Top 1000
Java
Interview Question & Answers

Knowledge Powerhouse

www.KnowledgePowerhouse.com

DEDICATION

To our readers!

CONTENTS

instance variable in an Object?

16. Why do we need constructor in Java?

17. Why do we need default constructor in Java classes?

18. What is the value returned by Constructor in Java?

19. Can we inherit a Constructor?

20. Why constructors cannot be final, static, or abstract in Java?

Inheritance

21. What is the purpose of 'this' keyword in java?

22. Explain the concept of Inheritance?

23. Which class in Java is superclass of every other class?

24. Why Java does not support multiple inheritance?

25. In OOPS, what is meant by composition?

26. How aggregation and composition are different concepts?

27. Why there are no pointers in Java?

28. If there are no pointers in Java, then why do we get NullPointerException?

29. What is the purpose of 'super' keyword in java?

30. Is it possible to use this() and super() both in same constructor?

31. What is the meaning of object cloning in Java?

Static

32. In Java, why do we use static variable?

33. Why it is not a good practice to create static variables in Java?

34. What is the purpose of static method in Java?

35. Why do we mark main method as static in Java?

36. In what scenario do we use a static block?

37. Is it possible to execute a program without defining a main() method?

38. What happens when static modifier is not mentioned in the signature of main method?

39. What is the difference between static method and instance method in Java?

Method Overloading and Overriding

40. What is the other name of Method Overloading?

41. How will you implement method overloading in Java?

42. What kinds of argument variations are allowed in Method Overloading?

43. Why it is not possible to do method overloading by changing return type of method in java?

44. Is it allowed to overload main() method in Java?

45. How do we implement method overriding in Java?

46. Are we allowed to override a static method in Java?

47. Why Java does not allow overriding a static method?

48. Is it allowed to override an overloaded method?

49. What is the difference between method overloading and method overriding in Java?

50. Does Java allow virtual functions?

51. What is meant by covariant return type in Java?

Polymorphism

52. What is Runtime Polymorphism?

53. Is it possible to achieve Runtime Polymorphism by data members in Java?

54. Explain the difference between static and dynamic binding?

Abstraction

55. What is Abstraction in Object Oriented programming?

56. How is Abstraction different from Encapsulation?

57. What is an abstract class in Java?

58. Is it allowed to mark a method abstract method without marking the class abstract?

59. Is it allowed to mark a method abstract as well as final?

60. Can we instantiate an abstract class in Java?

61. What is an interface in Java?

62. Is it allowed to mark an interface method as static?

63. Why an Interface cannot be marked as final in Java?

64. What is a marker interface?

65. What can we use instead of Marker interface?

66. How Annotations are better than Marker Interfaces?

67. What is the difference between abstract class and interface in Java?

68. Does Java allow us to use private and protected modifiers for variables in interfaces?

69. How can we cast to an object reference to an interface reference?

Final

70. How can you change the value of a final variable in Java?

71. Can a class be marked final in Java?

72. How can we create a final method in Java?

73. How can we prohibit inheritance in Java?

74. Why Integer class in final in Java?

75. What is a blank final variable in Java?

76. How can we initialize a blank final variable?

77. Is it allowed to declare main method as final?

Package

78. What is the purpose of package in Java?

79. What is java.lang package?

80. Which is the most important class in Java?

81. Is it mandatory to import java.lang package every time?

82. Can you import same package or class twice in your class?

83. What is a static import in Java?

84. What is the difference between import static com.test.Fooclass and import com.test.Fooclass?

Internationalization

85. What is Locale in Java?

86. How will you use a specific Locale in Java?

Serialization

87. What is the serialization?

88. What is the purpose of serialization?

89. What is Deserialization?

90. What is Serialization and Deserialization conceptually?

91. Why do we mark a data member transient?

92. Is it allowed to mark a method as transient?

93. How does marking a field as transient makes it possible to serialize an object?

94. What is Externalizable interface in Java?

95. What is the difference between Serializable and Externalizable interface?

Reflection

96. What is Reflection in Java?

97. What are the uses of Reflection in Java?

98. How can we access private method of a class from outside the class?

99. How can we create an Object dynamically at Runtime in Java?

Garbage Collection

100. What is Garbage Collection in Java?

101. Why Java provides Garbage Collector?

102. What is the purpose of gc() in Java?

103. How does Garbage Collection work in Java?

104. When does an object become eligible for Garbage Collection in Java?

105. Why do we use finalize() method in Java?

106. What are the different types of References in Java?

107. How can we reference an unreferenced object again?

108. What kind of process is the Garbage collector thread?

109. What is the purpose of the Runtime class?

110. How can we invoke an external process in Java?

111. What are the uses of Runtime class?

Inner Classes

112. What is a Nested class?

113. How many types of Nested classes are in Java?

114. Why do we use Nested Classes?

115. What is the difference between a Nested class and an Inner class in Java?

116. What is a Nested interface?

117. How can we access the non-final local variable, inside a Local Inner class?

118. Can an Interface be defined in a Class?

119. Do we have to explicitly mark a Nested Interface public static?

120. Why do we use Static Nested interface in Java?

String

121. What is the meaning of Immutable in the context of String class in Java?

122. Why a String object is considered immutable in java?

123. How many objects does following code create?

124. How many ways are there in Java to create a String object?

125. How many objects does following code create?

126. What is String interning?

127. Why Java uses String literal concept?

128. What is the basic difference between a String and StringBuffer object?

129. How will you create an immutable class in Java?

130. What is the use of toString() method in java ?

131. Arrange the three classes String, StringBuffer and StringBuilder in the order of efficiency for String processing operations?

Exception Handling

132. What is Exception Handling in Java?

133. In Java, what are the differences between a Checked and Unchecked?

134. What is the base class for Error and Exception classes in Java?

135. What is a finally block in Java?

136. What is the use of finally block in Java?

137. Can we create a finally block without creating a catch block?

138. Do we have to always put a catch block after a try block?

139. In what scenarios, a finally block will not be executed?

140. Can we re-throw an Exception in Java?

141. What is the difference between throw and throws in Java?

142. What is the concept of Exception Propagation?

143. When we override a method in a Child class, can we throw an additional Exception that is not thrown by the Parent class method?

Multi-threading

144. How Multi-threading works in Java?

145. What are the advantages of Multithreading?

146. What are the disadvantages of Multithreading?

147. What is a Thread in Java?

148. What is a Thread's priority and how it is used in scheduling?

149. What are the differences between Pre-emptive Scheduling Scheduler and Time Slicing Scheduler?

150. Is it possible to call run() method instead of start() on a thread in Java?

151. How will you make a user thread into daemon thread if it has already started?

152. Can we start a thread two times in Java?

153. In what scenarios can we interrupt a thread?

154. In Java, is it possible to lock an object for exclusive use by a thread?

155. How notify() method is different from notifyAll() method?

Collections

156. What are the differences between the two data structures: a Vector and an ArrayList?

157. What are the differences between Collection and Collections in Java?

158. In which scenario, LinkedList is better than ArrayList in Java?

159. What are the differences between a List and Set collection in Java?

160. What are the differences between a HashSet and TreeSet collection in Java?

161. In Java, how will you decide when to use a List, Set or a Map

collection?

