// JavaJub

Собес в Альфа-Банк

Alfa Digital

Java Middle

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

Стек

Java 11+ · Spring Boot · Kafka · PostgreSQL · Docker · Kubernetes · JUnit · Microservices

// содержание

Содержание

01 Про Альфа-Банк и формат собеса

Альфа-Банк — крупнейший частный банк России, №1 в рейтинге работодателей hh.ru (2025). IT-подразделение Alfa Digital разрабатывает мобильный банк, антифрод-системы, платёжные сервисы, кредитные конвейеры и десятки других продуктов. Для Java Middle это означает: микросервисы, высокие нагрузки, промышленный стек.

По отзывам кандидатов на DreamJob, собеседование в Альфу длится 1–2 часа и включает live-coding, вопросы по SQL, обсуждение опыта и проектов. Интервьюеры ценят умение рассуждать вслух и объяснять свои решения.

Как устроен процесс

Этап	Длительность	Что проверяют
Скрининг HR	20–30 мин	Мотивация, ЗП, стек, ожидания
Тех-интервью	60–90 мин	Java Core + live-coding + SQL + Spring
Системный дизайн	30–45 мин	Архитектура микросервисов, Kafka
Проверка СБ	до недели	Стандартная для банков (анкета, беседа)
Оффер	—	ДМС, стоматология, премии KPI, удалёнка

ФИШКА · Ключевая особенность

По отзывам кандидатов: в Альфе ценят умение рассуждать вслух и писать код в реальном времени. Это не экзамен по теории — это проверка инженерного мышления. Если не знаешь — скажи «не сталкивался, но предположу...» — честность ценят.

02 Стек по вакансии

На апрель 2026 года Альфа-Банк активно нанимает Java-разработчиков в Alfa Digital. Стек определён по вакансиям на hh.ru и job.alfabank.ru. Проекты: антифрод, кредитный конвейер, Альфа-Онлайн, электронное подписание.

Обязательный минимум

- Java 11+ (по некоторым вакансиям до Java 21), опыт от 1–2 лет
- Spring / Spring Boot — знать, как работает «под капотом» (прокси, автоконфигурация)
- Микросервисная архитектура — понимание принципов, декомпозиция, взаимодействие
- Kafka — брокеры сообщений, топики, партиции, consumer groups
- PostgreSQL / Oracle — SQL, индексы, транзакции, EXPLAIN ANALYZE
- Docker, Kubernetes / OpenShift — контейнеризация и оркестрация
- JUnit 5, Testcontainers, WireMock — обязательно умение писать тесты
- CI/CD (Jenkins / GitLab CI) — понимание пайплайна
- Git, code review — ежедневный рабочий инструмент

Будет плюсом

- MongoDB, Elasticsearch — NoSQL-хранилища
- Redis / Hazelcast / Infinispan — кэширование данных
- RxJava / Project Reactor — реактивное программирование
- Kotlin — встречается в некоторых командах
- Опыт в банковских / финтех проектах

ВНИМАНИЕ · Что это значит

Стек Альфы заточен под микросервисы и высокие нагрузки. Kafka, Kubernetes и кэширование — не «плюсы», а рабочие инструменты. На собесе спросят: «А что будет, если Kafka-consumer упадёт?» или «Как обеспечить идемпотентность?»

03 Java Core — что точно спросят

По отзывам кандидатов в банковские IT: equals/hashCode + HashMap — абсолютные чемпионы по частоте. Спрашивают все, абсолютно все. Далее идут Stream API, исключения и String Pool.

Базовые вопросы

- Что такое JDK, JRE, JVM?

  → *JVM — виртуальная машина, исполняет байт-код. JRE = JVM + стандартные библиотеки (нужно для запуска). JDK = JRE + инструменты разработки: javac, jdb, jar. С Java 9 JRE как отдельный дистрибутив не поставляется.*

- Области памяти JVM.

  → *Heap — хранение объектов, управляется GC. Stack — фрейм для каждого вызова метода (локальные переменные, ссылки, примитивы). Metaspace (до Java 8 — PermGen) — метаданные классов, пул интернированных строк. PC Register — адрес текущей инструкции потока. Native Method Stack — для JNI-вызовов.*

- Почему String immutable?

