){} }
```

Широковещательные приемники:

Широковещательные приемники заставляют наше приложение реагировать на любое полученное намерение, что делает их идеальными для создания приложений, управляемых событиями.

Намерения:

Намерения могут использоваться для запуска и остановки действий и служб, для трансляции сообщений в масштабах всей системы или для явного действия, службы или широковещательного приемника или для запроса выполнения действия над определенной частью данных.

Виджеты:

Это небольшие визуальные компоненты приложения, которые вы можете найти на главном экране устройств.

Уведомления:

Уведомления - это оповещения приложений, которые используются для привлечения внимания пользователя к какому-либо конкретному событию приложения, не отвлекая внимания и не прерывая текущую активность пользователя.



## 21. Структура проекта Android-приложения: характеристика

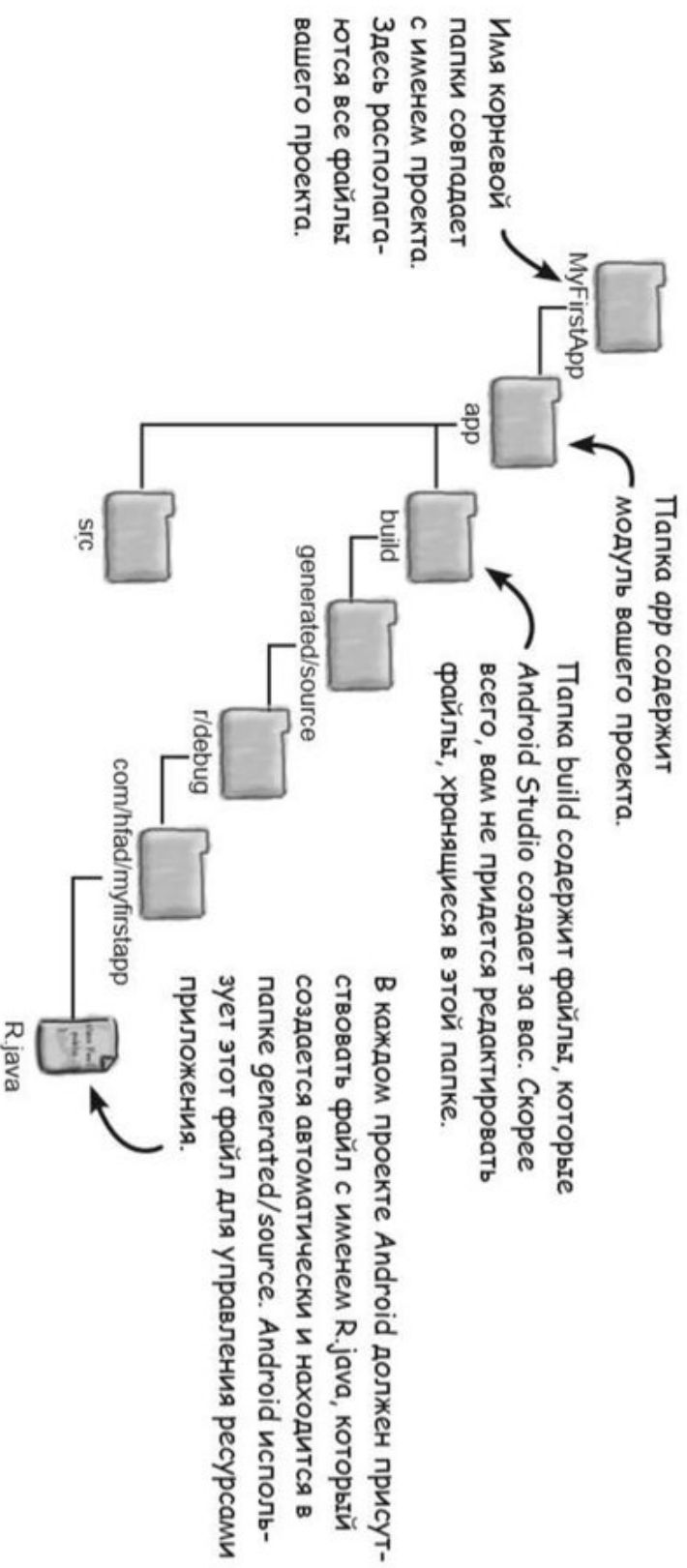
Android проект представлен 2 корневыми папками: **app** и **Gradle Scripts**. Папка **app** включает 3 подпапки:

1. Папка **manifests** содержит файлы конфигураций или файлы манифеста приложения
2. В папке **java** находится исходный код приложения.
3. Папка **res** содержит файлы используемых в Android приложении ресурсов

Основные файлы в проекте Android:

- *AndroidManifest.xml* : Это файл определения Android. Он содержит информацию о приложении Android, такую как минимальная версия Android, разрешение на доступ к возможностям устройства Android, таким как разрешение на доступ в Интернет, возможность использовать разрешение телефона и т. Д.
- *MainLayout.xml* : Этот файл описывает макет страницы. Это означает размещение каждого компонента (например, текстовых полей, меток, переключателей, пользовательских компонентов и т.д.) на экране приложения.
- Activity класс: для каждого приложения, занимающего весь экран устройства, необходим хотя бы один класс, который наследуется от Activity класса. Вызывается один из основных методов onCreate. Этот метод запускает приложение и загружает страницу макета.

# Структура проекта



## 22. Файл манифеста AndroidManifest.xml: элементы и структура.

**AndroidManifest.xml** — это мощный **файл** на платформе **Android**, который позволяет описать функциональные возможности и требования приложения к **Android**. Тем не менее, работа с ним не является простой. Xamarin.**Android** помогает свести к минимуму **эту** трудность, позволяя добавлять настраиваемые атрибуты в классы, которые затем будут использоваться для автоматического создания **манифеста**.

### Файл манифеста Android

Типичный *Android*.Файл манифеста выглядит следующим образом:

```
XML
<?xml version="1.0" encoding="utf-8"?>
<манифест xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.firstproject"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk для Android: minSdkVersion="7" />

    <приложение
        android: значок ="@drawable/ ic_launcher"
        android: ярлык ="@string/ app_name" >
        <действие
            android: имя ="MynewprojectActivity"
            android:label="@string/app_name" >
            <фильтр намерений>
                <действие android:name="android.intent.action.ГЛАВНАЯ" />
                <категория android:name="android.intent.category.ПУСКОВАЯ УСТАНОВКА" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Некоторые важные определения в этом файле:

- **Android:versionCode**: Это числовое значение. Это важный атрибут, потому что именно так Android-устройство узнает, когда нужно предупредить пользователя об обновлении приложения до более новой версии.
- **Android:sdkMinVersion**: Определяет, какая самая ранняя версия операционной системы Android доступна для этого приложения.
- **Activity** элемент: определяет, какие действия доступны в этом приложении и какие действия должны быть загружены при запуске.

AndroidManifest.xml создается как часть процесса сборки, а XML-код, найденный в свойствах или AndroidManifest.xml, объединяется с XML, созданным из пользовательских атрибутов. Результирующий объединенный AndroidManifest.xml находится в подкаталоге obj; например, он находится в obj/Debug/android/AndroidManifest.xml для сборок отладки. Процесс слияния прост: он использует настраиваемые атрибуты в коде для создания XML-элементов и *вставляет* эти элементы в AndroidManifest.xml.

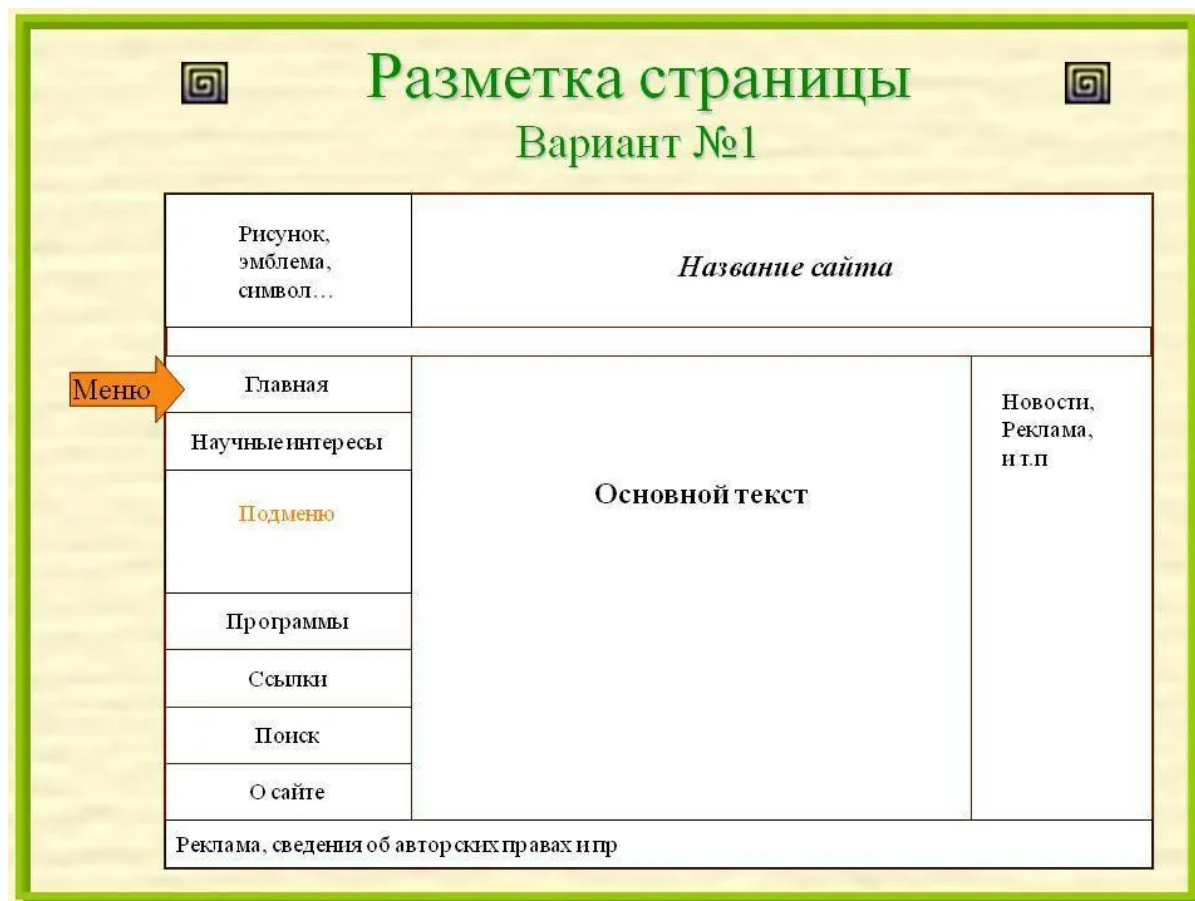
Манифест Android также предоставляет способ объявления свойств для всего приложения. Это делается с помощью <application> элемента и его аналога настраиваемого атрибута Application .

