// JavaJub

Собес в Яндекс Путешествия

Яндекс Вертикали

Java Middle

Вопросы, задачи и подготовка
к live-coding и техническому интервью

Стек

Java · Kotlin · Spring Boot · PostgreSQL · YDB · gRPC · Logbroker · Kubernetes

// содержание

Содержание

01 Яндекс Путешествия и формат собеса

Яндекс Путешествия — часть бизнес-юнита Яндекс Вертикали (вместе с Авто.ру, Недвижимостью, Арендой). Это единственное крупное JVM-направление Яндекса наряду с Маркетом — можно прийти с обычным Java-бэкграундом, не переучиваясь на C++, Go или Scala.

Команда ~40+ бэкенд-разработчиков. Продукты: отели, авиа, ж/д, автобусы, B2B-командировки, экстранет для отельеров, биллинговая платформа. Нагрузка — десятки тысяч RPS, миллионы DAU.

Как устроен процесс

Яндекс нанимает через общий поток «бэкенд в Путешествия». Уровень (Middle / Middle+ / Senior) определяется по итогам секций, а не указывается в вакансии. Секции оцениваются независимо — интервьюеры не обсуждают результаты до финальной калибровки.

Этап	Формат	Длительность
HR-скрининг	Zoom, мотивация, ЗП	30–60 мин
Техническое предварительное	Yandex.Code (без компиляции)	1 час
Advanced Code (с 2025)	IDE кандидата + автотесты + интернет	1 час
Алгоритмическая секция	Yandex.Code, 2–3 задачи	1 час
Java Core + многопоточность	Yandex.Code, теория + код	1–1.5 часа
System Design	Miro + Zoom	1 час (опц. для Middle)
Секция «про опыт» (STAR)	Разбор проектов	1 час
Финалы с командами	2–5 команд по 1 часу	2–5 часов

ФИШКА · Ключевая особенность

Можно провалить 1–2 секции и всё равно получить оффер — секции оцениваются независимо. В сомнительных случаях просто назначают дополнительную секцию. После отказа можно перезайти через 6–12 месяцев.

ВНИМАНИЕ · Про алгоритмы

Алгоритмическая секция — главный фильтр. Уровень сравним с Google. НО: Яндекс официально заявляет, что НЕ дают динамическое программирование, Дейкстру, KMP. Фокус — хэш-таблицы, два указателя, sliding window, BFS/DFS.

02 Стек по вакансии

Стек определён по вакансии «Разработчик на Java в Путешествия» (yandex.ru/jobs) и карточке на career.habr.com. Путешествия отличаются от остальных Вертикалей: Авто.ру пишет на Scala + ZIO, а Путешествия — на Java + Kotlin + Spring.

Основной стек

- Java + Kotlin — основные языки (Go и C++ для отдельных компонентов)
- Spring / Spring Boot — основной фреймворк
- Hibernate + JOOQ — ORM и типизированный SQL
- PostgreSQL — OLTP-база
- YDB — распределённая база Яндекса (высокая нагрузка, отказоустойчивость)
- gRPC — внутренний RPC, HTTP/REST — наружу
- Logbroker — внутренняя шина данных (поверх YDB Topics, API-совместима с Kafka)
- YT / YTsaurus — аналитика, MapReduce, холодное хранилище
- Arcadia — монорепозиторий Яндекса
- Nanny / Yandex Deploy — деплой и оркестрация

Что важно знать

- Требование: 5+ лет коммерческой разработки на Java или Kotlin
- Внутренние инструменты (YDB, Logbroker, Arcadia) осваиваются на месте — не нужно знать до собеса
- Новичок проводит двухнедельный буткемп в каждой команде перед выбором
- Опционально: MultiTrack-буткемп — 8 недель работы в 3 командах с полной зарплатой

> **СОВЕТ · Для Java-разработчика**
> Путешествия — наиболее «дружественное» подразделение Яндекса для классического Java-разработчика: привычный стек Spring/Hibernate/PostgreSQL + Kotlin. Внутренние технологии (YDB, Logbroker, Arcadia) изучаются уже на месте.

03 Java Core — что точно спросят

По отзывам кандидатов (Medium, Habr 968968, Habr 854956): equals/hashCode + HashMap — абсолютные чемпионы. Далее — Stream API, generics, String Pool, ссылочные типы.

