Java Backend Engineer Interview

Java Core (20 points)

1. Understanding of OOP principles: Encapsulation, Inheritance, Polymorphism, Abstraction.
2. Generics in Java: Use of type parameters, bounded types, and wildcard generics.
3. Multithreading in Java: Creating threads, thread lifecycle, and inter-thread communication.
4. JVM memory management: Heap, Stack, PermGen/Survivor spaces, garbage collection algorithms.
5. Exception handling: Checked and unchecked exceptions, try-catch blocks, finally, and multi-catch.
6. Serialization in Java: Serializable interface, custom serialization with writeObject and readObject.
7. Java Collections Framework: List, Set, Map, Queue interfaces and their implementations.
8. Lambda expressions and functional interfaces: Using predicates, consumers, suppliers, and functions.
9. Stream API: Intermediate and terminal operations, parallel streams, and stream pipelining.
10. Reflection API: Accessing classes, methods, and fields at runtime, annotation processing.
11. Java IO vs NIO: Differences in file handling, channel-based I/O, and non-blocking I/O.
12. Java Date and Time API: Working with LocalDate, LocalDateTime, and Duration.
13. Java Networking: Socket programming, URL connections, and HTTP clients.
14. Java Security: Cryptography, digital signatures, and secure coding practices.
15. Java Modules: Understanding of JPMS (Java Platform Module System) and modularity.
16. Java Enumerations: Use of enums, ordinal values, and custom methods in enums.
17. Java Annotations: Built-in annotations, custom annotations, and annotation processing.
18. Java Concurrency Utilities: CountDownLatch, CyclicBarrier, Semaphore, and Exchanger.
19. Java Memory Leaks: Causes, detection, and prevention strategies.
20. Java Performance Tuning: JVM options, profiling tools, and memory optimization techniques.

Spring Ecosystem (20 points)

21. Spring IoC container: Dependency injection, bean lifecycle, and scope.
22. Spring Boot auto-configuration: How Spring Boot automatically configures beans.
23. Spring Data JPA: Repository patterns, CRUD operations, and query methods.
24. Spring Security: Authentication, authorization, and securing REST APIs.

25. Spring MVC: Controller methods, request mapping, and view resolution.
26. Spring Cloud: Service discovery with Eureka, load balancing with Ribbon.
27. Spring AOP: Aspect Oriented Programming, cross-cutting concerns, and advice types.
28. Spring Boot Actuator: Monitoring endpoints, health checks, and metrics collection.
29. Spring Profiles: Environment-specific configurations and profile activation.
30. Spring Boot Starter Dependencies: Use of starters to simplify dependency management.
31. Spring Integration: Integrating different systems, messaging, and adapters.
32. Spring Batch: Batch processing, job scheduling, and step implementations.
33. Spring Cache: Caching strategies, annotations, and cache managers.
34. Spring WebFlux: Reactive programming, non-blocking I/O, and WebFlux frameworks.
35. Spring Cloud Config: Centralized configuration management for microservices.
36. Spring Cloud Gateway: API gateway patterns, routing, and filtering.
37. Spring Boot Testing: Using @SpringBootTest, MockMvc, and TestRestClient.
38. Spring Data REST: Exposing repositories as RESTful services.
39. Spring Cloud Stream: Integration with message brokers like RabbitMQ and Kafka.
40. Spring Cloud Sleuth: Distributed tracing and logging in microservices.

Microservices Architecture (20 points)

41. Service Discovery: How Eureka, Consul, and Zookeeper work.
42. API Gateway: Patterns, routing, and security in API gateways.
43. Circuit Breaker: Implementing resilience with Hystrix, Resilience4j.
44. Event-Driven Architecture: Event sourcing, message brokers, and event handlers.
45. RESTful API Design: HATEOAS, stateless design, and REST constraints.
46. GraphQL: Implementing GraphQL APIs, schema definitions, and resolvers.
47. Microservices Communication: Synchronous vs asynchronous communication.
48. Saga Pattern: Managing distributed transactions across services.
49. Health Checks: Implementing liveness and readiness probes.
50. Contract First Development: Using Swagger for API contracts.
51. API Versioning: Strategies for versioning RESTful APIs.

