Java Backend-Entwickler Vorstellungsgespräch

Java Core (20 Punkte)

1. Verständnis der OOP-Prinzipien: Kapselung, Vererbung, Polymorphismus, Abstraktion.
2. Generics in Java: Verwendung von Typparametern, beschränkten Typen und Wildcard-Generics.
3. Multithreading in Java: Erstellen von Threads, Thread-Lebenszyklus und Inter-Thread-Kommunikation.
4. JVM-Speicherverwaltung: Heap, Stack, PermGen/Survivor-Speicherbereiche, Garbage-Collection-Algorithmen.
5. Ausnahmebehandlung: Geprüfte und ungeprüfte Ausnahmen, try-catch-Blöcke, finally und multi-catch.
6. Serialisierung in Java: Serializable-Schnittstelle, benutzerdefinierte Serialisierung mit writeObject und readObject.
7. Java Collections Framework: List, Set, Map, Queue-Schnittstellen und deren Implementierungen.
8. Lambda-Ausdrücke und funktionale Schnittstellen: Verwendung von Prädikaten, Verbrauchern, Lieferanten und Funktionen.
9. Stream-API: Zwischen- und Endoperationen, parallele Streams und Stream-Pipelining.
10. Reflection-API: Zugriff auf Klassen, Methoden und Felder zur Laufzeit, Verarbeitung von Annotationen.
11. Java IO vs NIO: Unterschiede bei der Dateiverarbeitung, kanalbasiertes I/O und nicht blockierendes I/O.
12. Java Date and Time API: Arbeiten mit LocalDate, LocalDateTime und Duration.
13. Java-Netzwerkprogrammierung: Socket-Programmierung, URL-Verbindungen und HTTP-Clients.
14. Java-Sicherheit: Kryptographie, digitale Signaturen und sichere Codierungsmethoden.
15. Java-Module: Verständnis des JPMS (Java Platform Module System) und der Modularität.
16. Java-Enumerationen: Verwendung von Enums, Ordinalwerten und benutzerdefinierten Methoden in Enums.
17. Java-Annotationen: Eingebaute Annotationen, benutzerdefinierte Annotationen und Annotationsverarbeitung.
18. Java Concurrency Utilities: CountDownLatch, CyclicBarrier, Semaphore und Exchanger.
19. Java Memory Leaks: Ursachen, Erkennung und Verhinderungsstrategien.
20. Java Performance Tuning: JVM-Optionen, Profiling-Tools und Speicheroptimierungstechniken.

Spring Ecosystem (20 Punkte)

21. Spring IoC-Container: Dependency Injection, Bean-Lebenszyklus und Scope.

22. Spring Boot Auto-Konfiguration: Wie Spring Boot Beans automatisch konfiguriert.
23. Spring Data JPA: Repository-Muster, CRUD-Operationen und Abfragemethoden.
24. Spring Security: Authentifizierung, Autorisierung und Sicherung von REST-APIs.
25. Spring MVC: Controller-Methoden, Anforderungszuordnung und View-Auflösung.
26. Spring Cloud: Service-Discovery mit Eureka, Lastverteilung mit Ribbon.
27. Spring AOP: Aspektorientierte Programmierung, querverbindende Bedenken und Advice-Typen.
28. Spring Boot Actuator: Überwachungsendpunkte, Gesundheitsprüfungen und Metrikensammlung.
29. Spring Profiles: Umgebungspezifische Konfigurationen und Profilaktivierung.
30. Spring Boot Starter Abhängigkeiten: Verwendung von Startern zur Vereinfachung der Abhängigkeitsverwaltung.
31. Spring Integration: Integration verschiedener Systeme, Nachrichtenübermittlung und Adapter.
32. Spring Batch: Batch-Verarbeitung, Job-Planung und Schrittimplementierungen.
33. Spring Cache: Caching-Strategien, Annotationen und Cache-Manager.
34. Spring WebFlux: Reaktive Programmierung, nicht blockierendes I/O und WebFlux-Frameworks.
35. Spring Cloud Config: Zentralisierte Konfigurationsverwaltung für Microservices.
36. Spring Cloud Gateway: API-Gateway-Muster, Routing und Filterung.
37. Spring Boot Testing: Verwendung von @SpringBootTest, MockMvc und TestRestClient.
38. Spring Data REST: Exposing Repositories als RESTful Services.
39. Spring Cloud Stream: Integration mit Message Brokern wie RabbitMQ und Kafka.
40. Spring Cloud Sleuth: Verteilte Nachverfolgung und Protokollierung in Microservices.

