Типовые вопросы собеседования

Важно знать что вас ждет на интервью - обычно это выглядит примерно так

Здесь собраны вопросы которые я встречал на технических интервью и технических секциях hr-скринингов. Ответы я беру из интервью, часть их я формулирую сам (поэтому не стоит доверять им на 100%)

Архитектурная секция

Подробней изучить материал который я использовал при подготовке можно тут

kotlin(17)

Возможно ли создать data class с конструктором без параметра, но объявить его внутри?

Ответ: Нет, такое невозможно сделать. Первичный конструктор должен иметь как минимум один параметр.

Какие есть коллекции в Kotlin

Ответ:

1. List
2. Map
3. Set

Расскажите о “равенстве” в Kotlin

Ответ:

В Kotlin есть два типа равенства:

- Равенство структур (== - проверка через equals())
- Равенство ссылок (=== - две ссылки указывают на один и тот же объект)

Равенство структур
Структурное равенство проверяется оператором == и его отрицанием !=. По соглашению выражение a == b транслируется в:

a?.equals(b) ?: (b === null)

Если a не null, вызывается функция equals(Any?), иначе (т.е. если a указывает на null) b ссылочно сравнивается с null.
Заметьте, что в явном сравнении с null для оптимизации нет смысла: a == null будет автоматически транслироваться в a === null.
Чтобы обеспечить вашу реализацию проверки равенства, переопределите функцию equals(other: Any?): Boolean. Функции с тем же именем и другими сигнатурами, например equals(other: Foo), не влияют на проверку равенства с помощью операторов == и !=.
Структурное равенство не имеет ничего общего со сравнением, определяемым интерфейсом Comparable<...>, поэтому только пользовательская реализация equals(Any?) может повлиять на поведение оператора.

Равенство ссылок

Равенство ссылок проверяется с помощью оператора `===` и его отрицания `!==`.
Выражение `a === b` является истиной тогда и только тогда, когда `a` и `b` указывают на один и тот же объект. Для значений, представленных примитивными типами во время выполнения (например, `Int`), проверка равенства `===` эквивалентна проверке `==`.

Для чего нужно ключевое слово object

Ключевое слово object одновременно объявляет класс и создаёт его экземпляр.

С его помощью можно реализовать шаблон "Одиночка".

Иногда требуется класс, который должен существовать в одном экземпляре. В Kotlin имеется специальный синтаксис объявления объекта для подобных случаев.

```
object One {
    val cats = arrayListOf<Cat>()

    fun callCat() {
        for (cat in cats) {
            ...
        }
    }
}
```

Вы можете задавать свойства, методы, блоки инициализации, но не можете создавать конструкторы (как основные, так и вторичные).

Обращаться к методам и свойствам класса можно через имя объекта.

```
One.cats.add(Cat(...))
One.callCat()
```

Какие модификаторы доступа есть в kotlin?

Ответ:

- private
- public
- protected
- internal

Как НЕЛЬЗЯ объявлять data class

a) data class MyClass(val a: Int)

b) open data class MyClass(val a: Int)

c) data class MyClass(val a: Int): Base()

d) data class MyClass()

Каков порядок инициализации переменных и выполнения блоков кода при создании экземпляра ExampleClass? Для удобства можно ответить цифрами, указанными в комментариях.

Мой ответ: 4312

```
class ExampleClass(val first: String) { // 1

    val second = "Second" // 2

    init {

        println("Init block") // 3
```

```
    }
    companion object {
        val third = "Third" // 4
    }
}
```

Что означает знак равно здесь?

val a: String! = b.foo()

Класс Any имеет в себе лишь hashCode(), equals(), toString(). Каким образом его наследники могут использовать функции класса Object из Java?

Ответ:

Это функции расширения, которые генерируются самим kotlin.

Является ли kotlin Int аналогом Integer из Java?

Нет, не является. Этот класс репрезентует Java int. Сам Int наследуется от Numbers. Для нулабельных будет создаваться класс-оболочка.

Для чего нужен inline?

Встраиваемые функции, позволяют предотвратить создание объектов при компиляции кода. Здесь можно почитать подробнее.

Что такое sealed class?

Sealed class (изолированный класс) — это класс, который является абстрактным и используется в Kotlin для ограничения классов, которые могут наследоваться от него. Такое же можно провернуть и с интерфейсом. Подробнее тут

- Конструктор изолированного класса всегда приватен, и это нельзя изменить.
- У sealed класса могут быть наследники, но все они должны находиться в одном пакете с изолированным классом. Изолированный класс "открыт" для наследования по умолчанию, указывать слово open не требуется.

- Наследники sealed класса могут быть классами любого типа: data class, объектом, обычным классом, другим sealed классом. Классы, которые расширяют наследников sealed класса могут находиться где угодно.
- Изолированные классы абстрактны и могут содержать в себе абстрактные компоненты.
- Изолированные классы нельзя инициализировать.
- При использовании when, все подклассы, которые не были проверены в конструкции, будут подсвечены IDE.
- Не объявляется с ключевым словом inner.

Какая сигнатура функции let?

```
@kotlin.internal.InlineOnly
public inline fun <T, R> T.let(block: (T) -> R): R {
    contract {
        callsInPlace(block, InvocationKind.EXACTLY_ONCE)
    }
    return
```

Unit и Nothing

Тип Unit в Kotlin выполняет ту же функцию, что и void в Java.
Nothing является типом, который полезен при объявлении функции, которая ничего не возвращает и не завершается.

Подробней можно почитать тут

Что означает ! рядом с возвращаемым типом у функции?
Подробней почитать об этом тут. В целом это обозначение платформенных типов, они небезопасны могут быть как null так и не null.

Во что преобразуются extension-функции?

Extension-функции преобразуются в статические методы.

Как реализовать кастомный делегат в котлине?

В целом исчерпывающий ответ тут

Расскажите про typealias

Typealias — это механизм создания синонимов (псевдонимов) для существующих типов. То есть, можно создать новое имя для уже существующего типа данных.

Псевдонимы типов полезны, когда вы хотите сократить длинные имена типов, содержащих обобщения. К примеру, можно упрощать названия типов коллекций:

```
typealias NodeSet = Set<Network.Node>
```

```
typealias FileTable<K> = MutableMap<K, MutableList<File>>
```

Взял информацию [тут]

Корутины(12)

В чем отличие SharedFlow от StateFlow?

Ответ:

SharedFlow это аналог hot observable из RxJava. Про StateFlow можно сказать, что он является частным случаем SharedFlow с определенными параметрами.

```
// MutableStateFlow(initialValue) is a shared flow with the following
parameters:
val shared = MutableSharedFlow(
    replay = 1,
    onBufferOverflow = BufferOverflow.DROP_OLDEST
)
shared.tryEmit(initialValue) // emit the initial value
val state = shared.distinctUntilChanged() // get StateFlow-like
behavior
```

В чем отличие Dispatcher.IO от Dispatcher.Default?

Ответ

Dispatcher.IO предназначен для быстрых и частых операций, вроде запросов в сеть или записи в файл. Под капотом оперирует пулом тредов размером или в 64, или в количество ядер JVM.

Dispatcher.Defatult предназначен для операций требующих вычислений, под капотом ограниченный пул потоков размером в количество ядер. По умолчанию создается для вложенной корутины, если Dispatcher не был задан в контексте Scope.