52. Rate Limiting: Implementing rate limits to prevent abuse.
53. Circuit Breaker Patterns: Implementing fallbacks and retries.
54. Microservices Deployment: Using Docker, Kubernetes, and cloud platforms.
55. Service Mesh: Understanding Istio, Linkerd, and their benefits.
56. Event Collaboration: Saga vs Choreography patterns.
57. Microservices Security: OAuth2, JWT, and API gateways.
58. Monitoring and Tracing: Tools like Prometheus, Grafana, and Jaeger.
59. Microservices Testing: Integration testing, contract testing, and end-to-end testing.
60. Database per Service: Data management and consistency in microservices.

Databases and Caching (20 points)

61. SQL Joins: Inner, outer, left, right, and cross joins.
62. ACID Properties: Atomicity, Consistency, Isolation, Durability in transactions.
63. NoSQL Databases: Document stores, key-value stores, and graph databases.
64. Redis Caching: In-memory data store, data structures, and persistence options.
65. Memcached vs Redis: Comparing caching solutions.
66. Database Sharding: Horizontal partitioning and load balancing.
67. ORM Frameworks: Hibernate, MyBatis, and JPA specifications.
68. JDBC Connection Pooling: DataSource implementations and connection lifecycle.
69. Full-Text Search: Implementing search in databases like Elasticsearch.
70. Time-Series Databases: InfluxDB, OpenTSDB for time-based data.
71. Transaction Isolation Levels: Read uncommitted, read committed, repeatable read, serializable.
72. Indexing Strategies: B-tree, hash indexes, and composite indexes.
73. Database Replication: Master-slave, master-master setups.
74. Database Backup and Recovery: Strategies for data protection.
75. Database Profiling: Tools like SQL Profiler, slow query logs.
76. NoSQL Consistency Models: Eventual consistency, CAP theorem.
77. Database Migrations: Using Flyway, Liquibase for schema changes.
78. Caching Strategies: Cache-aside, read-through, write-through patterns.

79. Cache Invalidation: Managing cache expiration and invalidation.
80. Database Connection Pooling: HikariCP, Tomcat JDBC pool configurations.

Concurrency and Multithreading (20 points)

81. Thread Lifecycle: New, runnable, running, blocked, waiting, terminated.
82. Synchronization Mechanisms: Locks, synchronized blocks, and intrinsic locks.
83. Reentrant Locks: Benefits over synchronized blocks, fairness, and timeouts.
84. Executor Framework: ThreadPoolExecutor, ExecutorService, and thread pool configurations.
85. Callable vs Runnable: Differences and use cases.
86. Java Memory Model: Visibility, happens-before relationships, and memory consistency.
87. Volatile Keyword: Ensuring visibility of variable changes across threads.
88. Deadlock Prevention: Avoiding and detecting deadlocks.
89. Asynchronous Programming: Using CompletableFuture for non-blocking operations.
90. ScheduledExecutorService: Scheduling tasks with fixed rates and delays.
91. Thread Pools: Fixed, cached, and scheduled thread pools.
92. Lock Striping: Reducing lock contention with striped locks.
93. Read-Write Locks: Allowing multiple readers or a single writer.
94. Wait and Notify Mechanisms: Inter-thread communication using wait/notify.
95. Thread Interruption: Handling interrupts and designing interruptible tasks.
96. Thread-Safe Classes: Implementing thread-safe singleton patterns.
97. Concurrency Utilities: CountDownLatch, CyclicBarrier, Semaphore.
98. Java 8+ Concurrency Features: Parallel streams, fork-join framework.
99. Multicore Programming: Challenges and solutions for parallel processing.
100. Thread Dumps and Analysis: Identifying issues with thread dumps.

