ПРЕДИСЛОВИЕ. Серым помечены уебанские вопросы (и ответы на уебанские вопросы, на которые я хоть как-то могу ответить), которые некорректно сформулированы, непонятно что хотят или это бред.   
Поэтому, если в первом вопросе спрашивается про уровни архитектуры, а я отвечаю что-то, не завязанное на уровнях, то это не потому, что я в глаза долблюсь, а потому, что никаких четких уровней архитектуры андроид не существует и в нормальных источниках про это не говорится

1. Уровни архитектуры Android. Андроид работает на ядре Linux. До версии 5.0 андроид работал на виртуальной машине Dalvik, после чего и по сей день используется ART. Виртуальная машина – это то, что выполняет байт-код под платформу андроид. (Напоминаю, что в JVM-языках есть промежуточный компилируемый код (байт-код), который на лету должен интерпритироваться в машинный, этим и занимаются виртуальные машины). Выше этого находится android framework, который и предоставляет все возможности по созданию андроид-приложений, а еще выше уже находятся обычные библиотеки, упрощающие работу разработчикам.  
     
   2. Архитектура приложения, основные компоненты. Всего основных компонентов 4. Основные они потому, что каждый из них позволяет стартовать работу приложения, а еще совсем без них приложений не существует. Activity позволяет работать с UI, Service позволяет вести работу в фоне (музыку спотифая крутить, хотя телефон заблокирован). Broadcast receiver позволяет слушать сообщения системы, реагировать на них и кидать свои. Примером явл. уведомление о низком заряде батареи. Есть слушатель этих сообщений, и, как только он ловит сообщение о низком заряде телефона, кидает уведомление. Content Provider позволяет обмениваться данными между приложениями. Камера в инсте так и работает, что запрашивает данные от камеры через Content Provider.  
     
   3. Манифест приложения. Очень важный файлик на языке xml, в котором прописываются используемые основные компоненты, которые есть в приложении. Все разрешения на камеру, интернет и т.п., название приложения, иконка и другие возможности типа поддержки арабской верстки прописываются тут.  
     
   4. Ресурсы приложения. Отдельная папка res, куда должны прописываться все строки приложения, все вёрстки, картинки, шрифты, стили, темы, иконки и прочие ресурсы.  
     
   5. Активность (Activity). Жизненный цикл активности**.** Основной компонент Android. Позволяет отрисовывать интерфейс экрана и указывать реакции на взаимодействия с ним (реакции на клацание кнопок). Жизненный цикл – это этапы работы Activity. Если у нас приложение только открылось, то это один этап, потом оно переходит в другой этап к активной работе. Если приложение закрывается, то мы переходим на этап завершения работы активности (чистим память, например). Или если мы свернули приложение, то это тоже свой этап. Все этапы на фото, подробно объяснять… всё равно не запомните

  
6. Службы (Сервисы). Жизненный цикл. Способы запуска. Методы. Использование. Основной компонент андроид. С класса Service (простейшая реализация сервиса) можно было стартовать приложение, можно его было вызвать из всех основных компонентов. В жизненном цикле есть этапы onCreate, onStartCommand, onBind, onUnbind, onDestroy. С андроид 8.0 (8 лет назад) не работают, так как устарели и не поддерживаются. Сейчас поддерживаются Work Manager и Job Scheduler (это тоже сервисы, но современные). Сервисы позволяют заниматься фоновой работой, когда приложение свёрнуто или даже закрыто (ваша интимка из телеграмма всё равно отправится вашему отцу, даже если приложение вы закрыли, т.к. работает сервис. Музыка в спотифае так же работает)   
  
7. Android. Контент-провайдеры. Основной компонент андроид. Позволяет приложениям передавать друг другу данные. Именно на его основе в ваших приложениях работают камеры, именно они дают возможность телеграмму доступ к фоткам и файлам на вашем телефоне. С контент-провайдера можно стартануть приложение (когда, например, ты нажимаешь на .pdf файл, и тебе отрывается ридер. Он открывается и в него попадает этот pdf на отрисовку именно благодаря контент провайдеру)  
  
8. 8. Приемники широковещательных сообщений (Broadcast Receivers). Основной компонент андроид. Позволяет реагировать на широковещательные сообщения и создавать их. Широковещательные сообщения – это типа кричащегов толпе. Он орёт, и ты можешь при желании отреагировать как-то на его крик среди чужих фоновых криков. А также ты сам можешь крикнуть в толпу. Такие крики присылает и система всем приложениям. Например, кричит, что низкий уровень заряда, и приложение может на это как-то отреагировать (телеграмм позволяет настроить с какого уровня заряда будет энергосберегайка)  
  
9. Фрагменты. Жизненный цикл фрагмента. Думаю, вы знаете, что существуют активности (классы, в которых мы выводим интерфейс и прописываем реакии на взаимодействие с ним). И есть всякие кнопочки, тексты и т.п. Фрагмент – это среднее между активностями и кнопочкой. Он как кнопочки встраивается в активность и позволяет также выводит интерфейс внутри себя и прописывать реакии. Для чего он тогда нужен, если есть активности? Сейчас принято использовать именно фрагменты и помещать их внутри единственной активности на всё приложение. Это для того, что с фрагментами и одной активностью сильно проще навигация, также фрагменты имеют более тонкую настройку, они легче и модульнее. Жизненный цикл (жц) фрагмента связан с жц активности (типа создается активность -> создается фрагмент. Так же с уничтожением, удалением и т.п.), но имеет дополнительные стадии для более тонкой настройки.