Манифест Android представляет собой XML файл и выполняет несколько функций. Вот как их описывает Google:

- Определяет имя Java-пакета приложения. Имя пакета представляет собой уникальный идентификатор приложения.
- Описывает компоненты приложения — активити, сервисы, бродкаст-ресиверы и контент-провайдеры. Определяет имена классов, реализующие каждый из компонентов и оглашает их возможности (например, какие Intent-сообщения они могут обрабатывать). Эти объявления позволяют системе Android знать, какие компоненты и при каких условиях могут быть запущены.
- Предопределяет процессы, которые будут содержать компоненты приложения.
- Объявляет разрешения, которые приложение должно иметь для доступа к защищённым частям API и взаимодействия с другими приложениями.
- Также объявляет разрешения, которые требуются для доступа к компонентам приложения.
- Перечисляет классы Instrumentation, которые предоставляют профайлинг и другую информацию во время работы приложения. Эти объявления присутствуют в манифесте только пока приложение разрабатывается и тестируется; и они удаляются перед публикацией приложения.
- Объявляет минимальный уровень Android API, который требует приложение.
- Перечисляет библиотеки, с которыми приложение должно быть связано.

## 23. Характеристика разметок (макетов) страниц, их типы и основные элементы UI.

User interface (UI) элементы - это части, которые дизайнеры используют для создания приложений или веб-сайтов. Они добавляют интерактивность в пользовательский интерфейс, предоставляя пользователю точки соприкосновения при навигации по ним. Кнопки, полосы прокрутки, пункты меню и чекбоксы.



Зачастую основными элементами страницы являются: содержащий блок (wrapper, container), логотип, навигация, контент, футер (нижний колонтитул), свободное пространство

Содержащий блок (контейнер)

Роль контейнера на странице может выполнять непосредственно элемент body или же div. Ширина содержащего блока может быть резиновой (fluid), а может быть фиксированной (fixed).

Логотип

Текстовая или графическая составляющая проекта и выделяющая его среди других. Логотип чаще всего располагается в верхнем левом углу страницы или же посередине (в зависимости от идеи, макета).

## Навигация

Основная навигационная панель содержит ссылки на основные разделы сайта. Навигационная панель часто располагается в верхней части страницы (в независимости от того вертикально или горизонтально располагаются элементы навигации).

## Контент

Контент – это основная составляющая веб-страницы. Он занимает главенствующую роль в дизайне страницы, поэтому занимает большее пространство, подкреплён, помимо текста, графикой.

## Нижний колонтитул (footer)

Данный элемент располагается внизу страницы и обычно содержит информацию о правообладателе, контактные и юридические данные, ссылки на основные разделы сайта (зачастую дублирует основную навигацию), ссылки на социальные сети, форму обратной связи и пр.

## Резиновый и фиксированный макет

### Фиксированный макет

Фиксированный макет подразумевает под собой, что в независимости от разрешения экрана пользователя ваш сайт всегда будет занимать одинаковую ширину.

### Резиновый макет

«Резиновый» макет подразумевает, что страница сайта будет стараться занять всё доступное ей пространство на экране пользователя, подстраиваясь под разрешение.

## 24. Разметка (макеты) страниц на примере `FrameLayout` и `LinearLayout`.

Существует пять стандартных типов верстки:

- `AbsoluteLayout`
- `FrameLayout`
- `LinearLayout`
- `RelativeLayout`
- `TableLayout`

### **FrameLayout**

`FrameLayout`, также известный как однокадровая компоновка, - это самый простой метод компоновки среди методов компоновки, предоставляемых Android. Он определяет пустую область на экране и заполняет ее одним объектом. Например, картинки, текст, кнопки и т. Д.

Разработчики приложений не могут указать конкретную позицию заполнения для компонентов, заполненных в `FrameLayout`. По умолчанию эти компоненты будут зафиксированы в верхнем левом углу экрана, а компоненты, размещенные позже, будут помещены на предыдущий компонент, чтобы покрыть и заполнить, чтобы сформировать деталь. Заблокировать или заблокировать все.

Разработчики могут соответствующим образом изменить положение компонента с помощью атрибута `android:layout_gravity` компонента.

### **LinearLayout**

`LinearLayout`, также известный как линейный макет, должен быть наиболее часто используемым макетом в дизайне представлений Android. Макет может сделать размещенные в нем компоненты аккуратно расположенными по горизонтали или вертикали. Атрибут `android:Ориентация` указывает конкретное расположение, а атрибут веса устанавливает пропорцию каждого компонента в макете.

Внутри верстки возможно комбинировать вертикальную и горизонтальную разбивки, а кроме того, возможна комбинация нескольких разных типов верстки например использование `LinearLayout` внутри `FrameLayout`.

## 25. Разметка (макеты) страниц на примере RelativeLayout и ConstraintLayout.

RelativeLayout — тип верстки при котором позиционирование элементов происходит относительно друг друга и относительно главного контейнера.

За то, каким образом будут позиционироваться элементы отвечают следующие атрибуты:

Атрибуты позиционирования относительно контейнера

- android:layout\_alignParentBottom – Низ элемента находится внизу контейнера
- android:layout\_alignParentLeft – Левая часть элемента прилегает к левой части контейнера
- android:layout\_alignParentRight – Правая часть элемента прилегает к правой части контейнера
- android:layout\_alignParentTop – Элемент находится в верхней части контейнера
- android:layout\_centerHorizontal – Элемент позиционируется по центру относительно горизонтального размера контейнера
- android:layout\_centerInParent – Элемент позиционируется по центру относительно горизонтального и вертикального размеров размера контейнера
- android:layout\_centerVertical – Элемент позиционируется по центру относительно вертикального размера контейнера

Атрибуты позиционирования относительно других элементов.

В качестве значений этих атрибутов ставятся id элемента относительно которого будет производится позиционирование.

android:layout\_above – Располагает элемент над указанным

android:layout\_below – Располагает элемент под указанным

android:layout\_toLeftOf – Располагает элемент слева от указанного

android:layout\_toRightOf – Располагает элемент справа от указанного

ConstraintLayout представляет **контейнер, который позволяет создавать гибкие и масштабируемые визуальные интерфейсы.**

Для позиционирования элемента внутри ConstraintLayout необходимо указать ограничения (constraints). Есть несколько типов ограничений. В частности, для установки позиции относительно определенного элемента используются следующие ограничения:

layout\_constraintLeft\_toLeftOf: левая граница позиционируется относительно левой границы другого элемента

layout\_constraintLeft\_toRightOf: левая граница позиционируется относительно правой границы другого элемента



`layout_constraintRight_toLeftOf`: правая граница позиционируется относительно левой границы другого элемента

`layout_constraintRight_toRightOf`: правая граница позиционируется относительно правой границы другого элемента

`layout_constraintTop_toTopOf`: верхняя граница позиционируется относительно верхней границы другого элемента

`layout_constraintTop_toBottomOf`: верхняя граница позиционируется относительно нижней границы другого элемента

`layout_constraintBottom_toBottomOf`: нижняя граница позиционируется относительно нижней границы другого элемента

`layout_constraintBottom_toTopOf`: нижняя граница позиционируется относительно верхней границы другого элемента

`layout_constraintBaseline_toBaselineOf`: базовая линия позиционируется относительно базовой линии другого элемента

`layout_constraintStart_toEndOf`: элемент начинается там, где завершается другой элемент

`layout_constraintStart_toStartOf`: элемент начинается там, где начинается другой элемент

`layout_constraintEnd_toStartOf`: элемент завершается там, где начинается другой элемент

`layout_constraintEnd_toEndOf`: элемент завершается там, где завершается другой элемент.

Если элементы расположены по центру, `ConstraintLayout` позволяет их сдвигать по горизонтали и по вертикали. Для сдвига по горизонтали применяется атрибут `layout_constraintHorizontal_bias`, а для сдвига по вертикали - атрибут `layout_constraintVertical_bias`.

## 26. Разметка (макеты) страниц на примере `TableLayout` и `GridLayout`.

**TableLayout** — табличная верстка. Организует элементы в строки и столбцы таблицы. Для организации строк служит тег `Alternate Layouts`.

Макет, который упорядочивает свои дочерние элементы по строкам и столбцам. `TableLayout` состоит из нескольких `TableRow` объектов, каждый из которых определяет строку (на самом деле, у вас могут быть другие дочерние элементы, о которых будет рассказано ниже). Контейнеры `TableLayout` не отображают границы для своих строк, столбцов или ячеек. Каждая строка содержит ноль или более ячеек; каждая ячейка может содержать один `View` объект. В таблице столько столбцов, сколько строк с наибольшим количеством ячеек. Таблица может оставлять ячейки пустыми. Ячейки могут охватывать столбцы, как и в HTML.

Класс **GridLayout** — это менеджер компоновки, который раскладывает компоненты контейнера по прямоугольной сетке. Контейнер делится на прямоугольники одинакового размера, и в каждый прямоугольник помещается по одному компоненту. Например, ниже приведен апплет, который раскладывает шесть кнопок на три строки и два столбца

Менеджер компоновки сетки определяет размер отдельных компонентов, разделяя свободное пространство в контейнере на равные по размеру части в соответствии с количеством строк и столбцов в макете. Свободное пространство контейнера равно размеру контейнера за вычетом любых насекомых и любого заданного горизонтального или вертикального зазора. Всем компонентам в макете сетки присваивается одинаковый размер.

