**Вопросы к коллоквиуму 25.11.2020**

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

**Операцио́нная систе́ма**, сокр. **ОС** (англ. operating system, OS) — комплекс взаимосвязанных программ, предназначенных для управления ресурсами компьютера и организации взаимодействия с пользователем.

Логическая структура устройства: микроархитектура (машинный язык, прошивка) -> ОС -> прикладные программы.

По типу ядра ОС делятся на:

* монолитные системы,
* ОС с монолитным ядром,
* микроядерные ОС,
* гибридные системы.

**стра́иваемая систе́м**а (встро́енная систе́ма, англ. **embedded system**) — специализированная микропроцессорная система управления, контроля и мониторинга.

Особенности:

* 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

Область применения:

* средства автоматического регулирования и управления техпроцессами, например авионика, контроль доступа;
* станки с ЧПУ;
* банкоматы, платёжные терминалы;
* телекоммуникационное оборудование.

Встроенные системы находят массовое применение, например устройства RFID **RFID** (англ. Radio Frequency IDentification, радиочастотная идентификация) . Как результат появление вирусов для таких платформ или перехват управления устройствами и использование их вычислительных мощностей.

# 2. Графический интерфейс в Unix-системах.

1. Графический дисплейный менеджер — Display Manager (DM).
2. Графический дисплейный сервер — Display Server.
3. Среда рабочего стола — Desktop Environment

X Window System — оконная система, обеспечивающая стандартные инструменты и протоколы для построения графического интерфейса пользователя. Используется в UNIX-подобных ОС.

Разработана в 1984 году в Массачусетском технологическом институте (MIT).

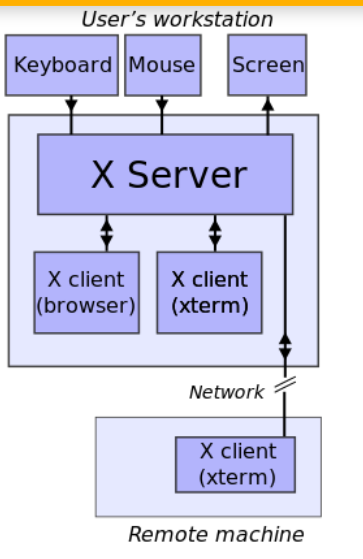
Текущая версия - X11 (2012 год).

X Window System часто называют X11 или просто X (в разговорной речи — «иксы́»).

Графический дисплейный менеджер для X Window System (Display Manager) — это программа, которая регистрирует пользователей в операционной системе, предлагая им ввести их логин и пароль, которая также позволяет выбирать графическую среду (сеанс) и локаль (набор языковых и региональных настроек).

Идея:

* X-сервер
* X-клиент
* X-терминал
* Xlib

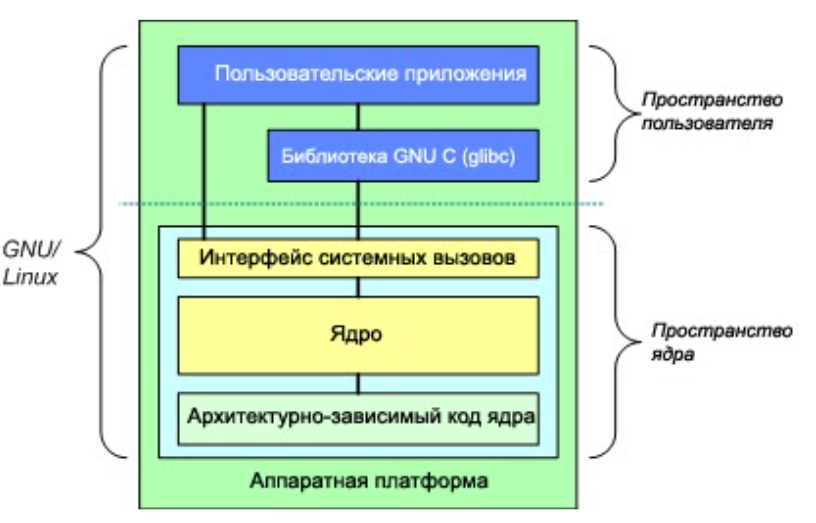


Результат - Graphical Application Programming Interface (GAPI)

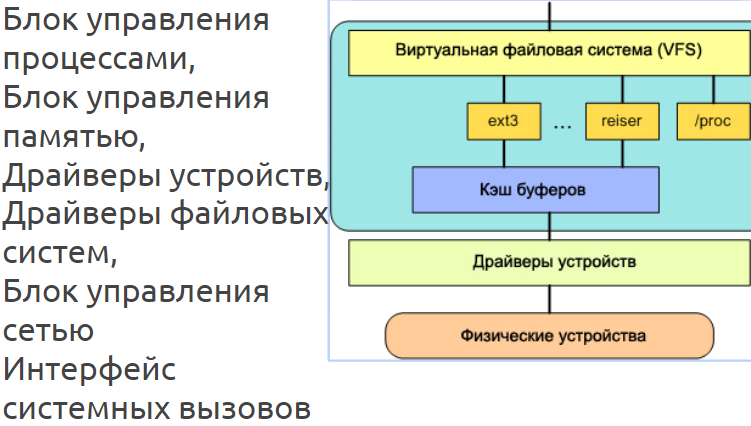
Gnome (GNU Network Object Model Environment) - графическая среда рабочего стола

# 3. Уровни и основные подсистемы ядра Linux

**Ядро́** (kernel) — центральная часть операционной системы (ОС), обеспечивающая приложениям координированный доступ к ресурсам компьютера, таким как процессорное время, память, внешнее аппаратное обеспечение, внешнее устройство ввода и вывода информации.

****

**Основные подсистемы**

****

**Модуль ядра** — это часть кода ядра, которая может быть динамически загружена и выгружена во время работы ядра.

**Модуль ядра**, загружаемый модуль ядра (loadable kernel module, LKM) — объект, содержащий код, который расширяет функциональность запущенного или т. н. базового ядра ОС.

* Автоматическая загрузка модулей ядра с kmod.
* Ядро и модули должны подходить друг к другу.

# 4. Виртуальная файловая система

***Виртуальная файловая система (VFS)*** - это абстракция, позволяющая представить многообразие различных файловых систем - располагающихся, возможно, на различных запоминающих устройствах, разделах или даже на различных компьютерах в сети, - в виде единой иерархии каталогов и файлов с унифицированным программным интерфейсом работы с файлами.

Официального единого стандарта для файловой системы UNIX не существует. Однако, основные элементы иерархии оказываются одинаковыми в различных версиях ОС. В Linux существует стандарт иерархии файловой системы FHS . Файлы в FHS распределяются по двум независимым категориям:

• с совместным доступом (sharable) или без него (unsharable) и

• изменяющиеся (variable) или статичные (static)

***Стандарт определяет следующие элементы иерархии:***

1. Корневая файловая система “/” - предназначена для загрузки и восстановления системы, содержит следующие директории

/bin основные команды ОС

/boot статичные файлы загрузчика ОС

/dev файлы устройств /etc конфигурационные файлы хоста

/lib разделяемые библиотеки и модули ядра

/media для монтирования флоппи, cd, usb и т.п. носителей

/mnt для временного монтирования файловых систем

/opt дополнительно установленные пакеты приложений

/sbin основные системные программы

/srv данные для сервисов, исполняющихся в системе

/tmp временные файлы

/usr иерархия системных ресурсов (Unix System Resources)

/var различные изменяющиеся файлы

/\****Монтирование*** — подключение дочерней файловой системы (устройства) к единому дереву родительской файловой системы в определенной логической точке.

**Точка монтирования** — логическое имя для подключаемого устройства (файловой системы)\*/

Кроме того, возможно создание директория для суперпользователя /root, директория /home – использующегося в качестве точки монтирования домашних каталогов пользователей, а также дополнительных библиотек /lib. Никакие приложения не должны создавать специальные файлы и директории в корневой файловой системе.

2. Иерархия /usr является совместно используемой, но доступной только для чтения секцией иерархии, не содержащей специфичной для данного хоста информации. ***Следующие директории являются обязательными:***

/usr/bin основные команды пользователя (включая компиляторы и интерпретаторы)

/usr/include include файлы языка С

/usr/lib библиотеки языков программирования

/usr/local локально установленные программы

/usr/sbin дополнительные системные команды и утилиты.

/usr/share статичные данные не зависящие от архитектуры, например man страницы

Кроме этого, /usr иерархия может содержать средства графического интерфейса /usr/X11R6, игровые приложения /usr/games, дополнительные библиотеки /usr/lib и исходные коды /usr/src ядра и др.

3. Иерархия /var содержит изменяющиеся в процессе функционирования системы файлы. Некоторые из файлов используются только локально, например, /var/log, /var/lock, /var/run. Другие — совместно с другими хостами, например, /var/mail, /var/cache/man, /var/cache/fonts, /var/spool/news.

4. Виртуальная файловая система /proc (Linux) – используется для хранения системной информации и информации об исполняющихся процессах для ядра. Эта файловая система не представляется сохраняемым каталогом на запоминающем устройстве, а существует только в адресном пространстве ядра и строится заново при перезагрузке системы.

Для физического размещения файлов и каталогов на разделах дисков ОС UNIX/Linux могут использовать смешанное разнообразие типов файловых систем: ext2fs, ext3fs, ext4fs, ufs, ffs, reiserfs и множество других, включая не-UNIX файловые системы, такие как, vfat (FAT16/32), ntfs и iso9660. Файлы всех типов файловых систем обрабатываются единообразно, что обеспечивается каркасом виртуальной файловой системы.

***Навигация по файловой системе:***

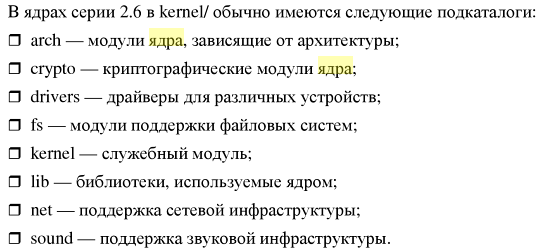
* / - корневой каталог
* Имена файлов чувствительны к регистру
* Длина файлов не превышает 255 символов (для файловой системы ext4fs)
* Файл, начинающийся с ‘.’, является скрытым.
* Текущий каталог - ‘.’
* Домашний каталог - ‘~’
* Родительский каталог текущего каталог - ‘..’
* Команды: pwd, cd

# 5. Модули ядра и уровни выполнения в Linux

***Модуль ядра*** — это часть кода ядра, которая может быть динамически загружена и выгружена во время работы ядра.

***Модуль ядра, загружаемый модуль ядра*** (loadable kernel module, LKM) — объект, содержащий код, который расширяет функциональность запущенного или т. н. базового ядра ОС.

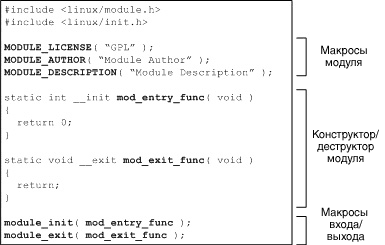
* Автоматическая загрузка модулей ядра с kmod.
* Ядро и модули должны подходить друг к другу.



Ядро Linux позволяет драйверам оборудования, файловых систем, и некоторым другим компонентам быть скомпилированными отдельно - как модули, а не как часть самого ядра. Таким образом, вы можете обновлять драйвера не пересобирая ядро, а также динамически расширять его функциональность. А еще это значит, что вы можете включить в ядре только самое необходимое, а все остальное подключать с помощью модулей.

Модули ядра Linux собираются только под определенную версию ядра, есть способ запуска модуля независимо от версии ядра, если они совместимы с помощью dkms, но об этом мы поговорим позже.

Находятся все модули в папке /lib/modules/. Учитывая, что модули рассчитаны только для определенной версии ядра, то в этой папке создается отдельная подпапка, для каждой установленной в системе версии ядра. В этой папке находятся сами модули и дополнительные конфигурационные файлы, модули отсортированы по категориям, в зависимости от назначения.



***Управление модулями ядра*** осуществляется следующими утилитами:

* **lsmod** вывод всех загруженных модулей в виде таблицы.
* **modinfo** вывод информации о модуле: файл модуля, краткое описание, авторы, лицензия, параметры.
* **insmod** утилита для загрузки модулей ядра. Повторяет функционал modprobe название\_модуля.
* **rmmod** простая программа для выгрузки модулей. Повторяет функционал modprobe -r название\_модуля.
* **modprobe** утилита для загрузки и выгрузки модулей.

***DKMS*** - фреймворк, который используется для генерации тех модулей ядра Linux, которые в общем случае не включены в дерево исходного кода.

| **Команда** | **Назначение** |
| --- | --- |
| # dkms status | Список модулей |
| # dkms autoinstall  # dkms autoinstall -k 3.16.4-1-ARCH  # dkms install -m nvidia -v 334.21 | Пересборка модулей |
| # dkms remove -m nvidia -v 331.49 --all | Удаление модуля |

***Уровень выполнения*** (*уровень инициализации*, *уровень запуска*) — нумерованный режим функционирования операционной системы, подразумевающий наличие в нём тех или иных функций: чем выше номер уровня, тем больше функций.

В Linux существует такое понятие как уровень выполнения (run level). Уровень выполнения обозначается числами от 0 до 6.

Система в определенный момент времени находится на соответствующем уровне выполнения. Вы, как администратор системы, можете переводить её с одного уровня выполнения на другой. Это делается при помощи программы init (или telinit). Для этого программе в качестве аргумента передается число, соответствующее уровню выполнения. Например, чтобы перевести систему на 3-й уровень выполнения, необходимо запустить init следующим образом:

init 3

В различных дистрибутивах Linux уровни выполнения используются для различных целей.

Современная версия программы init может использовать десять уровней выполнения, но обычно ***используются только семь***:

**0**  — действия по выключению системы

**1**  — однопользовательский режим

**2** — многопользовательский режим без поддержки сети

**3** — многопользовательский режим с поддержкой сети

**4**  — многопользовательский режим с поддержкой сети и графического входа в систему в Slackware. В других не сконфигурирован.