Microservices Architecture (20 Punkte)

41. Service Discovery: Wie Eureka, Consul und Zookeeper funktionieren.
42. API Gateway: Muster, Routing und Sicherheit in API-Gateways.
43. Circuit Breaker: Implementierung von Resilienz mit Hystrix, Resilience4j.
44. Event-Driven Architecture: Event-Sourcing, Message-Broker und Event-Handler.
45. RESTful API Design: HATEOAS, zustandslose Designs und REST-Beschränkungen.
46. GraphQL: Implementierung von GraphQL-APIs, Schema-Definitionen und Resolvern.
47. Microservices-Kommunikation: Synchrone vs. asynchrone Kommunikation.

48. Saga-Muster: Verwaltung verteilter Transaktionen über Dienste.
49. Gesundheitsprüfungen: Implementierung von Liveness- und Readiness-Probes.
50. Contract First Development: Verwendung von Swagger für API-Verträge.
51. API-Versionierung: Strategien zur Versionierung von RESTful-APIs.
52. Rate Limiting: Implementierung von Rate Limits zur Verhinderung von Missbrauch.
53. Circuit Breaker Muster: Implementierung von Fallbacks und Wiederholungen.
54. Microservices Deployment: Verwendung von Docker, Kubernetes und Cloud-Plattformen.
55. Service Mesh: Verständnis von Istio, Linkerd und deren Vorteilen.
56. Event Collaboration: Saga vs. Choreography-Muster.
57. Microservices Security: OAuth2, JWT und API-Gateways.
58. Überwachung und Nachverfolgung: Tools wie Prometheus, Grafana und Jaeger.
59. Microservices Testing: Integrationstests, Vertragstests und End-to-End-Tests.
60. Datenbank pro Service: Datenverwaltung und Konsistenz in Microservices.

Databases and Caching (20 Punkte)

61. SQL Joins: Inner, outer, left, right und cross joins.
62. ACID-Eigenschaften: Atomarität, Konsistenz, Isolation und Dauerhaftigkeit in Transaktionen.
63. NoSQL-Datenbanken: Dokumentenspeicher, Key-Value-Speicher und Graphdatenbanken.
64. Redis Caching: In-Memory-Datenspeicher, Datenstrukturen und Persistenzoptionen.
65. Memcached vs Redis: Vergleich von Caching-Lösungen.
66. Datenbank-Sharding: Horizontale Partitionierung und Lastverteilung.
67. ORM-Frameworks: Hibernate, MyBatis und JPA-Spezifikationen.
68. JDBC-Verbindungspool: DataSource-Implementierungen und Verbindungslebenszyklus.
69. Volltextsuche: Implementierung der Suche in Datenbanken wie Elasticsearch.
70. Zeitreihendatenbanken: InfluxDB, OpenTSDB für zeitbasierte Daten.
71. Transaktionsisolationsstufen: Read uncommitted, read committed, repeatable read, serializable.
72. Indexierungsstrategien: B-tree, Hash-Indexe und zusammengesetzte Indexe.
73. Datenbankreplikation: Master-Slave, Master-Master-Setups.
74. Datenbank-Sicherung und Wiederherstellung: Strategien zum Datenschutz.

75. Datenbankprofiling: Tools wie SQL Profiler, Slow Query Logs.
76. NoSQL-Konsistenzmodelle: Eventuelle Konsistenz, CAP-Theorem.
77. Datenbankmigrationen: Verwendung von Flyway, Liquibase für Schemaänderungen.
78. Caching-Strategien: Cache-aside, read-through, write-through-Muster.
79. Cache-Invalidation: Verwaltung von Cache-Ablauf und -Invalidierung.
80. Datenbank-Verbindungspool: HikariCP, Tomcat JDBC Pool-Konfigurationen.