162. What are the differences between a HashMap and a Hashtable in Java?

163. What are the differences between a HashMap and a TreeMap?

164. What are the differences between Comparable and Comparator?

165. In Java, what is the purpose of Properties file?

166. What is the reason for overriding equals() method?

167. How does hashCode() method work in Java?

168. Is it a good idea to use Generics in collections?

Mixed Questions

169. What are Wrapper classes in Java?

170. What is the purpose of native method in Java?

171. What is System class?

172. What is System, out and println in System.out.println method call?

173. What is the other name of Shallow Copy in Java?

174. What is the difference between Shallow Copy and Deep Copy in Java?

175. What is a Singleton class?

176. What is the difference between Singleton class and Static class?

Java Collection

177. What is the difference between Collection and Collections Framework in Java?

178. What are the main benefits of Collections Framework in Java?

179. What is the root interface of Collection hierarchy in Java?

180. What are the main differences between Collection and Collections?

181. What are the Thread-safe classes in Java Collections framework?

182. How will you efficiently remove elements while iterating a Collection?

183. How will you convert a List into an array of integers like- int[]?

184. How will you convert an array of primitive integers int[] to a List collection?

185. How will you run a filter on a Collection?

186. How will you convert a List to a Set?

187. How will you remove duplicate elements from an ArrayList?

188. How can you maintain a Collection with elements in Sorted order?

189. What is the difference between Collections.emptyList() and creating new instance of Collection?

190. How will you copy elements from a Source List to another list?

191. What are the Java Collection classes that implement List interface?

192. What are the Java Collection classes that implement Set interface?

193. What is the difference between an Iterator and ListIterator in Java?

194. What is the difference between Iterator and Enumeration?

195. What is the difference between an ArrayList and a LinkedList data structure?

196. What is the difference between a Set and a Map in Java?

197. What is the use of a Dictionary class?

198. What is the default size of load factor in a HashMap collection in Java?

199. What is the significance of load factor in a HashMap in Java?

200. What are the major differences between a HashSet and a HashMap?

201. What are the similarities between a HashSet and a HashMap in Java?

202. What is the reason for overriding equals() method?

203. How can we synchronize the elements of a List, a Set or a Map?

204. What is Hash Collision? How Java handles hash-collision in HashMap?

205. What are the Hash Collision resolution techniques?

206. What is the difference between Queue and Stack data structures?

207. What is an Iterator in Java?

208. What is the difference between Iterator and Enumeration in Java?

209. What is the design pattern used in the implementation of Enumeration in Java?

210. Which methods do we need to override to use an object as key in a HashMap?

211. How will you reverse a List in Java?

212. How will you convert an array of String objects into a List?

213. What is the difference between peek(), poll() and remove() methods of Queue interface in java?

214. What is the difference between Array and ArrayList in Java?

215. How will you insert, delete and retrieve elements from a HashMap collection in Java?

216. What are the main differences between HashMap and ConcurrentHashMap in Java?

217. What is the increasing order of performance for following collection classes in Java?

218. Why does Map interface not extend Collection interface in Java?

219. What are the different ways to iterate elements of a list in Java?

220. What is CopyOnWriteArrayList? How it is different from ArrayList in Java?

221. How remove() method is implemented in a HashMap?

222. What is BlockingQueue in Java Collections?

223. How is TreeMap class implemented in Java?

224. What is the difference between Fail-fast and Fail-safe iterator in Java?

225. How does ConcurrentHashMap work in Java?

226. What is the importance of hashCode() and equals() methods?

227. What is the contract of hashCode() and equals() methods in Java?

228. What is an EnumSet in Java?

229. What are the main Concurrent Collection classes in Java?

230. How will you convert a Collection to SynchronizedCollection in Java?

231. How IdentityHashMap is different from a regular Map in Java?

232. What is the main use of IdentityHashMap?

233. How can we improve the performance of IdentityHashMap?

234. Is IdentityHashMap thread-safe?

235. What is a WeakHashMap in Java?

236. How can you make a Collection class read Only in Java?

237.When is UnsupportedOperationException thrown in Java?

238.Let say there is a Customer class. We add objects of Customer class to an ArrayList. How can we sort the Customer objects in ArrayList by using customer firstName attribute of Customer class?

239.What is the difference between Synchronized Collection and Concurrent Collection?

240.What is the scenario to use ConcurrentHashMap in Java?

241. How will you create an empty Map in Java?

242.What is the difference between remove() method of Collection and remove() method of Iterator?

243.Between an Array and ArrayList, which one is the preferred collection for storing objects?

244.Is it possible to replace Hashtable with ConcurrentHashMap in Java?

245.How CopyOnWriteArrayList class is different from ArrayList and Vector classes?

246.Why ListIterator has add() method but Iterator does not have?

247.Why do we sometime get ConcurrentModificationException during iteration?

248.How will you convert a Map to a List in Java?

249.How can we create a Map with reverse view and lookup in Java?

250.How will you create a shallow copy of a Map?

251. Why we cannot create a generic array in Java?

252.What is a PriorityQueue in Java?

253.What are the important points to remember while using Java Collections Framework?

254.How can we pass a Collection as an argument to a method and ensure that method will not be able to modify it?

255.Can you explain how HashMap works in Java?

256.Can you explain how HashSet is implemented in Java?

257.What is a NavigableMap in Java?

258.What is the difference between descendingKeySet() and descendingMap() methods of NavigableMap?

259.What is the advantage of NavigableMap over Map?

260.What is the difference between headMap(), tailMap() and subMap() methods of NavigableMap?

261. How will you sort objects by Natural order in a Java List?

262.How can we get a Stream from a List in Java?

263.Can we get a Map from a Stream in Java?

264.What are the popular implementations of Deque in Java?

Advanced Multi-threading

265.What is a Thread in Java?

266.What is the priority of a Thread and how it is used in scheduling?

267.What is the default priority of a thread in Java?

268.What are the three different priorities that can be set on a Thread in Java?

269.What is the purpose of join() method in Thread class?

270.What is the fundamental difference between wait() and sleep() methods?

271. Is it possible to call run() method instead of start() on a thread in

Java?

272.What is a daemon thread in Java?

273.How can we make a regular thread Daemon thread in Java?

274.How will you make a user thread into daemon thread if it has already started?

275.Can we start a thread two times in Java?

276.What is a Shutdown hook in Java?

277.What is synchronization in Java?

278.What is the purpose of Synchronized block in Java?

279.What is static synchronization?

280.What is a Deadlock situation?

281. What is the meaning of concurrency?

282.What is the main difference between process and thread?

283.What is a process and thread in the context of Java?

284.What is a Scheduler?

285.What is the minimum number of Threads in a Java program?

286.What are the properties of a Java thread?

287.What are the different states of a Thread in Java?

288.How will you set the priority of a thread in Java?

289.What is the purpose of Thread Groups in Java?

290.Why we should not stop a thread by calling its stop() method?