Каких диспатчеров не существует?

a) Default

b) Computation

c) IO

d) Unconfined

e) Single

Что произойдет при вызове данной функции?

```
suspend fun unknownFunction() {
    coroutineScope {
        launch(Job()) {
            delay(1000)
            println("Hello!")
        }

        cancel()
    }
}
```

a) Запустится корутина launch, после задержки в 1 секунду, будет выведено “Hello!”, потом скоуп отменится.

b) Запустится корутина launch, scope будет сразу же отменен, отменяется корутина launch, ничего выведено не будет.

c) Запустится корутина launch, scope будет сразу же отменен, однако корутина launch продолжит свою работу, через секунду будет выведено “Hello!”.

d) Нет правильного ответа.

Как преобразовать StateFlow в Compose State?

Используем Composable-функцию collectAsState. Подробней о подходе тут

В чем разница между launch и async?

Первый возвращает значение типа Job и не возвращает никакого результата.
Второй возвращает значение типа Deferred и возвращает результат посредством вызова метода await().

Есть ли разница что использовать для следующей задачи(async, launch)

Есть два запроса, которые выполняются параллельно, надо получить из них результат. Объединить его и выполнить.

Ответ:
Здесь подойдет async.

Какие есть способы обработки ошибок в корутинах?

- CoroutineExceptionHandler
- try-catch
- catch(Flow)
- retry(Flow)
- retryWith(Flow)

Можно ли отменить запущенную корутину, как, что при этом произойдет?

Подробно это изложено здесь.
В целом, можно отменить при помощи cancel() метода. У джобы сменится статус активности - его надо проверять. саспенд функцию можно реализовать посредством suspendCancallableCoroutine и она автоматически подхватит это. Не забывай про флаг NonCancallable - это сделает ее неотменяемой, но это не рекомендуется.

Не забудь про CancellationException

Для чего нужен Dispatchers.Unconfined?

Кратко с примерами можно почитать тут. По сути отключается флаг диспатчеризации потоков, до первой остановки посредство саспенд функции.

В чем отличие Dispatchers.Main от Dispatcher.Main.immediate?

Последний выполнит код сразу, если мы сейчас находимся в главном потоке.

Что такое LaunchedEffect, DisposableEffect?

Тут раскрыто подробно

Что такое compositionlocalof?

Тут есть вся необходимая информация

ROOM(3)

Как сохранить сложный класс в ROOM?

Аннотация @Embedded. Чуть подробнее тут.

Как в Room сохранить BigInteger?

Написать класс с методами помеченными @TypeConverter . Один метод для конвертации, другой для восстановления. Через одноименную анотацию помечаем поле в Entity.

Как в реальном времени получать обновления из ROOM?

Пометить возвращаемый тип как Flow

Dagger2(3)

Как сделать так, чтобы класс можно было заинджектить куда либо?

Явно объявить его первичный конструктор и пометить его аннотацией @Inject . Прописать компоненту гет метод, который скажет даггеру что требуется сгенерировать под капотом провайдинг экземпляра этого класса.

Если это внешний класс, то можно создать билдер или фабрику и посредство нее обеспечить экземпляр некоторого библиотечного класса в нашем дереве зависимостей.

Что делает аннотация Binds?

Подробней и с примерами кода можно почитать тут. В целом, эта аннотация позволяет нам декларировать модуль как абстрактный класс или интерфей и не писать самим создание экземпляра класса. Стоит учитывать что при этом класс, который мы собираемся провайдить через Binds, должен содержать первичный конструктор помеченный аннотацией @Inject

Для интерфейса на вход модуля провайдера надо объявить входной аргумент класса реализующего ваш интерфейс(если вы им закрыли свою реализацию).

Как работает Dagger 2 - через кодогенерацию или рефлексию?

Кодогенерация

RxJava(5)

В чем отличие map от FlatMap?

Основное различие между map() и flatMap() состоит в том, что map() преобразует каждый элемент потока независимо, а flatMap() может преобразовывать один элемент в несколько и уплощать структуру данных.

Какие есть Subject'ы и какие у них отличия?

- Publish Subject
- BehaviorSubject
- AsyncSubject
- ReplaySubject
- UnicastSubject

Подробнее тут или тут

Сколько раз можно вызывать subscribeOn?

1

Какие Schedulers есть в RxJava?
Schedulers.io()
Schedulers.computation()
Schedulers.single()
Scheduler.newThread()

Scheduler.trampoline()
AndroidSchedulers.Main()

Подробней о шедулерах тут

В чем разница между flatMap, concatMap, switchMap?

Ответ тут

Как из холодного наблюдателя(cold observer)сделать горячего наблюдателя (hot observer) и наоборот?

Существует два способа трансформировать cold observable в hot.

Первый – это использование методов publish() и connect().
Метод publish() создает из observable объект типа ConnectableObservable. ConnectableObservable не начинает рассылать элементы, когда на него подписываются. Рассылка запускается после вызова метода connect(). Когда вызван метод connect(), начинается эмитинг элементов независимо от того, есть ли подписчики.

Второй способ – обернуть observable в subject, как показано на картинке. В этом случае эмитинг элементов оригинального observable стартует, когда на него подписывается subject. A subject, являясь hot observable, рассылает элементы независимо от наличия подписчиков.

Это можно сделать комбинацией методов replay() и autoConnect(0).

Метод replay() создает объект ConnectableObservable, который кэширует все элементы, отправляемые оригинальным hot observable.

Как было описано ранее, ConnectableObservable начинает эмитить элементы, когда на нем вызывается метод connect().

Чтобы получить поведение обычного cold observable, на ConnectableObservable вызывается метод autoConnect(numberOfSubscribers: Int). Этот метод принимает параметром переменную типа Int, которая задает количество подписчиков, необходимых для вызова connect(). Если параметром передано неположительное число, то connect() вызывается сразу.

В результате последовательного вызова replay() и autoConnect(0) создается observable, который эмитит все закэшированные элементы при вызове на нем subscribe(), т.е. ведет себя как cold observable.

Compose(3)

Какие есть виды layout в Compose?

Ответ:

1. Column
2. Row
3. Box
4. Кастомный layout при помощи функции layout

Как задать внешние отступы для Box?

Ответ:

Modifier.padding(...)

Что такое LaunchedEffect и для чего он нужен

Можно прочитать тут или тут

Android(38)

Жизненный цикл Activity

На каждом этапе жизненного цикла Activity применяется соответствующий метод.

- **`onCreate`**: вызывается при первоначальном создании Activity и используется для ее настройки, например для наполнения макета и инициализации переменных.
- **`onStart`**: вызывается, когда Activity становится видимой для пользователя, и используется для запуска необходимых процессов и анимации.
- **`onResume`**: вызывается, когда Activity становится приоритетной, то есть Activity, с которой пользователь взаимодействует в данный момент. Используется для запуска и возобновления процессов и анимаций, которые были приостановлены в методе `onPause`.

- `onPause`: вызывается, когда Activity больше не является приоритетной, но все еще видна пользователю. Используется для приостановки процессов и анимаций, которые не должны выполняться в фоновом режиме.
- `onStop`: вызывается, когда Activity больше не видна пользователю, и используется для остановки процессов и анимаций, которые больше не нужны. Гарантированно вызовется
- `onDestroy`: вызывается, когда система собирается завершить Activity, и используется для освобождения ресурсов и очистки оставшихся задач.

