// JavaJub

Собес в Сбер

Java Middle

Вопросы, задачи и реальные кейсы
с технических интервью

Темы

JavaCore · JVM · Collections · Multithreading · Spring · SQL · System Design

Содержание

01 Как устроен отбор

В Сбере больше 2 000 продуктовых команд, и каждая может адаптировать процесс под себя. Но общая схема для Java Middle в 2025–2026 годах выглядит так:

Этап	Длительность	Что проверяют
HR-скрининг	30–40 мин	Мотивацию, ожидания по ЗП, общий опыт, стек
Тех-скрининг	30–45 мин	Базовый синтаксис, SQL, Big-O, коллекции
Java + Code review	60–90 мин	JavaCore, JVM, многопоточность, контракты
System Design	60–90 мин	Проектирование (для Middle+ — опционально)
Team Match	60 мин	Soft skills, ценности, выбор команды

Как выглядит тех-секция

- Видео-созвон с демонстрацией экрана, совместный редактор кода.
- Обычно: теория + 1–2 практические задачи уровня LeetCode Easy/Medium или code review.
- Нужно рассуждать вслух — оценивается ход мыслей, а не только финальный ответ.
- AI-ассистенты и поисковики запрещены. За списыванием следят.

> **ФИШКА · Особенность Сбера**
>
> Реже дают чистые алгоритмы (в отличие от Яндекса и VK). Чаще — код с неочевидными багами, который компилируется и проходит тесты, но тихо ломает данные в проде. Смотрят, видишь ли ты контракты языка «под капотом».

02 Стек и темы, которые спрашивают

В 2026 году стандарт в Сбере — Java 21 LTS (местами уже Java 25), Spring Boot 3.x, PostgreSQL-совместимая Pangolin, Kafka-совместимая Corax, Kubernetes внутри Platform V.

Обязательный минимум для Middle

- Java Core: ООП, equals/hashCode, Generics, Exceptions, Stream API, Optional
- Collections Framework: HashMap, ArrayList/LinkedList, ConcurrentHashMap, TreeMap
- JVM: heap/stack, сборщики мусора (G1, ZGC), JIT, classloading
- Многопоточность: synchronized, volatile, wait/notify, Executors, CompletableFuture, JMM
- Spring / Spring Boot: DI, bean scopes, @Transactional, REST, Spring Data JPA
- БД: SQL, индексы, уровни изоляции, MVCC, EXPLAIN, оптимизация запросов
- Микросервисы: REST, gRPC, Kafka, паттерны (Saga, Circuit Breaker, Idempotency)
- Тестирование: JUnit 5, Mockito, Testcontainers
- Git, Docker, основы Kubernetes, CI/CD

Плюсом будет

- Virtual Threads (Project Loom) — как влияют на производительность
- Records, Sealed classes, Pattern Matching (Java 17–21)
- Sequenced Collections (Java 21)
- Spring Native / GraalVM
- OpenTelemetry, observability
- Специфика Platform V: Pangolin, Synapse, Corax

03 Java Core & ООП

Самый частый блок. Для Middle ждут не только знания, но и понимания «почему так».

Типовые вопросы

- Расскажите про ООП: инкапсуляция, наследование, полиморфизм.
  → *Будьте готовы объяснить полиморфизм пятикласснику.*
- В чём разница между композицией и агрегацией?
  → *Композиция = владение (жизненные циклы связаны), агрегация = ссылка.*
- Какие области памяти в JVM? Что хранится в heap, а что в stack?
  → *Heap — объекты и массивы; stack — фреймы методов, примитивы, ссылки.*
- Что такое String Pool? Почему String immutable?
  → *Безопасность, потокобезопасность, кэширование hashCode, работа String Pool.*
- Разница между == и equals(). Что вернёт someObj.equals(null)?
  → *== сравнивает ссылки; equals(null) по контракту должен возвращать false.*
- Контракт equals/hashCode. Что сломается при нарушении?
  → *Если a.equals(b), то hashCode совпадают. Иначе HashMap/HashSet теряют объекты.*
- Что будет, если hashCode() всегда возвращает одну константу?
  → *В Java 7 — деградация HashMap до связного списка O(n). В Java 8+ при 8 коллизиях в бакете он превращается в red-black tree, O(log n). В обоих случаях производительность резко падает.*
- Разница между checked и unchecked исключениями.
  → *Checked — от Exception (кроме RuntimeException), компилятор требует обработки. Unchecked — от RuntimeException/Error.*
- try-with-resources — какой интерфейс нужен?
  → *AutoCloseable (или Closeable). Ресурсы закрываются в обратном порядке.*
- Что такое functional interface? Приведите 3 примера из java.util.function.
  → *Интерфейс с одним абстрактным методом. Примеры: Function<T,R>, Predicate<T>, Consumer<T>, Supplier<T>.*
- Stream API: промежуточные vs терминальные операции. Что такое ленивость?
  → *Промежуточные (filter, map) не выполняются до вызова терминальной (collect, forEach).*
- Разница между map и flatMap.
  → *map: T → R; flatMap: T → Stream<R> с «уплощением».*
- Когда использовать Optional, а когда — нет?
  → *Для возвращаемых значений. НЕ для полей, параметров методов, коллекций.*
- Records (Java 16+): можно ли от них наследоваться?
  → *Нет. Record неявно наследуется от java.lang.Record, а Java не поддерживает множественное наследование классов.*
- Sealed classes — зачем?
  → *Ограничение иерархии наследования. Хорошо работает с pattern matching.*
- Generics: PECS. Что означает ? extends T и ? super T?
  → *Producer Extends, Consumer Super. extends — можно читать, нельзя писать; super — наоборот.*
- Что такое type erasure?