Базовые вопросы

- JDK, JRE, JVM?

  → *JVM — виртуальная машина (исполняет байт-код). JRE = JVM + стандартные библиотеки. JDK = JRE + инструменты (javac, jdb, jar). С Java 9 JRE отдельно не поставляется.*

- Области памяти JVM.

  → *Heap (Eden/Survivor/Old — объекты), Stack (фреймы методов), Metaspace (метаданные классов, раньше PermGen), PC Register, Native Method Stack, CodeCache (JIT-скомпилированный код).*

- Контракт equals/hashCode.

  → *Если a.equals(b), то hashCode одинаков. Обратное необязательно. Рефлексивность, симметричность, транзитивность. equals(null) → false. Переопределяешь один — переопределяй оба. Нарушение ломает HashMap/HashSet.*

- Что сломается, если hashCode() — константа?

  → *Все объекты в одном bucket. Java 7: список O(n). Java 8+: при 8 коллизиях И capacity ≥ 64 — red-black tree O(log n). Деградация по сравнению с O(1).*

- String Pool и immutability.

  → *Pool в heap хранит уникальные литералы. new String("hi") — новый объект вне пула. intern() добавляет в пул. Immutability: безопасность, потокобезопасность, кэширование hashCode, переиспользование в пуле.*

- Generics: type erasure и PECS.

  → *Дженерики стираются компилятором (type erasure) — в рантайме нет информации о типовом параметре. PECS: Producer Extends, Consumer Super. List<? extends Number> — читать, List<? super Integer> — писать.*

- WeakReference / SoftReference / PhantomReference.

  → *Weak — GC собирает при первой же сборке. Soft — GC собирает при нехватке памяти (для кэшей). Phantom — для постфинализационной очистки ресурсов. WeakHashMap — ключи через WeakReference.*

- Функциональные интерфейсы и Stream API.

  → *Один абстрактный метод (SAM). Function, Predicate, Consumer, Supplier, Comparator. Стрим: промежуточные (ленивые) → терминальные (запускают конвейер). Обработка вертикальная, не горизонтальная. Стрим одноразовый.*

- Optional — когда?

  → *Для возвращаемых значений. НЕ для полей, параметров, коллекций. Антипаттерны: get() без isPresent(), Optional в полях, Optional<List>. orElse vs orElseGet (ленивый).*

- final: класс, метод, поле.

  → *Класс — нельзя наследовать. Метод — нельзя переопределить. Поле — нельзя переприсвоить (но мутабельное содержимое можно менять). effectively final — для лямбд.*

- Абстрактный класс vs интерфейс.

  → *Абстрактный: состояние, конструкторы, один наследник. Интерфейс: контракт, множественная реализация. Java 8 — default/static. Java 9 — private методы.*

▸ Задачи «Что выведет?»

Тест 1. Integer cache

```
Integer i1 = 127, i2 = 127;
System.out.println(i1 == i2); // ?

Integer i3 = 128, i4 = 128;
System.out.println(i3 == i4); // ?
```

Ответ: true, false. IntegerCache: -128..127. Для 127 — один объект, ссылки совпадают. Для 128 — два разных. Мораль: для объектов — equals(), никогда ==.

Тест 2. Stream — ленивость

```
List.of(1,2,3,4,5).stream()
  .map(x -> { System.out.print(x+" "); return x; })
  .filter(x -> x > 2)
  .map(x -> { System.out.print(x+" "); return x; })
  .toList();
```

Ответ: 1 2 3 3 4 4 5 5. Стрим обрабатывает вертикально: каждый элемент проходит всю цепочку. 1 и 2 не проходят filter → печатаются раз. 3,4,5 проходят → дважды.

Тест 3. Передача по значению

```
Integer i = Integer.valueOf(1);
inc(i);
System.out.println(i); // ?

static void inc(Integer i) { i++; }
```

Ответ: 1. Java передаёт ссылки по значению. i++ создаёт новый Integer(2) и присваивает локальной переменной. Оригинал не меняется.

> **СОВЕТ · Лайфхак**
> На вопрос про equals/hashCode приведи практический пример: «положить объект в HashSet, изменить поле в hashCode — объект потеряется, contains() вернёт false». Это показывает понимание, а не заученность.