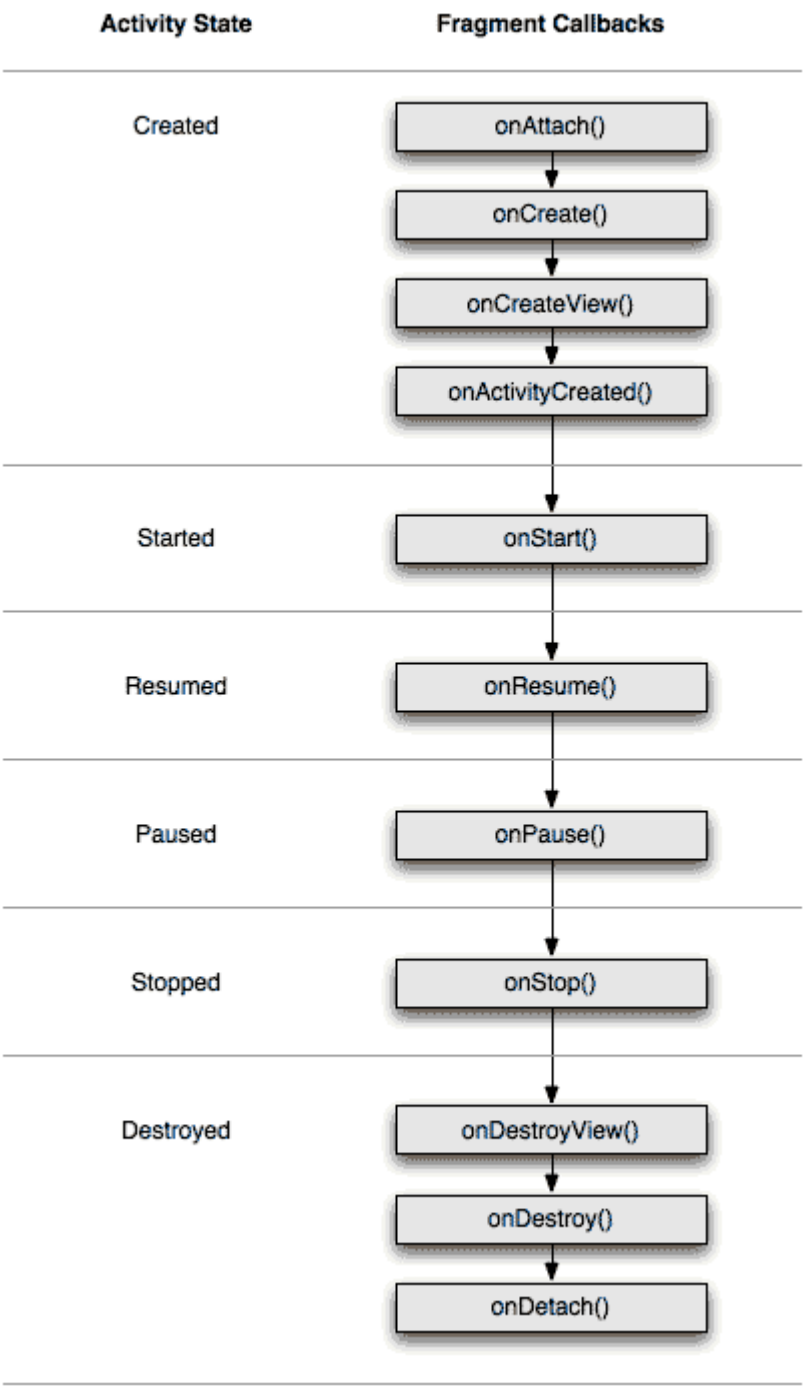
Какие есть основные компоненты Android

Ответ:

- Activity,
- Service,
- ContentProvider,
- BroadcastReceiver

Жизненный цикл фрагмента

Ответ:



onAttach(Activity)
Вызывается, когда фрагмент связывается с активностью. С этого момента мы можем получить ссылку на активность через метод getActivity()
onCreate()
В этом методе можно сделать работу, не связанную с интерфейсом. Например, подготовить адаптер.
onCreateView(LayoutInflater, ViewGroup, Bundle)
Вызывается для создания компонентов внутри фрагмента
onViewCreated()
Вызывается сразу после onCreateView()

onActivityCreated(Bundle)
Вызывается, когда отработает метод активности onCreate(), а значит фрагмент может обратиться к компонентам активности
onStart()
Вызывается, когда фрагмент видим для пользователя и сразу же срабатывает onStart() активности
onResume()
Вызываем после onResume() активности
onPause()
-
onStop()
-
onDestroyView()
Вызывается, когда набор компонентов удаляется из фрагмента
onDestroy()
-
onDetach()
Вызывается, когда фрагмент отвязывается от активности

Какие бывают интенты? Можно ли неявным intent’ом открыть activity

Интенты используются для старта базовых компонентов. Система понимает какой компонент стартовать по атрибутам объекта Intent.

Explicit intent явно содержит информацию о классе компонента. Это может быть объект Class, переданный в конструкторе Intent(context: Context, cls: Class<*>) или методом setClass(context: Context, cls: Class<*>), или объект класса ComponentName, переданный методом setComponent(componentName: ComponentName).
Явные интенты часто используются для старта компонентов внутри приложения, т.к. имена классов известны.
Например startActivity(Intent(context, MyHomeActivity::class.java)) – это старт активити с явным интентом.

Implicit intent не содержит информацию о конкретном компоненте. Система использует косвенные атрибуты, такие как action, type и category для выбора стартуемого компонента. Механизм поиска компонента по атрибутам неявного интента называется Intent Resolution.
Неявные интенты часто используются для старта компонентов других приложений.
Например startActivity(Intent(Intent.ACTION_CALL, Uri.parse("tel:$number"))) – неявный интент, стартующий активити, которая выполнит звонок по заданному номеру.

Более развернуто про явные и неявные intent тут

Что делает транзакция replace(Fragment)

Популярно раскрыто здесь

Какие есть ViewGroup в android?

LinearLayout
TableLayout
AbsoluteLayout
RelativeLayout
FrameLayout
ConstraintLayout

Как передать что-то во фрагмент?

Параметры передаются в конструктор Fragment-а через Bundle, с помощью метода Fragment.setArgument(Bundle). Переданный бандл может быть получен через Fragment.getArguments() в соответствующем методе жизненного цикла фрагмента.

Распространенная ошибка передавать данные через кастомный конструктор. Использовать не-дефолтные конструкторы фрагментов не рекомендуется, потому что фрагмент может быть уничтожен и пересоздан вследствие изменений конфигурации (например при повороте экрана).

Использование пары методов setArguments/getArguments гарантирует, что при пересоздании Bundle будет сериализован/десериализован, и данные восстановятся.

Fragment Result API - описано тут

Создать общую ViewModel подход описан тут или тут

WorkManager - что это и для чего?

WorkManager - позволяет запускать фоновые задачи последовательно или параллельно, передавать в них данные, получать из них результат, отслеживать статус выполнения и запускать только при соблюдении заданных условий.

Развернутей тут

На каком потоке вызовется метод Service.onStartCommand()?

Ответ:

На главном потоке(Main thread, GUI thread)

Что в андройде можно использовать для анимации, через что можно реализовать какую-либо анимацию?

Ответ:

- Через xml
- Property animation
- View animation
- Vector drawable animation
- transition framework
- Рисовать на канвасе View
- Рисовать на канвасе SurfaceView
- OpenGL

Подробнее тут

Что лучше подойдет в качестве коллекции внутри адаптера, для RecyclerView?