Concurrency and Multithreading (20 Punkte)

81. Thread-Lebenszyklus: New, runnable, running, blocked, waiting, terminated.
82. Synchronisationsmechanismen: Locks, synchronisierte Blöcke und intrinsische Locks.
83. Reentrant Locks: Vorteile gegenüber synchronisierten Blöcken, Fairness und Timeouts.
84. Executor Framework: ThreadPoolExecutor, ExecutorService und Thread-Pool-Konfigurationen.
85. Callable vs Runnable: Unterschiede und Anwendungsfälle.
86. Java Memory Model: Sichtbarkeit, happens-before-Beziehungen und Speicherkonsistenz.
87. Volatile-Schlüsselwort: Sicherstellung der Sichtbarkeit von Variablenänderungen zwischen Threads.
88. Deadlock-Verhinderung: Vermeidung und Erkennung von Deadlocks.
89. Asynchrone Programmierung: Verwendung von CompletableFuture für nicht blockierende Operationen.
90. ScheduledExecutorService: Planung von Aufgaben mit festen Raten und Verzögerungen.
91. Thread Pools: Fest, zwischengespeichert und geplante Thread-Pools.
92. Lock Striping: Reduzierung von Lock-Konflikten mit gestreiften Locks.
93. Read-Write Locks: Erlauben mehrerer Leser oder eines einzelnen Schreibers.
94. Wait and Notify Mechanisms: Inter-Thread-Kommunikation mit wait/notify.
95. Thread-Unterbrechung: Behandlung von Unterbrechungen und Design von unterbrechbaren Aufgaben.
96. Thread-Safe Klassen: Implementierung von thread-sicheren Singleton-Mustern.
97. Concurrency Utilities: CountDownLatch, CyclicBarrier, Semaphore.
98. Java 8+ Concurrency Features: Parallel Streams, Fork-Join-Framework.
99. Multicore-Programmierung: Herausforderungen und Lösungen für parallele Verarbeitung.
100. Thread Dumps and Analysis: Identifizierung von Problemen mit Thread-Dumps.

Web Servers and Load Balancing (20 Punkte)

101. Apache Tomcat Konfiguration: Einrichten von Connectors, context.xml und server.xml.
102. Nginx als Reverse Proxy: Konfiguration von proxy_pass, Upstream-Servern und Lastverteilung.
103. HAProxy für Hochverfügbarkeit: Einrichten von Failover und Sitzungsbeständigkeit.
104. Web Server Security: SSL/TLS-Konfigurationen, Sicherheitsheader und Firewall-Regeln.
105. Lastverteilungsalgorithmen: Round Robin, Least Connections, IP Hash.
106. Serverseitiges Caching: Verwendung von Varnish, Redis oder In-Memory-Caches.
107. Überwachungstools: Verwendung von Prometheus, Grafana und New Relic für Serverüberwachung.
108. Logging in Production: Zentralisierte Protokollierung mit ELK Stack oder Graylog.
109. Horizontal vs. Vertikal Skalierung: Verständnis von Trade-offs und Anwendungsfällen.
110. Web Server Performance Tuning: Anpassen von Worker-Threads, Verbindungszeitüberschreitungen und Puffern.
111. Reverse Proxy Caching: Konfiguration von Cache-Headern und Ablaufzeiten.
112. Web Server Load Testing: Tools wie Apache JMeter, Gatling für Leistungstests.
113. SSL Offloading: Handhabung von SSL/TLS-Terminierung am Lastverteiler.
114. Web Server Hardening: Sicherheitsbest Practices und Schwachstellenbewertungen.
115. Dynamischer vs. statischer Inhaltsauslieferung: Optimierung von Serverkonfigurationen.
116. Web Server Clustering: Einrichten von Clustern für Hochverfügbarkeit.
117. Web Server Authentifizierung: Implementierung von Basic, Digest und OAuth Authentifizierung.
118. Web Server Logging Formats: Gängige Logformate und Parsing-Tools.
119. Web Server Ressourcenlimits: Konfiguration von Limits für Verbindungen, Anfragen und Bandbreite.
120. Web Server Backup and Recovery: Strategien für Katastrophenwiederherstellung.