**5** — многопользовательский режим с поддержкой сети и графического входа в систему в Debian и Ubuntu. В других не сконфигурирован

**6**  — действия по перезагрузке системы

Суперпользователь может остановить систему, переведя её на нулевой уровень:

init 0

Или перегрузить систему:

init 6

В любой момент времени ***проверить текущий уровень*** можно командой runlevel:

runlevel

**6. Пакет BusyBox: состав, возможности и применение**

**BusyBox** — набор UNIX-утилит командной строки, используемой в качестве основного интерфейса во встраиваемых операционных системах.

BusyBox объединяет малые версии многих распространенных утилит UNIX в один небольшой исполняемый файл. Он предоставляет замену большинству утилит, которые вы обычно найдете в GNU fileutils, shellutils и т. д. Утилиты в BusyBox обычно имеют меньше опций, чем их полнофункциональные собратья GNU; тем не менее, включенные опции обеспечивают ожидаемую функциональность и ведут себя так же, как их аналоги GNU. BusyBox предоставляет достаточно полную среду для любой небольшой или встроенной системы.

BusyBox был написан с учетом оптимизации размера и ограниченных ресурсов. Кроме того, он чрезвычайно модульный, поэтому вы можете легко включать или исключать команды (или функции) во время компиляции. Это упрощает настройку ваших встроенных систем. Чтобы создать рабочую систему, просто добавьте несколько узлов устройств в / dev, несколько файлов конфигурации в / etc и ядро ​​Linux.

**Преимущества:**

* Малый размер.
* Низкие требования к аппаратуре.

**Команды BusyBox:**

cat – объединяет файлы и выводит их в стандартный вывод. ,

chmod – изменяет биты файлового режима файлов. ,

chown – изменяет владельца и / или группу файлов.,

cut – выводит выбранные поля из каждого входного файла в стандартный вывод,

date – отображение времени (в формате) или установка времени,

echo – выводит указанные аргументы в стандартный вывод.

gunzip – распаковывает файлы gzip.

gzip – сжимайте файлы (или стандартный ввод) с помощью алгоритма gzip.

kill – отправляет сигнал (по умолчанию: TERM) на заданные PID.

ln – создаёт ссылку на указанную цель.

login – начинает новый сеанс в системе.

ls – список содержимого каталога.

mkdir – создаёт директорию, netstat - отображает сетевуб информацию.

pidof – список PID всех процессов с именами, соответствующими шаблону , ping – отправляет пакеты ICMP ECHO\_REQUEST на сетевые узлы.

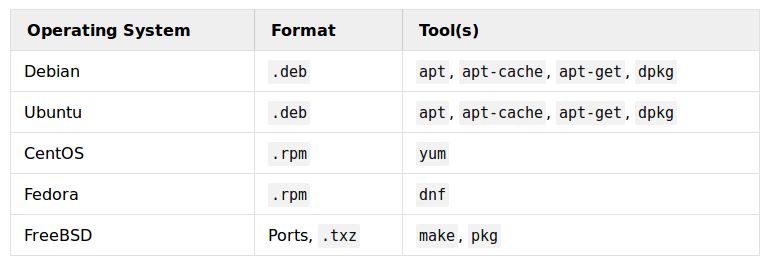
ps – отчет о состоянии процесса.

pwd – выводит полное имя текущего рабочего каталога. rm – удаляет файлы.

rmdir – удаляет директорию, если она пуста.

# 7. Установка ПО в ОС Linux: пакетные менеджеры

* **Пакет** — программа, библиотека, исходники или любые другие файлы, а также их метаданные, упакованные в особым образом сформированный архив.

****

**Основные функции пакетных менеджеров:**

* установка в системе новых программ;
* удаление программ из системы;
* проверка;
* модернизация версий установленных в системе программ;
* запрос установленного программного обеспечения;
* извлечение.

**Безопасность:**

Для обеспечения защиты от вредоносного ПО, Linux не позволяет неприви­легированным пользователям устанавливать программы.

**Проверка подлинности пакета:**

Основная форма проверки подлинности пакетов — функция хеширования.

При наличии хеша из доверенного источника:

* шансы получить модифицированный файл с идентичным хешем очень неве­лики;
* вероятность того, что злоумышленник может взять файл, модифицировать его по своему усмотрению и сгенерировать идентичный хеш, бесконечно мала.

**Если проверка подлинности не удалась:**

* Используйте исходный код для сборки своих приложений.
* Проверяйте сценарии, исполняемые в процессе установки.
* Проверяйте содержимое.

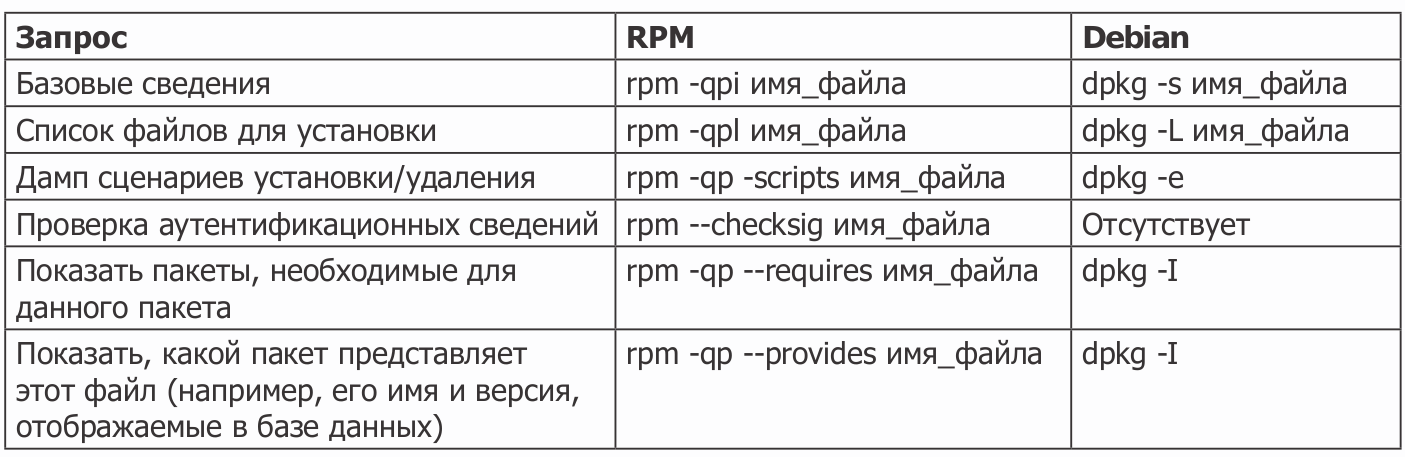
**Состав пакета:**

1. Архив с файлами для установки;
2. Сценарии, исполнямые во время установки и удаления пакета;
3. Сведения о зависимостях;
4. Текстовые данные о самом пакете.

**Типы сценариев установки:**

* Прединсталляционные
* Постинсталляционные
* Преддеинсталляционные
* Постдеинсталляционные

**Проверка пакетов:**



**Команды APT:**

* apt-cache search [mask]
* apt-cache show [package]
* apt-get install [package]
* apt-get remove [package] или apt-get install --remove [package]
* apt-get purge [package]
* apt-get update
* apt-get upgrade
* apt-get autoremove

**Преимущества пакетных менеджеров:**

* одни и те же библиотеки используются для нескольких программ, что экономит место;
* понятная файловая структура, да и, вообще, unix-way;
* конкретный разработчик обеспечивает работоспособность отдельного пакета;
* обеспечивается целостность системы;
* программа устанавливается один раз и для всех пользователей.

**Недостатки пакетных менеджеров:**

* старые версии пакетов в репозиториях, а если нужно свежее и новее, то приходится использовать PPA или другие средства;
* зависимость от конкретных версий библиотек, поставляемых совместно с системой;
* сложность установки программ разных версий;
* приходится скачивать большое количество зависимостей, которые зачастую не столь необходимых;
* необходимо устанавливать программы из-под root-пользователя;

# 8. Работа с файлами: типы и управление разрешениями на доступ

**Типы файлов:**

* **-** -регулярный или обычный файл (Regular ﬁle);
* **d** - каталог (Directory);
* **c** - файл символьного устройства (Character device ﬁle);
* **b** - файл блочного устройства (Block device ﬁle);
* **s** - файл локального сокета (Local domain socket);
* **p** - именованный канал (Named pipe);
* **l** - символическая ссылка (Symbolic link).

Регулярный файл — это обычный файл, который содержит разнообразные данные или команды.

Каталоги — это файлы, которые содержат указатели на другие файлы.

Файлы символьных и блочных устройств — системные файлы, предназначенные для организации обмена данными между такими устройствами.

Локальные сокеты — это файлы, которые позволяют организовать обмен данными между локальными процессами в двух направлениях.

Именованные каналы или FIFO — это файлы, которые позволяют организовать обмен данными между локальными процессами в одном направлении.

Символическая ссылка — это файл, который содержит абсолютный или относительный путь к другому файлу или каталогу

**Модель разрешений на доступ:**

Концепции модели разрешений на доступ к файлам:

* классы пользователей для доступа к файлу;
* разрешения на доступ к файлу.

Пользователи:

* владелец файла;
* группа файла;
* другие пользователи системы.

**Права на доступ к файлам:**

* r — разрешение на чтение файла;
* w — разрешение на запись в файл;
* x — разрешение на исполнение файла;
* X — подтверждение разрешения на исполнение файла при наличии уже прав на исполнение;
* s — устанавливает флаг setuid или setgid для владельца или группы, флаг x установлен;
* S — аналогичен s, флаг x НЕ установлен;
* t — установить sticky бит, флаг x установлен;
* T — установить sticky бит, флаг x НЕ установлен;

**Изменение разрешения на доступ к файлам:**

chmod [класс] [изменение] [права] [файлы]

u — владелец

g — группа

o — все остальные

a — все

+ - разрешить (добавить)

- - запретить (убавить)

= - установить (переписать)

r — чтение

w — запись

x — выполнение

**Примеры**:

chmod g­w,a+r file

chmod ug=rw,o= file

Установка маски разрешений на доступ:

umask [-S] [mask-definition]

где

- опция -S указывает на то, что маска разрешений задана или должна быть отображена в символьном формате;

- аргумент mask-definition задает маску разрешений на доступ к файлу или каталогу в числовом или символьном формате.

mask-definition задается в формате:

[user-class-symbol(s)][permissions-operator]

[permission-symbol(s)][,]...

**Примеры:**

# umask 200

# umask -S

# 9. Процессы в ОС Linux: создание, атрибут, просмотр информации о процессах

**Процесс** — это экземпляр программы, исполняемый процессором или ожидающий этого момента в очереди.

Процессом рассматриваем как программу, запущенную на исполнение.

**Новые процессы создаются**:

1. Родительский процесс вызывает функцию fork(), которая создает дочерний процесс.

2. Дочерний процесс вызывает одну из функций семейства exec, которая загружает в адресное пространство дочернего процесса новый исполняемый файл.

**Атрибуты процесса**:

Процесс в ядре представляется просто как структура с множеством полей. Данные поля и являются атрибутами процесса. Приведем примеры некоторых атрибутов процесса.

· Идентификатор процесса (Process ID (PID)) – это уникальный порядковый номер процесса. В операционной системе Linux PID хранится в переменной целочисленного типа.

· PPID (Parent Process ID) - PPID родительского процесса

· RUID (Real User ID) — ID пользователя, от имени которого выполняется процесс;

· RGID (Real Group ID) — идентификатор группы пользователя, который запустил процесс;

· PGID — идентификатор группы процесса;

**Просмотр информации о процессах**:

1. ps - просмотр информации об исполняемых процессах;

На выходе будут отображаться строки данных, содержащих следующую информацию: PID - это уникальный идентификатор процесса, TTY - тип терминала, TIME - это общее время использования процессорного времени процессом, CMD - это имя команды, которая запустила этот процесс.

Примеры различных ключей для команды ps:

- A : вывод всех процессов. (Аналогом является ключ -e).

- p: вывод только перечисленных процессов. Например, если мы знаем PID, то можно просто воспользоваться этим ключом для просмотра информации о процессе.

- r : вывод информации только о работающих процессах.

2. pstree - просмотр дерева исполняемых процессов;

Древовидный формат является более удобным способом отображения иерархии процессов и делает вывод более визуально привлекательным. Верхний/корневой элемент в дереве будет является родительским процессом всех системных процессов.

Примеры ключей для команды pstree:

С помощью ключа –s, за которым следует PID процесса показываются родительские процессы данного процесса.

С помощью ключа –p можем вывести PID процесса, который будет указан в скобках после каждого процесса. Например, systemd(1)).

Также pstree объединяет идентичные ветви (например, |-2\*[proc]), но с помощью ключа –c мы можем отключить объединение. По итогу получим:

|-proc

|-proc

3. jobs - просмотр информации о заданиях.

Задание — команда, запущенная на выполнение пользователем с помощью оболочки.

Например, с помощью данной команды можно отобразить список текущих фоновых задач. На выходе мы получим PID, задачу, а также состояние процесса – Running, Stopped, Done, Exit.

# 10.Процессы в ОС Linux: планирование процессов и перенаправление ввода-вывода

**Процесс** — экземпляр программы, исполняемый процессором или ожидающий этого момента в очереди.

***Планирование процессов***

В операционной системе Linux можно сообщить системе о том, что процесс не очень важен и должен выполняться с пониженным приоритетом. Это можно сделать путем повышения фактора уступчивости процесса. По умолчанию у каждого процесса нулевой фактор уступчивости. Повышение этого значения свидетельствует о снижении приоритета процесса, и наоборот, т.е. процессы с низким (т.е. отрицательным) фактором уступчивости получают больше времени на выполнение.

Для запуска программы с ненулевым фактором уступчивости необходимо воспользоваться командой nice -n. Например: nice -n 10 sort input.txt > output.txt

По умолчанию nice устанавливает значение приоритета 10. Значение nice может быть от -20(самый высокий приоритет) до 19(самый низкий приоритет).