## 27. Разметка (макеты) страниц на примере ScrollView, ListView и GridView.

В Android ScrollView-это группа представлений, которая используется для создания вертикально прокручиваемых представлений. Представление прокрутки содержит только один прямой дочерний элемент. Чтобы разместить несколько представлений в представлении прокрутки, нужно создать группу представлений(например, LinearLayout) в качестве прямого дочернего элемента, а затем мы можем определить внутри нее множество представлений.

ScrollView определяет следующие свойства:

- Contentc типом Viewпредставляет содержимое, отображаемое в ScrollView.
- ContentSizec типом Sizeпредставляет размер содержимого. Это свойство доступно только для чтения.
- HorizontalScrollBarVisibiltyc типом ScrollBarVisibilityпредставляет, когда отображается горизонтальная полоса прокрутки.
- Orientationc типом ScrollOrientationпредставляет направление прокрутки ScrollViewобъекта . Значение по умолчанию этого свойства равно Vertical.
- ScrollXc типом doubleуказывает текущую позицию прокрутки X. Значение по умолчанию этого свойства только для чтения равно 0.
- ScrollYc типом doubleуказывает текущую позицию прокрутки по оси Y. Значение по умолчанию этого свойства только для чтения равно 0.
- VerticalScrollBarVisibiltyc типом ScrollBarVisibilityпредставляет, когда видима вертикальная полоса прокрутки.

Класс ListView — это специализированный **класс спискового типа, предназначенный для отображения различных представлений одних и тех же данных**. Он особенно удобен, когда требуется создать представление, состоящее из множества столбцов и отображающее о каждом элементе данных несколько различных фрагментов информации.

Класс ListView унаследован от ListBox и дополняет его одной единственной деталью: свойством View.

**GridView** — исключительно гибкий табличный элемент управления, предназначенный для демонстрации данных в виде двумерной сетки (**grid**), или экранной таблицы, состоящей из строк и столбцов.

Столбцы GridView представлены объектами GridViewColumn, размер которых может автоматически изменяться в соответствии с содержимым. При необходимости для GridViewColumn можно задать явным образом определенную ширину.

## 28. Ресурсы Android-приложения: строковые, булевы, числовые, меню, ресурсы разметки.

**Ресурс в приложении Android** представляет собой файл, например, файл разметки интерфейса или некоторое значение, например, простую строку. То есть **ресурсы** представляют собой и файлы разметки, и отдельные строки, и звуковые файлы, файлы изображений и т.д. Все **ресурсы** находятся в проекте в каталоге `res`.

**Строковые ресурсы** представляются в виде пар имя/значение, где имя — **строка**, определяющая **ресурс**, а значение — **строка ресурса**, которая возвращается при передаче имени методу извлечения **ресурсов**, например, `ResourceManager.GetString`. Имя и значение должны быть разделены знаком равенства (=).

Можно также хранить в ресурсах булевы значения `true` или `false` в файле с произвольным именем в папке `res/values`.

В файле с корневым элементом `<resources>` вы определяете элемент `bool` с нужным значением. У элемента есть атрибут `name` - строка, определяющая имя булевого ресурса.

В ресурсах можно хранить числа типа `Integer`. Хранить можно в произвольном имени XML-файла в папке `res/values/` в корневом элементе `<resources>`

У элемента `<integer>` есть атрибут `name`, определяющий имя числового значения.

### Ресурсы меню

Меню, описанное в формате XML, загружается в виде программного объекта с помощью метода `inflate`, принадлежащего сервису `MenuInflater`. Как правило, это происходит внутри обработчика `onOptionsItemSelected` (смотри урок Меню).

Описание меню хранится в отдельном файле в каталоге `res/menu`. Имена файлов без расширения автоматически становятся идентификаторами ресурсов.

В XML-файле меню есть три элемента:

- `<menu>` — корневой элемент файла меню;
- `<group>` — контейнерный элемент, определяющий группу меню;
- `<item>` — элемент, определяющий пункт меню

Элементы `<item>` и `<group>` могут быть дочерними элементами `<group>`.

Элемент `item` может иметь несколько атрибутов:

**id** - Идентификатор пункта меню, по которому приложение может распознать при выделении пункта меню пользователем

**title** - Текст, который будет выводиться в меню

### icon

Значок для пункта меню. Можно использовать графический ресурс ресурсы разметки, которые отвечают за внешний вид приложения. Данные ресурсы представлены в формате XML. Ресурс разметки формы (`layout resource`) - это ключевой тип ресурсов, применяемый при программировании пользовательских интерфейсов в Android. Каждый ресурс, описывающий разметку, хранится в отдельном файле каталога `res/layout`. Имя файла без расширения выступает как идентификатор ресурса.

## 29. Ресурсы Android-приложения: цветовые, размеров, визуальных стилей и тем, drawable.

Для работы со цветом используется тег `<color>`, а цвет указывается в специальных значениях.

- `#RGB`;
- `#RRGGBB`;
- `#ARGB`;
- `#AARRGGBB`;

Также существуют predefined названия цветов. Такие ID доступны в пространстве имен `android.R.color`.

В Android используются следующие единицы измерения: пиксели, дюймы, точки. Все они могут входить в состав XML-шаблонов и кода Java. Данные единицы измерения также можно использовать в качестве ресурсов при помощи тега `<dimen>` (обычно используют файл `dimens.xml`): Метод Java использует в названии целое слово `Dimension`, а в коде и в XML используется сокращённая версия `dimen`.

Ресурсы со стилями позволяют поддерживать единство внешнего вида приложения. Чаще всего визуальные стили и темы используются для хранения цветовых значений и шрифтов. Обычно используют файл `styles.xml`.

Вместо описания каждого стиля, вы можете использовать ссылки, предоставляемые Android, с помощью которых вы можете использовать стили из текущей темы.

Ресурсы в Android являются декларативными. В основном ресурсы хранятся в виде XML-файлов в каталоге `res` с подкаталогами `values`, `drawable-ldpi`, `drawable-mdpi`, `drawable-hdpi`, `layout`, но также бывают и другие типы ресурсов.

### 30. Ресурсы Android-приложения: отрисовываемых объектов, mipmap, анимации, необработанных объектов.

Кроме обычных изображений в Android поддерживается и другой тип отрисовываемых ресурсов, которые называются отрисовываемыми цветами (color-drawable). По сути, это просто цветные прямоугольники.

Чтобы задать цветной прямоугольник, нужно указать тег <drawable> в имени узла XML-файла, находящегося в подкаталоге res/values/.

#### Mipmap

С появлением Nexus 6 и Nexus 9, Гугл добавила в Android новые типы ресурсов - Mipmap. По сути, это замена ресурсам Drawable для значков приложения. Нужно подготовить значки и расположить их в папках mipmap-mdpi, mipmap-hdpi и т.д. Основная идея заключается в том, что при компиляции неиспользуемые drawable-ресурсы могут быть удалены в целях оптимизации. Перенос значков приложения в новые папки с разными разрешениями позволяет избежать потенциальной проблемы с удалением нужных файлов.

Android поддерживает два типа анимации. Первый тип основан на расчете промежуточных кадров и может использоваться для поворота, перемещения, растягивания, затемнения элементов. Второй тип - пошаговая анимация, т.е. последовательный вывод заранее подготовленных изображений.

При использовании анимации промежуточных кадров каждый экземпляр анимации хранится в отдельном XML-файле внутри каталога res/anim. Имена файлов без расширения являются идентификаторами для ресурсов.

Анимацию можно задать в виде изменений параметров alpha (затемнение), scale (масштабирование), translate (перемещение) или rotate (вращение).

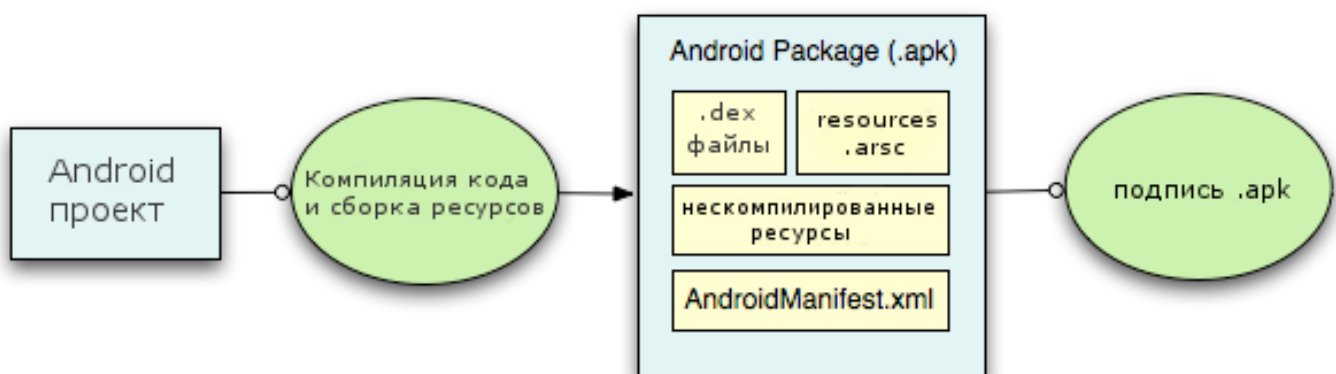
Если разместить файлы, в том числе написанные на XML, в каталоге res/raw, они не будут скомпилированы в двоичном формате, а попадают в пакет прикладных программ как есть. Для считывания таких файлов нужно использовать явные API с поддержкой потоков. К категории raw относятся аудио- и видеофайлы.