10. Статическое и динамическое использование фрагментов. Статическое использование фрагментов – это когда у нас есть Activity, внутри которой мы строго прописываем фрагмент, который там должен быть. Этот подход в реальности не используется. Используется динамический подход, когда внутри Activity мы устанавливаем контейнер фрагментов. И внутри этого контейнера прописываем в коде любой фрагмент с возможностью его менять, производить навигацию и т.п. На данный момент рекомендованной практикой считается использование одной активности и контейнера для фрагментов, которые и переключаются между собой как экраны  
  
11.Списки и адаптеры. Разновидности стандартых адаптеров. Списки позволяют отобразить кучу типизированных элементов внутри одного элемента интерфейса. Например, чаты телеграмма списком и отрисовываются. Раньше использовался listView, однако он устарел и ныне не используется, т.к. отрисовывал все элементы списка сразу же. Т.е. если у тебя миллион чатов в тг, то listView сразу же и пытался бы их отобразить. На его смену пришел и используется RecyclerView, смысл которого в отрисовке только того, что видно на экране а также переипользовании контейнеров элементов списка. Т.е. отрисуется не миллион чатов, а лишь 10, которые видны на экране, и по парочке сверху и снизу, чтобы прогрузка не была заметна глазу. Каждый чат при этом отображается внутри контейнера интерфейса элемента списка. А эти контейнеры переиспользуются, чтобы оптимизировать нагрузку на телефон. И, если ты скроллишь вниз, то контейнеры сверху незаменто для тебя будут перенесены вниз и вместо отрисовки новых элементов с нуля на существующих просто поменяется иконка и ник пользователя. Адаптеры же – это отдельные классы, которые позволяют настраивать работу списков, например.  
  
13.14.15. Пользовательские компоненты ( custom views ) и составные (compound\composite component).  
  
Андроид-система предоставляет всякие кнопочки, текстовые поля и прочее, но она не имеет чего-то очень необычного, типа радиального графика для отрисовки розы ветров. Поэтому эту хуню рисовать надо самому, это и есть Custom View, ты разрабатываешь свой компонент (элемент интерфейса) для своих нужд. Любой свой элемент, каким способом он ни был бы сделан – это custom view, но они подразделяются на composite (это когда ты составил из кучи уже существующих кнопочек что-то свое) и compound (когда ты при момощи класса Canvas реально рисуешь треугольники, квадратики и кружочки, рисуя из них то, что хочешь).  
Все Custom View имеют свой жизненный цикл отрисовки, состоящий из 3 методов: onMeasure, onLayout и onDraw. В методе onMeasure считаются размеры, которые будет занимать твоя CustomView на экране. В onLayout считается, куда ты это разместишь. И только в onDraw ты отображаешь своё творчество.

15. Многопоточность. Использование Thread + Handler  
16. Многопоточность. Использование AsyncTask. Этой хуйней уже лет 10 не пользуются, она устарела. Сама многопоточность позволяет приложению совершать несколько действий одновременно. По умолчанию всё делается на главном потоке, и, если ты решил сделать сложные математические вычисления на главном потоке, то ты еблан, так как в это время у тебя всё приложение зафризит, ибо интерфейс будет ждать, пока освободится основной поток. Поэтому время секундомер считает на второстепенном потоке, иначе бы приложение застывало, ибо основной поток считал бы время, а не отрисовывал интерфейс. На данный момент Для многопоточности используются rxJava/rxKotlin и Kotlin coroutines. Они позволяют отдавать работу главному потоку, слабому и сильному второстепенному, а также переключаться между ними последи работы. Kotlin coroutines работает не с самими потоками, а с быстрой абстракцией над ними.

17. Хранение данных. Preferences. Shared Preferences – это класс, позволяющий в память телефона сохранить какую-то базовую НЕ сложную инфу типа выбранных настроек приложения, чтобы при перезапуске приложения настройки не слетали. Для сложной инфы используются локальные базы данных Realm и Room.  
  
18. Хранение данных. Файлы. Хуйня какая-то… Все данные через Shared Preferences, Room и Realm и так в файлах хранятся. А другим для локального сохранения данных приложения не пользуются…  
  
19. Хранение данных. SQLite. MySql и PostgreSQL помните? Ну так SQLite – это подобная, но упрощенная библиотека для работы с языком SQL, чтобы разворачивать на телефоне локальные базы данных Realm и Room для сохранения всяких данных типа кеша на телефоне. И при помощи запросов GET, DELETE и т.п. можно с ними работать