Здесь активизируется длительная операция сортировки, которая, благодаря пониженному приоритету, не приведет к сильному снижению производительности системы. Далее, изменить фактор уступчивости выполняющегося процесса позволяет команда renice.

Например, изменим приоритет работающего процесса с 10 до 15: renice -n 15 -p 5645, где 15 – приоритет процесса, а 5645 – pid процесса.

Если требуется менять фактор уступчивости программным путем, воспользуйтесь функцией nice(). Ее аргумент — это величина приращения, добавляемая к фактору уступчивости вызывающего процесса. В результате приоритет процесса снижается.

Также стоит отметить, что только программа с привилегиями пользователя root может запускать процессы с отрицательным фактором уступчивости или понижать это значение у выполняющегося процесса. Это означает, что вызывать команды nice и renice с отрицательными аргументами можно, лишь войдя в систему как пользователь root, и только процесс, выполняемый от имени суперпользователя, может передавать функции nice() отрицательное значение.

***Перенаправление ввода-вывода***

В системе по-умолчанию всегда открыты три "файла" -- stdin (клавиатура), stdout (экран) и stderr (вывод сообщений об ошибках на экран). Эти, и любые другие открытые файлы, могут быть перенапрвлены. В данном случае, термин "перенаправление" означает получить вывод из файла, команды, программы, сценария или даже отдельного блока в сценарии и передать его на вход в другой файл, команду, программу или сценарий.

С каждым открытым файлом связан дескриптор файла. Дескрипторы файлов stdin, stdout и stderr -- 0, 1 и 2, соответственно. При открытии дополнительных файлов, дескрипторы с 3 по 9 остаются незанятыми. Иногда дополнительные дескрипторы могут сослужить неплохую службу, временно сохраняя в себе ссылку на stdin, stdout или stderr. Это упрощает возврат дескрипторов в нормальное состояние после сложных манипуляций с перенаправлением.

Есть некоторые правила перенаправления:

· если дескриптор файла опущен, то подразумевается канал STDOUT;

· для стандартных потоков вывода и ошибок дескриптор файла должен быть равен 1 или 2;

· если файл с именем filename не существует, то он создается.

Приведем некоторые примеры:

ls > file\_list.txt

ls -lR > dir-tree.list

head < input.txt > output.txt

Для однонаправленной передачи данных между процессами в операционной системе Linux предназначены каналы (pipes), которые также называют анонимными каналами. (command1 | command2 | ... | commandN)

Например, ls -lt | head

# 11. Командный интерпретатор bash: возможности, режимы работы, файлы запуска

Bash (Bourne Again Shell) - это командный интерпретатор, с которым пользователь может работать в интерактивном режиме, используя консоль или терминал.

Командный интерпретатор Bash имеет свой язык команд. Bash также может читать команды из файла, который называется скриптом. Сценарий или скрипт (script) - программа, содержащая команды и инструкции, хранится в текстовом файле и исполняется интерпретатором. Также он поддерживает автодополнение имён файлов и каталогов, подстановку вывода результата команд, переменные, контроль над порядком выполнения, операторы ветвления и цикла. Ключевые слова, синтаксис и другие основные особенности языка были заимствованы из sh. Другие функции, например, история, были скопированы из csh и ksh. Bash в основном соответствует стандарту POSIX, но с рядом расширений.

**Режимы работы:**

Интерактивный и неинтерактивный режим работы.

В интеракивном режиме, оболочка читает команды, вводимые пользователем, с устройства tty. Кроме того, такая оболочка считывает конфигурационные файлы на запуске, выводит строку приглашения к вводу (prompt), и, по-умолчанию, разрешает управление заданиями. Пользователь имеет возможность взаимодействия с оболочкой.

Сценарий всегда запускается в неинтерактивном режиме. Но, не смотря на это, он сохраняет доступ к своему tty. И даже может эмулировать интерактивный режим работы.

**Файлы запуска:**

Вызов в интерактивном режиме с регистрацией в системе: /etc/profile

Файлы .bash\_profile, .bash\_login и .profile

· устанавливаются цвета для вывода;

· задается маска разрешений на доступ к файлам по умолчанию

· устанавливаются переменные окружения для локализации;

· конфигурируется настройки клавиатуры;

· cоздается каталог временных файлов, имя которого задается в переменной TMPDIR;

· корректирует переменную PATH

· исполняет сценарий из файла ~/.bashrc, который исполняет сценарий из файла /etc/bashrc

· ~/.bash\_logout

Вызов интерактивной командной оболочки без регистрации в системе:

· /etc/bashrc

· ~/.bashrc

Неинтерактивное обращение к оболочке:

· переменная среды окружения BASH\_ENV

# 12.Программирование в bash: исполнительная среда переменные,исполнение

**команд и сценарии**

Bash — Bourne Again Shell

**Bash** - это командный интерпретатор, с которым пользователь может работать в интерактивном режиме, используя консоль или терминал.

Командный интерпретатор Bash имеет свой **язык команд**.

**Переменная** - это именованное значение, которое в командном интерпретаторе Bash может принимать числовые, символьные, строковые значения и массивы.

* **глобальные** переменные (global variables) или переменные окружения (environment variables) (пример — команды env или printenv)
* **локальные** переменные (local variables) (пример — команда set)

**Присваивание значения локальной переменной:**

VAR="VALUE"

**Присваивание значения глобальной переменной:**

export VAR="VALUE"

Локальные переменные можно экспортировать в среду окружения командного интерпретатора:

export VAR

**Правила для создания переменных:**

* имена переменных не могут начинаться с цифры;
* имена переменных чувствительны к регистру;
* при присваивании не допускаются пробелы;
* использование двойных кавычек предотвращает обработку специальных символов, за исключением символов $, ‘ (обратная кавычка) и \ (обратный слэш).

**Ссылка** на значение переменной: ${VAR}

**Операции над переменными**

| **Операция** | **Оператор** |
| --- | --- |
| Определение количества символов в переменной | ${#VAR} |
| Замена значения переменной | ${VAR:-WORD} |
| Замена значения существующей переменной | ${VAR:+WORD} |
| Присваивание значения переменной | ${VAR:=WORD} |
| Вывод сообщения об ошибке, если переменная не существует или ее значение не определено | ${VAR:?"error message"} |

**Удаление переменной**

unset [-v] name

где опция -v задает удаление переменной с именем name

**Создание массива**

ARR[INDEX]="VALUE"

или ARR=(value1 value2 ... valueN)

**Ссылка** на значение элементов массива:

${VAR[INDEX]}

Определение количества символов в элементе массива: ${#VAR[INDEX]}

Определение количества элементов в массиве: ${#VAR[@]}

ARR[INDEX]="VALUE" - добавляет в массив элемент с индексом INDEX, если этого элемента не существует

**Удаление массива**

Используется встроенная команда unset:

* если в команде задано имя массива, то удаляется весь массив;
* если в команде задан элемент массива, то удаляется только этот элемент.

При удалении элемента массива, индексы следующих за ним элементов уменьшаются на 1.

**Псевдонимы** или синонимы — это другие, как правило, краткие обозначения команд.

**Создание псевдонима**:

alias [-p] [name[=value] ...]

где

- -p — опция, предписывающая вывести предыдущее значение псевдонима с именем name;

- name — имя псевдонима;

- value — значение псевдонима.

Для **выхода** из оболочки используются команды:

exit

logout [n]

Команды бывают **системные** и  **встроенные**

**Примеры встроенных:**

alias, bind, command, echo, help, printf, read, type.

**Порядок исполнения команды:**

1. Чтение командной строки
2. Разбиение командной строки на слова и операторы
3. Расширение псевдонимов
4. Расширение выражения в { }
5. Расширение выражения с ~
6. Подстановка значений вместо параметров и переменных
7. Подстановка команды $( ) и ‘‘
8. Разбиение командной строки на слова снова
9. Расширение символов глобализации
10. Исполнение команды
11. Ожидание завершения команды и обработка кода возврата из команды

**Сценарии bash**

**Сценарий командной оболочки Bash** — это текстовый файл, который содержит команды или инструкции этой командной оболочки и исполняется интерпретатором..

* Создается в любом текстовом редакторе.
* Автоматизирует повторяющиеся работы;

**Правила для сценариев:**

* Одна строка содержит одну команду.
* Несколько команд разделяются ;
* Для продолжения длинной команды на следующей строке используется \
* Комментарии начинаются с символа #
* Первая строка имеет вид:

#! /bin/bash

**Как запустить сценарий:**

* вызвать командную оболочку Bash и передать ей в качестве аргумента имя файла со сценарием;
* запустить сценарий на выполнение из командной строки оболочки.

**При этом:**

* имя файла должно быть первым отличным от опции аргументом оболочки;
* командная оболочка Bash должна быть вызвана без опций -c или -s;
* командная оболочка ищет заданный файл в текущем каталоге, а затем в каталогах, хранящихся в переменной окружения $PATH;
* найдя файл, командная оболочка читает и исполняет команды из этого файла, а потом завершает свою работу.
* сценарий должен храниться в файле, для которого установлено разрешение на исполнение;
* файл должен находиться в каталоге, путь к которому хранится в переменной окружения $PATH.

**При завершении своей работы** команды устанавливают статус возврата, который хранится в специальной переменной $?.

Формат:

exit n

где n - устанавливаемый статус возврата из сценария.

# 13.Программирование в bash: выражения, условное выполнение команд, циклы и

**функции**

**Условное и безусловное выполнение команд**

**Безусловное выполнение команд:**

* + Команды в сценарии могут располагаться по одной команде в строке.
  + Если в одной строке располагается несколько команд, то они должны разделяться символом точка с запятой ;

Любая встроенная команда, выражение или функция, а также любая внешняя команда или скрипт могут выполняться условно с использованием && (и) и || (или) операторов.

**Условное выполнение команд:**

* + command1 || command2
  + !command1
  + command1&

**Арифметические выражения**

expr expression  
где expression - вычисляемое арифметическое выражение.

В арифметических выражения допустимо использование только целых чисел и следующих операторов:  
унарные арифметические операторы +, -, ++, --  
бинарные арифметические операторы +, -, \*, /, %, \*\*  
(возведение в степень)  
унарные логические операторы ! (отрицание)  
бинарные логические операторы &&, ||  
операторы сравнения ==, !=, >, >=, <, <=  
операторы присваивания =, +=, -=, \*=, /=, %=  
условные операторы expr ? expr : expr  
разделитель выражений , (запятая)

**Условные выражения**

Условные выражения могут использоваться для изменения порядка исполнения команд в управляющих инструкциях if-then, while и until.

Для вычисления значений условных выражений в командной оболочке предназначена команда test, которая может использоваться в любом из следующих форматов:  
test EXPR  
где EXPR - условное выражение.

также поддерживается инструкция выбора **case**

**Циклы**

**for**

for variable [in list-of-values];

do commands;

done for (( expr1 ; expr2 ; expr3 )) ;

do commands ; done  
где  
variable — переменная;  
list-of-values — список значений разделенных, как правило пробелами, или значением переменной окружения $IFS;  
commands — последовательность команд;  
expr1, expr2, expr3 — арифметические выражения.

**while**

while test-commands;

do consequent-commands;

done

**until**

until test-commands; do consequent-commands; done

**continue**

continue [n]  
где n - должно быть не меньше 1 и обозначает возобновление n-го вложенного цикла.

**break**

break [n]  
где n - должно быть не меньше 1 и обозначает завершение n-го вложенного цикла.

**Функции**

**Объявление функции**

[function] name() { command-list; }  
где  
function — необязательное зарезервированное слово;  
name — имя функции;  
command-list — список команд.

**Вызов функции**

name arg1 arg2 ...  
**Исполнение функции**

* функция исполняется в контексте текущей оболочки, новый процесс не создается;
* когда функция исполняется, её аргументы становятся позиционными параметрами на время исполнения функции;
* специальная переменная $# содержит количество аргументов функции;
* переменная FUNCNAME содержит имя исполняемой функции.

**Локальные переменные**

local name[=value]  
где  
name — имя переменной;  
value — значение переменной.

Локальные переменные видимы только в теле функции.

**Возврат** и статус возврата функции

return [n]

# 14.Архитектура Android.

Android — самая популярная операционная система и платформа для приложений.Android основан на ядре Linux. 2008 г. — первая стабильная версия

В Android ядро Linux обеспечивает управление памятью, защиту данных, поддержку мультипроцессности и многопоточности.

В Android нет ряда привычных компонентов GNU/Linux-систем: нет утилит GNU, не используется X.Org и systemd.

Ядро Linux в Android модифицировано:

* ashmem (механизм разделяемой памяти),
* Binder driver (межпроцессорное взаимодействие),
* wakelocks (управление спящим режимом)
* low memory killer (динамическая система управления запущенными приложениями)

В качестве libc (стандартной библиотеки языка C) в Android используется минималистичная реализация ***bionic***, оптимизированная для встраиваемых (embedded) систем .

В Android есть оболочка командной строки (shell) и множество команд/программ.

Во встраиваемых системах используется пакет Busybox; в Android используется его аналог под названием Toybox.

Приложение-эмулятор терминала — Terminal Emulator for Android, Material Terminal, Termux

Подключение к Android-устройству с компьютера через Android Debug Bridge (adb).

Другие знакомые компоненты, используемые в Android:

* библиотека FreeType (для отображения текста),
* графические API OpenGL ES, EGL и Vulkan,
* легковесная СУБД SQLite.

Ранее для реализации WebView использовался браузерный движок WebKit, но начиная с версии 7.0 используется установленное приложение Chrome.

**Архитектура Android состоит из следующих частей:**

* Applications (системные)

home, contacts, phone, browser, calendar ...