04 Коллекции

HashMap — чемпион. По опыту кандидатов в Яндекс: «обсасывают мапу со всех сторон — устройство, коллизии, treeify, resize, ключи-массивы, что будет если положить и не найти».

- Как устроен HashMap?

  → *Массив Node<K,V>[]. Размер — степень двойки (default 16). Индекс: (n-1) & hash(key), hash — XOR верхних 16 бит с нижними. Коллизии — связный список. Java 8+: при TREEIFY_THRESHOLD=8 И capacity ≥ 64 → red-black tree. При 6 → обратно (UNTREEIFY).*

- load factor и resize.

  → *Порог 0.75. При size >= capacity * loadFactor — resize: массив вдвое + перехеширование ВСЕХ элементов. Совет: initialCapacity = expectedSize / 0.75 + 1.*

- HashMap vs ConcurrentHashMap.

  → *HashMap: не потокобезопасен, допускает null-ключ. ConcurrentHashMap: Java 7 — Segment-блокировки, Java 8+ — CAS + synchronized на головах бакетов. null запрещён. computeIfAbsent — атомарная операция.*

- ArrayList vs LinkedList.

  → *ArrayList: O(1) доступ, O(n) вставка в середину, дружелюбен к CPU-кэшу. LinkedList: O(1) вставка/удаление в начало, O(n) доступ. На практике ArrayList почти всегда лучше.*

- TreeMap vs LinkedHashMap.

  → *TreeMap: red-black tree, O(log n), ключи отсортированы. LinkedHashMap: HashMap + двусвязный список, порядок вставки. accessOrder=true → LRU-кэш.*

- Fail-fast итератор.

  → *ConcurrentModificationException при изменении коллекции не через итератор. modCount-счётчик. Не гарантирован в многопоточке. Альтернатива: CopyOnWriteArrayList (fail-safe).*

- Можно ли byte[] как ключ HashMap?

  → *Технически можно, но hashCode у массива — по адресу (identityHashCode), а equals — по ссылке. Два одинаковых по содержимому массива дадут разные хэши. Нужно обернуть в ByteBuffer или свой класс с переопределёнными equals/hashCode.*

05 Многопоточность и JMM

В Яндексе многопоточность спрашивают глубоко — volatile vs synchronized давали на каждом собесе. JMM, happens-before, CAS, пулы потоков — обязательно.

- synchronized.
  → *Захват монитора. Instance-метод → this. Static → Class. Блок → указанный объект. Reentrant (счётчик). Гарантирует: mutual exclusion + happens-before (видимость).*
- volatile.
  → *Видимость (запрет кэширования в регистрах/L1) + запрет reordering. НЕ гарантирует атомарность i++ (read-increment-write). Для атомарных — AtomicInteger/LongAdder.*
- happens-before.
  → *unlock → lock того же монитора. volatile write → read. Thread.start() → первая инструкция потока. Последняя инструкция → join(). final-поля видны после конструктора.*
- ConcurrentHashMap: Java 7 vs Java 8.
  → *Java 7: массив Segment (ReentrantLock), каждый сегмент — свой HashMap. Java 8: убрали Segment, CAS + synchronized на головах бакетов (лучше параллелизм). Treeify при 8 коллизиях.*
- ThreadPoolExecutor — параметры.
  → *corePoolSize, maxPoolSize, keepAliveTime, workQueue (LinkedBlockingQueue/ArrayBlockingQueue/SynchronousQueue), threadFactory, rejectedExecutionHandler (AbortPolicy/CallerRunsPolicy/DiscardPolicy/DiscardOldestPolicy).*
- CountDownLatch vs CyclicBarrier vs Semaphore.
  → *CountDownLatch: одноразовый счётчик, потоки ждут обнуления. CyclicBarrier: переиспользуемый, все потоки ждут друг друга. Semaphore: ограничение числа одновременных потоков (пул ресурсов).*
- Deadlock — как возникает?
  → *Два+ потока ждут друг друга. Условия Коффмана. Избежать: упорядочить захват мониторов, tryLock с таймаутом. Обнаружение: jstack, VisualVM, Thread.getState().*
- Реализуй ограниченную BlockingQueue.
  → *synchronized + wait/notify. while(!condition) wait() — не if! (spurious wakeups). put() ждёт если полная, take() ждёт если пустая. notifyAll() после каждой операции.*

ЛОВУШКА · volatile counter++

volatile long counter; counter++ — НЕВЕРНО. Это три операции (read-increment-write), volatile не обеспечивает атомарность. Три корректных варианта: synchronized, AtomicLong.incrementAndGet(), LongAdder.increment().