CI/CD and DevOps (20 Punkte)

121. Jenkins Pipeline as Code: Schreiben von Jenkinsfiles für CI/CD-Pipelines.
122. Docker Containerisierung: Dockerfile-Erstellung, mehrstufige Builds und Container-Orchestrierung.
123. Kubernetes Orchestration: Deployments, Services, Pods und Skalierungsstrategien.
124. GitOps-Prinzipien: Verwendung von Git für Infrastruktur und Konfigurationsverwaltung.
125. Maven und Gradle Build Tools: Abhängigkeitsverwaltung, Plugins und Build-Lebenszyklus.
126. Unit und Integration Testing: Schreiben von Tests mit JUnit, Mockito und TestNG.

127. Code Coverage Tools: Verwendung von Jacoco zur Messung der Codeabdeckung.
128. Statische Code-Analyse: Tools wie SonarQube für Codequalitätsprüfungen.
129. Infrastructure as Code (IaC): Verwendung von Terraform, CloudFormation für Infrastrukturbereitstellung.
130. Blue/Green Deployments: Minimierung von Ausfallzeiten während der Bereitstellungen.
131. Canary Deployments: Schrittweise Einführung neuer Funktionen.
132. Automatisiertes Testen in CI-Pipelines: Integration von Tests mit Build-Stufen.
133. Umgebungsverwaltung: Verwendung von Ansible, Chef oder Puppet für Konfigurationsverwaltung.
134. CI/CD Best Practices: Kontinuierliche Integration, kontinuierliche Bereitstellung und kontinuierliche Lieferung.
135. Rollback-Strategien: Implementierung automatisierter Rollbacks bei Bereitstellungsfehlern.
136. Sicherheitsprüfungen: Einbeziehung von Sicherheitsprüfungen wie SAST, DAST in Pipelines.
137. CI/CD-Pipelines für Microservices: Verwaltung von Pipelines für mehrere Dienste.
138. Überwachung von CI/CD-Pipelines: Alarmierung bei Pipeline-Fehlern und Leistungsproblemen.
139. DevOps-Tools-Ökosystem: Verständnis von Tools wie Docker, Kubernetes, Jenkins, Ansible.
140. CI/CD für Cloud-native Anwendungen: Bereitstellung von Anwendungen auf Cloud-Plattformen.

Design Patterns and Best Practices (20 Punkte)

141. Singleton-Muster: Implementierung von thread-sicheren Singletons.
142. Factory-Muster: Erstellen von Objekten ohne Angabe der genauen Klasse.
143. Strategy-Muster: Einkapselung von Algorithmen und Umschalten zwischen ihnen.
144. SOLID-Prinzipien: Verständnis und Anwendung von Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion.
145. Dependency Injection: Reduzierung der Kopplung und Erhöhung der Codewartbarkeit.
146. Event Sourcing Pattern: Speichern von Ereignissen zur Rekonstruktion des Anwendungszustands.
147. CQRS Architektur: Trennung von Befehls- und Abfrageverantwortlichkeiten.
148. Designing for Scalability: Verwendung von horizontaler Skalierung, Sharding und Lastverteilung.
149. Code Refactoring Techniques: Extrahieren von Methoden, Umbenennen von Variablen und Vereinfachen von Bedingungen.
150. Clean Code Practices: Schreiben von lesbarem, wartbarem und selbstdokumentierendem Code.

151. Test-Driven Development (TDD): Schreiben von Tests vor der Implementierung.
152. Code Versioning: Verwendung von Git-Verzweigungsstrategien wie GitFlow, Trunk-Based Development.
153. Designing for Maintainability: Verwendung von modularer Gestaltung, Trennung von Bedenken.
154. Anti-Patterns to Avoid: God-Klassen, Spaghetti-Code und enge Kopplung.
155. Designing for Security: Implementierung von least privilege, defense in depth.
156. Designing for Performance: Optimierung von Algorithmen, Reduzierung von I/O-Operationen.
157. Designing for Reliability: Implementierung von Redundanz, Fehlertoleranz und Fehlerbehandlung.
158. Designing for Extensibility: Verwendung von Plugins, Erweiterungen und offenen APIs.
159. Designing for Usability: Sicherstellen, dass APIs intuitiv und gut dokumentiert sind.
160. Designing for Testability: Schreiben von Code, der leicht zu testen und zu mocken ist.