291. How will you create a Thread in Java?

292.How can we stop a thread in the middle of execution in Java?

293.How do you access the current thread in a Java program?

294.What is Busy waiting in Multi-threading?

295.How can we prevent busy waiting in Java?

296.Can we use Thread.sleep() method for real-time processing in Java?

297.Can we wake up a thread that has been put to sleep by using Thread.sleep() method?

298.What are the two ways to check if a Thread has been interrupted?

299.How can we make sure that Parent thread waits for termination of Child thread?

300.How will you handle InterruptedException in Java?

301. Which intrinsic lock is acquired by a synchronized method in Java?

302.Can we mark a constructor as synchronized in Java?

303.Can we use primitive values for intrinsic locks?

304.Do we have re-entrant property in intrinsic locks?

305.What is an atomic operation?

306.Can we consider the statement i++ as an atomic operation in Java?

307.What are the Atomic operations in Java?

308.Can you check if following code is thread-safe?

309.What are the minimum requirements for a Deadlock situation in a program?

310. How can we prevent a Deadlock?

311. How can we detect a Deadlock situation?

312. What is a Livelock?

313. What is Thread starvation?

314. How can a synchronized block cause Thread starvation in Java?

315. What is a Race condition?

316. What is a Fair lock in multi-threading?

317. Which two methods of Object class can be used to implement a Producer Consumer scenario?

318. How JVM determines which thread should wake up on notify()?

319. Check if following code is thread-safe for retrieving an integer value from a Queue?

320. How can we check if a thread has a monitor lock on a given object?

321. What is the use of yield() method in Thread class?

322. What is an important point to consider while passing an object from one thread to another thread?

323. What are the rules for creating Immutable Objects?

324. What is the use of ThreadLocal class?

325. What are the scenarios suitable for using ThreadLocal class?

326. How will you improve the performance of an application by multi-threading?

327. What is scalability in a Software program?

328. How will you calculate the maximum speed up of an application by using multiple processors?

329. What is Lock contention in multi-threading?

330. What are the techniques to reduce Lock contention?

331. What technique can be used in following code to reduce Lock contention?

332. What is Lock splitting technique?

333. Which technique is used in ReadWriteLock class for reducing Lock contention?

334. What is Lock striping?

335. What is a CAS operation?

336. Which Java classes use CAS operation?

337. Is it always possible to improve performance by object pooling in a multi-threading application?

338. How can techniques used for performance improvement in a single thread application may degrade the performance in a multi-threading application?

339. What is the relation between Executor and ExecutorService interface?

340. What will happen on calling submit() method of an ExecutorService instance whose queue is already full?

341. What is a ScheduledExecutorService?

342. How will you create a Thread pool in Java?

343. What is the main difference between Runnable and Callable interface?

344. What are the uses of Future interface in Java?

345. What is the difference in concurrency in HashMap and in Hashtable?

346. How will you create synchronized instance of List or Map Collection?

347. What is a Semaphore in Java?

348.What is a CountDownLatch in Java?

349.What is the difference between CountDownLatch and CyclicBarrier?

350.What are the scenarios suitable for using Fork/Join framework?

351.What is the difference between RecursiveTask and RecursiveAction class?

352.In Java 8, can we process stream operations with a Thread pool?

353.What are the scenarios to use parallel stream in Java 8?

354.How Stack and Heap work in Java multi-threading environment?

355.How can we take Thread dump in Java?

356.Which parameter can be used to control stack size of a thread in Java?

357.There are two threads T1 and T2? How will you ensure that these threads run in sequence T1, T2 in Java?

Java 8

358.What are the new features released in Java 8?

359.What are the main benefits of new features introduced in Java 8?

360.What is a Lambda expression in Java 8?

361.What are the three main parts of a Lambda expression in Java?

362.What is the data type of a Lambda expression?

363.What is the meaning of following lambda expression?

364.Why did Oracle release a new version of Java like Java 8?

365.What are the advantages of a lambda expression?

366.What is a Functional interface in Java 8?

367.What is a Single Abstract Method (SAM) interface in Java 8?

368.How can we define a Functional interface in Java 8?

369.Why do we need Functional interface in Java?

370.Is it mandatory to use @FunctionalInterface annotation to define a Functional interface in Java 8?

371.What are the differences between Collection and Stream API in Java 8?

372.What are the main uses of Stream API in Java 8?

373.What are the differences between Intermediate and Terminal Operations in Java 8 Streams?

374.What is a Spliterator in Java 8?

375.What are the differences between Iterator and Spliterator in Java 8?

376.What is Type Inference in Java 8?

377.Does Java 7 support Type Inference?

378.How does Internal Iteration work in Java 8?

379.What are the main differences between Internal and External Iterator?

380.What are the main advantages of Internal Iterator over External Iterator in Java 8?

381.What are the applications in which we should use Internal Iteration?

382.What is the main disadvantage of Internal Iteration over External Iteration?

383.Can we provide implementation of a method in a Java Interface?

384.What is a Default Method in an Interface?

385.Why do we need Default method in a Java 8 Interface?

386.What is the purpose of a Static method in an Interface in Java 8?

387.What are the core ideas behind the Date/Time API of Java 8?

388.What are the advantages of new Date and Time API in Java 8 over old Date API?

389.What are the main differences between legacy Date/Time API in Java and Date/Time API of Java 8?

390.How can we get duration between two dates or time in Java 8?

391. What is the new method family introduced in Java 8 for processing of Arrays on multi core machines?

392.How does Java 8 solve Diamond problem of Multiple Inheritance?

393.What are the differences between Predicate, Supplier and Consumer in Java 8?

394.Is it possible to have default method definition in an interface without marking it with default keyword?

395.Can we create a class that implements two Interfaces with default methods of same name and signature?

396.How Java 8 supports Multiple Inheritance?

397.In case we create a class that extends a base class and implements an interface. If both base class and interface have a default method with same name and arguments, then which definition will be picked by JVM?

398.If we create same method and define it in a class , in its parent class and in an interface implemented by the class, then definition will be invoked if we access it using the reference of Interface and the object of class?

399.Can we access a static method of an interface by using reference of

the interface?

400.How can you get the name of Parameter in Java by using reflection?

401. What is Optional in Java 8?

402.What are the uses of Optional?

403.Which method in Optional provides the fallback mechanism in case of null value?

404.How can we get current time by using Date/Time API of Java 8?

405.Is it possible to define a static method in an Interface?

406.How can we analyze the dependencies in Java classes and packages?

407.What are the new JVM arguments introduced by Java 8?

408.What are the popular annotations introduced in Java 8?

409.What is a StringJoiner in Java 8?

410. What is the type of a Lambda expression in Java 8?

411. What is the target type of a lambda expression ?

412. What are the main differences between an interface with default method and an abstract class in Java 8?

Java Tricky Questions

413. Is there any difference between a = a + b and a += b expressions?

414.What does the expression 1.0 / 0.0 return? Will there be any compilation error?