→ *Стирание дженериков во время компиляции. Последствия: нельзя new T(), нельзя instanceof T, нельзя создать массив T[].*

СОВЕТ · Лайфхак

На любой вопрос про контракт equals/hashCode обязательно приведите пример: «если я положу объект в HashSet, а потом изменю поле, участвующее в hashCode, — я его больше не найду». Это показывает понимание, а не заученность.

04 Collections Framework

HashMap — абсолютный чемпион по числу упоминаний. Спрашивают все и всегда.

- Как устроен HashMap? Что такое bucket, как разрешаются коллизии?

  → *Массив связных списков (Node[]). Бакет = ячейка массива. Коллизия → цепочка.*

- Что поменялось в HashMap с Java 8?

  → *При 8 элементах в бакете и размере таблицы ≥ 64 список превращается в red-black tree. Поиск становится O(log n) вместо O(n) в худшем случае.*

- Сложность операций HashMap: put, get, remove.

  → *Амортизированная O(1), в худшем случае O(log n) (Java 8+) или O(n) (Java 7).*

- Что такое load factor?

  → *Порог заполнения (по умолчанию 0.75), при котором происходит resize — удвоение массива и rehashing всех элементов.*

- Можно ли использовать как ключ изменяемый объект (например, массив)?

  → *Технически можно, но опасно: если изменить объект после добавления — его hashCode может измениться, и найти элемент станет невозможно.*

- HashMap vs ConcurrentHashMap — ключевые различия.

  → *HashMap не потокобезопасен, допускает null-ключ/значение. ConcurrentHashMap потокобезопасен, не допускает null, в Java 7 использовал сегментную блокировку, в Java 8+ — CAS + synchronized на головах бакетов.*

- HashMap vs TreeMap vs LinkedHashMap — когда что?

  → *HashMap — быстрый доступ без порядка. TreeMap — сортировка ключей, O(log n). LinkedHashMap — сохраняет порядок вставки или access order.*

- ArrayList vs LinkedList — где быстрее вставка в середину?

  → *У LinkedList O(1) на саму вставку, но O(n) на поиск позиции. В реальности ArrayList почти всегда быстрее из-за локальности в кэше CPU.*

- Как работает resize в ArrayList?

  → *Новый массив в 1.5 раза больше, копирование через Arrays.copyOf.*

- Fail-fast vs fail-safe итераторы.

  → *Fail-fast (HashMap, ArrayList) кидают ConcurrentModificationException при изменении коллекции. Fail-safe (ConcurrentHashMap, CopyOnWriteArrayList) работают со снимком.*

- List.of(1,2,3) vs new ArrayList<>() — в чём разница?

  → *List.of возвращает неизменяемый список. add/remove бросают UnsupportedOperationException.*

- Sequenced Collections (Java 21) — что это?

  → *Новый интерфейс с методами getFirst, getLast, addFirst, addLast, reversed. Унифицирует работу с упорядоченными коллекциями.*

05 JVM, память, GC

- JVM, JRE, JDK — в чём разница?

  → *JVM — виртуальная машина. JRE = JVM + библиотеки. JDK = JRE + инструменты разработки (javac, jdb).*

- Области памяти JVM.

  → *Heap (shared), Stack (per thread), Metaspace (class metadata), PC Register, Native Method Stack.*

- Young / Old generation — как работает поколенческая сборка?

  → *Young: Eden + 2 Survivor. После нескольких переживаний minor GC — объект попадает в Old.*

- Какие бывают сборщики мусора?