Security (20 Punkte)

161. OAuth2 und JWT: Implementierung von tokenbasierter Authentifizierung.
162. Role-Based Access Control (RBAC): Zuweisung von Rollen und Berechtigungen an Benutzer.
163. Security Headers: Implementierung von Content Security Policy, X-Frame-Options.
164. SQL-Injection Prevention: Verwendung von vorbereiteten Anweisungen und parametrisierten Abfragen.
165. Cross-Site Scripting (XSS) Protection: Eingaben und Ausgaben bereinigen.
166. Verschlüsselung und Entschlüsselung: Verwendung von AES, RSA zum Datenschutz.
167. Secure Coding Practices: Vermeidung häufiger Schwachstellen wie Pufferüberläufe.
168. Implementing Audit Trails: Protokollierung von Benutzeraktionen und Systemereignissen.
169. Handling Sensitive Data: Sicheres Speichern von Passwörtern mit Hashing-Algorithmen.
170. Compliance with Regulations: GDPR, PCI-DSS und Datenschutzgesetze.
171. Implementing Two-Factor Authentication (2FA): Hinzufügen einer zusätzlichen Sicherheitsstufe.
172. Security Testing: Penetrationstests, Schwachstellenbewertungen.
173. Secure Communication Protocols: Implementierung von SSL/TLS zur Datenverschlüsselung.
174. Secure Session Management: Verwaltung von Sitzungstoken und Zeitüberschreitungen.
175. Implementing Web Application Firewalls (WAF): Schutz vor häufigen Angriffen.
176. Security Monitoring and Alerting: Verwendung von Tools wie SIEM zur Bedrohungserkennung.
177. Security Best Practices in Microservices: Sicherung der Kommunikation zwischen Diensten.

178. Implementing CAPTCHA for Bot Protection: Verhinderung automatisierter Angriffe.
179. Security in CI/CD Pipelines: Scannen auf Schwachstellen während der Builds.
180. Implementing Security by Design: Einbeziehung von Sicherheit von Anfang des Entwicklungsprozesses an.

Performance Tuning and Optimization (20 Punkte)

181. Profiling Java Applications: Verwendung von Tools wie JProfiler, VisualVM zur Leistungsanalyse.
182. Garbage Collection Tuning: Anpassen von GC-Parametern zur Leistungsoptimierung.
183. Datenbankabfrageoptimierung: Indexierung, Abfrageumschreibung und Verwendung von Explain-Plänen.
184. Caching-Strategien: Verwendung verteilter Caches, Cache-Invalidierungsmechanismen.
185. Load Testing and Stress Testing: Identifizierung von Leistungsengpässen.
186. Optimizing RESTful APIs: Reduzierung der Antwortzeiten, Minimierung der Datenübertragung.
187. Reducing Network Latency: Verwendung von CDNs, Optimierung von API-Aufrufen.
188. Connection Pool Sizing: Bestimmung optimaler Poolgrößen für Datenbanken und Verbindungen.
189. Monitoring and Alerting Setups: Verwendung von Prometheus, Grafana für Echtzeitüberwachung.
190. Identifying and Resolving Bottlenecks: Profiling von CPU, Speicher und I/O-Nutzung.
191. Optimizing Java Heap Settings: Einstellen geeigneter Heap-Größen für verschiedene Umgebungen.
192. Reducing Garbage Collection Pauses: Verwendung von G1GC, ZGC für latenzarme Anwendungen.
193. Optimizing Disk I/O: Verwendung von SSDs, RAID-Konfigurationen und Dateisystemoptimierungen.
194. Caching vs Storing: Entscheidung, wann Daten zwischengespeichert oder in einer Datenbank gespeichert werden sollen.
195. Optimizing Logging: Reduzierung des Logging-Aufwands und Verwaltung der Logvolumen.
196. Optimizing Concurrent Access: Effiziente Verwendung von Locks und Minimierung von Konflikten.
197. Profiling Memory Usage: Identifizierung von Speicherlecks und Optimierung von Objektzuweisungen.
198. Optimizing Thread Pool Sizes: Ausgleich zwischen zu wenigen und zu vielen Threads.
199. Optimizing Data Structures: Wahl der richtigen Datenstrukturen für spezifische Anwendungsfälle.
200. Performance Metrics and KPIs: Definition und Verfolgung von Schlüsselleistungsindikatoren für Anwendungen.