// JavaJub

Собес в Лигу
Цифровой Экономики
Java Middle

Вопросы, задачи и подготовка
к live-coding и интервью

Стек
Java 11+ · Spring · Hibernate · PostgreSQL · JUnit · Docker · Jenkins

// содержание

Содержание

01 Про Лигу и формат собеса

Лига Цифровой Экономики — крупный российский IT-интегратор (6 000+ человек, больше 1 000 проектов в год). Клиенты: Сбер, ВТБ, Альфа-Банк, Ростелеком, МТС, Роснефть, МВД, министерства. Для Java Middle это значит: проекты чаще всего банковские или государственные, с жёсткими требованиями к качеству и тестированию.

Как устроен процесс

Этап	Длительность	Что проверяют
Скрининг HR	20–30 мин	Мотивация, ЗП-ожидания, согласие на ТК РФ
Тех-интервью	60–90 мин	Java Core + live-coding + SQL
Интервью с руководителем	30–60 мин	Проект, команда, матч по софтам
Проверка СБ	до недели	Стандартная для госпроектов
Оффер	—	По ТК РФ, ДМС, стоматология

Как проходит техническая секция

- Видео-созвон с демонстрацией экрана + совместный редактор кода.
- Обычно одним днём — собес часто занимает одну встречу на ~1 час.
- Меньше теоретической зубрёжки, больше практики и рассуждений.
- Обязательно будет живое кодирование и вопросы по SQL.
- Могут попросить разобрать кусок чужого кода или объяснить баг.

> **ФИШКА · Ключевая особенность Лиги**
>
> По отзывам кандидатов на DreamJob: «Лайфкодинг, понимание SQL. Спрашивают твоё умение мыслить, а не теоретическую подготовку». Готовься не пересказывать учебник, а писать код и объяснять, почему именно так.

02 Стек по вакансии

На апрель 2026 года на Хабр Карьере актуальна вакансия Java Middle в Лиге — в команду антифрод-решения ведущего банка. Стек оттуда — наш основной ориентир.

Обязательный минимум

- Java 11+, опыт от 2 лет
- Spring 5–6, Spring Boot 2–3
- Hibernate, Spring Data
- PostgreSQL (реляционные БД)
- Maven
- JUnit 5, Mockito
- Hamcrest, AssertJ
- Docker, Openshift
- Linux (уверенные знания)
- Jenkins
- TDD и умение писать unit-тесты

Будет плюсом

- Опыт разработки на Kotlin
- Микросервисная архитектура
- Опыт code review
- Опыт работы в банковских проектах

> **ВНИМАНИЕ · Что это значит для подготовки**
>
> Стек приземлённый — никаких Virtual Threads, ZGC, GraalVM и прочего хайпа. Ждут крепкое владение Java 11 + Spring + Hibernate + SQL + тестами. Всё остальное — бонус.

03 Java Core — что точно спросят

Базовые вопросы

- Что такое JDK, JRE, JVM?

  → *JVM — виртуальная машина (исполняет байт-код). JRE = JVM + стандартные библиотеки. JDK = JRE + инструменты разработки (javac, jdb, jar).*

- Области памяти JVM.

  → *Heap (объекты), Stack (фреймы методов, примитивы, ссылки), Metaspace (метаданные классов), PC Register, Native Method Stack.*

- Почему String immutable?

  → *Безопасность (передача в файлы, БД, сеть), потокобезопасность, кэширование hashCode, работа String Pool.*

- Что такое String Pool?

  → *Область в heap, где хранятся уникальные строковые литералы. String s = "hi" кладёт в пул, new String("hi") — создаёт новый объект в heap.*

- Разница между == и equals() для строк.

  → *== сравнивает ссылки, equals() — содержимое. Для литералов == может дать true из-за String Pool, но полагаться на это нельзя.*

- Что вернёт someObj.equals(null)?

  → *false. По контракту equals не должен бросать NPE на null-аргумент.*

- Контракт equals/hashCode.

  → *Если a.equals(b), то a.hashCode() == b.hashCode(). Обратное не обязательно. Переопределяешь один — переопределяй и второй.*

- Что сломается, если hashCode() вернуть константу?

  → *Все объекты попадут в один bucket HashMap. В Java 8+ при 8 коллизиях список превращается в red-black tree, но O(log n) вместо O(1) — всё равно деградация производительности.*

- Разница между checked и unchecked исключениями.

  → *Checked наследуются от Exception (но не RuntimeException) — компилятор требует throws или catch. Unchecked — от RuntimeException/Error — обрабатывать необязательно.*

- try-with-resources — какой интерфейс нужен?

  → *AutoCloseable (или Closeable). Ресурсы закрываются автоматически в обратном порядке объявления.*

- Что такое функциональный интерфейс? Примеры.

  → *Интерфейс с ровно одним абстрактным методом. @FunctionalInterface — опциональная аннотация. Примеры: Function, Predicate, Consumer, Supplier, Runnable, Comparator.*

- Stream API: промежуточные vs терминальные операции.