[Следует подумать над вопросом]

Есть подходящее рассуждение тут

Нужно учитывать что связанный список хранится в памяти более хаотично, ведь это просто набор объектов хранящих в себе ссылку на разрозненные участки памяти и ссылку/и на другие объекты. Когда ArrayList(под капотом массив) хранит все данные отдельным слитным блоком.

Что делает метод View.invalidate()?

Метод View.invalidate() шедулит перерисовку view. Результат вызова этого метода – асинхронный(или синхронный тут не понять) вызов onDraw().
invalidate() используется, когда нужно перерисовать view без изменения размеров, например когда изменяется цвет бэкграунда.

Что такое RenderThread?

Он называется RenderThread. Он отвечает за преобразование списков отображения в OpenGL команды и отправку их графическому процессору. Как только это происходит, поток UI может переходить к обработке следующего кадра. Время, затраченное потоком UI, чтобы передать все необходимые

ресурсы RenderThread, отражается на данном этапе. Если у нас длинные/тяжелые списки отображения, этот шаг может занять больше времени.

Как отловить любой exception?

https://habr.com/ru/articles/129582/

Как выполнить код на UI потоке?

Обычно такой вопрос задается в контексте работы со View.

Краткий ответ приведен тут

В чем отличие между статическим и динамическим BroadcastReceiver?

Ответом может послужить следующая информация

Какие методы жизненного цикла фрагмента существуют?

a) onInflate()

b) onResume()

c) onReload()

d) onStart()

Можно ли создать приложение без Activity?

a) Да, можно

b) Нет, у приложения всегда должна быть хотя бы одна активити

Является ли метод FragmentManager.commit() синхронным?

a) Да, он синхронный

b) Нет, он асинхронный

c) Такого метода нет

Сколько дочерних элементов может содержать ScrollView?

a) 2

b) 1

c) 3

d) Сколько угодно(V)

С помощью какого компонента лучше выполнять отложенный (по запросу) inflate дочерней View?

a) LazyView

b) AsnycLayoutInflater

c) ViewStub

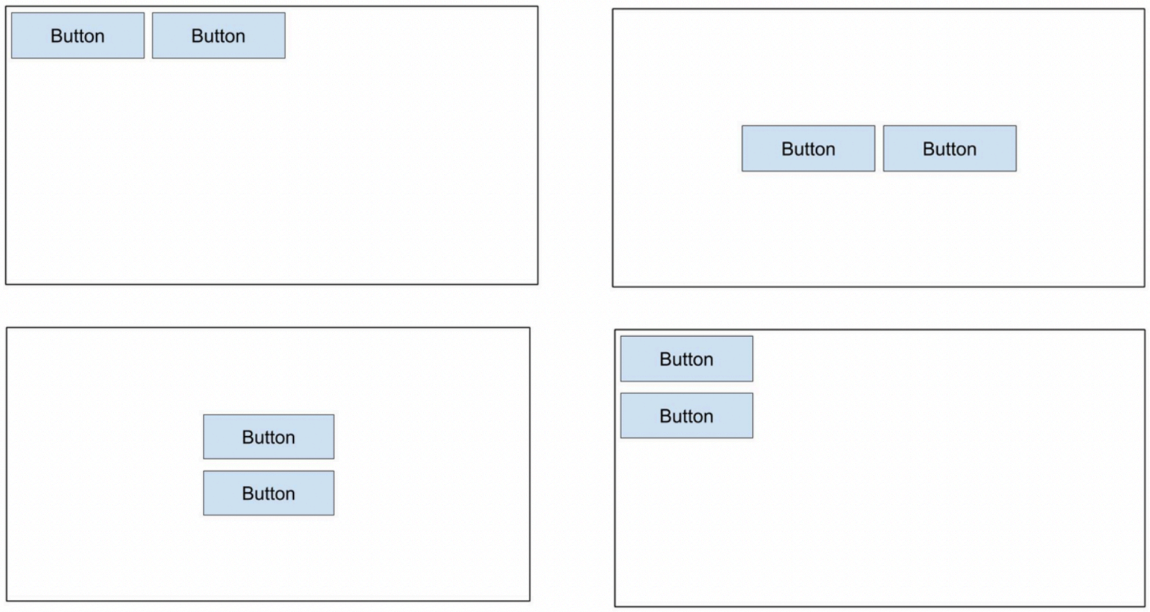
Что отрисуется на экране при inflate этого xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="horizontal">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button" />