Web Servers and Load Balancing (20 points)

101. Apache Tomcat Configuration: Setting up connectors, context.xml, and server.xml.
102. Nginx as Reverse Proxy: Configuring proxy_pass, upstream servers, and load balancing.
103. HAProxy for High Availability: Setting up failover and session persistence.

104. Web Server Security: SSL/TLS configurations, security headers, and firewall rules.
105. Load Balancing Algorithms: Round Robin, Least Connections, IP Hash.
106. Server-Side Caching: Using Varnish, Redis, or in-memory caches.
107. Monitoring Tools: Using Prometheus, Grafana, and New Relic for server monitoring.
108. Logging in Production: Centralized logging with ELK stack or Graylog.
109. Horizontal vs Vertical Scaling: Understanding trade-offs and use cases.
110. Web Server Performance Tuning: Adjusting worker threads, connection timeouts, and buffers.
111. Reverse Proxy Caching: Configuring cache headers and expiration.
112. Web Server Load Testing: Tools like Apache JMeter, Gatling for performance testing.
113. SSL Offloading: Handling SSL/TLS termination at the load balancer.
114. Web Server Hardening: Security best practices and vulnerability assessments.
115. Dynamic vs Static Content Serving: Optimizing server configurations.
116. Web Server Clustering: Setting up clusters for high availability.
117. Web Server Authentication: Implementing basic, digest, and OAuth authentication.
118. Web Server Logging Formats: Common log formats and parsing tools.
119. Web Server Resource Limits: Configuring limits on connections, requests, and bandwidth.
120. Web Server Backup and Recovery: Strategies for disaster recovery.

CI/CD and DevOps (20 points)

121. Jenkins Pipeline as Code: Writing Jenkinsfiles for CI/CD pipelines.
122. Docker Containerization: Dockerfile creation, multi-stage builds, and container orchestration.
123. Kubernetes Orchestration: Deployments, services, pods, and scaling strategies.
124. GitOps Principles: Using Git for infrastructure and configuration management.
125. Maven and Gradle Build Tools: Dependency management, plugins, and build lifecycle.
126. Unit and Integration Testing: Writing tests with JUnit, Mockito, and TestNG.
127. Code Coverage Tools: Using Jacoco for measuring code coverage.
128. Static Code Analysis: Tools like SonarQube for code quality checks.
129. Infrastructure as Code (IaC): Using Terraform, CloudFormation for infrastructure provisioning.
130. Blue/Green Deployments: Minimizing downtime during deployments.

131. Canary Deployments: Gradual rollout of new features.
132. Automated Testing in CI Pipelines: Integrating tests with build stages.
133. Environment Management: Using Ansible, Chef, or Puppet for configuration management.
134. CI/CD Best Practices: Continuous integration, continuous deployment, and continuous delivery.
135. Rollback Strategies: Implementing automated rollbacks on deployment failures.
136. Security Scanning: Incorporating security checks like SAST, DAST in pipelines.
137. CI/CD Pipelines for Microservices: Managing pipelines for multiple services.
138. Monitoring CI/CD Pipelines: Alerting on pipeline failures and performance issues.
139. DevOps Tools Ecosystem: Understanding tools like Docker, Kubernetes, Jenkins, Ansible.
140. CI/CD for Cloud-Native Applications: Deploying applications on cloud platforms.

Design Patterns and Best Practices (20 points)

141. Singleton Pattern: Implementing thread-safe singletons.
142. Factory Pattern: Creating objects without specifying the exact class.
143. Strategy Pattern: Encapsulating algorithms and switching between them.
144. SOLID Principles: Understanding and applying Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion.
145. Dependency Injection: Reducing coupling and increasing code maintainability.
146. Event Sourcing Pattern: Storing events to reconstruct application state.
147. CQRS Architecture: Separating command and query responsibilities.
148. Designing for Scalability: Using horizontal scaling, sharding, and load balancing.
149. Code Refactoring Techniques: Extracting methods, renaming variables, and simplifying conditionals.
150. Clean Code Practices: Writing readable, maintainable, and self-documenting code.
151. Test-Driven Development (TDD): Writing tests before implementation.
152. Code Versioning: Using Git branching strategies like GitFlow, Trunk-Based Development.
153. Designing for Maintainability: Using modular design, separation of concerns.
154. Anti-Patterns to Avoid: God classes, spaghetti code, and tight coupling.
155. Designing for Security: Implementing least privilege, defense in depth.
156. Designing for Performance: Optimizing algorithms, reducing I/O operations.