* Application Framework (Java API)

managers = (Activity manager, window manager, package manager, … ), view system, content provider

* Libraries (C/C++)

surface manager, media framework, SQLite, SGL, SSL, openGL

* Android runtime

Core libraries, Dalvik Virtual machine

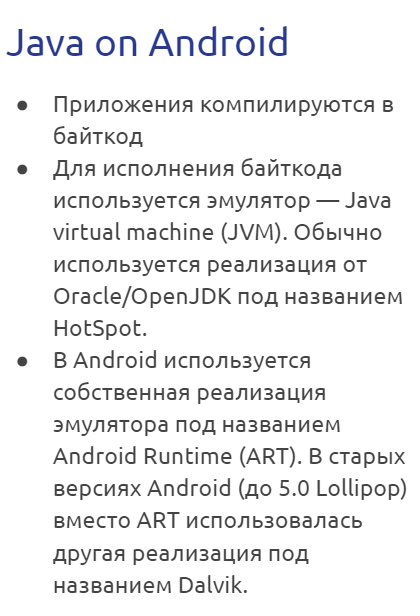
* Hardware Abstraction Layer

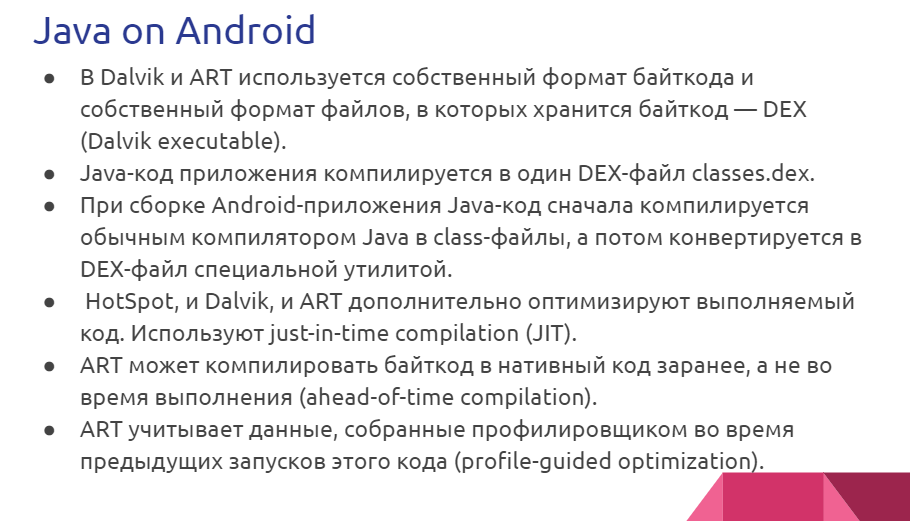
Camera, bluetooth, audio ...

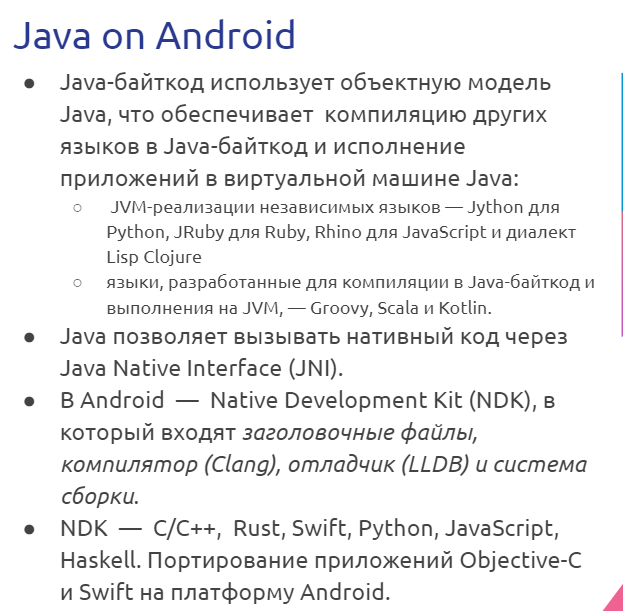
* Linux Kernel

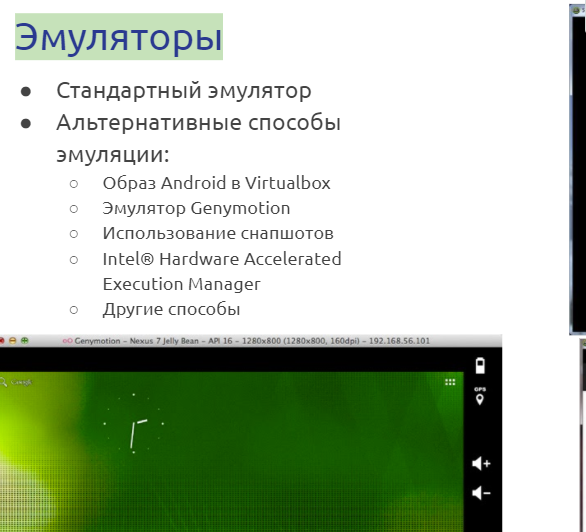
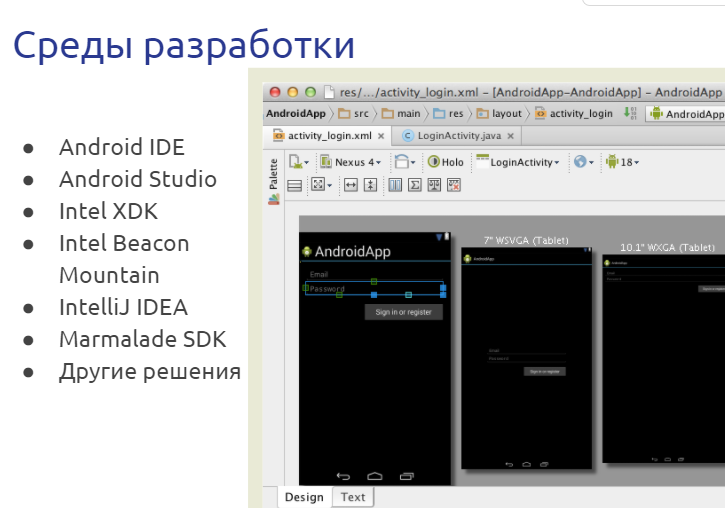
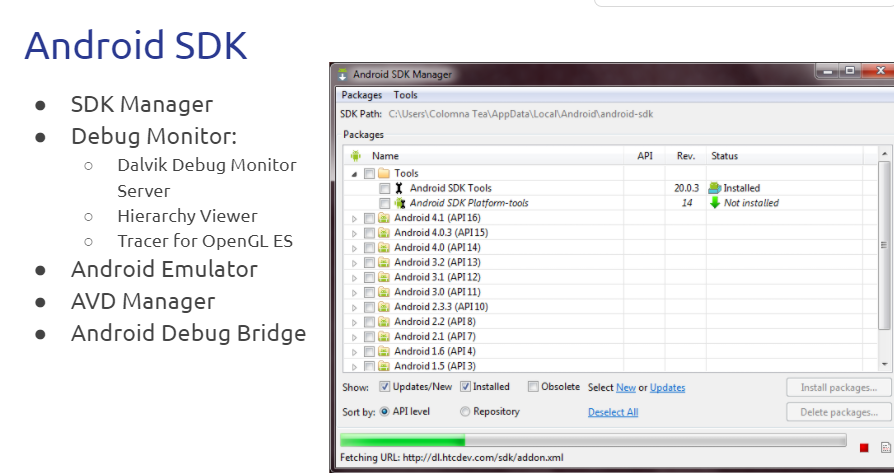
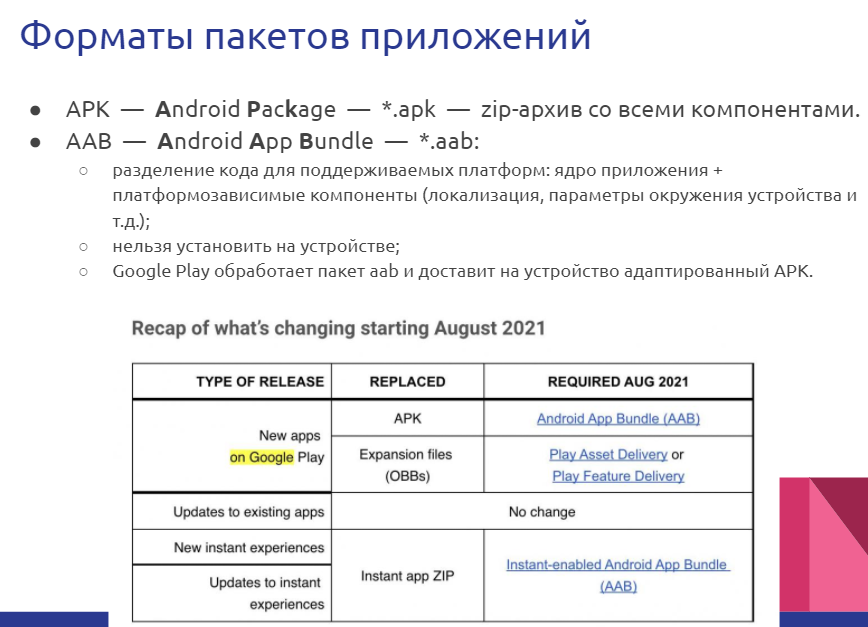
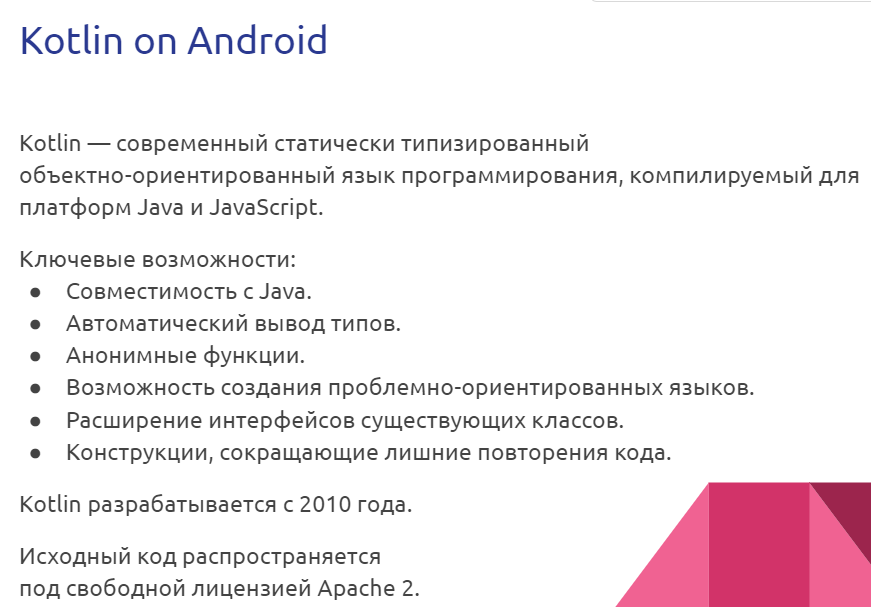
drivers, power management

# 15. Инструментарий разработчика и эмуляторы

****

****

****

****

# 16.Основные виды Android-приложений.

У Android приложений есть очень большое количество видов.

Среди основных видов Android-приложений можно выделить:

* Приложения переднего плана
* Фоновые приложения
* Смешанные приложения
* Виджеты

**Приложения переднего плана**:

* Выполняют свои функции только когда видимы на экране
* В свернутом виде выполнение приложений приостанавливается

Например:

* игры
* текстовые редакторы
* видеопроигрыватели

**Фоновые приложения**:

* После настройки не предполагают взаимодействия с пользователем.
* Большую часть времени находятся и работают в скрытом состоянии

Например:

* службы экранирования звонков
* SMS-автоответчики

Важно помнить, что совершенно невидимые сервисы будут неуправляемыми. Пользователю позволяем только:

* Санкционирование запуска сервиса
* Настройка сервиса
* Приостановка работы сервиса
* Прерывание работы сервиса

**Смешанные приложения**:

* Большую часть времени работают в фоновом режиме.
* Допускают взаимодействие с пользователем даже после настройки.

Например:

* Мультимедиа-проигрыватели
* Чаты
* Почтовые клиенты

Смешанные приложения, не теряя работоспособности в фоновом режиме, должны реагировать на пользовательский ввод. При взаимодействии с пользователем должны учитывать свое текущее состояние.

**Виджеты** - это небольшие приложения, отображаемые в виде графического объекта на рабочем столе.

В качестве примера можно привести любое приложение для отображения

динамической информации: вывод даты, погоды, температуры и прочей информации.

Сделаем небольшие выводы по видам Android приложений. Невозможно создать осмысленное приложение, не изучив внутреннюю организацию, свойственную приложениям, работающим на определенной платформе. Нам важно изучить структуру и основные компоненты приложений, разрабатываемых для работы на смартфонах под управлением ОС Android. Ведь от типа мобильного устройства внутренняя организация приложений не зависит, т. е. Android-приложения, разработанные для смартфонов вполне смогут выполняться и на планшетах. При разработке Android приложений нам следует стремиться к идее многократного использования компонентов, которые являются основными строительными блоками. Именно поэтому так важно разбираться в деталях основных компонентов Android приложений.

# **17.Архитектура приложений и основные компоненты.**

Архитектура Android приложений основана на идее многократного использования компонентов, которые являются основными строительными блоками. Каждый компонент является отдельной сущностью и помогает определить общее поведение приложения.

**Основные компоненты Android-приложений**

* Операции (Activities)
* Сервисы (Services)
* Контент-провайдеры (Content providers)
* Приемники широковещательных сообщений (Broadcast receivers)

**Операции**

Видимая часть приложения (экран, окно, форма), отвечает за отображение графического интерфейса пользователя.

Операция (activity) — это один “экран” приложения.

Каждой операции соответствует свое окно, которое чаще всего развернуто на весь экран.

Android поддерживает мультиоконный режим — split-screen, picture-in-picture и даже freeform. Android поддерживает подключение нескольких дисплеев.

activity может намеренно занимать небольшую часть экрана (Theme\_Dialog).

Приложение может иметь несколько операций (activities).

Намерение (Intent) — это сообщение, которое указывает системе, что нужно «сделать».