</LinearLayout>
```



Правый верхний

Чем отличается serializable от parcelable

Ответ: Parcelable создан специально для андройда, он работают без рефлексии, что было преимущество на ранних версиях андройда. Расплачиваться за это приходилось большим количеством кода. Но со временем это преимущество сошло на нет, а с помощью методов read/writeObject можно сделать более шустрый механизм сериализация

Что такое AAPT?

Android Assets Package Tools - Приложение упаковки ресурсных файлов для Андроид (Android Asset Packaging Tool). Собирает и упаковывает ресурсы приложения Android.

Что такое ART?

Android Runtime - это среда выполнения андройд приложений. Подробнее можно почитать вот тут

Чем отличается activity context от application context?

В контексте Android, application context используется для предоставления доступа к глобальным ресурсам приложения, таким как настройки, ресурсы и службы системы. Он является singleton'ом и существует на протяжении всего жизненного цикла приложения. Это отличает его от activity context, который привязан к жизненному циклу

конкретной активности и может привести к утечкам памяти, если используется неправильно. Избежать утечек памяти с application context можно, используя его в долгоживущих объектах и службах, вместо activity context.

Activity Context создается при создании активити и уничтожается вместе с активити. Контекст – тяжелый объект. Когда говорят об утечке памяти в андроиде, имеют в виду утечку контекста, т.е. ситуацию, когда контекст активити хранится после вызова Activity.onDestroy(). Не передавайте контекст активити в другой объект, если не известно как долго этот объект проживет. Подробнее о memory leak.

Application Context – синглтон. Application Context создается при создании объекта Application и живет, пока жив процесс приложения. По этой причине Application Context можно безопасно инжектить в другие синглтоны в приложении. Не рекомендуется использовать Application Context для старта активити, потому что необходимо создание новой задачи, и для layout inflation, потому что используется дефолтная тема.

Взял тут

Также стоит отметить что различия есть и в порядке наследования, а значит функциях которые способен выполнять activity conttext и application context. К примеру, бекстек или стили

Чуть развернутей можно почитать тут или тут

Что такое ForegroundService?

Foreground Service – это сервис, о котором пользователь осведомлен. Это достигается с помощью отображения нотификации в статус-баре.
Пример foreground сервиса – отображение нотификации при проигрывании музыки в приложении-плеере.

Об этом можно почитать тут и тут

Что такое ContentProvider и для чего он нужен?

Можно почитать тут

Какой метод-callback будет вызван гарантировано, при сворачивании приложения и мы знаем что системы уничтожит процесс?

Метод onPause() будет вызван в том случае, когда система собирается вернуть на передний план другую Activity или когда пользователю необходимо перейти к другим частям системы. Это последний метод, который гарантированно будет вызван до того, как Activity может быть уничтожена системой.

Какие способы сохранения данных при смене конфигурации у Activity тебе известны?

- onSavedInstance() с Bundle [Переживет смерть процесса]
- onRetainNonConfigurationInstance/getLastNonConfigurationInstance [Не переживет смерть процесса]

В чем особенность RecyclerView?

Можно почитать тут

Почему RecyclerView может “лагать”?

Возможно ответ тут

Разница между compoileSDK и targetSDK

Неплохо объяснено тут

Какие бывают BroadcastReceiver?

Статический и динамический

В чем отличие PendingIntent от Intent?

Краткий ответ тут. Более подробно тут

Какие виды долгосрочно хранения есть в Android?

Можно посмотреть тут

SSL Pinning

В ресурсах есть ссылка, можно начать с нее.

Для чего нужен файл Proguard?

Тут есть некоторая базовая информация

Java(12)

Назовите методы класса Object?

Ответ

- hashCode
- equals
- toString
- copy
- notify
- notifyAll
- wait(3 in)
- getClass
- finalize
- clone

Небольшие пояснения по назначению тут

hashCode и equals расскажи как работают

Ответ:

Эти два метода придуманы для использования в Java Collections Framework и связаны общим контрактом, для соблюдения которого переопределять их необходимо вместе. Методы обязательно нужно переопределить чтобы эффективно использовать экземпляры как ключи в HashMap или HashSet. HashMap работает тем эффективнее, чем «лучше» распределение хэшей.

Контракт:
1. Если объекты equals, у них должны быть одинаковые hashCode (не обязательно наоборот – коллизии допустимы!)
2. equals должен быть отношением эквивалентности
3. Ничто не может быть equals(null)
4. equals и hashCode должны возвращать одни и те же значения для одного и того же объекта при каждом последующем вызове, даже если состояние объекта изменилось. Это делает реализацию для изменяемых (mutable) объектов крайне сложной.

По умолчанию equals сравнивает на ==. С умолчательным hashCode дела обстоят интереснее: он зависит от реализации JVM, и может быть неожиданным. Например в OpenJDK 7 это случайное число.

Что делают методы wait, notify, notifyAll?

Часто этот вопрос формулируется как задача Producer-consumer. Эту задачу и практические задачи на многопоточность вообще при возможности лучше реализовывать на высокоуровневых примитивах синхронизации. Другой подход – воспользоваться также низкоуровневой, но оптимистической блокировкой на compareAndSet. Но обычно использование notify/wait (пессимистическая блокировка) – условие этого задания, то есть требуется реализовать уже существую BlockingQueue.

Эти методы вместе с synchronized – самый низкий уровень пессимистических блокировок в Java, использующийся внутри реализации примитивов синхронизации. Еще с Java 5 в непосредственном использовании этих методов нет необходимости, но теоретические знания всё еще часто спрашивают на интервью.

Чтобы вызывать эти методы у объекта, необходимо чтобы был захвачен его монитор (т.е. нужно быть внутри synchronized-блока на этом объекте). В противном случае будет выброшено IllegalMonitorStateException. Так что для полного ответа нужно понимать, как работает monitor lock (блок synchronized).

Вызов wait тормозит текущий поток на ожидание на этом объекте и отпускает его монитор. Исполнение продолжится, когда другой поток вызовет notify и отпустит блокировку монитора. Если на объекте ожидают несколько потоков, notify разбудит один случайный, notifyAll - все сразу.

В теории, ожидание wait может быть прервано без вызова notify, по желанию JVM (spurious wakeup). На практике это бывает крайне редко, но нужно страховаться и после вызова wait добавлять дополнительную проверку условия завершения ожидания.

Еще два нештатных случая завершения wait – прерывание потока извне и таймаут ожидания. В случае прерывания выбрасывается InterruptedException. Для таймаута нужно указать время ожидания параметрами метода wait. Значение 0 проигнорируется.

Различные проблемы реализации блокировок рассмотрены в Java Concurrency in Practice 14.1.3, 14.2. Для желающих разобраться, как блокировки работают в кишках JVM, написана статья на хабре.

Взял тут

Здесь объяснено чуть понятнее, но тред называется нитью

Какие root’ы у GC могут быть?

Thread’s и статичные элементы(из-зп ClassLoader) и многие другие

Некоторые руты перечисленны тут
Локальная переменная метода тоже корень.

Если два объекта ссылаются друг на друга - сможет ли GC удалить их?

Да, мы определяем это по достижимости объектов из корней, а не по количеству ссылок.

Виды ссылок в Java

Strong
Weak
Soft
Phantom

Разница между Weak и Soft ссылками?

Отношение со стороны Garbage Collector к ним в общем одинаковое, но для Soft-ссылок отдельно проговаривается, что их буду стараться удалять только когда память будет подходить к концу.

В чем разница между @Volatile и Atomic-типами? Что предпочтительней для задачи по инкременту переменных?

Модификатор volatile накладывает некоторые дополнительные условия на чтение/запись переменной. Важно понять две вещи о volatile переменных:

1. Операции чтения/записи volatile переменной являются атомарными.
2. Результат операции записи значения в volatile переменную одним потоком, становится виден всем другим потокам, которые используют эту переменную для чтения из нее значения.