  → *Serial, Parallel, CMS (deprecated с Java 9, удалён в Java 14), G1 (default с Java 9), ZGC, Shenandoah. Для low-latency выбирают ZGC/Shenandoah.*

- Что такое Stop-the-World?

  → *Пауза приложения для сборки мусора. У ZGC и Shenandoah паузы < 10 мс даже на огромных heap'ах.*

- Виды OutOfMemoryError.

  → *Java heap space, Metaspace, Unable to create new native thread, GC overhead limit exceeded.*

- Как диагностировать утечку памяти?

  → *Heap dump (jmap, -XX:+HeapDumpOnOutOfMemoryError) → анализ в Eclipse MAT или VisualVM.*

- Classloader: parent delegation model.

  → *Bootstrap → Platform (ext) → Application. Дочерний сначала делегирует родителю, потом грузит сам.*

- Что такое JIT-компиляция?

  → *Оптимизация «горячего» кода в нативный. C1 — быстрый прогрев, C2 — агрессивная оптимизация.*

- Escape analysis — что это даёт?

  → *JIT определяет, «сбегает» ли объект из метода. Если нет — может аллоцировать на стеке вместо heap (scalar replacement).*

06 Многопоточность и JMM

Для Middle этот блок — обязательный. Готовьтесь подробно.

- Чем процесс отличается от потока?

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

- Способы создать поток в Java.

  → *extends Thread, implements Runnable / Callable, ExecutorService, CompletableFuture, Virtual Thread (Java 21+).*

- Жизненный цикл потока.

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

- Что такое монитор объекта? Как работает synchronized?

  → *У каждого объекта есть монитор. synchronized захватывает его при входе, освобождает при выходе (включая исключения).*

- Разница между synchronized-методом и synchronized-блоком.

  → *Метод захватывает this (для static — Class-объект). Блок — любой указанный объект. Блок обычно эффективнее.*

- wait / notify / notifyAll — правила использования.

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

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

  → *Из-за spurious wakeup — поток может проснуться сам по себе. После пробуждения условие может быть уже невалидным.*

- Зачем нужен volatile?

  → *Гарантирует видимость изменений между потоками и запрет reorderings. НЕ гарантирует атомарность составных операций (i++).*

- Что такое happens-before?

  → *Отношение в JMM, гарантирующее видимость операций. Примеры: synchronized release→acquire, volatile write→read, Thread.start→действия потока.*

- Atomic-классы — как работают без блокировок?

  → *Через CAS (compare-and-swap) — атомарную инструкцию процессора. Lock-free алгоритм.*

- Что такое ABA-проблема?

  → *Значение сменилось с A на B и обратно на A. CAS это не видит. Решение: AtomicStampedReference с версией.*

- Виды пулов потоков в Executors.

  → *newFixedThreadPool, newCachedThreadPool, newSingleThreadExecutor, newScheduledThreadPool, newWorkStealingPool. Для прода обычно создают ThreadPoolExecutor вручную, чтобы контролировать очередь.*

- Почему опасно использовать Executors.newCachedThreadPool() в проде?

  → *Unbounded очередь и неограниченное количество потоков — при всплеске нагрузки можно положить JVM OutOfMemoryError.*

- ThreadLocal — где пригождается и где опасен?

  → *Удобен для контекста (например, SecurityContext). Опасен при использовании с пулами потоков — значение «протекает» между задачами. Обязательно remove() в finally.*

- CompletableFuture: thenApply vs thenCompose vs thenCombine.

→ *thenApply: преобразование результата. thenCompose: flatMap для CompletableFuture. thenCombine: объединение двух результатов.*

- Virtual Threads — чем отличаются от обычных?

  → *Управляются JVM, а не ОС. Лёгкие (несколько КВ против ~1 МВ у platform). Можно писать блокирующий код, который масштабируется как асинхронный.*

- Как сделать потокобезопасный singleton?

  → *Лучшее — enum. Альтернативы: static holder (lazy init via classloader), double-checked locking с volatile.*

- Deadlock, livelock, starvation — в чём разница?

  → *Deadlock: два потока ждут друг друга. Livelock: потоки активны, но не прогрессируют. Starvation: поток не получает ресурс.*

> **ЛОВУШКА · Классика Сбера**
>
> Попросить написать double-checked locking singleton. Если кандидат забыл volatile на instance — это минус. Спросите себя: зачем здесь volatile? (Ответ: без него возможна публикация «полусозданного» объекта другим потоком из-за reordering).

// JavaJub

Это только начало гайда.

Сбер · 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