  → *Четыре причины: 1) безопасность — строки передаются в конструкторы файлов, БД, сетевых соединений, изменение привело бы к уязвимостям; 2) потокобезопасность — можно шарить между потоками без синхронизации; 3) кэширование hashCode — вычисляется один раз; 4) String Pool — экономия памяти.*

- Контракт equals/hashCode.

  → *Если a.equals(b) == true, то a.hashCode() == b.hashCode(). Обратное необязательно (коллизии допустимы). equals(null) → false. Переопределяешь один — переопределяй оба. Практический пример: положить объект в HashSet, изменить поле в hashCode — объект потеряется, contains() вернёт false.*

- Что сломается, если hashCode() вернёт константу?

  → *Все объекты попадут в один bucket HashMap. Java 7: связный список O(n). Java 8+: при 8 коллизиях И capacity ≥ 64 — перестройка в red-black tree O(log n). Это лучше O(n), но деградация по сравнению с O(1).*

- checked vs unchecked исключения.

  → *Checked: от Exception (не RuntimeException) — компилятор требует throws/catch. Примеры: IOException, SQLException. Unchecked: от RuntimeException и Error. Примеры: NullPointerException, IllegalArgumentException, ClassCastException. Современные фреймворки (Spring, Hibernate) предпочитают unchecked.*

- try-with-resources.

  → *Автоматически вызывает close() в finally. Ресурс реализует AutoCloseable. Закрытие в обратном порядке объявления. Если и try, и close бросают исключение — close-исключение подавляется (getSuppressed()). С Java 9 можно передать effectively final переменную.*

- Функциональный интерфейс.

  → *Ровно один абстрактный метод (SAM). @FunctionalInterface — опциональная аннотация, но защищает от случайного добавления второго метода. Основные: Function<T,R>, Predicate<T>, Consumer<T>, Supplier<T>, UnaryOperator<T>, BiFunction<T,U,R>, Comparator<T>, Runnable, Callable<V>.*

- Stream API: промежуточные vs терминальные.

  → *Промежуточные (filter, map, flatMap, sorted, distinct, peek, limit, skip) — ленивые, возвращают Stream, не выполняются пока нет терминальной. Терминальные (collect, forEach, count, reduce, findFirst, toList) — запускают конвейер. Стрим одноразовый — после терминальной повторно нельзя.*

- map vs flatMap.

  → *map: T → R, один элемент → один. flatMap: T → Stream<R> с уплощением. Пример: List<List<String>> → flatMap(Collection::stream) → плоский Stream<String>. Для Optional: flatMap избегает Optional<Optional<T>>.*

- Optional — когда?

  → *Для возвращаемых значений методов. НЕ для полей (не Serializable), НЕ для параметров, НЕ для коллекций (пустая лучше). Антипаттерны: Optional.get() без isPresent(), Optional в полях, Optional<List>.*

- final — класс, метод, поле.

  → *Класс — нельзя наследовать (String, Integer). Метод — нельзя переопределить. Поле — нельзя переприсвоить ссылку, но можно менять внутреннее состояние (final List — можно add/remove). effectively final — нужна для лямбд.*

- Абстрактный класс vs интерфейс.

  → *Абстрактный класс: состояние, конструкторы, реализация. Наследуется один. Интерфейс: контракт, любое число. Java 8 — default/static. Java 9 — private. Когда что: общее состояние → абстрактный класс, контракт для разных иерархий → интерфейс.*

Задачи «Что выведет?»

В банках любят такие задачи — проверяют понимание, а не заученность. Вот типичные примеры из реальных собесов:

Тест 1. Передача по значению

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

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

Ответ: 1. Java передаёт ссылки по значению. i++ создаёт новый объект Integer(2) и присваивает локальной переменной. Оригинал не меняется. То же самое со String — неизменяемый объект.