06 Spring и Spring Boot

Spring — основной фреймворк Путешествий. Спрашивают глубоко: прокси, жизненный цикл бинов, автоконфигурация, AOP.

- IoC и DI.

  → *IoC — контейнер управляет жизненным циклом. DI — зависимости внедряются извне. Три способа: конструктор (лучший), сеттер, поле (@Autowired). Constructor injection: поля final, явные зависимости, легко тестировать.*

- Жизненный цикл бина.

  → *BeanDefinition → инстанцирование → DI → Aware-интерфейсы → BPP.postProcessBefore → @PostConstruct → InitializingBean → init-method → BPP.postProcessAfter → ГОТОВ → @PreDestroy → DisposableBean → destroy-method. BPP — точка создания прокси.*

- Scope бинов.

  → *singleton (default), prototype, request, session, application, websocket. Инжект prototype в singleton: через Provider<T>, ObjectFactory<T> или @Lookup. Иначе prototype создастся один раз.*

- @Transactional — прокси.

  → *Spring создаёт прокси (JDK/CGLIB). Прокси открывает транзакцию, вызывает метод, commit/rollback. self-call минует прокси → @Transactional/@Cacheable/@Async не работают. Решение: вынести в другой бин.*

- Автоконфигурация Spring Boot.

  → *@EnableAutoConfiguration + @Conditional. Если DataSource в classpath — автоматически настроит JPA. Список: META-INF/spring/...AutoConfiguration.imports (Spring Boot 3+, раньше spring.factories).*

- @SpringBootApplication.

  → *@Configuration + @EnableAutoConfiguration + @ComponentScan.*

- AOP: JDK proxy vs CGLIB.

  → *JDK dynamic proxy — если бин реализует интерфейс (через Proxy.newProxyInstance). CGLIB — наследование от класса (final-классы нельзя проксировать). Spring Boot 3 по умолчанию использует CGLIB.*

- Обработка исключений.

  → *@RestControllerAdvice + @ExceptionHandler. ErrorDto: code, message, timestamp. HTTP-статусы: 400 (валидация), 404, 409 (конфликт), 500.*

07 Hibernate и JPA

- Состояния сущности.
  → *Transient → Persistent (persist) → Detached (close/detach) → Removed (remove). Persistent: dirty checking — изменения автоматически синхронизируются.*
- N+1 проблема.
  → *1 findAll() + N доп. запросов на lazy-коллекции. Решения: JOIN FETCH, @EntityGraph, @BatchSize(100), DTO-проекция. EAGER — неправильный ответ.*
- LazyInitializationException.
  → *Обращение к lazy-полю после закрытия сессии. Решения: JOIN FETCH, @EntityGraph, DTO. OpenSessionInView — анти-паттерн.*
- Оптимистичная блокировка.
  → *@Version (int/long). UPDATE ... WHERE version = ?. Не совпала → OptimisticLockException. Для low-contention (веб-приложения). Пессимистичная: SELECT FOR UPDATE.*
- JOOQ vs Hibernate.
  → *JOOQ: типизированный SQL, compile-time проверка запросов, нет dirty checking. Hibernate: ORM, lazy loading, кэширование. В Путешествиях используют оба — JOOQ для сложных запросов, Hibernate для CRUD.*
- Кэши Hibernate.
  → *First-level: привязан к Session, автоматический. Second-level: общий (EhCache, Caffeine), опциональный. Query cache: кэширует JPQL-результаты.*

ЛОВУШКА · EAGER = решение N+1?

Нет. EAGER грузит коллекцию ВСЕГДА, даже когда она не нужна — медленнее и съедает память. Правильно: LAZY по умолчанию + FETCH/EntityGraph там, где нужны данные.

// JavaJub

Это только начало гайда.

Яндекс Путешествия · Middle

Полная версия — бесплатно в Telegram-канале:

- все вопросы с ответами по грейду
- задачи: live-coding, SQL, System Design — с разбором
- чек-лист «за 2 часа до собеса»

@java_jub

t.me/java_jub · подписка бесплатная