Пакет java.util.concurrent.atomic содержит девять классов для выполнения атомарных операций. Операция называется атомарной, если её можно безопасно выполнять при параллельных вычислениях в нескольких потоках, не используя при этом ни блокировок, ни синхронизацию synchronized. Прежде, чем перейти к рассмотрению атомарных классов, рассмотрим выполнение наипростейших операций инкремента и декремента целочисленных значений. Подробнее тут.

Для инкремента лучше всего подойдут атомарные классы, так как они обеспечат безопасность этой операции в многопоточном приложении и будут работать

производительней за счет отстутсвия блокировок, посредством compare and swap - стратегии.

Какие есть средства синхронизации в многопоточном приложении помимо synchronized?

RetaintLock
Semaphore
CycleBarrier
CountDownLatch

В чем разница между notify() и notifyAll()? И может ли случиться так что зайдут два потока и они заблокируются

В отличие от notify(), метод notifyAll() уведомляет все потоки, ожидающие доступа к определенному монитору. Однако это не означает, что все потоки сразу получат доступ к ресурсу. Только один из них сможет захватить монитор, а остальные продолжат ожидание. Выбор потока, который получит доступ к монитору, осуществляется планировщиком потоков системы.

Чуть подробнее тут

Про возможность deadlock написано тут

Какие виды Executors существуют?

Интерфейсы

- Executors
- ExecutorService
- ScheduledExecutorService

AbstractExecutorService — абстрактный класс для построения ExecutorService'а. Внутри есть имплементация методов submit, invokeAll, invokeAny. От этого класса наследуются **ThreadPoolExecutor**, **ScheduledThreadPoolExecutor** и **ForkJoinPool**.

Подробнее тут

Что означает модификатор static в Java?

Подробно можно прочитать тут

В целом, поле или метод помеченный static принадлежат классу, а не его экземплярам. То есть в памяти будут хранится в единственном экземпляре.

Общая теория(18)

Расскажите про паттерн Singleton

Одиночка — порождающий шаблон проектирования, гарантирующий, что в однопоточном приложении будет единственный экземпляр некоторого класса, и предоставляющий глобальную точку доступа к этому экземпляру.

Назовите 4 принципа ООП

Ответ:

1. Абстракция — отделение концепции от ее экземпляра;
2. Полиморфизм — реализация задач одной и той же идеи разными способами;
3. Наследование — способность объекта или класса базироваться на другом объекте или классе. Это главный механизм для повторного использования кода. Наследственное отношение классов четко определяет их иерархию;
4. Инкапсуляция — размещение одного объекта или класса внутри другого для разграничения доступа к ним.

Назовите принципы SOLID?

Ответ:
S - Single Responsibility Principle - принцип единственной ответственности. Каждый класс должен иметь только одну зону ответственности.

O - Open closed Principle - принцип открытости-закрытости. Классы должны быть открыты для расширения, но закрыты для изменения.

L - Liskov substitution Principle - принцип подстановки Барбары Лисков. Должна быть возможность вместо базового (родительского) типа (класса) подставить любой его подтип (класс-наследник), при этом работа программы не должна измениться.

I - Interface Segregation Principle - принцип разделения интерфейсов. Данный принцип означает, что не нужно заставлять клиента (класс) реализовывать интерфейс, который не имеет к нему отношения.

D - Dependency Inversion Principle - принцип инверсии зависимостей. Модули верхнего уровня не должны зависеть от модулей нижнего уровня. И те, и другие должны зависеть от абстракции. Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.

Расскажите про принцип YAGNI

Ответ:

You Aren’t Gonna Need It / Вам это не понадобится

Этот принцип прост и очевиден, но ему далеко не все следуют. Если пишете код, то будьте уверены, что он вам понадобится. Не пишите код, если думаете, что он пригодится позже.

Этот принцип применим при рефакторинге. Если вы занимаетесь рефакторингом метода, класса или файла, не бойтесь удалять лишние методы. Даже если раньше они были полезны – теперь они не нужны.

Может наступить день, когда они снова понадобятся – тогда вы сможете воспользоваться git-репозиторием, чтобы воскресить их из мертвых.

Расскажите принцип DRY

Ответ:
Don’t Repeat Yourself / Не повторяйтесь
Дублирование кода – пустая трата времени и ресурсов. Вам придется поддерживать одну и ту же логику и тестировать код сразу в двух местах, причем если вы измените код в одном месте, его нужно будет изменить и в другом.

В большинстве случаев дублирование кода происходит из-за незнания системы. Прежде чем что-либо писать, проявите прагматизм: осмотритесь. Возможно, эта функция где-то реализована. Возможно, эта бизнес-логика существует в другом месте. Повторное использование кода – всегда разумное решение.

Сложность чтения и добавления в ArrayList, HashMap

ArrayList в среднем чтени = o(1), вставка = O(n). HashMap(Хеш-таблица) в среднем чтение = 1, вставка = 1

Подробней тут

Расскажи про паттерн Адаптер и его применение в Android. Основные методы адаптера.

Адаптер (англ. *Adapter*) — структурный шаблон проектирования, предназначенный для организации использования функций объекта, недоступного для модификации, через специально созданный интерфейс. Другими словами — это структурный паттерн проектирования, который позволяет объектам с несовместимыми интерфейсами работать вместе.

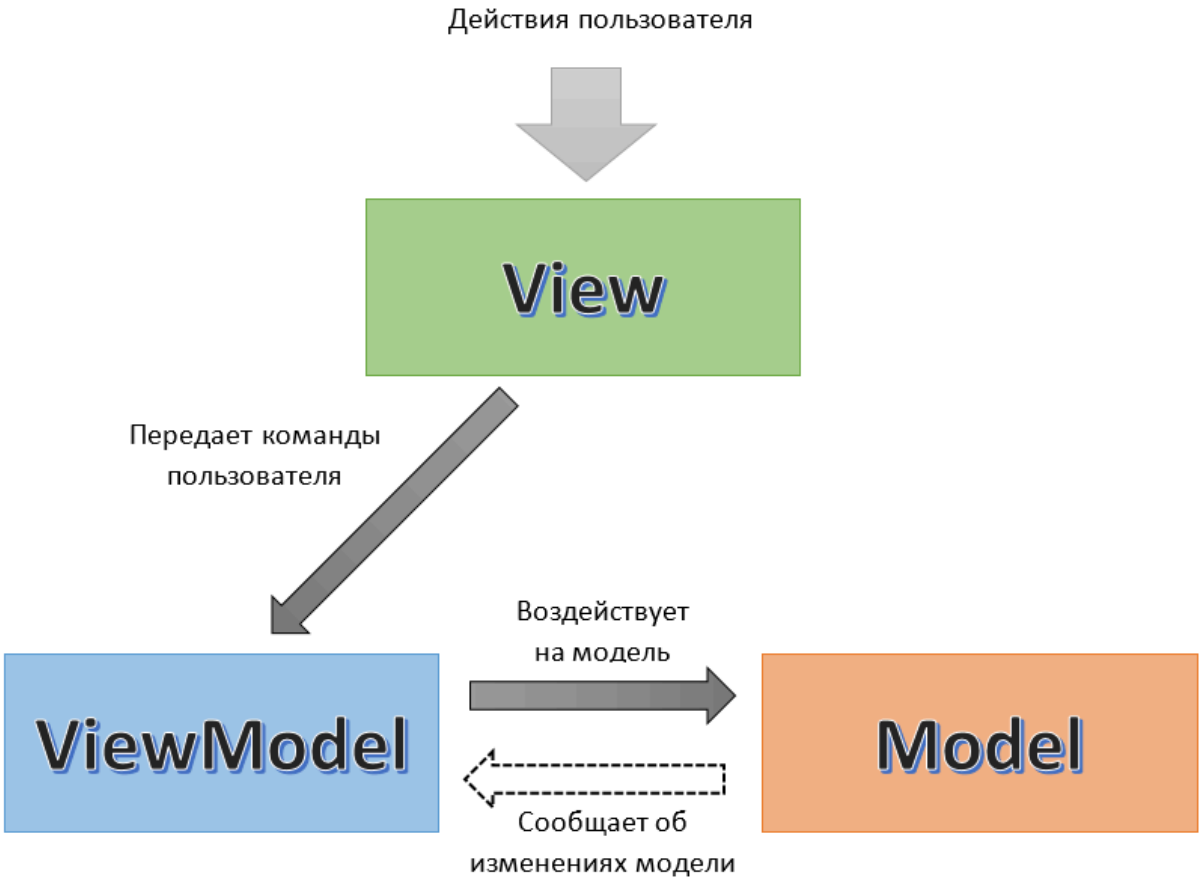
Основные методы:

- onCreateViewHolder
- getItemCount
- onBindViewHolder

В чем разница между interface и abstract class

- Переменные в интерфейсе по умолчанию final
- В абстрактном классе могут использоваться модификаторы доступа, в интерфейсе по умолчанию всё public
- Можно наследовать несколько интерфейсов, но только один класс

Расскажите про MVVM



Какое самое большое число поместиться в 32битный int(целочисленный)?

Ответ:

Уточнить, беззнаковое или со знаком

Primitive Type	Size	Minimum Value	Maximum Value	Wrapper Type
char	16-bit	Unicode 0	Unicode 2^{16}-1	Character
byte	8-bit	-128	+127	Byte
short	16-bit	-2^{15} *(-32,768)*	$+2^{15}$-1 *(32,767)*	Short
int	32-bit	-2^{31} *(-2,147,483,648)*	$+2^{31}$-1 *(2,147,483,647)*	Integer
long	64-bit	-2^{63} *(-9,223,372,036,854,775,808)*	$+2^{63}$-1 *(9,223,372,036,854,775,807)*	Long
float	32-bit	32-bit IEEE 754 floating-point numbers		Float
double	64-bit	64-bit IEEE 754 floating-point numbers		Double

В чем отличие процесса от потока?

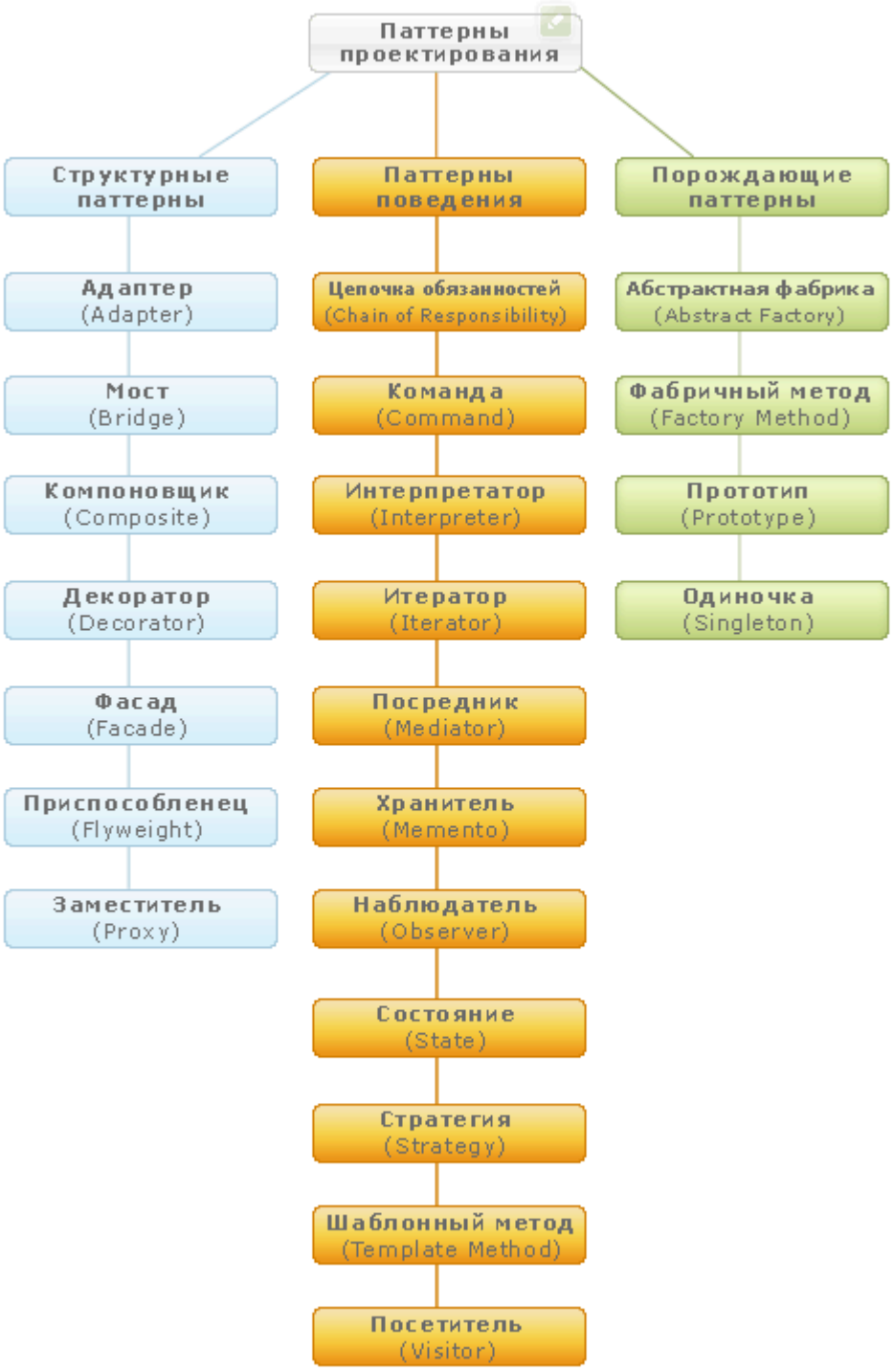
Процесс(по-умолчанию один на приложение) это сущность которая включает в себя потоки. потоки это часть процесса. В одном процессе может быть(и есть) множество потоков. Мы можем создать поток программно, а вот запустить новый процесс мы можем только опосредованно.

К какому виду паттерна проектирования относится …(рандом)(Поведенческие, Порождающие, Структурные)

Порождающие:
Singleton (Одиночка)
Factory (Фабрика)
Abstract Factory (Абстрактная фабрика)
Builder (Строитель)

Структурные:
Adapter (Адаптер)
Facade (Фасад)
Bridge (Мост)
Decorator (Декоратор)

Поведенческие:
Observer (Наблюдатель)
Strategy (Стратегия)
Command (Команда)



Расскажите про устройство памяти в Java/Android программах? для чего нужен Стек, для чего нужна куча?

Можно почитать об этом тут

Если мы в рамках своего приложения запустили другой процесс(например launchMode = SingleInstace) - будит ли куча общей для обоих процессов?

Нет

Можно ли вызывать Garbage Collector из приложения?

Нет, нельзя. Даже если обратиться к методу System.gc(). Это передаст лишь пожелание для GC, но не гарантирует его мгновенный вызов.

Есть ли какой-то метод который сигнализирует что выполнился GC?

Application.onTrimMemmory()

Что такое утечка памяти?

Уте́чка па́мяти — процесс неконтролируемого уменьшения объёма свободной оперативной или виртуальной памяти компьютера, связанный с ошибками в работающих программах, вовремя не освобождающих ненужные участки памяти, или с ошибками системных служб контроля памяти.

Как обнаружить утечки памяти?

LeakCanary
Можно средствами профилирования студии. Подробней об этом тут

Задачи(без решений)

Задача 1

Есть класс