### 31. Ресурсы Android-приложения: создание псевдонимов ресурсов, компиляция ресурсов, доступ из кода

Чтобы избежать дублирования ресурсов, можно использовать псевдонимы, которые будут ссылаться на один и тот же ресурс. Предположим вы создали два файла `res/layout-land/activity_main.xml` и `res/layout-large/activity_main.xml` с одинаковым содержанием для разметки с альбомной ориентацией для смартфонов и планшетов. Создайте теперь ещё один файл с таким же содержанием, например под именем `res/layout/activity_main_horizontal.xml`. Теперь два одинаковых файла можете удалить. Вместо них создайте два файла `res/values-land/refs.xml` и `res/values-large/refs.xml`.

```
<resources>
  <item type="layout" name="activity_main">@layout/activity_main_horizontal</item>
</resources>
```

В общих чертах процесс сборки приложения выглядит так:



#### [Получение доступа к ресурсу в коде Java]

Вы можете использовать ресурс в коде путем передачи resource ID методу (функции) как параметр. Например, Вы можете установить `ImageView` для использования ресурса картинки `res/drawable/myimage.png`, используя `setImageResource()`:

```
ImageView imageView = (ImageView) findViewById(R.id.myimageview);
imageView.setImageResource(R.drawable.myimage);
```

Вы также можете запросить отдельные ресурсы с использованием методов в классе `Resources`, экземпляр которого можно получить вызовом `getResources()`.

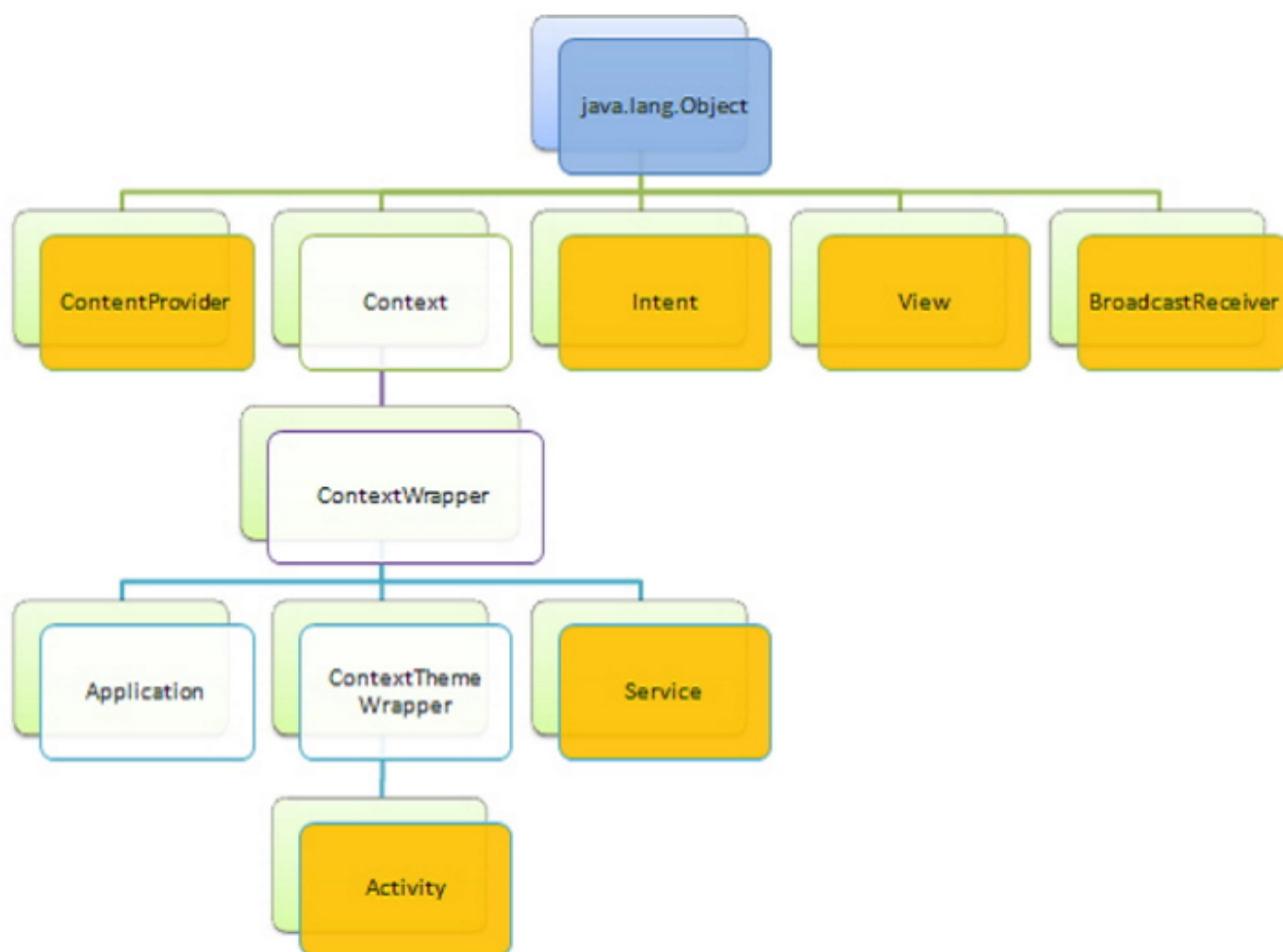
## 32. Иерархия классов в Android SDK.

Из чего состоит Android-приложение

- Java-классов, являющихся подклассами основных классов из Android SDK (View, Activity, ContentProvider, Service, BroadcastReceiver, Intent) и Java-классов, у которых нет родителей в Android SDK.
- Манифеста приложения.
- Ресурсов, на подобии строк, изображений и т.д. и т.п.
- Файлов.

Java классы

На следующей диаграмме представлена иерархия основных классов из Android SDK, с которыми предстоит иметь дело разработчику:





Интерфейс Android приложения представляет собой иерархию UI компонентов (см. рис. 2), можно описать эту иерархию программно, но более простым и эффективным способом задать расположение элементов интерфейса является XML файл, который предоставляет удобную для восприятия структуру компоновки (layout file).

Во время исполнения XML файл автоматически превращается в дерево соответствующих объектов.

View — базовый класс для всех виджетов пользовательского интерфейса (GUI widgets).

Класс Activity и его подклассы содержат логику, лежащую за пользовательским интерфейсом.

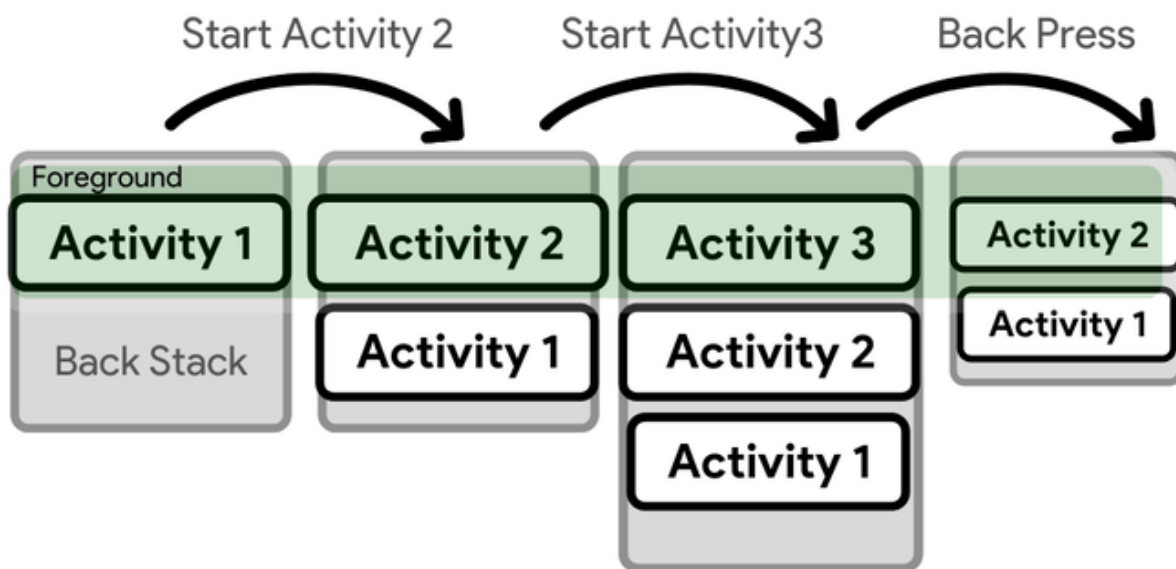
Класс ContentProvider и его подклассы представляют model в архитектуре MVVM.

Класс BroadcastReceiver и его подклассы представляют собой «подписчика» реализованного в архитектуре Android, в механизме взаимодействия издатель/подписчик.

Манифест Android — ещё одна важная часть Android-приложения.

### 33. Задачи и стек переходов назад в мобильных приложениях для ОС Android

При выполнении задания пользователи взаимодействуют с задачей, которая представляет собой набор действий. Действия складываются в том порядке, в котором они были открыты, в стек, называемый обратным стеком. Например, одно действие в почтовом приложении может отображать список свежих сообщений. Когда пользователь выбирает сообщение, появляется новое действие, в котором пользователь может прочитать сообщение. Это новое действие было отправлено в конец очереди. Когда пользователь нажимает кнопку "Назад", новое действие завершается и удаляется из стека. Когда множество приложений запущено в многооконной среде, что **разрешено в Android 7.0 (уровень API 24) и выше**, система поддерживает задачи для каждого окна по отдельности; каждое окно может содержать несколько задач. Система организует задачи или группы задач для каждого окна для приложений Android, работающих на Chromebook.



Для планирования задач на Android доступно несколько API:

- Alarm Manager
- Job Scheduler
- GCM Network Manager
- Firebase Job Dispatcher
- Sync Adapter
-

## Alarm Manager

AlarmManager обеспечивает доступ к службам уведомлений. Это дает возможность выполнять любые операции за пределами жизненного цикла вашего приложения. Таким образом, вы можете инициировать события или действия, даже если ваше приложение не запущено. AlarmManager может запустить сервис в будущем.

## Job Scheduler

Этот API позволяет выполнять задания, когда у устройства больше доступных ресурсов или при соблюдении правильных условий. Все условия могут быть определены при создании задания.