Типы намерений:

* явное (explicit);
* неявное (implicit).

Стандартный способ создавать intent’ы — через соответствующий класс в Android Framework

**Сервис**

Компонент, который работает в фоновом режиме, выполняет длительные по времени операции или работу для удаленных процессов.

Сервис может быть запущен другим компонентом и после этого работать самостоятельно, а может остаться связанным с этим компонентом и взаимодействовать с ним.

Типы сервисов:

* Foreground Service — сервис, выполняющий действие, состояние которого важно для пользователя.
* Background Service — сервис, выполняющий фоновое действие, состояние которого не интересует пользователя.
* Bound Service — сервис, обрабатывающий входящее Binder-подключение.

**Контент-провайдеры**

Content provider — компонент, позволяющий приложению предоставлять другим приложениям доступ к данным, которыми оно управляет. Управляет распределенным множеством данных приложения.

Данные могут храниться в файловой системе, в базе данных SQLite, в сети.

Позволяет другим приложениям при наличии у них соответствующих прав делать запросы или даже менять данные.

**Приемники широковещательных сообщений**

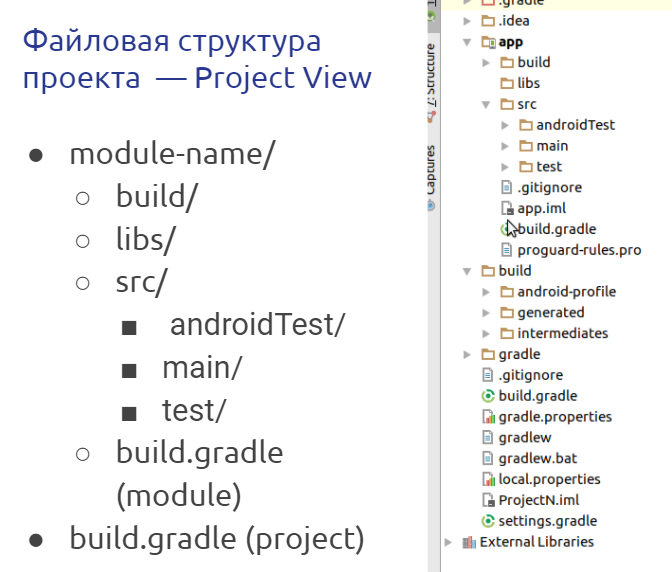
Broadcast receiver (приемник широковещательных сообщений) — компонент, позволяющий приложению принимать широковещательные извещения (broadcast’ы), специальный вид сообщений от системы или других приложений.

Приемник — компонент, который реагирует на широковещательные извещения.

Характеристики приемников:

* Инициирование широковещания
* Не отображают пользовательского интерфейса
* Могут создавать уведомление на панели состояний

# 18.Структура проекта Android-приложения.Характеристика

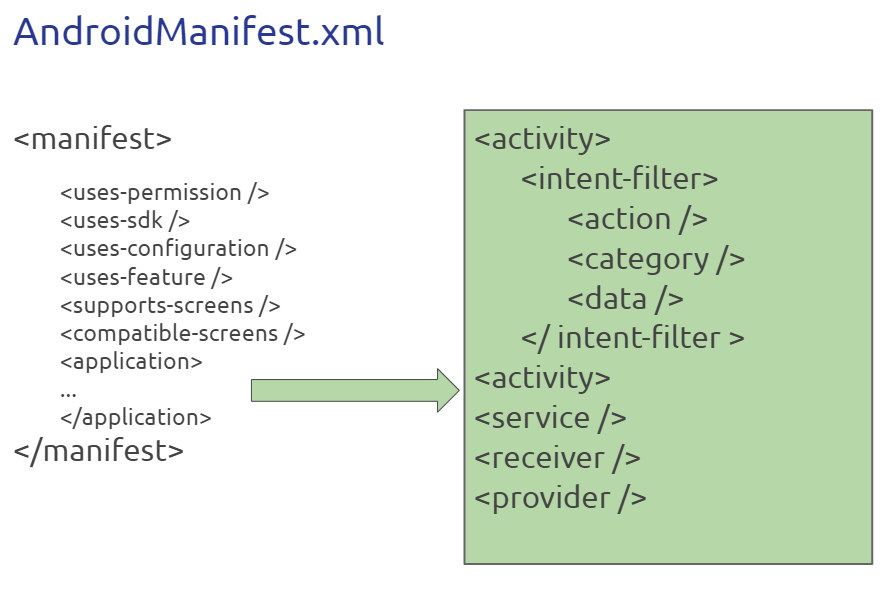


# 19.Файл манифеста AndroidManifest.xml: элементы и структура

**Манифест приложения**

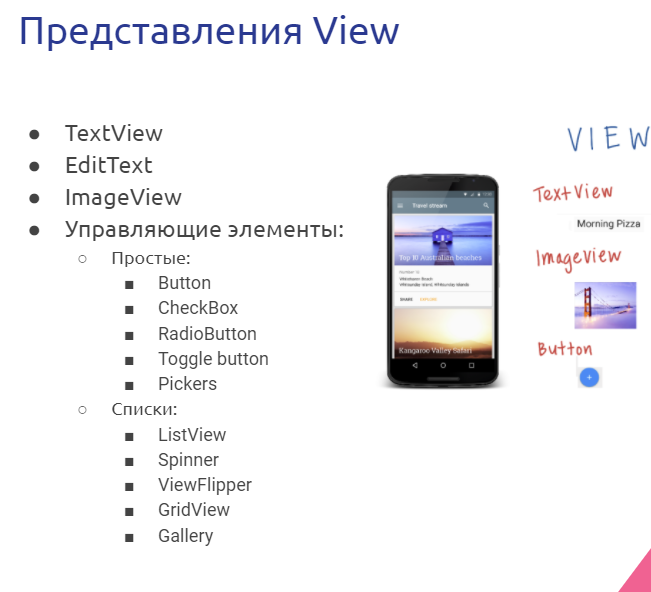
Файл манифеста представляет основную информацию о программе системе.

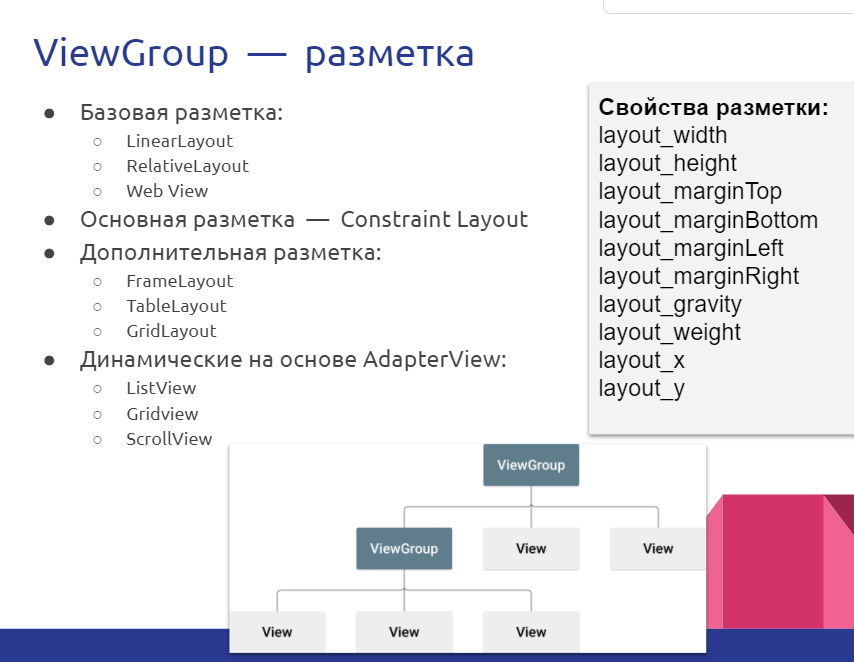
Основная информация в манифесте:

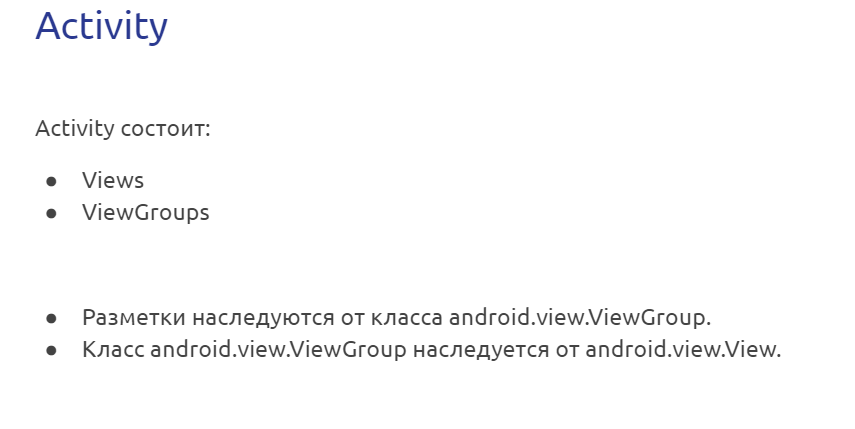
* Имя Java пакета приложения
* Описание компонентов приложения
* Определение процессов
* Объявление полномочий, которыми должно обладать приложение для доступа к защищенным частям API и взаимодействия с другими приложениями
* Объявление полномочий, которыми должны обладать другие приложения для взаимодействия с компонентами данного
* Список вспомогательных классов
* Определение минимального уровня Android API для приложения
* Список библиотек связанных с приложением
* 

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

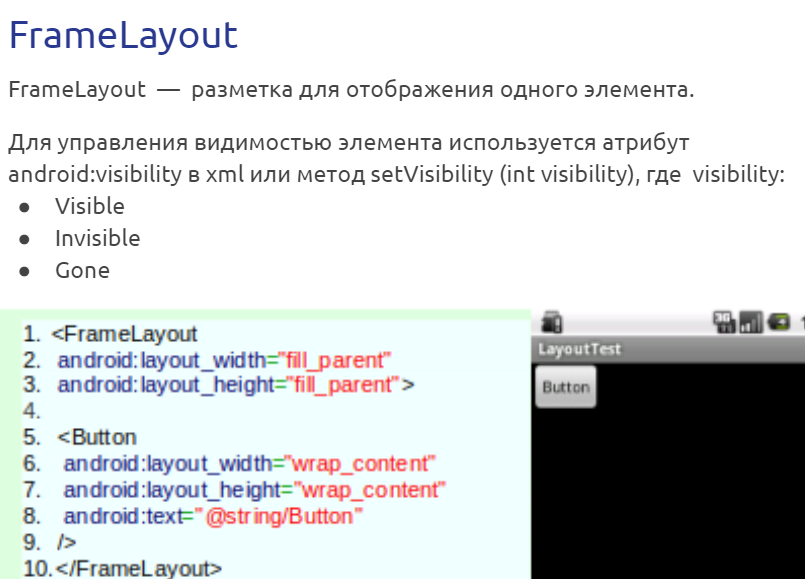
# 20.Характеристика разметок (макетов) страниц, их типы и основные элементы UI.

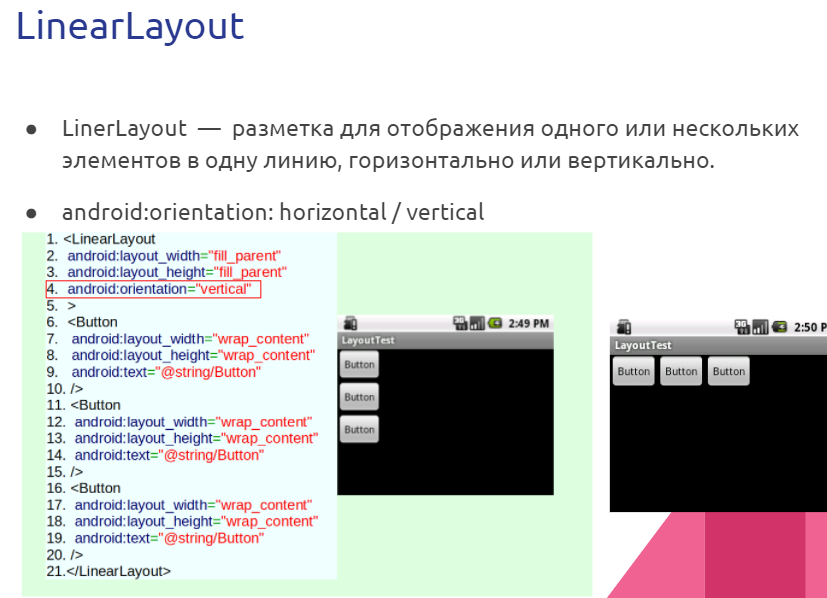
****

****

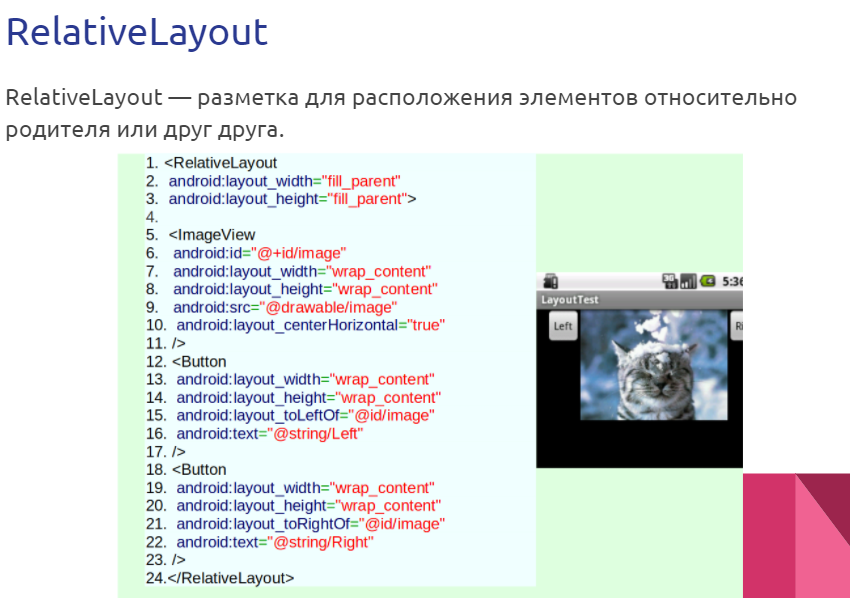
****

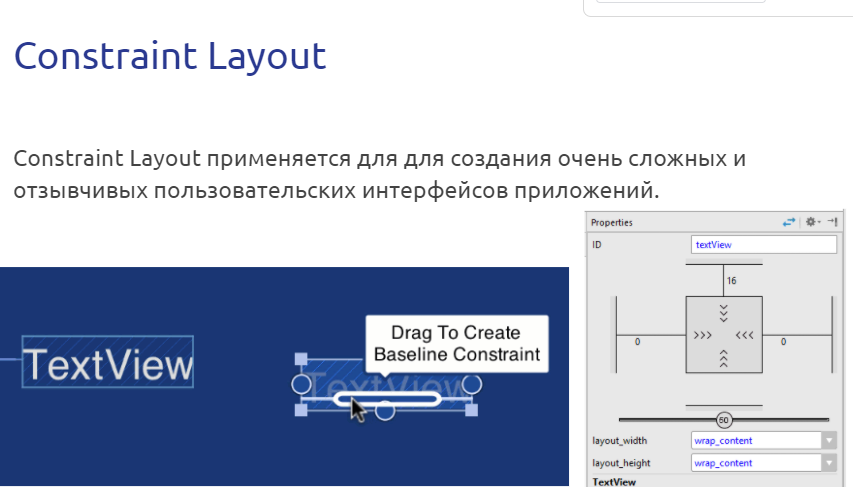
# 21.Разметка (макеты) страниц на примере FrameLayout и LinearLayout.