415. Can we use multiple main methods in multiple classes?

416. Does Java allow you to override a private or static method?

417. What happens when you put a key object in a HashMap that is already present?

418. How can you make sure that N threads can access N resources without deadlock?

419. How can you determine if JVM is 32-bit or 64-bit from Java Program?

420. What is the right data type to represent Money (like Dollar/Pound) in Java?

421. How can you do multiple inheritances in Java?

422. Is ++ operation thread-safe in Java?

423. How can you access a non-static variable from the static context?

424. Let say there is a method that throws NullPointerException in the superclass. Can we override it with a method that throws RuntimeException?

425. How can you mark an array volatile in Java?

426. What is a thread local variable in Java?

427. What is the difference between sleep() and wait() methods in Java?

428. Can you create an Immutable object that contains a mutable object?

429. How can you convert an Array of bytes to String?

430. What is difference between CyclicBarrier and CountDownLatch class?

431. What is the difference between StringBuffer and StringBuilder?

432. Which class contains clone method? Cloneable or Object class?

433. How will you take thread dump in Java?

434. Can you cast an int variable into a byte variable? What happens if the value of int is larger than byte?

435. In Java, can we store a double value in a long variable without explicit

casting?

436.What will this return 5*0.1 == 0.5? true or false?

437.Out of an int and Integer, which one takes more memory?

438.Can we use String in the switch case statement in Java?

439.Can we use multiple main methods in same class?

440.When creating an abstract class, is it a good idea to call abstract methods inside its constructor?

441.How can you do constructor chaining in Java?

442.How can we find the memory usage of JVM from Java code?

443.What is the difference between x == y and x.equals(y) expressions in Java?

444. How can you guarantee that the garbage collection takes place?

445.What is the relation between x.hashCode() method and x.equals(y) method of Object class?

446.What is a compile time constant in Java?

447.Explain the difference between fail-fast and fail-safe iterators?

448. You have a character array and a String. Which one is more secure to store sensitive data (like password, date of birth, etc.)?

449.Why do you use volatile keyword in Java?

450.What is the difference between poll() and remove() methods of Queue in Java?

451.Can you catch an exception thrown by another thread in Java?

452.How do you decide which type of Inner Class – Static or Non-Static to use in Java?

453.What are the different types of Classloaders in Java?

454.What are the situations in which you choose HashSet or TreeSet?

455.What is the use of method references in Java?

456.Do you think Java Enums are more powerful than integer constants?

457.Why do we use static initializers in Java?

458.Your client is complaining that your code is throwing NoClassDefFoundError or NoSuchMethodError, even though you are able to compile your code without error and method exists in your code. What could be the reason behind this?

459.How can you check if a String is a number by using regular expression?

460.What is the difference between the expressions String s = "Temporary" and String s = new String("Temporary ")? Which one is better and more efficient?

461. In Java, can two equal objects have the different hash code?

462.How can we print an Array in Java?

463.Is it ok to use random numbers in the implementation of hashcode() method in Java?

464.Between two types of dependency injections, constructor injection and setter dependency injection, which one is better?

465.What is the difference between DOM and SAX parser in Java?

466.Between Enumeration and Iterator, which one has better performance in Java?

467.What is the difference between pass by reference and pass by value?

468.What are the different ways to sort a collection in Java?

469.Why Collection interface doesn’t extend Cloneable and Serializable interfaces?

470.What is the difference between a process and a thread in Java?

471. What are the benefits of using an unordered array over an ordered array?

472.Between HashSet and TreeSet collections in Java, which one is better?

473.When does JVM call the finalize() method?

474.When would you use Serial Garabage collector or Throughput Garbage collector in Java?

475.In Java, if you set an object reference to null, will the Garbage Collector immediately free the memory held by that object?

476.How can you make an Object eligible for Garbage collection in Java?

477.When do you use Exception or Error in Java? What is the difference between these two?

478.What is the advantage of PreparedStatement over Statement class in Java?

479.In Java, what is the difference between throw and throws keywords?

480.What happens to the Exception object after the exception handling is done?

481. How do you find which client machine is sending request to your servlet in Java?

482.What is the difference between a Cookie and a Session object in Java?

483.Which protocol does Browser and Servlet use to communicate with each other?

484. What is HTTP Tunneling?

485.Why do we use JSP instead of Servlet in Java?

486.Is empty '.java' file name a valid source file name in Java?

487.How do you implement Servlet Chaining in Java?

488.Can you instantiate this class?

489.Why Java does not support operator overloading?

490.Why String class is Immutable or Final in Java?

491. What is the difference between sendRedirect and forward methods?

492.How do you fix your Serializable class, if it contains a member that is not serializable?

493.What is the use of run time polymorphism in Java?

494.What are the rules of method overloading and method overriding in Java?

495.What is the difference between a class and an object in Java?

496.Can we create an abstract class that extends another abstract class?

497.Why do you use Upcasting or Downcasting in Java ?

498.What is the reason to organize classes and interfaces in a package in Java?

499.What is information hiding in Java?

500.Why does Java provide default constructor?

501. What is the difference between super and this keywords in Java?

502.What is the advantage of using Unicode characters in Java?

503.Can you override an overloaded method in Java?

504.How can we change the heap size of a JVM?

505.Why should you define a default constructor in Java?

506.How will you make an Object Immutable in Java?

507.How can you prevent SQL Injection in Java Code?

508.Which two methods should be always implemented by HashMap key Object?

509.Why an Object used as Key in HashMap should be Immutable?

510.How can we share an object between multiple threads?

511. How can you determine if your program has a deadlock?

JSP

512.What are the implicit objects in JSP?

513.How will you extend JSP code?

514.How will you handle runtime exceptions in JSP?

515.How will you prevent multiple submits of a page that come by clicking refresh button multiple times?

516.How will you implement a thread safe JSP page?

517.How will you include a static file in a JSP page?

518.What are the lifecycle methods of a JSP?

519.What are the advantages of using JSP in web architecture?

520.What is the advantage of JSP over Javascript?

521.What is the Lifecycle of JSP?

522.What is a JSP expression?

523.What are the different types of directive tags in JSP?

524.What is session attribute in JSP?

525.What are the different scopes of a JSP object?

526.What is pageContext in JSP?

527.What is the use of jsp:useBean in JSP?

528.What is difference between include Directive and include Action of JSP?

529.How will you use other Java files of your application in JSP code?

530.How will you use an existing class and extend it to use in the JSP?

531.Why _jspService method starts with _ symbol in JSP?

532.Why do we use tag library in JSP?

533.What is the different type of tag library groups in JSTL?

534.How will you pass information from one JSP to another JSP?

535.How will you call a stored procedure from JSP?

536.Can we override _jspService() method in JSP?

537.What is a directive in JSP?

538.How will you implement Session tracking in JSP?

539.How do you debug code in JSP?

540.How will you implement error page in JSP?

541.How will you send XML data from a JSP?

542.What happens when we request for a JSP page from web browser?

543.How will you implement Auto Refresh of page in JSP?

544.What are the important status codes in HTTP?

545.What is the meaning of Accept attribute in HTTP header?

546.What is the difference between Expression and Scriptlet in JSP?

547.How will you delete a Cookie in JSP?

548.How will you use a Cookie in JSP?