```
data class A(var a: String) {
var b: String? = null
}
fun main() {
val a = A(“Ivan”)
```

```
val b = A(“Ivan”)

print( “${a == b}”)
}
```

Что даст вывод или каким будет результат?

Ответ:
Вывод будет true

Задача 2.

// Условия задачи: Необходимо реализовать поиск через EditText и ApiService внутри Activity.
// Результаты поиска нужно вывести в Log.
// Без какой-либо архитектуры и доп.классов (ViewModel, Repository, UseCase).
// Для реализации решения можно использовать любую технологию (RxJava, Coroutines и другое).

```
interface ApiService {
    @GET("products")
    fun getProducts(@Query("query") query: String): List<Product>
}

class MainActivity : AppCompatActivity() {

    @Inject
    lateinit var apiService: ApiService

    val subjectSearch = BehaviorSubject<String>.create()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val binding = ActivityLoginBinding.inflate(layoutInflater)
        setContentView(binding.root)

        val editText = binding.editText
    }
}
```

Задача 3

Junior разработчик реализовал логику нового продукта и ушел в отпуск. Его код пришел на ревью, нужно сделать его чище и исправить явные баги, если таковые есть.

Исправления в коде и улучшения нужно аргументировать, чтобы джуниор чему-то научился.
По ходу редактирования, интервьюер будет задавать теоретические вопросы.