****

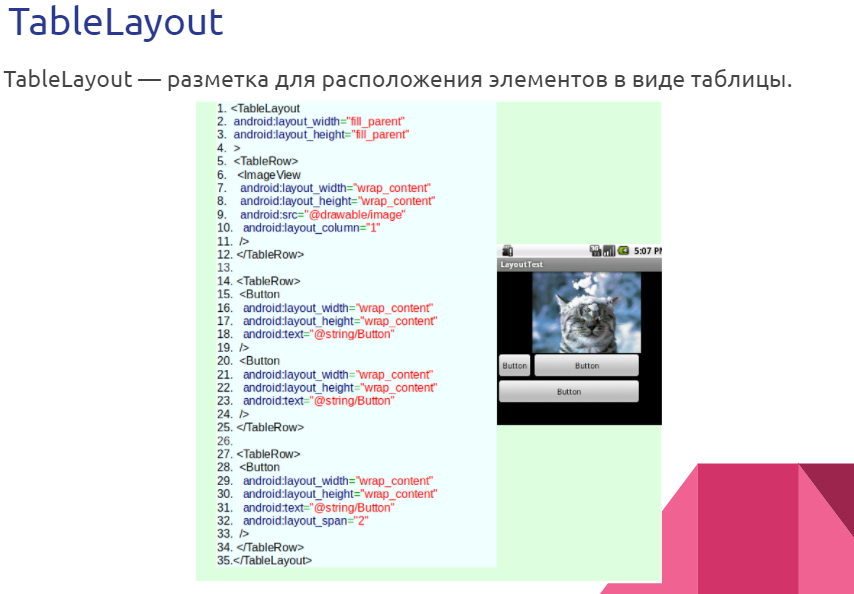
****

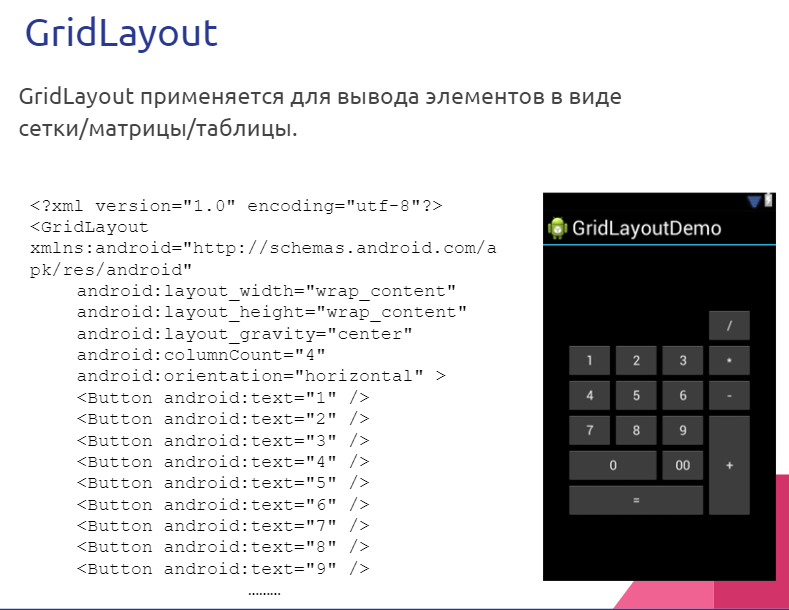
# 22.Разметка (макеты) страниц на примере RelativeLayout и ConstraintLayout.

****

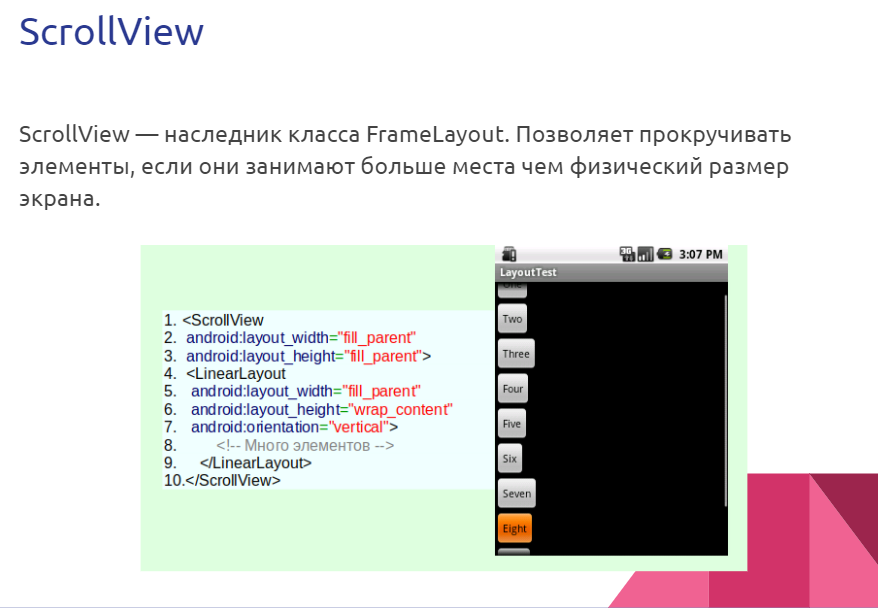
****

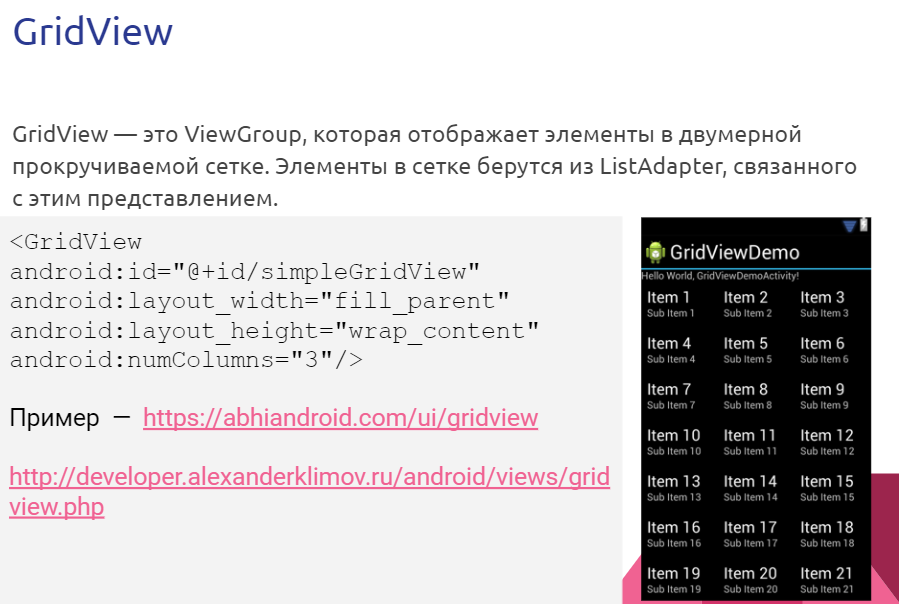
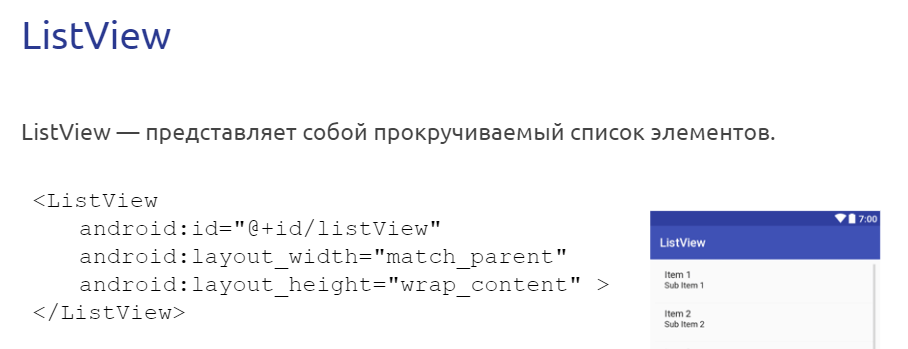
# 23.Разметка (макеты) страниц на примере TableLayout и GridLayout.

****

****

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

****

****

# 25.Ресурсы Android-приложения: строковые, булевые, числовые, меню, ресурсы разметки.

Каждое приложение на Android содержит каталог для ресурсов res и каталог для активов assets. Реальное различие между ресурсами и активами заключается в следующем:

* информация в каталоге ресурсов будет доступна в приложении через класс R, который автоматически генерируется средой разработки. То есть хранение файлов и данных в ресурсах (в каталоге res) делает их легкодоступными для использования в коде программы;
* для чтения информации, помещенной в каталог активов assets (необработанный формат файла), необходимо использовать AssetManager для чтения файла как потока байтов.

**Строковые ресурсы**

Строковые ресурсы помогают упростить процесс создания локализованных версий. Мы храним строки в файле string.xml при вызываем их по идентификатору. Благодаря этому процесс локализации становится проще.

Строковые ресурсы обозначаются тегом <string> и обычно хранятся в файле string.xml.

**Булевые ресурсы**

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.

Определение булевого ресурса:

<resources>  
 <bool name="autostart">true</bool>  
 <bool name="sound">no</bool>  
</resources>

Получить значение через код:

Resources res = getResources();  
boolean autostartSetting = res.getBoolean(R.bool.autostart);

**Числовые ресурсы**

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

Определение числового ресурса

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

<resources>

<integer name="max\_speed">75</integer>

<integer name="min\_speed">5</integer>

</resources>

Получить значение через код:

Resources res = getResources();

int maxSpeed = res.getInteger(R.integer.max\_speed);

**Ресурсы меню:**

Хорошая практика — создавать меню в формате xml, используя отдельные файлы.

Меню, описанное в формате XML, загружается в виде программного объекта с помощью метода inflate, принадлежащего сервису MenuInflater.

В ХМL-файле меню есть три элемента:

* <menu> — корневой элемент файла меню;
* <group> — контейнерный элемент, определяющий группу меню;
* <item> — элемент, определяющий пункт меню:
  + атрибуты item: id, title, icon

**Ресурсы разметки:**

Еще один из важных видов ресурсов - 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.

Пример кода разметки: setContentView(R.layout.main)

Пример xml-файла с разметкой:

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

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

>

<TextView

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:text="@string/hello"

/>

</LinearLayout>

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

Цветовые ресурсы

Для работы со цветом используется тег <color>:

* #RGB;
* #RRGGBB;
* #ARGB;
* #AARRGGBB;

Предопределенные названия цветов:

* background\_dark
* background\_light
* black
* darker\_gray
* holo\_blue\_bright
* holo\_blue\_dark и др

Обычно используется файл colors.xml в подкаталоге res/values:

· <?xml version="1.0" encoding="utf-8"?>

· <resources>

· <color name="red">#f00</color>

· <color name="yellow">#FFFF00</color>

· <color name="green">#FF00FF00</color>

· </resources>

Программное использование цветовых ресурсов:

· int myRedColor = activity.getResourses.getColor(R.color.red); // получаем значение красного цвета

· linearLayout.setBackgroundResource(R.color.yellow); // устанавливаем фон в желтый цвет

· linearLayout.setBackgroundResource(android.R.color.holo\_orange\_dark); // устанавливаем фон в оранжевый цвет

Еще один вариант использования ресурсов:

· Resources res = getResources(); // получим объект для работы с ресурсами

· int title\_color = res.getColor(R.color.title\_red); // получим значение цвета, заданного в ресурсах

Использование цветовых ресурсов в xml-файлах:

· <TextView

· android:layout\_width="match\_parent"

· android:layout\_height="wrap\_content"

· android:text="@string/anytext"

· android:color="@color/red" />

Ресурсы размеров

В Android используются следующие единицы измерения: пиксели, дюймы, точки. Все они могут входит в состав ХМL-шаблонов и кода Jаvа. Данные единицы измерения также можно использовать в качестве ресурсов при помощи тега <dimen> (обычно используют файл dimens.xml):

<resources>

<dimen name="in\_pixels">1px</dimen>

<dimen name="in\_dp">5dp</dimen>

<dimen name="in\_sp">100sp</dimen>

</resources>

Ресурсы визуальных стилей и тем

Визуальные стили и темы используются для хранения цветовых значений и шрифтов — styles.xml.