## GCM Network Manager

GCM (Google Cloud Messaging) Network Manager имеет все функции расписания из JobScheduler. GCM Network Manager также предназначен для выполнения многократной или одноразовой, неминуемой работы при сохранении времени автономной работы. Он используется для поддержки обратной совместимости и может также использоваться под Android 5.0 (API уровня 21).

## Firebase Job Dispatcher

Эта библиотека также будет работать, если на устройстве нет установленных сервисов Google Play. В этом состоянии эта библиотека внутренне использует AlarmManager. Если на устройстве доступно приложение Google Play, он использует механизм планирования в службах Google Play.

## Sync Adapter

Sync adapters разработаны специально для синхронизации данных между устройством и облаком. Он должен использоваться только для этого типа задач. Синхронизация может быть вызвана изменениями данных в облаке или на устройстве или по истекшему времени.

## 34. Фрагменты и управление фрагментами

Существует два основных подхода в использовании фрагментов.

Первый способ основан на замещении родительского контейнера. Создаётся стандартная разметка и в том месте, где будут использоваться фрагменты, размещается контейнер, например, `FrameLayout`. В коде контейнер замещается фрагментом. При использовании подобного сценария в разметке не используется тег `fragment`, так как его нельзя менять динамически. Также вам придётся обновлять `ActionBar`, если он зависит от фрагмента.

Второй вариант - используются отдельные разметки для телефонов и планшетов, которые можно разместить в разных папках ресурсов. Например, если в планшете используется двухпанельная разметка с двумя фрагментами на одной активности, мы используем эту же активность для телефона, но подключаем другую разметку, которая содержит один фрагмент. Когда нам нужно переключиться на второй фрагмент, то запускаем вторую активность.

Второй подход является наиболее гибким и в целом предпочтительным способом использования фрагментов. Активность проверяет в каком режиме (свои размеры) он запущен и использует разную разметку из ресурсов. Графически это выглядит следующим образом.

Основные классы

Сами фрагменты наследуются от `androidx.fragment.app.Fragment`. Существует подклассы фрагментов: `ListFragment`, `DialogFragment`, `PreferenceFragment`, `WebViewFragment` и др. Не исключено, что число классов будет увеличиваться, например, появился ещё один класс `MapFragment`.

Для взаимодействия между фрагментами используется класс `android.app.FragmentManager` - специальный менеджер по фрагментам.

`FragmentManager`

Класс `FragmentManager` имеет два метода, позволяющих найти фрагмент, который связан с активностью:

**`findFragmentById(int id)`**

Находит фрагмент по идентификатору

**`findFragmentByTag(String tag)`**

Находит фрагмент по заданному тегу

Методы транзакции

Мы уже использовали некоторые методы класса `FragmentManager`. Познакомимся с ними поближе

**`add()`** Добавляет фрагмент к активности

**`remove()`** Удаляет фрагмент из активности

**`replace()`** Заменяет один фрагмент на другой

**`hide()`** Прячет фрагмент (делает невидимым на экране)

**`show()`** Выводит скрытый фрагмент на экран

**`detach()` (API 13)** Отсоединяет фрагмент от графического интерфейса, но экземпляр класса сохраняется

**`attach()` (API 13)** Присоединяет фрагмент, который был отсоединён методом `detach()`

### 35. Намерения в Android-приложении: класс Intent и методы класса.

Намерение (Intent) - это механизм для описания одной операции - выбрать фотографию, отправить письмо, сделать звонок, запустить браузер и перейти по указанному адресу. В Android-приложениях многие операции работают через намерения.

Намерения могут применяться для трансляции сообщений по системе. Любое приложение способно зарегистрировать широковещательный приёмник и отслеживать эти намерения с возможностью на них реагировать. Это позволяет создавать приложения, использующие событийную модель, в основе которой лежат внутренние, системные или сторонние события, передаваемые внешними программами.

Android транслирует намерения для объявления о системных событиях, например об изменениях в состоянии сетевого подключения или в уровне заряда батареи. Системные приложения в Android, такие как программы дозвона или управления SMS, регистрируют компоненты, отслеживающие заданные намерения, например *входящий звонок* или *получено новое SMS-сообщение*, и соответствующим образом реагируют на них.

Объект Intent содержит информацию, представляющую интерес для компонента, который получает намерение, и данные, которые передаются этому компоненту. Кроме того, объект Intent содержит информацию, представляющую интерес для системы Android, — имя компонента, который должен обработать намерение и набор параметров запуска этого компонента.

Для работы с категориями в классе Intent определена группа методов:

- `addCategory()` — помещает категорию в объект Intent
- `removeCategory()` — удаляет категорию, которая была добавлена ранее
- `getCategories()` — получает набор всех категорий, находящихся в настоящее время в объекте Intent

### 36. Операции в Android-приложении: класс Activity и методы класса.

**Activity** — это отдельный экран в **Android**. Это как окно в приложении для рабочего стола, или фрейм в программе на Java. **Activity** позволяет вам разместить все ваши компоненты пользовательского интерфейса или виджеты на этом экране. Важно понимать, что у **Activity** есть жизненный цикл, проще говоря, это означает, что она может быть в одном из различных стадий, в зависимости от того, что происходит с приложением при действиях пользователя.

**Activity** — это отдельный экран в **Android**. Это как окно в приложении для рабочего стола, или фрейм в программе на Java. **Activity** позволяет вам разместить все ваши компоненты пользовательского интерфейса или виджеты на этом экране. Важно понимать, что у **Activity** есть жизненный цикл, проще говоря, это означает, что она может быть в одном из различных стадий, в зависимости от того, что происходит с приложением при действиях пользователя.

Android SDK включает набор классов, наследованных от Activity. Они предназначены для упрощения работы с виджетами, которые часто встречаются в обычном пользовательском интерфейсе. Перечислим некоторые из них (наиболее полезные).

- **MapActivity**. Инкапсулирует обработку ресурсов, необходимых для поддержки элемента MapView внутри Активности.
- **ListActivity**. Обертка для класса Activity, главная особенность которой — виджет ListView, привязанный к источнику данных, и обработчики, срабатывающие при выборе элемента из списка.
- **ExpandableListActivity**. То же самое, что и ListActivity, но вместо ListView поддерживает ExpandableListView.
- **TabActivity**. Позволяет разместить несколько Активностей или Представлений в рамках одного экрана, используя вкладки для переключения между элементами.

методы - [http://developer.alexanderklimov.ru/android/theory/activity\\_methods.php](http://developer.alexanderklimov.ru/android/theory/activity_methods.php)

### 37. Сервисы в Android-приложении: класс `Service` и методы класса.

Службы (Сервисы) в Android работают как фоновые процессы и представлены классом `Android.app.Service`. Они не имеют пользовательского интерфейса и нужны в тех случаях, когда не требуется вмешательства пользователя. Сервисы работают в фоновом режиме, выполняя сетевые запросы к веб-серверу, обрабатывая информацию, запуская уведомления и т.д. Служба может быть запущена и будет продолжать работать до тех пор, пока кто-нибудь не остановит её или пока она не остановит себя сама.

Реализуя эти методы обратного вызова в своей службе, вы можете контролировать жизненные циклы службы. В полном жизненном цикле службы существует два вложенных цикла:

- полная целая жизнь службы — промежуток между временем вызова метода `onCreate()` и временем возвращения `onDestroy()`. Подобно активности, для служб производят начальную инициализацию в `onCreate()` и освобождают все остающиеся ресурсы в `onDestroy()`
- активная целая жизнь службы — начинается с вызова метода `onStartCommand()`. Этому методу передаётся объект `Intent`, который передавали в метод `startService()`.

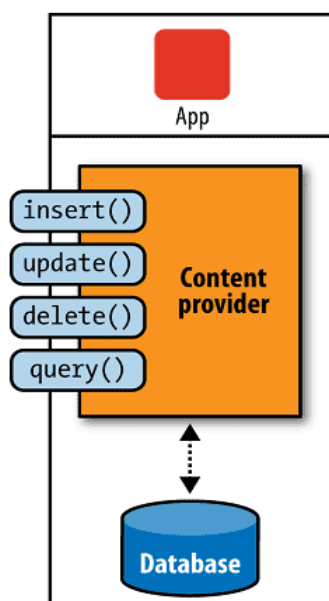
Из своего приложения службу можно запустить вызовом метода `Context.startService()`, остановить через `Context.stopService()`. Служба может остановить сама себя, вызывая методы `Service.stopSelf()` или `Service.stopSelfResult()`.

### 38. Контент-провайдеры. Класс **ContentProvider** и методы класса.

Контент-провайдер или "Поставщик содержимого" (Content Provider) - это оболочка (wrapper), в которую заключены данные. Если ваше приложение использует базу данных SQLite, то только ваше приложение имеет к ней доступ. Но бывают ситуации, когда данные желательно сделать общими. Простой пример - ваши контакты из телефонной книги тоже содержатся в базе данных, но вы хотите иметь доступ к данным, чтобы ваше приложение тоже могло выводить список контактов. Так как вы не имеете доступа к базе данных чужого приложения, был придуман специальный механизм, позволяющий делиться своими данными всем желающим.

Поставщик содержимого применяется лишь в тех случаях, когда вы хотите использовать данные совместно с другими приложениями, работающих в устройстве. Но даже если вы не планируете сейчас делиться данными, то всё-равно можно подумать об реализации этого способа на всякий случай.

Основные **методы** — это Query (запрос), Insert (вставка), Update (обновление) and Delete (удаление). ContentResolver — статический прокси-сервер, взаимодействующий с **ContentProvider** данными, либо из одного приложения, либо из другого приложения.





### 39. Приёмники широковещательных сообщений (Broadcast Receivers): основные классы и методы.

Широковещательные сообщения делают приложение более открытым; передавая события, использующие сообщения, вы открываете компоненты своего приложения для сторонних приложений, и сторонние разработчики реагируют на события без необходимости изменять ваше оригинальное приложение. В своём приложении вы можете прослушивать широковещательные сообщения других приложений, заменить или улучшить функциональность собственного (или стороннего) приложения или реагировать на системные изменения и события приложений.