  → *Промежуточные (filter, map, sorted) возвращают Stream и ленивые. Терминальные (collect, forEach, count, reduce) запускают pipeline.*

- Разница между map и flatMap.

  → *map: T → R. flatMap: T → Stream с уплощением результата.*

- Когда использовать Optional?

  → *Для возвращаемых значений методов, где результат может отсутствовать. НЕ для полей, параметров методов, коллекций — это антипаттерн.*

- Можно ли переопределить static-метод?

→ *Нет. Static-методы не участвуют в полиморфизме. В подклассе это будет сокрытие (hiding), а не переопределение.*

- final у класса, метода и поля.

  → *Класс — нельзя наследовать. Метод — нельзя переопределить. Поле — нельзя переприсваивать (но можно менять внутреннее состояние, если поле — ссылка на mutable-объект).*

- Абстрактный класс vs интерфейс.

  → *Абстрактный класс может иметь состояние и реализацию, наследуется только один. Интерфейс — контракт, реализуется любое число. С Java 8 в интерфейсах появились default и static методы, с Java 9 — private.*

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

04 Коллекции

HashMap — чемпион по частоте вопросов. Готовься рассказывать «как устроено».

- Основные интерфейсы Collection Framework.

  → *Collection (List, Set, Queue), Map (отдельно, не Collection). List — упорядоченный с дубликатами. Set — без дубликатов. Map — пары ключ-значение.*

- Как устроен ArrayList?

  → *Динамический массив. При нехватке места создаёт новый массив в 1.5 раза больше и копирует через Arrays.copyOf.*

- ArrayList vs LinkedList.

  → *ArrayList: O(1) доступ по индексу, быстрая итерация, локальность в кэше. LinkedList: O(1) вставка в начало, но O(n) поиск. На практике почти всегда выигрывает ArrayList.*

- Как устроен HashMap?

  → *Массив бакетов (Node[]). Индекс бакета: (n - 1) & hash(key), где n — размер массива. Коллизии — связный список. С Java 8: при 8 элементах в бакете и размере таблицы ≥ 64 список превращается в red-black tree.*

- Что такое load factor?

  → *Порог заполнения, по умолчанию 0.75. При size >= capacity * loadFactor — resize: новый массив вдвое больше + перехеширование всех элементов.*

- Сложность операций HashMap.

  → *put, get, remove — O(1) в среднем. В худшем случае O(log n) благодаря дереву (Java 8+).*

- Можно ли null-ключ в HashMap?

  → *Да, один null-ключ, кладётся в bucket 0. В ConcurrentHashMap нельзя ни ключ, ни значение.*

- HashMap vs ConcurrentHashMap.

  → *HashMap не потокобезопасен. ConcurrentHashMap потокобезопасен: в Java 7 — Segment-блокировки, в Java 8+ — CAS + synchronized на головах бакетов.*

- HashMap vs TreeMap vs LinkedHashMap.

  → *HashMap — хэш, без порядка, O(1). TreeMap — красно-чёрное дерево, отсортирован по ключам, O(log n). LinkedHashMap — сохраняет порядок вставки или access order.*

- Fail-fast итератор.

  → *Кидает ConcurrentModificationException при изменении коллекции не через сам итератор. Так работают итераторы HashMap, ArrayList.*

- Как сделать List неизменяемым?

  → *List.copyOf(list) (Java 10+) или Collections.unmodifiableList(list). На add/remove — UnsupportedOperationException.*

- List.of(1,2,3) vs new ArrayList.

  → *List.of — immutable, add/remove → UnsupportedOperationException, null-элементы запрещены. ArrayList — обычный mutable список.*

05 Многопоточность и JMM

Для Middle ждут уверенное понимание synchronized, volatile, JMM и пулов потоков.

- Чем процесс отличается от потока?

  → *Процесс — изолированная единица ОС со своей памятью. Поток — единица исполнения внутри процесса, память общая.*

- Способы создать поток.

  → *extends Thread, implements Runnable/Callable, через ExecutorService, CompletableFuture. Предпочтительнее — пулы потоков через Executors.*

- Жизненный цикл потока.

  → *NEW → RUNNABLE → (BLOCKED / WAITING / TIMED_WAITING) → TERMINATED.*

- Как работает synchronized?

  → *Захватывает монитор объекта. На методе — монитор this (на static — Class-объект). На блоке — указанного объекта. Только один поток одновременно.*

- wait / notify / notifyAll — правила.

  → *Только внутри synchronized на том же объекте. wait() освобождает монитор и засыпает. notify будит один, notifyAll — все.*

- Почему wait() проверяют в while, а не if?

  → *Spurious wakeup — поток может проснуться сам по себе. После пробуждения условие может быть уже невалидным, нужна повторная проверка.*

- Зачем volatile?