Тест 2. Integer cache

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

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

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

Тест 3. Stream API — ленивость

```
List<Integer> nums = List.of(1, 2, 3, 4, 5);
nums.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 элементы вертикально: каждый элемент проходит всю цепочку. Элементы 1 и 2 не проходят filter → печатаются один раз. Элементы 3,4,5 проходят → печатаются дважды.

СОВЕТ · Лайфхак

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

04 Коллекции

HashMap — абсолютный чемпион. По опыту кандидатов в банки: «обсасывают бедную мапу со всех сторон». Готовься рассказывать устройство, коллизии, treeify threshold, resize, null-ключи.

- Основные интерфейсы Collection Framework.

  → *Collection (List, Set, Queue) и Map (отдельно, НЕ наследует Collection). List — упорядоченный с дубликатами. Set — без дубликатов. Queue — очередь. Map — пары ключ-значение. Реализации: ArrayList, LinkedList, HashSet, TreeSet, HashMap, TreeMap, LinkedHashMap, ConcurrentHashMap, ArrayDeque, PriorityQueue.*

- Как устроен HashMap?

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

- load factor и resize.

  → *Порог 0.75 по умолчанию. При size >= capacity * loadFactor — resize: массив вдвое + перехеширование ВСЕХ элементов (дорого!). Совет: если знаешь количество элементов — задай initialCapacity = expectedSize / 0.75 + 1.*

- HashMap vs ConcurrentHashMap.

  → *HashMap: не потокобезопасен, допускает 1 null-ключ, null-значения. ConcurrentHashMap: потокобезопасен. Java 7 — Segment. Java 8+ — CAS + synchronized на головах бакетов (лучше параллелизм). null-ключи и null-значения ЗАПРЕЩЕНЫ. computeIfAbsent — атомарная «проверь и вставь».*

- ArrayList vs LinkedList.

  → *ArrayList: O(1) доступ по индексу, O(n) вставка в середину, дружелюбно к CPU-кэшу (непрерывная память). LinkedList: O(1) вставка в начало/конец (если есть ссылка), O(n) доступ по индексу. На практике ArrayList почти всегда лучше — даже вставка в середину быстрее из-за локальности кэша.*

- Fail-fast итератор.

  → *ConcurrentModificationException при изменении коллекции не через сам итератор. Реализован через modCount. Не гарантирован в многопоточной среде — только best effort. Альтернатива: CopyOnWriteArrayList (fail-safe, работает с копией).*

- TreeMap vs LinkedHashMap.

  → *TreeMap: красно-чёрное дерево, O(log n), ключи отсортированы (Comparable/Comparator). LinkedHashMap: HashMap + двусвязный список — порядок вставки. accessOrder=true — LRU-кэш. removeEldestEntry() для ограничения размера.*

- Как сделать List неизменяемым?

  → *List.of(1,2,3) — Java 9+, immutable, null запрещён. List.copyOf(list) — Java 10+, глубокая неизменяемая копия. Collections.unmodifiableList(list) — обёртка (изменения в оригинале видны!). Предпочитай List.of или List.copyOf.*

05 Многопоточность и JMM

Для Middle ждут уверенное понимание synchronized, volatile, happens-before и пулов потоков. JMM — обязательная тема для банков с высоконагруженными системами.

- synchronized.

  → *Захватывает монитор объекта. Instance-метод — монитор this. Static — монитор Class. Блок — указанный объект. Только один поток. Reentrant: поток может повторно захватить свой монитор (счётчик). Гарантирует: mutual exclusion + happens-before (видимость).*

- volatile.

  → *Видимость между потоками (запрет кэширования в регистрах/L1-кэше) + запрет reordering вокруг volatile. НЕ гарантирует атомарность i++ (read-increment-write = три операции). Для атомарных — AtomicInteger.*

- happens-before.

  → *Ключевое отношение JMM. Если A happens-before B, то записи в A видны в B. Примеры: unlock → lock того же монитора. volatile write → read. Thread.start() → первая инструкция потока. Последняя инструкция → Thread.join(). final-поля видны после конструктора.*

- Atomic-классы.

  → *AtomicInteger, AtomicLong, AtomicReference. CAS (Compare-And-Swap) — атомарная инструкция CPU. Lock-free. Методы: get, set, compareAndSet, incrementAndGet. Для счётчиков с высоким contention — LongAdder (лучше масштабируется).*