```
package ru.tinkoff.interview.formula

import android.content.Context
import retrofit2.http.GET
import java.util.LinkedList

object {
     const val TRaceVolume = 2
}
const val TRaceCylinders = 8

class Formula1DataSource private constructor(
    val context: Context,
    val service: ApiService
) {

    fun loadAllData(): Single<LinkedList<Engine>> {
        return Single.fromCallable {
            val engines: LinkedList<Engine> = service.loadAvailableEngines(
            val result = LinkedList<Engine>()

            for (i in 0..engines.size) {
                if (engines[i].volume >= TRaceVolume)
                    result.add(engines[i])
                if (engines[i].cylinders >= TRaceCylinders)
                    result.add(engines[i])
            }

            if (result.size == 0)
                throw Exception("No engines available")
            }

         return result
        }
    }

    @Synchronized
    fun getLastCar(): FormulaCar? = car

    @Synchronized
    fun saveCar(engine: Engine, name: String) {
```

```
        if (car.get() == null) {
            car = FormulaCar(engine, formattedName, "Tinkoff") // magic
        } else {
            car.engine = engine
            car.name = formattedName
        }
        service.saveCar(car)
    }

    companion object {

        @Volatile
        var car: AtomicReference<FormulaCar?> = null

        fun getInstance(context: Context, service: ApiService)
Formula1DataSource(context.applicationContext, service)
    }
}

// interfaces just for info
data class FormulaCar(var engine: Engine, var name: String, val sponsor: String)
data class Engine(val engineType: String, val cylinders: Int, val volume: Double)

interface ApiService {

    @GET
    fun loadAvailableEngines(): ApiEnvelope<LinkedList<Engine>>

    @POST
    fun saveCar(car: FormulaCar): ApiEnvelope<Unit>
}

interface ApiEnvelope<T> {

    val payload: T
    val responseCode: Int
}
```

Задача 4.

Необходимо рассчитать ширину всех элементов поступающих в метод.
В метод поступает общая ширина ViewGroup - parentWidth
В метод поступает список значений ширины вьюх - mesureSpec
Если пункт >= 0, то ширина View фиксированная

Если пункт < 0, то ширина указана в долях от оставшегося пространства, не занятого View с фиксированной шириной
Пример: measureWidth(100, listOf(50, -3, -2))
Результат: listOf(50, 30, 20)

```
fun measure(parentWidth: Int, childSpecs: List<Int>): List<Int> {
    // Тут надо реализовать задачу(внешние методы можно создавать)
}
```

Задача 5.

Каков порядок инициализации переменных и выполнения блоков кода при создании экземпляра ExampleClass? Для удобства можно ответить цифрами, указанными в комментариях.

```
class ExampleClass(val first: String) { // 1
    val second = "Second" // 2

    init {
        println("Init block") // 3
    }

    companion object {
        val third = "Third" // 4
    }
}
```

Задача 6

В каком порядке отработают функции?

```
val state1 = MutableStateFlow(1)
val state2 = MutableStateFlow(2)
val state3 = MutableStateFlow(3)

launch {
    state1.coolect {
        //1
    }
    state2.coolect {
        //2
    }
    state3.coolect {
      //3
```

```
    }
}
```

Задача 7

Будет ли тут утечка памяти?

```
class MainActivity {
val repository = Repository()

fun onCreate() {
    val list = repository.findItems()
}
}

object Repository {

    fun findItems(): List<Item>
}
```

Задача 8

Будет ли тут утечка памяти?

```
class MainActivity {

lateinit var presenter: Presenter

override fun onCreate() {
presenter = Presenter(this)
}
}

class Presenter(activity: Activity)
```

Задача 9

Мы хотим складывать очень большие числа, которые превышают емкость базовых типов, поэтому мы храним их в виде массива неотрицательных чисел. Нужно написать функцию которая примет на вход два таких массива, вычислит сумму чисел, представленных массивами и вернет результат в виде такого же массива.
Пример 1
Ввод
arr1 = [1,2,3]// число 123
arr2 = [4,5,6]// число 456
Вывод:

res = [5,7,9]//число 579, допустим ответ с первым незначимым 0 – [0,5,7,9]

Пример 2
Ввод
arr1 = [5,4,4]// число 544
arr2 = [4,5,6]// число 456
Вывод:
res = [1,0,0,0]//число 1000

Пример 3
Ввод
arr1 = [9,9,9,9]// число 9999
arr2 = [1]// число 1
Вывод:
res = [1,0,0,0,0]//число 10000

Задача 10

Какие проблемы при работе с многопоточностью наблюдаются в этом коде?

```
class Holder {
    private val  list: MutableList<String> = LinkedList<String>

   @Syncronized
   fun getState(): List<String> {
         return list.toList()
   }

   @Syncronized
   fun log() {
      for(item in list) {
          println(item)
      }
   }

   fun add(item: String) {
         syncronized(list) {
               list.add(item)
         }
   }
}
```

Задача 11

Напишите свою реализацию функции let

Подсказка/ответ:

```
@kotlin.internal.InlineOnly
```

```
public inline fun <T, R> T.let(block: (T) -> R): R {
    contract {
        callsInPlace(block, InvocationKind.EXACTLY_ONCE)
    }
    return block(this)
}
```

Задача 12

Напишите программу которая будет принимать на вход произвольную строку и подсчитывать количество палиндромов в ней.

```
fun main() {
    val text: String = getTextInput()
    val result = text.filter {it: Char -> it.isLetterOrDigit() || it. == ' '}

    val words: List<String> = result.trim().toLowerCase().split(" ")
    var polindromCount = 0

    words.forEach { word ->
        if(isPalindrome(word)) {
            ++polindromCount
        }
    }


    println(polindromCount )
}

fun isPalindrome(input: String): Boolean {
    return input == input.asReversed()
}
```

Тестовые данные: Каза шел по дороге; Казак, привет!

Разные ресурсы

- Start android
- solid-примеры
- Разные принципы
- 25 основных вопросов(перевод англоязычной статьи, не очень качественный, картинок нет)
- Жизненный цикл фрагментов
- Есть много классических вопросов по Java и Android, но нету Kotlin
- О Kotlin-равенстве
- Про ООП тут и тут
- Про О-нотацию или оценку операций со структурами данных
- Kotlin playground

- Про object
- Про явный и неявный intent
- Неплохой набор вопросов-ответов
- Коротко о том во что разворачиваются kotlin-”сахар”
- Очень малополезный ресурс, но можно ознакомиться
- Делегирование
- Неплохой набор вопросов по Kotlin
- Неплохой ресурс
- Внутреннее устройство ArrayList Java
- Внутреннее устройство LinkedList
- Внутреннее устройство HashMap
- Полезно про stable/immutable
- Про RxJavaSchedulers
- Алгоритмические задачки Яндекс
- Память в Java/Android приложениях
- Реализации различных сортировок на котлин с комментариями. тут
- Про низкоуровневые инструменты синхронизации в многопоточности тут
- Подробно про AtomicReferences тут
- Здесь много примеров по атомику и синхронным коллекциям. ссылка
- Про синхронизацию в корутинах тут
- Неплохое короткое видео о синхронизации в корутинах. ссылка
- Джава песочница
- Здесь инфа про потокобезопасность у lazy-делегата. Ссылка
- Довольно любопытно о подкапотном устройстве permission. тут
- О том как закрыт другую активити пермиссион тут
- Про активити резалт апи часть 1, часть 2. Текстовая версия про активити, фрагменты
- Популярно раскрыто что такое ssl pinning тут. Для продвинутых ребят можно почитать тут и тут
- Avito code playground
- Вопросы для Java-программистов