Класс `BroadcastReceiver` является базовым для класса, в котором должны происходить получение и обработка сообщений, посылаемых клиентским приложением с помощью вызова метода `sendBroadcast()`.

Зарегистрировать экземпляр класса `BroadcastReceiver` можно динамически в коде или статически в манифесте.

- Динамическая регистрация происходит с помощью метода `Context.registerReceiver()`.
- Перед этим создаётся класс, расширяющий базовый класс `BroadcastReceiver` и реализуется метод обратного вызова `onReceive()` обработчика событий.
- Для отмены регистрации используется метод `unregisterReceiver()` в контексте приложения, передавая ему в качестве параметра экземпляр широковещательного приёмника

Есть три способа отправки трансляций:

- Порядковые сообщения о намерениях (Ordered broadcasts), которые посылаются методом `Context.sendOrderedBroadcast()`
- Нормальные сообщения о намерениях (Normal broadcasts) — посылаемые вызовом метода `context.sendBroadcast()` и являющиеся полностью асинхронными.
- Метод `LocalBroadcastManager.sendBroadcast()` отправляет широковещательные сообщения только получателям, определённым в вашем приложении, и не выходит за рамки вашего приложения

## 40. Рекомендации по разработке и проектированию интерфейсов мобильных приложений.

О чем же следует помнить при разработке интерфейса мобильного приложения?

1

Размер экрана. Очевидно, что экран телефона меньше экрана компьютера или ноутбука. На дисплее мобильного устройства может поместиться гораздо меньше элементов, поэтому при разработке дизайна важно продумать, что это будет, а также упростить навигацию для пользователя.

2

Кликабельность. Редко кто сегодня использует стилусы. Поэтому все кликабельные элементы должны быть такого размера, чтобы пользователь мог легко нажать на каждый из них пальцем. Если для этого ему придется увеличить страницу или неоднократно пытаться попасть по кнопке, вряд ли он останется доволен приложением.

3

Трафик и производительность. Все знают так называемые тяжелые сайты, которые очень неудобно смотреть с телефона — страницы долго грузятся, возникают ошибки. Все, что проектируется для мобильного устройства, должно быть легким — и приложения в том числе. Во-первых, тяжелые файлы много весят, и это может оказаться дорого для аудитории. Во-вторых, все та же скорость работы, которая очень важна. Также стоит следить, чтобы количество обращений к API не снижало общую производительность сервера.

4

Ориентация страницы. Большую часть времени (около 94%) пользователь держит устройство вертикально. И тем не менее, важно продумать, как будет выглядеть интерфейс при повороте телефона в горизонтальную позицию — это не должно повлиять на удобство.

5

Управление жестами. То есть можно отключить стандартные кнопки типа «вперед», «назад» и т. д., и все эти команды будут выполняться с помощью определенных жестов. Это отличительная особенность всех современных мобильных устройств, и ее нужно использовать при разработке мобильного приложения.

6

Камера и микрофон. Ими оснащены все гаджеты. Чаще всего их используют для обеспечения безопасности устройства: помимо стандартного ввода пароля, можно сделать распознавание лица или голоса. Стоит подумать и о других вариантах применительно к конкретному мобильному приложению.

## Этапы разработки mobile design

Существуют два основных этапа работы над дизайном:

1

UX, или User experience — это разработка алгоритма, понимание того, как пользователь будет взаимодействовать с приложением. Иными словами, это некий каркас, архитектура ресурса.

2

UI, или User Interface Design. UI дизайн определяет внешний вид, удобство и эстетику интерфейса.

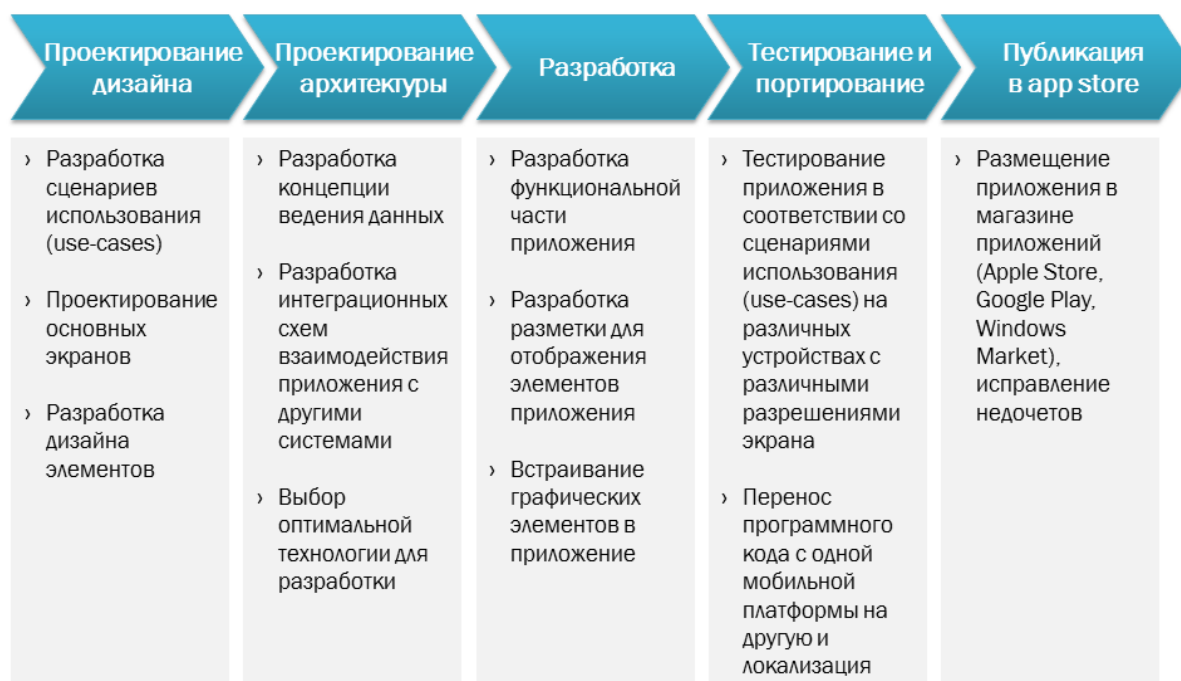
Если говорить простым языком, то UX дизайн отвечает за то, как система будет работать, а UI — за то, как все это будет выглядеть. Оба этапа неразрывно связаны между собой, и очень часто всю работу выполняет один человек, особенно в небольших дизайнерских студиях.

## 41. Основы разработки многооконных приложений для ОС Android.

### Основы разработки многооконных Android-приложений

Для мобильных приложений главным ограничением является размер экрана устройства. Очень часто невозможно разместить все элементы полнофункционального приложения так, чтобы их можно было увидеть одновременно. Очевидным решением этой проблемы является разделение интерфейса на части по какому-либо принципу. Основные пути решения этой проблемы:

- Использовать различные сообщения (диалоговые окна, уведомления, всплывающие подсказки). Этот способ наиболее прост и не требует редактирования файла манифеста, однако очевидно, что так можно решить только часть задач.
- Использовать в одном приложении несколько активностей. Способ универсальный и подходит для любых приложений, однако прежде чем его реализовывать, необходимо очень хорошо продумать структуру будущего приложения. Здесь требуется редактировать манифест и организовать переключение между различными активностями удобным для пользователя способом.
- Разместить компоненты на активности таким образом, что в нужный момент можно будет легко переключиться на работу с другой частью интерфейса.



### Диалоговые окна

Диалог - это небольшое окно, позволяющее пользователю получить или ввести дополнительную информацию. Диалоговое окно занимает только часть экрана и обычно используется в модальном режиме.

В ОС Android можно выделить три вида диалоговых окон:

- **Класс Dialog и его производные.** Помимо традиционного набора диалоговых окон, он содержит несколько дополнительных вариантов, в которых используются возможности сенсорного интерфейса. Диалоги этого типа не создают новых активностей и их не нужно регистрировать в файле манифеста (см. следующие разделы лекции), что существенно упрощает разработку. Однако они работают в модальном режиме и требуют немедленного ответа пользователя, поэтому для простого информирования рекомендуется использовать сообщения следующих двух типов.
- **Уведомления (notifications).** Это сообщения, которые отображаются в верхней панели в области уведомлений. Для того чтобы прочитать это сообщение, необходимо на домашнем экране потянуть вниз верхнюю шторку. Пользователь может это сделать в любой момент времени, следовательно, уведомления стоит использовать, когда сообщение является важным, однако не требует немедленного прочтения и ответа.
- **Всплывающие подсказки (toasts).** Сообщения, которые появляются прямо на экране приложения, перекрывая его интерфейс, и через некоторое время (обычно несколько секунд) автоматически пропадают. Их рекомендуется использовать для простых уведомлений, не требующих ответа пользователя, но важных для продолжения его работы.

## 42. Сенсорное управление в смартфонах.

**Управление сенсорным** экраном заключается в том, что он реагирует на прикосновение пальцами или стилусом, и, если надо, повторяет в точности эти движения. Если устройство распознает прикосновения или движения как команды, то они выполняются именно как команды. Задача пользователя простая – освоить нажатия на **сенсорный** экран так, чтобы они приводили к выполнению тех команд, которые будут понятны устройству и которые нужны пользователю.

### 1. Касание (нажатие)

Касание (или нажатие) — это основное действие, которое чаще всего применяется на сенсорном экране.

Касание необходимо во многих случаях:

для включения функций,  
для запуска какого-либо приложения,  
активации того или иного параметра, элемента,  
для выбора нужного значка или соответствующего параметра в списке,  
для ввода текста

### 2. Двойное касание

Двойное касание (или двойное нажатие) применяется для быстрого масштабирования, для запуска каких-то приложений и других действий.

### 3. Касание с удержанием

Нажатие (касание) с удержанием используется для того, чтобы открыть дополнительные опции, если они есть.