549.What is the main difference between a Session and Cookie in JSP?

550.How will you prevent creation of session in JSP?

551. What is an output comment in JSP?

552.How will you prevent caching of HTML output by web browser in JSP?

553.How will you redirect request to another page in browser in JSP code?

554.What is the difference between sendRedirect and forward in a JSP?

555.What is the use of config implicit object in JSP?

556.What is the difference between init-param and context-param?

557.What is the purpose of RequestDispatcher?

558.How can be read data from a Form in a JSP?

559.What is a filter in JSP?

560.How can you upload a large file in JSP?

561. In which scenario, Container initializes multiple JSP/Servlet objects?

Java Design Patterns

562.When will you use Strategy Design Pattern in Java?

563.What is Observer design pattern?

564.What are the examples of Observer design pattern in JDK?

565.How Strategy design pattern is different from State design pattern in Java?

566.Can you explain Decorator design pattern with an example in Java?

567.What is a good scenario for using Composite design Pattern in Java?

568.Have you used Singleton design pattern in your Java project?

569.What are the main uses of Singleton design pattern in Java project?

570.Why java.lang.Runtime is a Singleton in Java?

571.What is the way to implement a thread-safe Singleton design pattern in Java?

572.What are the examples of Singleton design pattern in JDK?

573.What is Template Method design pattern in Java?

574.What are the examples of Template method design pattern in JDK?

575.Can you tell some examples of Factory Method design pattern implementation in Java?

576.What is the benefit we get by using static factory method to create object?

577.What are the examples of Builder design pattern in JDK?

578.What are the examples of Abstract Factory design pattern in JDK?

579.What are the examples of Decorator design pattern in JDK?

580.What are the examples of Proxy design pattern in JDK?

581.What are the examples of Chain of Responsibility design pattern in JDK?

582.What are the main uses of Command design pattern?

583.What are the examples of Command design pattern in JDK?

584.What are the examples of Interpreter design pattern in JDK?

585.What are the examples of Mediator design pattern in JDK?

586.What are the examples of Strategy design pattern in JDK?

587.What are the examples of Visitor design pattern in JDK?

588.How Decorator design pattern is different from Proxy pattern?

589.What are the different scenarios to use Setter and Constructor based injection in Dependency Injection (DI) design pattern?

590.What are the different scenarios for using Proxy design pattern?

591. What is the main difference between Adapter and Proxy design pattern?

592.When will you use Adapter design pattern in Java?

593.What are the examples of Adapter design pattern in JDK?

594.What is the difference between Factory and Abstract Factory design pattern?

595.What is Open/closed design principle in Software engineering?

596.What is SOLID design principle?

597.What is Builder design pattern?

598.What are the different categories of Design Patterns used in Object Oriented Design?

599.What is the design pattern suitable to access elements of a Collection?

600.How can we implement Producer Consumer design pattern in Java?

601. What design pattern is suitable to add new features to an existing object?

602.Which design pattern can be used when to decouple abstraction from the implementation?

603. Which is the design pattern used in Android applications?

604. How can we prevent users from creating more than one instance of singleton object by using clone() method?

605. What is the use of Interceptor design pattern?

606. What are the Architectural patterns that you have used?

607. What are the popular uses of Façade design pattern?

608. What is the difference between Builder design pattern and Factory design pattern?

609. What is Memento design pattern?

610. What is an AntiPattern?

611. What is a Data Access Object (DAO) design pattern?

Spring Questions

612. What is Spring framework?

613. What are the benefits of Spring framework in software development?

614. What are the modules in Core Container of Spring framework?

615. What are the modules in Data Access/Integration layer of Spring framework?

616. What are the modules in Web layer of Spring framework?

617. What is the main use of Core Container module in Spring framework?

618. What kind of testing can be done in Spring Test Module?

619. What is the use of BeanFactory in Spring framework?

620. Which is the most popular implementation of BeanFactory in Spring?

621. What is XMLBeanFactory in Spring framework?

622.What are the uses of AOP module in Spring framework?

623.What are the benefits of JDBC abstraction layer module in Spring framework?

624.How does Spring support Object Relational Mapping (ORM) integration?

625.How does Web module work in Spring framework?

626.What are the main uses of Spring MVC module?

627.What is the purpose of Spring configuration file?

628.What is the purpose of Spring IoC container?

629.What is the main benefit of Inversion of Control (IOC) principle?

630.Does IOC containers support Eager Instantiation or Lazy loading of beans?

631.What are the benefits of ApplicationContext in Spring?

632.How will you implement ApplicationContext in Spring framework?

633.Explain the difference between ApplicationContext and BeanFactory in Spring?

634.Between ApplicationContext and BeanFactory which one is preferable to use in Spring?

635.What are the main components of a typical Spring based application?

636.Explain Dependency Injection (DI) concept in Spring framework?

637.What are the different roles in Dependency Injection (DI)?

638.Spring framework provides what kinds of Dependency Injection mechanism?

639.In Spring framework, which Dependency Injection is better? Constructor-based DI or Setter-based DI?

640.What are the advantages of Dependency Injection (DI)?

641. What are the disadvantages of Dependency Injection (DI)?

642.What is a Spring Bean?

643.What does the definition of a Spring Bean contain?

644.What are the different ways to provide configuration metadata to a Spring Container?

645.What are the different scopes of a Bean supported by Spring?

646.How will you define the scope of a bean in Spring?

647.Is it safe to assume that a Singleton bean is thread safe in Spring Framework?

648.What are the design-patterns used in Spring framework?

649.What is the lifecycle of a Bean in Spring framework?

650.What are the two main groups of methods in a Bean's lifecycle?

651. Can we override main lifecycle methods of a Bean in Spring?

652.What are Inner beans in Spring?

653.How can we inject a Java Collection in Spring framework?

654.What is Bean wiring in Spring?

655.What is Autowiring in Spring?

656.What are the different modes of Autowiring supported by Spring?

657.What are the cases in which Autowiring may not work in Spring framework?

658.Is it allowed to inject null or empty String values in Spring?

659.What is a Java-based Configuration in Spring?

660.What is the purpose of @Configuration annotation?

661.What is the difference between Full @Configuration and 'lite' @Beans mode?

662.In Spring framework, what is Annotation-based container configuration?

663.How will you switch on Annotation based wiring in Spring?

664.What is @Autowired annotation?

665.What is @Required annotation?

666.What are the two ways to enable RequiredAnnotationBeanPostProcessor in Spring?

667.What is @Qualifier annotation in Spring?

668.How Spring framework makes JDBC coding easier for developers?

669.What is the purpose of JdbcTemplate?

670.What are the benefits of using Spring DAO?

671.What are the different ways to use Hibernate in Spring?

672.What types of Object Relational Mapping (ORM) are supported by Spring?

673.How will you integrate Spring and Hibernate by using HibernateDaoSupport?

674.What are the different types of the Transaction Management supported by Spring framework?

675.What are the benefits provided by Spring Framework's Transaction Management?

676.Given a choice between declarative and programmatic Transaction Management, which method will you choose?