Вместо описания каждого стиля можно использовать ссылки, предоставляемые Android.

Для создания стилей используется тег <style>:

<style name="StyleName">

<item name="attributeName">value</item>

</style>

Создание ссылок на ресурсы:

android:textColor="?android:textColor"

android:background="?colorPrimary"

Ресурсы /res/drawable

/res/drawable

* android:shape: rectangle, oval, line, ring;
* <corners>;
* <gradient>: Linear, Radial и Sweep;
* <size>;
* <solid>.

Разрешения экрана — drawable-hdpi, drawable-mdpi, drawable-ldpi и др.

Размеры экрана — drawable-normal, drawable-large и др.

Ссылка на изображение в xml-файле:

<Button

android:id="@+id/button1"

...

android:background="@drawable/cat"

</Button>

Программное использование:

// вызываем getDrawable для получения изображения

BitmapDrawable bd = activity.getResources().getDrawable(R.drawable.cat);

// Затем можно использовать полученный объект, чтобы установить фон

button.setBackgroundDrawable(bd);

// или можно установить фон непосредственно по идентификатору ресурса

button.setBackgroundResource(R.drawable.icon);

Получение имени ресурса:

getResources().getIdentifier("image\_name","drawable", getPackageName())

getResources().getIdentifier("your.full.package.name:drawable/image\_name", null, null);

Получение идентификатора ресурса:

String mDrawableName = "cat1"; // файл cat1.png в папке drawable

int resID = getResources().getIdentifier(mDrawableName , "drawable", getPackageName());

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

Ресурсы отрисовываемых объектов

<resources>

<drawable name="black\_rectangle">#000000</drawable>

<drawable name="white\_rectangle">#ffffff</drawable>

</resources>

Использование в xml-шаблонах:

<TextView

android:layout\_width="fill\_parent"

...

android:background="@drawable/white\_rectangle" />

Программное использование:

// Получение отрисовываемого объекта

ColorDrawable whiteDrawable = (ColorDrawable)activity.getResources().getDrawable(R.drawable.white\_rectangle);

// установление его в качестве фона для текстового вида

textView.setBackground(whiteDrawable);

Ресурсы mipmap

· Mipmap — это замена ресурсам Drawable для значков приложения.

· Основная идея — 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.

· Подготовить значки и расположить их в папках mipmap-mdpi, mipmap-hdpi и т.д.

Ресурсы анимации

· Анимация свойств (Property Animation)

· Анимация представления (View Animation):

1. Tween Animation — основана на расчете промежуточных кадров и может использоваться для поворота, перемещения, растягивания, затемнения элементов.

1. TranslateAnimation

2. ScaleAnimation

3. RotateAnimation

4. AlphaAnimation

· Frame Animation — пошаговая анимация, т.е. последовательный вывод заранее подготовленных изображений.

Необработанные ресурсы

· Необработанные ресурсы хранятся в каталоге /res/raw.

· Каждый файл, помещённый в res/raw, имеет свой идентификатор, генерируемый в R.java.

· Для получения доступа к ресурсам используется метод openRawResource(), принадлежащий объекту Resource.

· В /assets так же хранятся необработанные файлы.

· Для файлов в /assets не генерируются идентификаторы ресурсов в R.java.

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

**Создание псевдонимов ресурсов**

В некоторых случаях создать альтернативный ресурс, действующий в качестве псевдонима для ресурса, сохраненного в каталоге ресурсов по умолчанию.

Порядок создания псевдонимов

1. Разместить ресурс в общий каталог, например *res/drawable/icon\_ca.png*

2. Создать xml-файл в соответствующих каталогах со ссылкой на исходный, например:

создать файл icon.xml в каталогах res/drawable-en-rCA/ и res/drawable-fr-rCA/, который ссылается на ресурс icon\_ca.png с помощью элемента <bitmap>;

**Компиляция ресурсов**

· Для компиляции ресурсов используется инструмент aapt.

· Res/values компилируются в Resources.arsc

· Без изменений

o Файлы ресурсов из assets и res/raw.

o Растровые файлы из каталога drawable(\*.png,\*.bmp,\*.jpg).

· Каждому ресурсу (исключая assets) присваивается идентификатор ресурса ID.

· Resources.arsc создается, чтобы хранить индекс ресурсов.

· R.java содержит перечень идентификаторов ресурсов.

· Другие xml-файлы, включая AndroidManifests.xml, компилируются в бинарный xml-файл.

· Каждому ресурсу (исключая assets) назначается уникальный идентификатор ID.

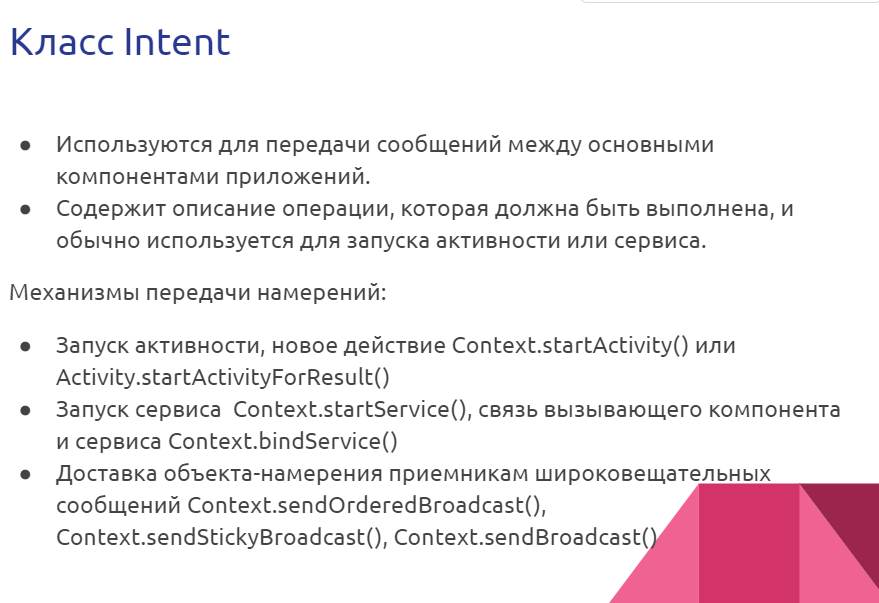
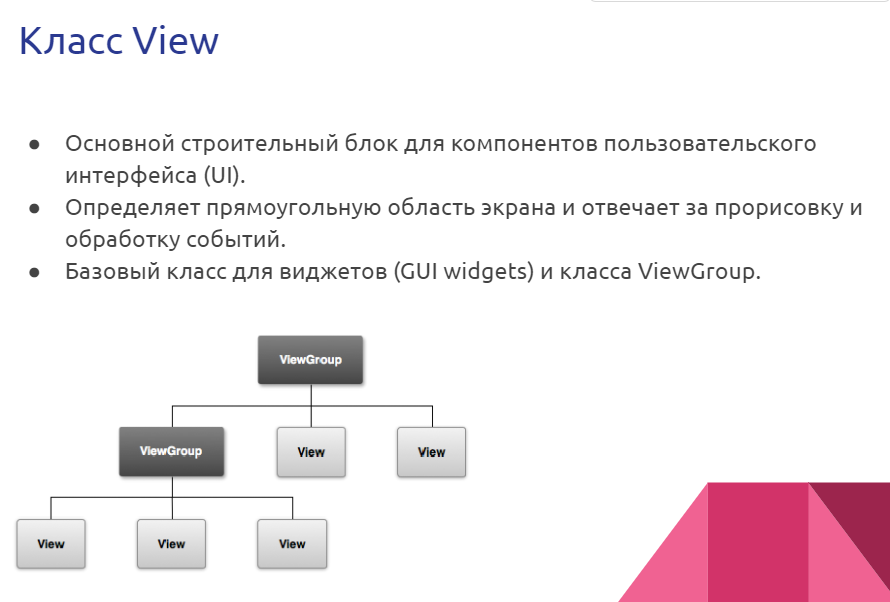
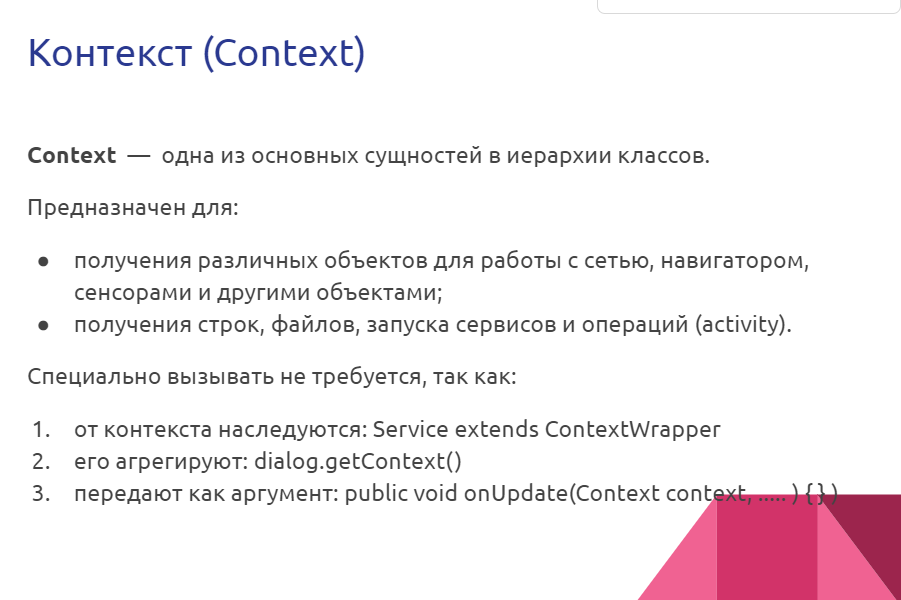
**Доступ к ресурсам из кода**

С помощью метода setImageResource() можно указать использование виджетом ImageView ресурса res/drawable/myimage.png:

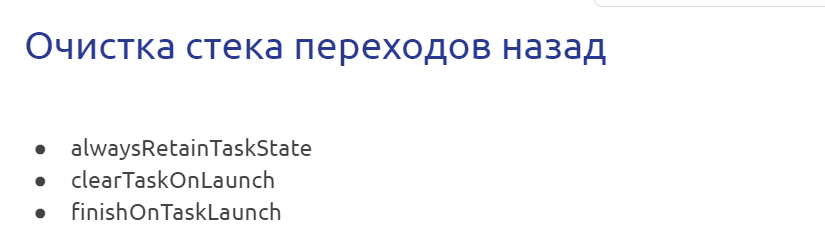
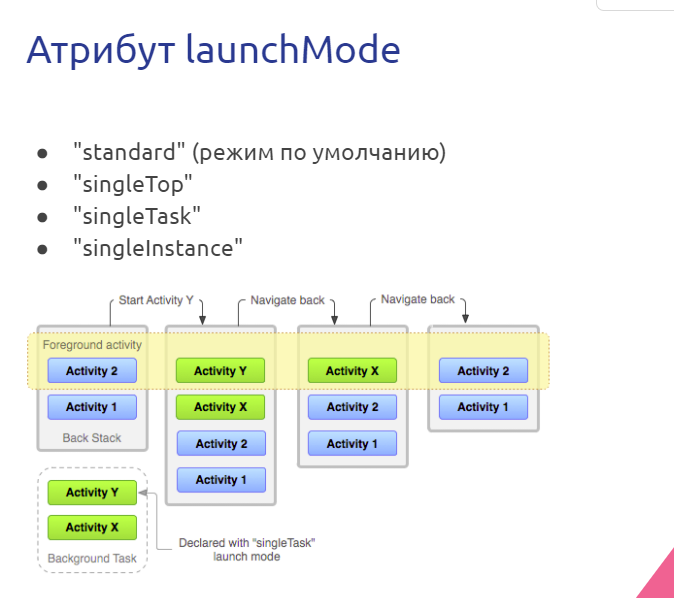
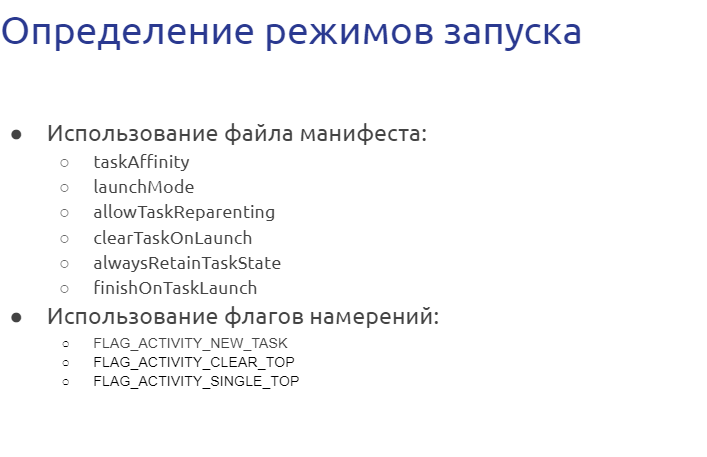
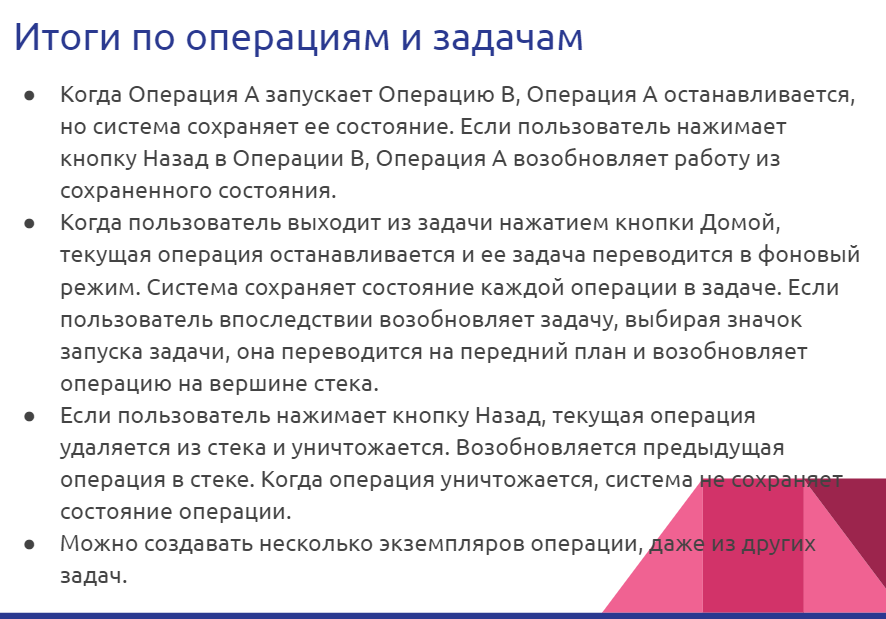
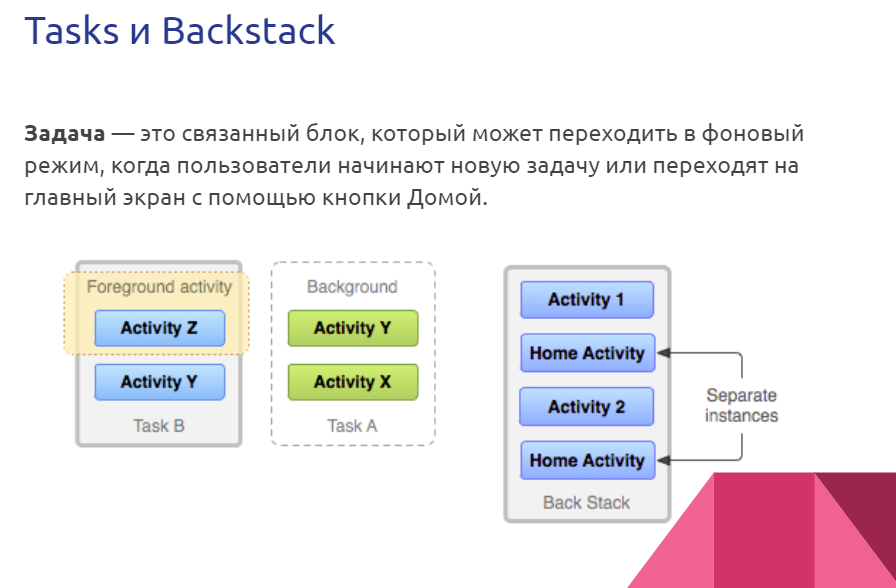
ImageView imageView = (ImageView) findViewById(R.id.myimageview);