### 4. Смахивание (пролистывание)

Смахивание (пролистывание) применяется для переключения между страницами на Рабочем столе, для прокрутки различных списков

### 5. Перетягивание (перетаскивание)

Перетягивание (или перетаскивание) требуется для перемещения приложений на экране, а также для переноса папок, значков и т.п.

### 6. Сведение и разведение пальцев

Простая и полезная функция для изменения масштаба того, что отображается на экране: картинка, текст, карта, схема и т.п

### 7. Изменение ориентации экрана

Книжная (вертикальная) ориентация удобна, чтобы читать книги. Альбомная (горизонтальная) ориентация хорошо подойдет при просмотре видеороликов и различных карт.

### 43. Хранение данных в ОС Android на примере SharedPreferences и DataStore.

Хранилище **данных** — это цифровая система **хранения**, которая выполняет объединение и согласование больших объемов **данных** из разных источников. Она предоставляет **данные** для бизнес-аналитики, отчетов и анализа, а также обеспечивает поддержку нормативных требований. С ее помощью компании превращают свои **данные** в ценную информацию и принимают взвешенные решения на основе **данных**.

SharedPreferences — **постоянное хранилище на платформе Android, используемое приложениями для хранения своих настроек, например**. Это хранилище является относительно постоянным, пользователь может зайти в настройки приложения и очистить данные приложения, тем самым очистив все данные в хранилище.

Для работы с данными постоянного хранилища нам понадобится экземпляр класса SharedPreferences, который можно получить у любого объекта, унаследованного от класса *android.content.Context* (например, Activity или Service). У объектов этих классов (унаследованных от Context) есть метод `getSharedPreferences`, который принимает 2 параметра:

`name` — выбранный файл настроек. Если файл настроек с таким именем не существует, он будет создан при вызове метода `edit()` и фиксации изменений с помощью метода `commit()`.  
`mode` — режим работы.

DataStore — это **библиотека из семейства Jetpack, которая предоставляет новое решение для хранения данных**. Скорее всего, оно заменит общие настройки. В настоящее время библиотека находится в альфа-стадии. Библиотека разработана с использованием корутин Kotlin и Flow API, что делает её более безопасной и надежной, чем общие настройки.

Библиотека разработана с использованием корутин Kotlin и Flow API, что делает её более безопасной и надежной, чем общие настройки. Предлагаются два различных подхода к хранению данных:

- **Preferences DataStore.** Они похожи на SharedPreferences, поскольку не могут определить схему или обеспечить доступ к ключам с правильным типом.
- **Proto DataStore.** С ее помощью можно создавать схемы, используя буферы протокола, которые позволяют сохранять строго типизированные данные. Они быстрее, меньше, проще и менее неоднозначны, чем XML и другие подобные форматы данных

#### 44. Хранение данных в ОС Android на примере внутреннего и внешнего хранилища: основные классы и методы.

##### Android Internal Storage:

**Это** место хранения личных данных каждого приложения, эти данные хранятся и используются этим же приложением. Другие приложения не имеют доступа. Обычно, когда приложение удаляется с устройства **Android**, связанные файлы данных так же будут удалены. Еще одна особенность, **это** при работе с файлами данных во Внутренней **памяти**, вы можете работать только с простыми файлами, и не сможете работать с файлами имеющими ссылку.

Внутреннее хранилище можно использовать, когда вам нужны некоторые конфиденциальные данные для вашего приложения. Вы должны получить разрешение на **чтение ВНЕШНЕГО хранилища** от пользователя, чтобы получить доступ к внешнему хранилищу. В результате любое приложение с этим разрешением имеет доступ к данным вашего приложения.

Целесообразнее всего использовать внешнее хранилище, если данные, которые хранит ваше приложение, могут использоваться другими приложениями.

Чтобы сохранить файл во внутреннем хранилище, вы должны сначала получить его из внутреннего каталога. Вы можете сделать это, вызвав методы `getFilesDir()` или `getCacheDir()`. Метод `getFilesDir()` возвращает абсолютный путь к каталогу, в котором создаются файлы в файловой системе. `getCacheDir()` возвращает абсолютный путь к каталогу кэша файловой системы для конкретного приложения.

##### Вложенные классы:

class

StorageManager.StorageVolumeCallback

Обратный вызов, который доставляет StorageVolume связанные события.

Методы:

ateBytes(FileDescriptor fd, long bytes)

елите запрошенное количество байт для вашего приложения для использования в данн  
ытом файле.

- allocateBytes(UUID storageUuid, long bytes)

Выделите запрошенное количество байт для вашего приложения для использования на данном томе хранилища.

- getAllocatableBytes(UUID storageUuid)

Возвращает максимальное количество новых байтов, которое ваше приложение может выделить для себя на данном томе хранилища.

- getCacheQuotaBytes(UUID storageUuid)

Возвращает размер квоты в байтах для всех кэшированных данных, принадлежащих вызывающему приложению, на данном томе хранилища.

#### 45. Основы работы с базами данных SQLite. Запросы в sqlite.

+

#### 46. Классы и основные методы для работы с SQLite.

Базы данных SQLite - это **механизм базы данных, написанной на языке C**. Это не отдельное приложение, а скорее библиотека, которую разработчики программного обеспечения встраивают в свои приложения. Как таковая, она относится к семейству встроенных баз данных.

Обращения к базе данных SQL выполняются посредством запросов, существует три основных вида SQL запросов: DDL, Modification и Query.

**DDL запросы.** Такие запросы используются для создания таблиц. Каждая таблица характеризуется именем и описанием столбцов, которое содержит имя столбца и тип данных. В файле базы данных может быть несколько таблиц.

Пример запроса для создания таблицы:

```
create Table_Name (  
    _id integer primary key autoincrement,  
    field_name_1 text, field_name_2 text);
```

**Modification запросы.** Такие запросы используются для добавления, изменения или удаления записей.

Пример запроса на добавление строки:

```
insert into Table_Name values(null, value1, value2);
```

**Query запросы.** Такие запросы позволяют получать выборки из таблицы по различным критериям. Пример запроса:

```
select * from Table_Name where _id = smth
```

```
select Field_Name_2 from Table_Name WHERE Field_Name_1 = smth
```

В Android предусмотрен класс для работы с базой данных SQLite напрямую, этот класс называется SQLiteDatabase и содержит методы:

|            |  |
|------------|--|
| Database() | зоялет открыть базу данных;                    |
| ie()       | зоялет обновить строки таблицы базы данных;    |
| :( )       | зоялет добавлять строки в таблицу базы данных; |
| э()        | зоялет удалять строки из таблицы базы данных;  |
| '()        | зоялет составлять запросы к базе данных;       |
| SQL()      | зоялет выполнять запросы к базе данных.        |

SQLite3 даёт множество преимуществ в отличии от других СУБД. Множество фреймворков таких как Django, Ruby on Rails и web2py по умолчанию используют SQLite3.

Многие браузеры используют данный инструмент для хранения локальных данных. Так же она используется в качестве хранилища данных таких ОС как Android и Windows Phone 8.

Для работы с SQLite3 можно воспользоваться и программами с графическим интерфейсом. К примеру: DB Browser for SQLite и SQLiteStudio.



## 47. 2D и 3D графика в мобильных приложениях для ОС Android.

мы можем использовать различные методы:

- **Canvas и Drawable:** в этом случае мы можем расширить существующие виджеты пользовательского интерфейса, чтобы мы могли настроить их поведение или создать собственную 2D-графику, используя стандартный метод, предоставленный классом Canvas .
- **Аппаратное ускорение:** мы можем использовать аппаратное ускорение при рисовании с помощью Canvas API. Это возможно с Android 3.0.
- **OpenGL:** Android изначально поддерживает OpenGL с использованием NDK. Эта техника очень полезна, когда у нас есть приложение, которое интенсивно использует графический контент (например, игры).

Android предоставляет два разных API для создания 2D-графики. Один — это высокоуровневый декларативный подход, а другой — программный низкоуровневый API:

- Рисуемые ресурсы — они используются для создания пользовательской графики программными средствами или (чаще всего) путем внедрения инструкций рисования в XML-файлы. Рисуемые ресурсы обычно определяются как XML-файлы, содержащие инструкции или действия для Android для отрисовки 2D-рисунка.
- Canvas — это низкоуровневый API, который включает в себя рисование непосредственно на базовом растровом рисунке. Он обеспечивает очень детализированный контроль над отображаемыми сведениями.

Помимо этих 2d графических методов Android также предоставляет несколько различных способов создания анимаций:

Анимации с поддержкой рисования — Android также поддерживает анимации по кадрам, известные как анимация с поддержкой рисования. Это самый простой API анимации. Android последовательно загружает и отображает ресурсы, доступные для рисования в последовательности

Анимации представлений — анимации представления — это API исходной анимации в Android и доступны во всех версиях Android. Этот API ограничен тем, что он будет работать только с объектами View и может выполнять только простые преобразования для этих представлений. Анимации представления обычно определяются в XML-файлах, найденных в папке /Resources/anim .

Анимации свойств — Android 3.0 появился новый набор API анимации, известный как анимация свойств. В этих новых API появилась расширяемая и гибкая система, которую можно использовать для анимации свойств любого объекта, а не только для просмотра объектов. Такая гибкость позволяет инкапсулировать анимации в различных классах, которые упрощают совместное использование кода.

#### 48. Анимация в ОС Android и типы анимации.

Анимация-это **процесс добавления эффекта движения к любому виду, изображению или тексту**. С помощью анимации вы можете добавить движение или изменить форму определенного вида. Анимация в Android обычно используется для придания вашему пользовательскому интерфейсу насыщенного внешнего вида.

**Андроид** поддерживает два **типа анимации**:

**анимация** преобразований (Tweened Animations)

**анимация** «кадр за кадром» (Frame-by-frame)

**анимация** преобразований применяется к объектам **типа** View и позволяет изменять их размер, положение и прозрачность. Например, вы можете задать эффект затухания для представления (View) или повернуть его.

Покадровая анимация (frame-by-frame) показывает различные эффекты рисования в представлении. Этот тип анимации ограничен оригинальными размерами представления (View).