677.What is Aspect Oriented Programming (AOP)

678.What is an Aspect in Spring?

679.In Spring AOP, what is the main difference between a Concern and a Cross cutting concern?

680.What is a Joinpoint in Spring AOP?

681.What is an Advice in Spring AOP?

682.What are the different types of Advice in Spring AOP?

683.What is a Pointcut in Spring AOP?

684.What is an Introduction in Spring AOP?

685.What is a Target object in Spring AOP?

686.What is a Proxy in Spring AOP?

687.What are the different types of AutoProxy creators in Spring?

688.What is Weaving in Spring AOP?

689.In Spring AOP, Weaving is done at compile time or run time?

690.What is XML Schema-based Aspect implementation?

691.What is Annotation-based aspect implementation in Spring AOP?

692.How does Spring MVC framework work?

693.What is DispatcherServlet?

694.Can we have more than one DispatcherServlet in Spring MVC?

695.What is WebApplicationContext in Spring MVC?

696.What is Controller in Spring MVC framework?

697.What is @RequestMapping annotation in Spring?

698.What are the main features of Spring MVC?

699.What is the difference between a Singleton and Prototype bean in Spring?

700.How will you decide which scope- Prototype or Singleton to use for a bean in Spring?

701.What is the difference between Setter and Constructor based Dependency Injection (DI) in Spring framework?

702.What are the drawbacks of Setter based Dependency Injection (DI) in Spring?

703.What are the differences between Dependency Injection (DI) and Factory Pattern?

704.In Spring framework, what is the difference between FileSystemResource and ClassPathResource?

705.Name some popular Spring framework annotations that you use in your project?

706.How can you upload a file in Spring MVC Application?

707.What are the different types of events provided by Spring framework?

708.What is the difference between DispatcherServlet and ContextLoaderListener in Spring?

709.How will you handle exceptions in Spring MVC Framework?

710.What are the best practices of Spring Framework?

711. What is Spring Boot?

Hibernate

712.What is Hibernate framework?

713. What is an Object Relational Mapping (ORM)?

714. What is the purpose of Configuration Interface in Hibernate?

715. What is Object Relational Impedance Mismatch?

716. What are the main problems of Object Relational Impedance Mismatch?

717. What are the key characteristics of Hibernate?

718. Can you tell us about the core interfaces of Hibernate framework?

719. How will you map the columns of a DB table to the properties of a Java class in Hibernate?

720. Does Hibernate make it mandatory for a mapping file to have .hbm.xml extension?

721. What are the steps for creating a SessionFactory in Hibernate?

722. Why do we use POJO in Hibernate?

723. What is Hibernate Query Language (HQL)?

724. How will you call a stored procedure in Hibernate?

725. What is Criteria API in Hibernate?

726. Why do we use HibernateTemplate?

727. How can you see SQL code generated by Hibernate on console?

728. What are the different types of collections supported by Hibernate?

729. What is the difference between session.save() and session.saveOrUpdate() methods in Hibernate?

730. What are the advantages of Hibernate framework over JDBC?

731. How can we get statistics of a SessionFactory in Hibernate?

732.What is the Transient state of an object in Hibernate?

733.What is the Detached state of an object in Hibernate?

734.What is the use of Dirty Checking in Hibernate?

735.What is the purpose of Callback interface in Hibernate?

736.What are the different ORM levels in Hibernate?

737.What are the different ways to configure a Hibernate application?

738.What is Query Cache in Hibernate?

739.What are the different types of Association mappings supported by Hibernate?

740.What are the different types of Unidirectional Association mappings in Hibernate?

741. What is Unit of Work design pattern?

742.In Hibernate, how can an object go in Detached state?

743.How will you order the results returned by a Criteria in Hibernate?

744.How does Example criterion work in Hibernate?

745.How does Transaction management work in Hibernate?

746.How can we mark an entity/collection as immutable in Hibernate?

747.What are the different options to retrieve an object from database in Hibernate?

748.How can we auto-generate primary key in Hibernate?

749.How will you re-attach an object in Detached state in Hibernate?

750.What is the first level of cache in Hibernate?

751. What are the different second level caches available in Hibernate?

752.Which is the default transaction factory in Hibernate?

753.What are the options to disable second level cache in Hibernate?

754.What are the different fetching strategies in Hibernate?

755.What is the difference between Immediate fetching and Lazy collection fetching?

756.What is 'Extra lazy fetching' in Hibernate?

757.How can we check is a collection is initialized or not under Lazy Initialization strategy?

758.What are the different strategies for cache mapping in Hibernate?

759.What is the difference between a Set and a Bag in Hibernate?

760.How can we monitor the performance of Hibernate in an application?

761.How can we check if an Object is in Persistent, Detached or Transient state in Hibernate?

762.What is 'the inverse side of association' in a mapping?

763.What is ORM metadata?

764.What is the difference between load() and get() method in Hibernate?

765.When should we use get() method or load() method in Hibernate?

766.What is a derived property in Hibernate?

767.How can we use Named Query in Hibernate?

768.What are the two locking strategies in Hibernate?

769.What is the use of version number in Hibernate?

770.What is the use of session.lock() method in Hibernate?

771.What inheritance mapping strategies are supported by Hibernate?

Maven

772. What is Maven?

773. What are the main features of Maven?

774. What areas of a Project can you manage by using Maven?

775. What are the main advantages of Maven?

776. Why do we say "Maven uses convention over configuration"?

777. What are the responsibilities of a Build tool like Maven?

778. What are the differences between Ant and Maven?

779. What is MOJO in Maven?

780. What is a Repository in Maven?

781. What are the different types of repositories in Maven?

782. What is a local repository in Maven?

783. What is a central repository in Maven?

784. What is a Remote repository in Maven?

785. Why we should not store jars in CVS or any other version control system instead of Maven repository?

786. Can anyone upload JARS or artifacts to Central Repository?

787. What is a POM?

788. What is Super POM?

789. What are the main required elements in POM file?

790. What are the phases in Build lifecycle in Maven?

791. What command will you use to package your Maven project?

792.What is the format of fully qualified artifact name of a Maven project?

793.What is an Archetype in Maven?

794.What is the command in Maven to generate an Archetype?

795.What are the three main build lifecycles of Maven?

796.What are the main uses of a Maven plugin?

797.How will you find the version of a plugin being used?

798.What are the different types of profile in Maven? Where will you define these profiles?

799.What are the different setting files in Maven? Where will you find these files?

800.What are the main elements we can find in settings.xml?

801. How will you check the version of Maven in your system?

802.How will you verify if Maven is installed on Windows?

803.What is a Maven artifact?

804.What are the different dependency scopes in Maven?

805.How can we exclude a dependency in Maven?

806.How Maven searches for JAR corresponding to a dependency?

807.What is a transitive dependency in Maven?

808.What are Excluded dependencies in Maven?

809.What are Optional dependencies in Maven?