- wait/notify — почему while?

  → *Spurious wakeup: JVM/ОС может разбудить поток без notify. while (!condition) { wait(); }. Важно: вызывать ТОЛЬКО внутри synchronized на том же объекте, иначе — IllegalMonitorStateException. notify() будит один поток, notifyAll() — все.*

- Пулы потоков.

  → *newFixedThreadPool(n) — фикс. число, LinkedBlockingQueue. newCachedThreadPool — неограниченно (ОПАСНО — OOM). newScheduledThreadPool — периодические задачи. В проде: ThreadPoolExecutor вручную с corePoolSize, maxPoolSize, BlockingQueue и RejectedExecutionHandler.*

- ThreadLocal — опасность.

  → *С пулами потоков: поток переиспользуется, ThreadLocal остаётся от предыдущей задачи (утечка данных/памяти). Обязательно: try { ... } finally { threadLocal.remove(); }. В Reactor/WebFlux — не работает, нужен Context.*

- Deadlock.

  → *Два+ потока ждут друг друга. Условия Коффмана: mutual exclusion, hold and wait, no preemption, circular wait. Избегать: упорядочить захват мониторов (lock A → lock B), tryLock с таймаутом. Обнаружение: jstack, VisualVM.*

- CompletableFuture.

  → *thenApply: T → R. thenCompose: T → CF<R> (flatMap). thenCombine: объединение двух CF. exceptionally: обработка ошибки. Async-варианты выполняются в ForkJoinPool.commonPool() или указанном Executor.*

06 Spring и Spring Boot

В вакансии Альфы прямо указано: «знаешь, как работает Spring/Spring Boot под капотом». Поверхностного понимания недостаточно — будут копать в прокси, автоконфигурацию, жизненный цикл бинов.

- IoC и DI.

  → *IoC — не ты создаёшь зависимости, а контейнер. DI — реализация: зависимости внедряются через конструктор/сеттер/поле. Преимущества: слабая связанность, подмена реализаций, тестируемость (можно подставить мок).*

- Какой DI лучше?

  → *Constructor injection. Поля можно final, явно видны зависимости, легко тестировать без контекста (new Service(mockRepo)), невозможен циклический DI при старте (сигнал о проблеме в дизайне). С Spring 4.3: если один конструктор — @Autowired не нужен.*

- Жизненный цикл бина.

  → *Инстанцирование → DI → Aware-интерфейсы → BPP.postProcessBefore → @PostConstruct → InitializingBean → init-method → BPP.postProcessAfter → ГОТОВ → @PreDestroy → DisposableBean → destroy-method. BeanPostProcessor — точка для создания прокси (@Transactional, @Async).*

- @Transactional — под капотом.

  → *Spring создаёт прокси-обёртку. JDK dynamic proxy (если интерфейс) или CGLIB (наследование). Прокси: открывает транзакцию (PlatformTransactionManager), вызывает метод, commit при успехе, rollback при RuntimeException/Error. Checked НЕ откатывают — нужно rollbackFor.*

- self-invocation.

  → *this.method() минует прокси → @Transactional/@Async/@Cacheable не работают. Решения: вынести метод в другой бин (рекомендуемый), инжектить self через @Lazy или ObjectProvider, использовать AspectJ compile-time weaving.*

- Propagation.

  → *REQUIRED (default) — присоединяется/создаёт. REQUIRES_NEW — всегда новая, приостанавливает текущую (аудит/логирование). NESTED — savepoint внутри текущей. SUPPORTS — присоединяется если есть, нет — без. MANDATORY — требует существующей. NOT_SUPPORTED — приостанавливает. NEVER — если есть → исключение.*

- @SpringBootApplication.

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

- Scopes бина.

  → *singleton (default — один на контекст), prototype (новый при каждом запросе), request (один на HTTP-запрос), session, application, websocket. Важно: инжект prototype в singleton через Provider<T> или ObjectFactory<T>, иначе prototype-бин создастся один раз.*

- Обработка исключений.

  → *@RestControllerAdvice + @ExceptionHandler. ErrorDto с code, message, timestamp. HTTP-статусы: 400 (валидация), 404 (не найдено), 409 (бизнес-конфликт), 500. MethodArgumentNotValidException для @Valid, ConstraintViolationException для @Validated на query-параметрах.*

// JavaJub

Это только начало гайда.

Альфа-Банк · 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