Анимация может быть применена не только к представлениям, но и при переходе от одного Activity к другому.

Для этого вы должны использовать метод `overridePendingTransition()` в текущем Activity. Анимация преобразований — это экземпляр абстрактного класса `Animation`. Вы имеете предопределённые Java-классы: `AlphaAnimation`, `RotateAnimation`, `ScaleAnimation` и `TranslateAnimation`.

Анимация может задаваться в виде XML-файла в ресурсах. В таком случае файл должен располагаться в каталоге «res/anim».

Скорость анимации можно задать в классе «Interpolator».

## 49. Использование встроенной камеры и работа с мультимедиа в Android-приложениях.

Встроенная камера - это встроенное системное приложение, которое в основном поставляется с каждым смартфоном, а фильтры намерений предусмотрены внутри:

Все классы, предоставляемые Android SDK, которые мы можем использовать для добавления мультимедийных возможностей в наши приложения, находятся в пакете `android.media`. В этом пакете класс сердца называется MediaPlayer. Этот класс имеет несколько методов, которые мы можем использовать для воспроизведения аудио и видео файлов, хранящихся на нашем устройстве или передаваемых с удаленного сервера.

Этот класс реализует конечный автомат с четко определенными состояниями, и мы должны знать их перед воспроизведением файла. Упрощая диаграмму состояний, как показано в официальной документации, мы можем определить эти макро-состояния:

- **Состояние бездействия:** когда мы создаем новый экземпляр класса `MediaPlayer`.
- **Состояние инициализации:** это состояние срабатывает, когда мы используем `setDataSource` для установки источника информации, который должен использовать `MediaPlayer`.
- **Подготовленное состояние:** В этом состоянии подготовительные работы завершены. Мы можем войти в это состояние, вызывая метод `prepareAsync` или `prepare`.
- **Завершенное состояние:** достигнут конец потока.

## 50. Классификация сенсоров и датчиков для ОС Android и их характеристики.

Датчики, следящие за физическими свойствами и состоянием окружающей среды, предоставляют инновационные способы для улучшения мобильных приложений. Наличие в современных телефонах электронных компасов, датчиков равновесия, яркости и близости открывает целый ряд новых возможностей для взаимодействия с устройством, таких как дополненная реальность и ввод данных, основанный на перемещениях в пространстве.

Датчики в Android делятся на несколько категорий: движения, положения и окружающей среды. Ниже перечислены некоторые виды популярных датчиков:

- Акселерометр (TYPE\_ACCELEROMETER)
- Гироскоп (TYPE\_GYROSCOPE)
- Датчик освещения (TYPE\_LIGHT)
- Датчик расстояния (TYPE\_PROXIMITY)
- Датчик магнитных полей (TYPE\_MAGNETIC\_FIELD)
- Барометр (TYPE\_PRESSURE)
- Датчик температуры окружающей среды (TYPE\_AMBIENT\_TEMPERATURE)
- Измеритель относительной влажности (TYPE\_RELATIVE\_HUMIDITY)

В каждом телефоне может быть свой набор датчиков. В большинстве аппаратов есть — акселерометр и гироскоп.

Чтобы узнать, какие сенсоры есть в смартфоне, следует использовать метод **getSensorList** объекта **SensorManager**:

```
List<Sensor> sensors = sensorManager.getSensorList(Sensor.TYPE_ALL);
```

За работу с сенсорами отвечает класс **SensorManager**, содержащий несколько констант, которые характеризуют различные аспекты системы датчиков Android, в том числе:

### Тип датчика

Ориентация, акселерометр, свет, магнитное поле, близость, температура и т.д.

### Частота измерений

Максимальная, для игр, обычная, для пользовательского интерфейса. Когда приложение запрашивает конкретное значение частоты отсчётов, с точки зрения сенсорной подсистемы это лишь рекомендация. Никакой гарантии, что измерения будут производиться с указанной частотой, нет.

### Точность

Высокая, низкая, средняя, ненадёжные данные

## **51. Взаимодействие с системами позиционирования: права доступа, основные классы и методы.**

Системы позиционирования позволяют определить местоположение в некоторой системе координат, обычно определяются широта и долгота.

Так как смартфон является мобильным телефоном, ему доступны методы, обычно используемые мобильными телефонами для определения своего местоположения.

Во-первых, смартфон постоянно связывается с сотовой вышкой, в зоне действия которой он находится. Каждая сотовая вышка в мире имеет уникальный идентификатор, называемый идентификатором соты (Cell ID), а также для нее точно известны широта и долгота ее расположения. В связи с этим, смартфон, зная идентификатор соты, в которой он находится, может получить географические координаты центра этой соты.

Во-вторых, чаще всего смартфон оказывается в зоне действия более, чем одной сотовой вышки. В случае, когда телефон находится в зоне действия двух или трех сотовых вышек, они могут выполнять триангуляцию его местоположения. Телефон может запросить у сети информацию о том, где он находится. Такая техника определения местоположения может быть очень точной и не требует установки дополнительного оборудования.

Android предоставляет приложениям доступ к геолокационным возможностям мобильного устройства, через классы пакета `android.location`. Центральным классом этого пакета является класс `LocationManager`, который предоставляет доступ к системным сервисам для определения координат устройства.

Ключевым классом в Google Maps Android API является класс `MapView`, который отображает карту с данными полученными из сервиса Google Maps. Когда `MapView` имеет фокус, он может перехватывать нажатия клавиш и сенсорные жесты для выполнения автоматического перемещения и изменения масштаба карты, а также может управлять сетевыми запросами для получения дополнительных фрагментов карты. Этот класс так же предоставляет все элементы пользовательского интерфейса, необходимые для управления картой.

Google Maps Android API не является частью платформы Android, но доступен на любом устройстве с Google Play Store, работающем, начиная с Android 2.2, через Google Play services. Чтобы обеспечить возможность интеграции Google Maps в приложения, в Android SDK необходимо установить библиотеку Google Play services.

## 52. Сетевые соединения: класс **ConnectivityManager** и класс **NetworkInfo**.

Класс **ConnectivityManager**, который отвечает на запросы о состоянии сетевого подключения. Он также уведомляет приложения об изменении сетевого подключения.

Основными обязанностями этого класса являются:

1. Мониторинг сетевых подключений (Wi-Fi, GPRS, UMTS и т.д.)
2. Отправка широковещательных намерений при изменении сетевого подключения
3. Попытка "аварийного переключения" на другую сеть при потере подключения к сети
4. Предоставьте API, который позволяет приложениям запрашивать крупнозернистое или мелкозернистое состояние доступных сетей
5. Предоставьте API, который позволяет приложениям запрашивать и выбирать сети для своего трафика данных

**Класс NetworkInfo** инкапсулирует параметры мобильной сети Интернет и содержит набор методов для определения параметров сети:

`getType()` – возвращает целочисленное значение, определяющее тип сети. **Это** может быть одна из констант `TYPE_MOBILE`, `TYPE_WiFi`, `TYPE_WIMAX`, `TYPE_Ethernet`, `TYPE_Bluetooth` или какое-то другое выражение, определенное в **классе** `ConnectivityManager`.

`getSubtype()` – возвращает значение, определяющее тип подсети.

`getSubtypeName()` – возвращает описательное имя типа подсети.

`isAvailable()` – проверяет доступность сети, в случае положительного ответа возвращает `true`.

`isConnected()` – возвращает `true`, если сетевое соединение установлено и через него можно передавать данные.

`isConnectedOrConnecting()` – возвращает `true` в случае обнаружения установленного или устанавливаемого сетевого соединения.

`isRoaming()` – возвращает `true`, если мобильное устройство находится в роуминге относительно данной сети.

### 53. Использование Wi-Fi в Android: основные классы и их характеристики.

Android SDK предоставляет разработчикам программный интерфейс WiFi, который очень удобен в использовании.

Связанные пакеты:

**android.net.wifi** (Вам нужно только импортировать пакет при написании приложения, чтобы использовать функции, связанные с WiFi)

В основном связанные категории:

\* **WifiManager** WiFi программирование входа, большинство функций WIFI предусмотрены в виде такого метода

\* **WifiInfo** Используется для описания статуса WIFI-соединения

\* **ScanResult** Используется для описания точки доступа, например SSID, уровня сигнала, метода безопасности

В коде фреймворка отношения между несколькими классами, относящимися к Wi-Fi, выглядят следующим образом:

- WifiService наследуется от IWifiManager.Stub;
- IWifiManager.Stub наследуется от Binder и реализует интерфейс IWifiManager;
- WifiManager.Stu.proxy также реализует интерфейс IWifiManager;

## 54. Использование Bluetooth в Android

Bluetooth — это распространенный протокол беспроводной связи, используемый для соединения двух устройств на сравнительно коротких расстояниях, например, для подключения беспроводных наушников к смартфону на Android.

В составе Android есть пакет `android.bluetooth`, имеющий необходимые классы для работы с синим зубом:

- `BluetoothAdapter` - локальный Bluetooth-адаптер, то есть устройство, на котором работает ваше приложение. Предоставляет интерфейс обнаружения и установки Bluetooth-соединений
- `BluetoothClass` - общие характеристики Bluetooth-устройства
- `BluetoothDevice` - информация об удалённом Bluetooth-устройстве, к которому вы хотите подключиться
- `BluetoothSocket` - сокет или точка соединения для данных, которыми наша система обменивается с другим устройством
- `BluetoothServerSocket` - сокет для прослушивания входящих Bluetooth-соединений. Дает возможность отслеживать входящие сетевые запросы, поступающие от удалённых устройств

Для передачи данных с использованием Bluetooth требуется сделать следующее:

- Включить адаптер Bluetooth
- Найти доступные устройства с включёнными Bluetooth-адаптерами
- Подключиться к выбранному устройству
- Обмениваться данными (переслать картинку, файл, музыку)

Для работы с Bluetooth обычно требуются два разрешения:

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

Второе разрешение `BLUETOOTH_ADMIN` используется в тех случаях, когда требуется изменить какие-то свойства адаптера.