810. Where will you find the class files after compiling a Maven project successfully?

811. What are the default locations for source, test and build directories in Maven?

812. What is the result of jar:jar goal in Maven?

813. How can we get the debug or error messages from the execution of Maven?

814. What is the difference between a Release version and SNAPSHOT version in Maven?

815. How will you run test classes in Maven?

816. Sometimes Maven compiles the test classes but doesn't run them? What could be the reason for it?

817. How can we skip the running of tests in Maven?

818. Can we create our own directory structure for a project in Maven?

819. What are the differences between Gradle and Maven?

820. What is the difference between Inheritance and Multi-module in Maven?

821. What is Build portability in Maven?

GIT

822. How can we see n most recent commits in GIT?

823. How can we know if a branch is already merged into master in GIT?

824. What is the purpose of git stash drop?

825. What is the HEAD in GIT?

826. What is the most popular branching strategy in GIT?

827. What is SubGit?

828. What is the use of git instaweb?

829.What are git hooks?

830.What is GIT?

831. What is a repository in GIT?

832.What are the main benefits of GIT?

833.What are the disadvantages of GIT?

834.What are the main differences between GIT and SVN?

835.How will you start GIT for your project?

836.What is git clone in GIT?

837.How will you create a repository in GIT?

838.What are the different ways to start work in GIT?

839.GIT is written in which language?

840.What does 'git pull' command in GIT do internally?

841. What does 'git push' command in GIT do internally?

842.What is git stash?

843.What is the meaning of 'stage' in GIT?

844. What is the purpose of git config command?

845.How can we see the configuration settings of GIT installation?

846.How will you write a message with commit command in GIT?

847.What is stored inside a commit object in GIT?

848.How many heads can you create in a GIT repository?

849.Why do we create branches in GIT?

850.What are the different kinds of branches that can be created in GIT?

851. How will you create a new branch in GIT?

852. How will you add a new feature to the main branch?

853. What is a pull request in GIT?

854. What is merge conflict in GIT?

855. How can we resolve a merge conflict in GIT?

856. What command will you use to delete a branch?

857. What command will you use to delete a branch that has unmerged changes?

858. What is the alternative command to merging in GIT?

859. What is Rebasing in GIT?

860. What is the 'Golden Rule of Rebasing' in GIT?

861. Why do we use Interactive Rebasing in place of Auto Rebasing?

862. What is the command for Rebasing in Git?

863. What is the main difference between git clone and git remote?

864. What is GIT version control?

865. What GUI do you use for working on GIT?

866. What is the use of git diff command in GIT?

867. What is git rerere?

868. What are the three most popular version of git diff command?

869. What is the use of git status command?

870. What is the main difference between git diff and git status?

871. What is the use of git rm command in GIT?

872.What is the command to apply a stash?

873.Why do we use git log command?

874.Why do we need git add command in GIT?

875.Why do we use git reset command?

876.What does a commit object contain?

877.How can we convert git log messages to a different format?

878.What are the programming languages in which git hooks can be written?

879.What is a commit message in GIT?

880.How GIT protects the code in a repository?

881. How GIT provides flexibility in version control?

882.How can we change a commit message in GIT?

883.Why is it advisable to create an additional commit instead of amending an existing commit?

884.What is a bare repository in GIT?

885.How do we put a local repository on GitHub server?

886.How will you delete a branch in GIT?

887.How can we set up a Git repository to run code sanity checks and UAT tests just before a commit?

888.How can we revert a commit that was pushed earlier and is public now?

889.In GIT, how will you compress last n commits into a single commit?

890.How will you switch from one branch to a new branch in GIT?

891. How can we clean unwanted files from our working directory in GIT?

892. What is the purpose of git tag command?

893. What is cherry-pick in GIT?

894. What is shortlog in GIT?

895. How can you find the names of files that were changed in a specific commit?

896. How can we attach an automated script to run on the event of a new commit by push command?

897. What is the difference between pre-receive, update and post-receive hooks in GIT?

898. Do we have to store Scripts for GIT hooks within same repository?

899. How can we determine the commit that is the source of a bug in GIT?

900. How can we see differences between two commits in GIT?

901. What are the different ways to identify a commit in GIT?

902. When we run git branch <branchname>, how does GIT know the SHA-1 of the last commit?

903. What are the different types of Tags you can create in GIT?

904. How can we rename a remote repository?

905. Some people use git checkout and some use git co for checkout. How is that possible?

906. How can we see the last commit on each of our branch in GIT?

907. Is origin a special branch in GIT?

908. How can we configure GIT to not ask for password every time?

909. What are the four major protocols used by GIT for data transfer?

910. What is GIT protocol?

911. How can we work on a project where we do not have push access?

912. What is git grep?

913. How can your reorder commits in GIT?

914. How will you split a commit into multiple commits?

915. What is filter-branch in GIT?

916. What are the three main trees maintained by GIT?

917. What are the three main steps of working GIT?

918. What are ours and theirs merge options in GIT?

919. How can we ignore merge conflicts due to Whitespace?

920. What is git blame?

921. What is a submodule in GIT?

AWS

922. What do you know about AWS Region?

923. What are the important components of IAM?

924. What are the important points about AWS IAM?

925. What are the important features of Amazon S3?

926. What is the scale of durability in Amazon S3?

927. What are the Consistency levels supported by Amazon S3?

928. What are the different tiers in Amazon S3 storage?

929. How will you upload a file greater than 100 megabytes in Amazon S3?

930. What happens to an Object when we delete it from Amazon S3?

931. What is the use of Amazon Glacier?

932. Can we disable versioning on a version-enabled bucket in Amazon S3?

933. What are the use cases of Cross Region Replication Amazon S3?

934. Can we do Cross Region replication in Amazon S3 without enabling versioning on a bucket?

935. What are the different types of actions in Object Lifecycle Management in Amazon S3?

936. How do we get higher performance in our application by using Amazon CloudFront?

937. What is the mechanism behind Regional Edge Cache in Amazon CloudFront?

938. What are the benefits of Streaming content?

939. What is Lambda@Edge in AWS?

940. What are the different types of events triggered by Amazon CloudFront?

941. What is Geo Targeting in Amazon CloudFront?

942. What are the main features of Amazon CloudFront?

943. What are the security mechanisms available in Amazon S3?

Cloud Computing

944. What are the benefits of Cloud Computing?

945. What is On-demand computing in Cloud Computing?

946. What are the different layers of Cloud computing?

947. What resources are provided by Infrastructure as a Service (IAAS) provider?

948. What is the benefit of Platform as a Service?

949.What are the main advantages of PaaS?

950.What is the main disadvantage of PaaS?

951. What are the different deployment models in Cloud computing?

952.What is the difference between Scalability and Elasticity?

953.What is Software as a Service?

954.What are the different types of Datacenters in Cloud computing?

955.Explain the various modes of Software as a Service (SaaS) cloud environment?

956.What are the important things to care about in Security in a cloud environment?

957.Why do we use API in cloud computing environment?

958.What are the different areas of Security Management in cloud?

959.What are the main cost factors of cloud based data center?

960.How can we measure the cloud-based services?

961. How a traditional datacenter is different from a cloud environment?

962.How will you optimize availability of your application in a Cloud environment?

963.What are the requirements for implementing IaaS strategy in Cloud?