  → *Гарантирует видимость изменений между потоками (запрет кэширования в регистрах/локальном кэше CPU) и запрет reordering. НЕ гарантирует атомарность составных операций (i++).*

- Что такое happens-before?

  → *Отношение в JMM, гарантирующее видимость. Примеры: synchronized release → acquire, volatile write → read, Thread.start() → действия потока, final-поля после конструктора.*

- Atomic-классы — как работают без блокировок?

  → *Через CAS (compare-and-swap) — атомарную инструкцию CPU. Lock-free. Классы: AtomicInteger, AtomicLong, AtomicReference.*

- Виды пулов в Executors.

  → *newFixedThreadPool, newCachedThreadPool, newSingleThreadExecutor, newScheduledThreadPool. В проде часто создают ThreadPoolExecutor вручную, чтобы контролировать очередь и политику отказа.*

- Чем опасен Executors.newCachedThreadPool() в проде?

  → *Неограниченное количество потоков — при всплеске нагрузки может положить JVM через OutOfMemoryError (unable to create new native thread).*

- ThreadLocal — где пригодится и где опасен?

  → *Для контекста (например, SecurityContext, транзакционный контекст). Опасен с пулами потоков — значение протекает между задачами, нужен remove() в finally.*

- CompletableFuture: thenApply vs thenCompose vs thenCombine.

  → *thenApply — преобразование результата. thenCompose — flatMap для CompletableFuture (нужен, когда лямбда возвращает CompletableFuture). thenCombine — объединение двух.*

- Как реализовать потокобезопасный singleton?

  → *Лучший вариант — enum. Альтернативы: static holder (Initialization-on-demand), double-checked locking с volatile.*

- Deadlock, livelock, starvation.

  → *Deadlock — два потока ждут друг друга. Livelock — активны, но не прогрессируют (оба уступают). Starvation — поток не получает CPU/ресурс.*

06 Spring и Spring Boot

В вакансии указан Spring 5–6 и Spring Boot 2–3. Блок обязательный, готовься основательно.

- Что такое IoC и DI?

  → *IoC — принцип: контейнер управляет жизненным циклом объектов. DI — реализация: зависимости внедряются извне (конструктор/сеттер/поле).*

- Какой способ внедрения предпочтительнее?

  → *Constructor injection. Поля можно final, явно видны обязательные зависимости, удобно тестировать без контекста, нет скрытых циклических зависимостей.*

- Жизненный цикл бина.

  → *Инстанцирование → инжект зависимостей → BeanNameAware/BeanFactoryAware/ApplicationContextAware → @PostConstruct → InitializingBean.afterPropertiesSet → init-method → готов → @PreDestroy → DisposableBean.destroy → destroy-method.*

- Scopes бина.

  → *singleton (default — один на контекст), prototype (новый при каждом запросе), request, session, application, websocket.*

- Чем @Component отличается от @Service, @Repository, @Controller?

  → *Технически почти ничем, все регистрируют бин. Семантически: @Service — бизнес-логика, @Repository — работа с БД + перевод исключений в DataAccessException, @Controller — веб-слой. @RestController = @Controller + @ResponseBody.*

- @Autowired + несколько подходящих бинов — что делать?

  → *@Qualifier("beanName") на точке инжекта, или @Primary на одном из бинов, или имя поля/параметра должно совпадать с именем бина.*

- Как работает @Transactional под капотом?

  → *Spring создаёт прокси (JDK dynamic proxy для интерфейсов или CGLIB для классов). Прокси открывает транзакцию перед методом, коммитит после или откатывает при исключении.*

- Почему @Transactional не работает при self-invocation?

  → *Вызов this.method() идёт минуя прокси. Решение: вынести метод в другой бин или инжектить self через ApplicationContext / @Lazy self.*

- На какие исключения откатывается транзакция по умолчанию?

  → *На RuntimeException и Error. Checked-исключения НЕ откатывают по умолчанию — нужно rollbackFor = Exception.class.*

- Propagation-уровни транзакций.

  → *REQUIRED (default) — присоединяется или создаёт. REQUIRES_NEW — всегда новая, приостанавливает текущую. NESTED — savepoint. SUPPORTS, MANDATORY, NOT_SUPPORTED, NEVER.*

- Что такое Spring Boot?

  → *Spring + автоконфигурация + встроенный сервер (Tomcat/Jetty/Undertow) + стартеры. Минимизирует boilerplate.*

- Что такое стартер?

  → *Готовый набор зависимостей под задачу. Например, spring-boot-starter-web подтянет Spring MVC, встроенный Tomcat, Jackson. Один импорт — и веб-приложение работает.*

- Как работает автоконфигурация?

  → *@EnableAutoConfiguration (входит в @SpringBootApplication) + @Conditional. Список конфигов в META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports (в Spring Boot 3+, раньше был spring.factories).*

- Что делает @SpringBootApplication?

// JavaJub

Это только начало гайда.

Лига Цифровой Экономики · 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