imageView.setImageResource(R.drawable.myimage);

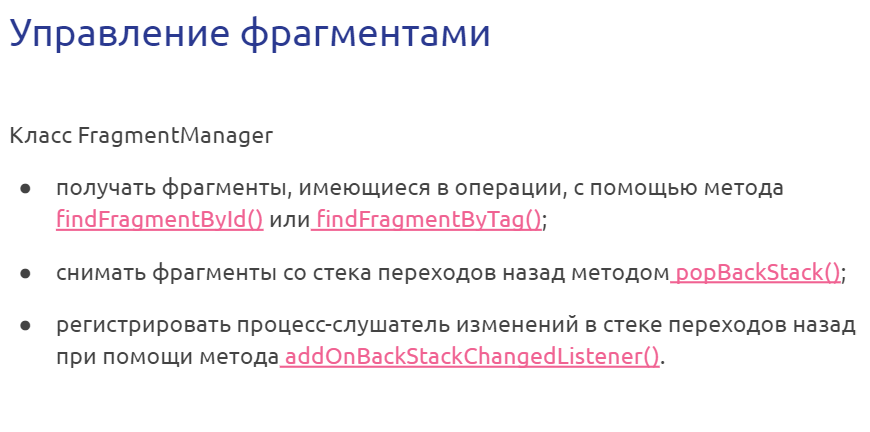
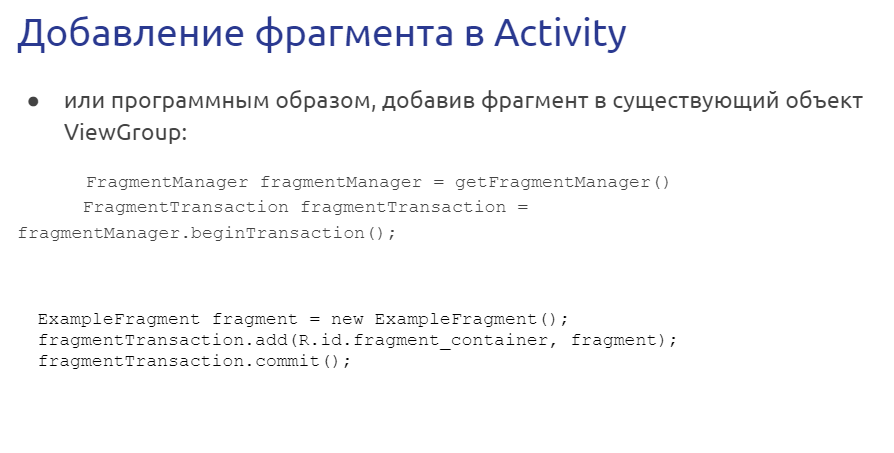
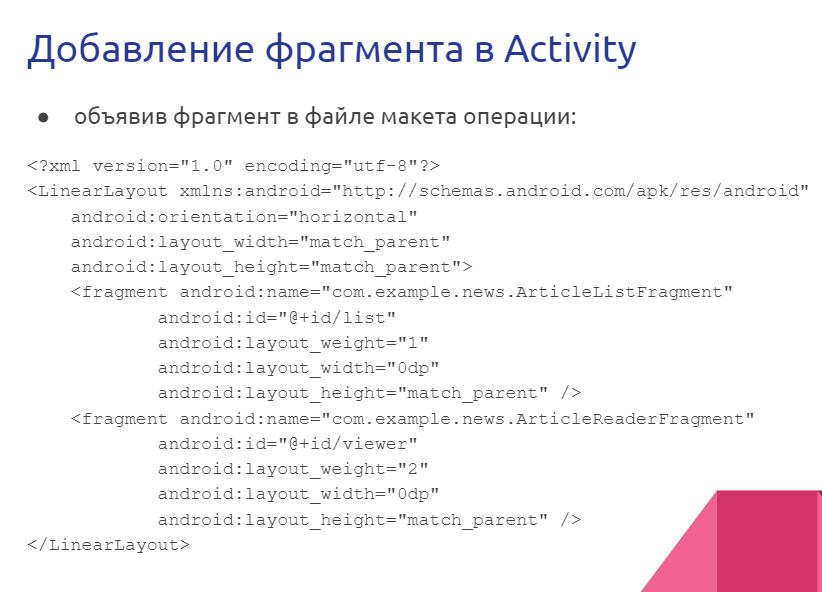
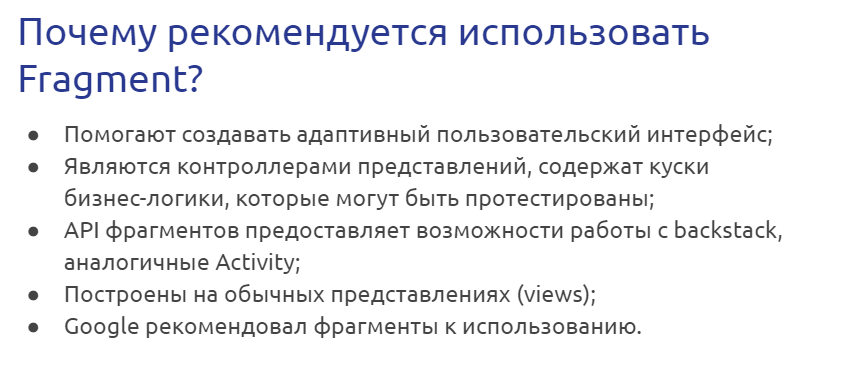
# 29.Иерархия классов в Android SDK.

****

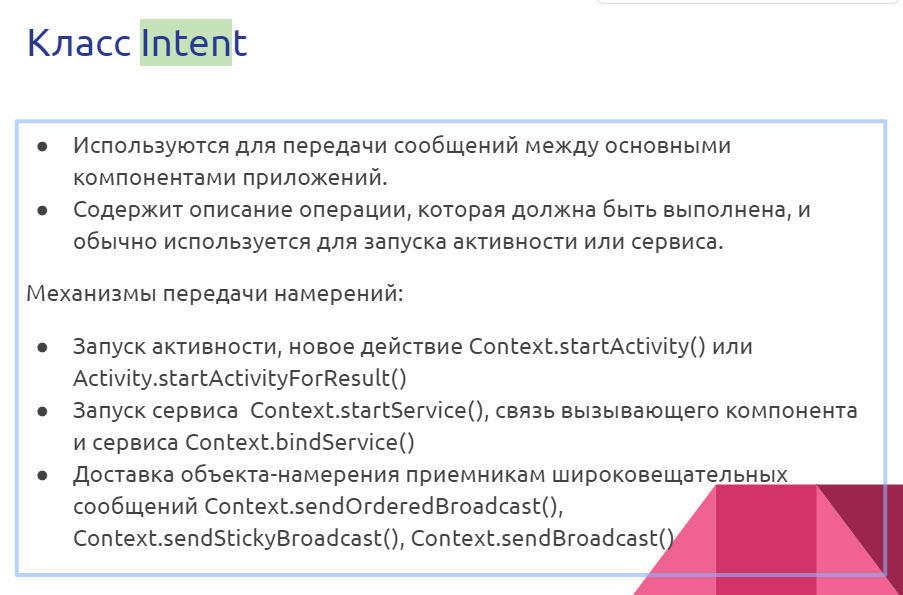
# 30.Задачи и стек переходов назад.

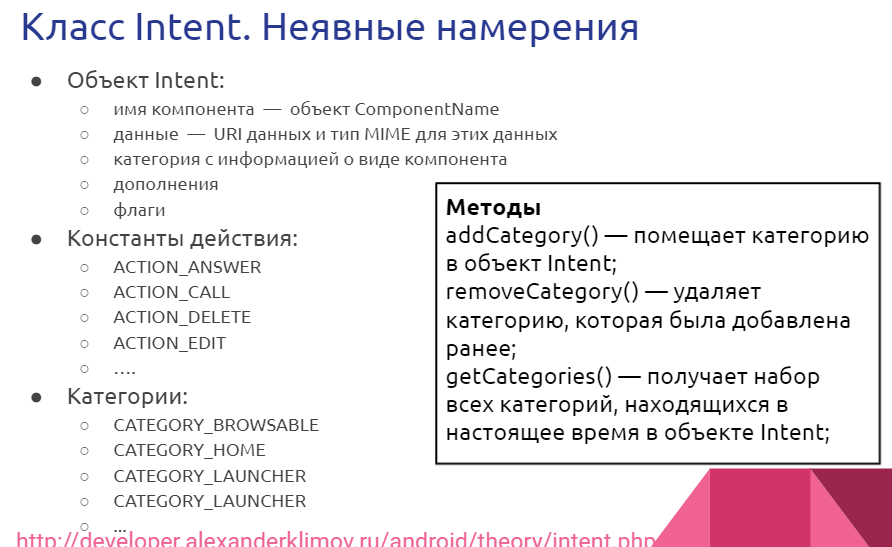
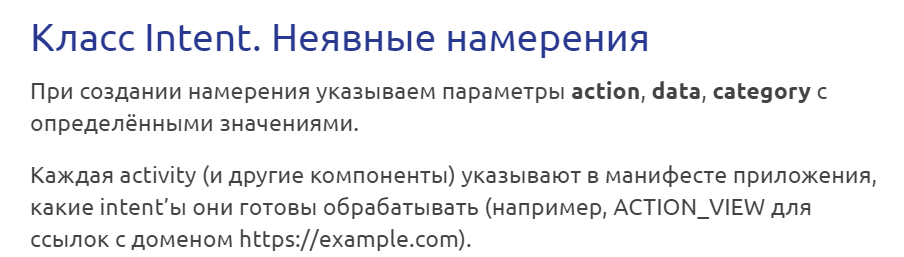
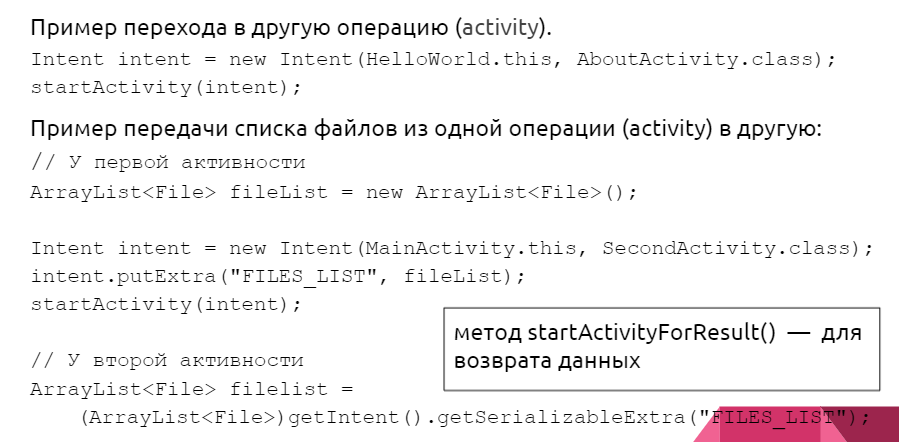
****

# 31.Фрагменты и управление фрагментами.

Фрагмент представляет собой многократно используемую часть пользовательского интерфейса вашего приложения. Фрагмент определяет и управляет своим собственным макетом, имеет собственный жизненный цикл и может обрабатывать свои собственные входные события. Фрагменты не могут существовать сами по себе - они должны размещаться в действии или другом фрагменте.

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

****

****

# 33.Активности в Android-приложении: класс Activity и методы класса. Жизненный цикл:

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

# 35.Контент-провайдеры. Класс ContentProvider и методы класса.

# 36.Приёмники широковещательных сообщений (Broadcast Receivers) и методы класса.