157. Designing for Reliability: Implementing redundancy, fault tolerance, and error handling.
158. Designing for Extensibility: Using plugins, extensions, and open APIs.
159. Designing for Usability: Ensuring APIs are intuitive and well-documented.
160. Designing for Testability: Writing code that is easy to test and mock.

Security (20 points)

161. OAuth2 and JWT: Implementing token-based authentication.
162. Role-Based Access Control (RBAC): Assigning roles and permissions to users.
163. Security Headers: Implementing Content Security Policy, X-Frame-Options.
164. SQL Injection Prevention: Using prepared statements and parameterized queries.
165. Cross-Site Scripting (XSS) Protection: Sanitizing inputs and outputs.
166. Encryption and Decryption: Using AES, RSA for data protection.
167. Secure Coding Practices: Avoiding common vulnerabilities like buffer overflows.
168. Implementing Audit Trails: Logging user actions and system events.
169. Handling Sensitive Data: Storing passwords securely with hashing algorithms.
170. Compliance with Regulations: GDPR, PCI-DSS, and data protection laws.
171. Implementing Two-Factor Authentication (2FA): Adding an extra layer of security.
172. Security Testing: Penetration testing, vulnerability assessments.
173. Secure Communication Protocols: Implementing SSL/TLS for data encryption.
174. Secure Session Management: Managing session tokens and timeouts.
175. Implementing Web Application Firewalls (WAF): Protecting against common attacks.
176. Security Monitoring and Alerting: Using tools like SIEM for threat detection.
177. Security Best Practices in Microservices: Securing service-to-service communication.
178. Implementing CAPTCHA for Bot Protection: Preventing automated attacks.
179. Security in CI/CD Pipelines: Scanning for vulnerabilities during builds.
180. Implementing Security by Design: Incorporating security from the start of the development process.

Performance Tuning and Optimization (20 points)

181. Profiling Java Applications: Using tools like JProfiler, VisualVM for performance analysis.

182. Garbage Collection Tuning: Adjusting GC parameters for performance.
183. Database Query Optimization: Indexing, query rewriting, and using explain plans.
184. Caching Strategies: Using distributed caches, cache invalidation mechanisms.
185. Load Testing and Stress Testing: Identifying performance bottlenecks.
186. Optimizing RESTful APIs: Reducing response times, minimizing data transfer.
187. Reducing Network Latency: Using CDNs, optimizing API calls.
188. Connection Pool Sizing: Determining optimal pool sizes for databases and connections.
189. Monitoring and Alerting Setups: Using Prometheus, Grafana for real-time monitoring.
190. Identifying and Resolving Bottlenecks: Profiling CPU, memory, and I/O usage.
191. Optimizing Java Heap Settings: Setting appropriate heap sizes for different environments.
192. Reducing Garbage Collection Pauses: Using G1GC, ZGC for low-latency applications.
193. Optimizing Disk I/O: Using SSDs, RAID configurations, and file system optimizations.
194. Caching vs Storing: Deciding when to cache data versus storing it in a database.
195. Optimizing Logging: Reducing logging overhead and managing log volumes.
196. Optimizing Concurrent Access: Using locks efficiently and minimizing contention.
197. Profiling Memory Usage: Identifying memory leaks and optimizing object allocations.
198. Optimizing Thread Pool Sizes: Balancing between too few and too many threads.
199. Optimizing Data Structures: Choosing the right data structures for specific use cases.
200. Performance Metrics and KPIs: Defining and tracking key performance indicators for applications.