DOCKER

964.What is Docker?

965.What is the difference between Docker image and Docker container?

966.How will you remove an image from Docker?

967.How is a Docker container different from a hypervisor?

968.Can we write compose file in json file instead of yaml?

969.Can we run multiple apps on one server with Docker?

970.What are the common use cases of Docker?

971.What are the main features of Docker-compose?

972.What is the most popular use of Docker?

973.What is the role of open source development in the popularity of Docker?

UNIX Shell

974.How will you remove all files in current directory? Including the files that are two levels down in a sub-directory.

975.What is the difference between the –v and –x options in Bash shell scripts?

976.What is a Filter in Unix command?

977.What is Kernel in Unix operating system?

978.What is a Shell in Unix OS?

979.What are the different shells in Unix that you know about?

980.What is the first character of the output in ls –l command ?

981.What is the difference between Multi-tasking and Multi-user environment?

982.What is Command Substitution in Unix?

983.What is an Inode in Unix?

984.What is the difference between absolute path and relative path in Unix file system?

985.What are the main responsibilities of a Unix Shell?

986.What is a Shell variable?

Microservices

987.What is a Microservice?

988.What are the benefits of Microservices architecture?

989.What is the role of architect in Microservices architecture?

990.What is the advantage of Microservices architecture over Service Oriented Architecture (SOA)?

991. Is it a good idea to provide a Tailored Service Template for Microservices development in an organization?

992.What are the disadvantages of using Shared libraries approach to decompose a monolith application?

993.What are the characteristics of a Good Microservice?

994.What is Bounded Context?

995.What are the points to remember during integration of Microservices?

996.Is it a good idea for Microservices to share a common database?

997.What is the preferred type of communication between Microservices? Synchronous or Asynchronous?

998.What is the difference between Orchestration and Choreography in Microservices architecture?

999.What are the issues in using REST over HTTP for Microservices?

1000. Can we create Microservices as State Machines?

ACKNOWLEDGMENTS

We thank our readers who constantly send feedback and reviews to motivate us in creating these useful books with the latest information!

INTRODUCTION

Java is one of the most popular programming language. There is a growing demand for Java Developer jobs in technology companies.

This book contains technical interview questions that an interviewer asks for Java technology and related topics like Spring, Hibernate, Maven, Git, Microservices, AWS etc.

Each question is accompanied with an answer so that you can prepare for job interview in short time.

We have compiled this list after attending dozens of technical interviews in top-notch companies like- Facebook, Oracle, Netflix, Amazon etc.

Once you go through them in the first pass, mark the questions that you could not answer by yourself. Then, in second pass go through only the difficult questions.

After going through this book 2-3 times, you will be well prepared to face a technical interview for a Java Developer position from Software Engineer level to Principal Engineer level.

All the best!!

Java Interview Questions

Java Basics

1. What is the difference between JDK and JRE?

JDK stands for Java Development Kit. It contains the tools and libraries for development of Java programs. It also contains compilers and debuggers needed to compile Java program,

JRE stands for Java Runtime Environment. This is included in JDK. JRE provides libraries and JVM that is required to run a Java program.

2. What is Java Virtual Machine (JVM)?

Java Virtual Machine (JVM) is an abstract machine that executes Java Bytecode. There are different JVM for different hardware and software platforms. So JVM is platform dependent. JVM is responsible for loading, verifying and executing the Bytecode on a platform.

3. What are the different types of memory areas allocated by JVM?

In java, JVM allocates memory to different processes, methods and objects. Some of the memory areas allocated by JVM are:

1. ClassLoader: It is a component of JVM used to load class files.
2. Class (Method) Area: It stores per-class structures such as the runtime constant pool, field and method data, and the code for methods.
3. Heap: Heap is created a runtime and it contains the runtime data area in which objects are allocated.
4. Stack: Stack stores local variables and partial results at runtime. It also helps in method invocation and return value. Each thread creates a private JVM stack at the time of thread creation.
5. Program Counter Register: This memory area contains the address of the Java virtual machine instruction that is currently being executed.
6. Native Method Stack: This area is reserved for all the native methods used in the application.

https://www.baeldung.com/java-stack-heap#summary

4. What is JIT compiler?

Just In Time compiler also known as JIT compiler is used for performance improvement in Java. It is enabled by default. It is compilation done at execution time rather earlier.
Java has popularized the use of JIT compiler by including it in JVM.

```
Java compiler converts Java code in to byte code that can be interpreted by JVM
```

5. How Java platform is different from other platforms?

Java is a platform independent language. Java compiler converts Java code in to byte code that can be interpreted by JVM. There are JVM written for almost all the popular platforms in the world.

Java byte code can run on any supported platform in same way. Where as other languages require libraries compiled for a specific platform to run.

6. Why people say that Java is 'write once and run anywhere' language?

You can write Java code on Windows and compile it in Windows platform. The class and jar files that you get from Windows platform can run as it is on Unix environment. So it is a truly platform independent language.

Behind all this portability is Java byte code. Byte code generated by Java compiler can be interpreted by any JVM. So it becomes much easier to write programs in Java and expect those to run on any platform.

Java compiler javac compiles java code and JVM java runs that code.

7. How does ClassLoader work in Java?

In Java, ClassLoader is a class that is used to load files in JVM. ClassLoader loads files from their physical file locations e.g. Filesystem, Network location etc.

There are three main types of ClassLoaders in Java.

1. Bootstrap ClassLoader: This is the first ClassLoader. It loads classes from rt.jar file.
2. Extension ClassLoader: It loads class files from jre/lib/ext location.
3. Application ClassLoader: This ClassLoader depends on CLASSPATH to find the location of class files. If you specify your jars in CLASSPATH, then this ClassLoader will load them.

8. Do you think ‘main’ used for main method is a keyword in Java?

No, main is just a name of method. There can be multiple methods with same name main in a class file. It is not a keyword in Java.

9. Can we write main method as public void static instead of public static void?

No, you cannot write it like this. Any method has to first specify the modifiers and then the return value. The order of modifiers can change.

We can write static public void main() instead of public static void main().

```
Any method has to first specify the modifiers and then the return value.
```

10. In Java, if we do not specify any value for local variables, then what will be the default value of the local variables?

Java does not initialize local variables with any default value. So these variables will be just null by default.

11. Let say, we run a java class without passing any arguments. What will be the value of String array of arguments in Main method?

By default, the value of String array of arguments is empty in Java. It is not null.

12. What is the difference between byte and char data types in Java?

Both byte and char are numeric data types in Java. They are used to represent numbers in a specific range.

Major difference between them is that a byte can store raw binary data where as a char stores characters or text data.

Usage of char is E.g. char ch = ‘x’;

Byte values range from -128 to 127.

A byte is made of 8 bits. But a char is made of 16 bits. So it is equivalent to 2 bytes.

OOPS

13. What are the main principles of Object Oriented Programming?

Main principles of Object Oriented Programming (OOPS) are:

1. Abstraction
2. Encapsulation
3. Inheritance
4. Polymorphism

14. What is the difference between Object Oriented Programming language and Object Based Programming language?

Object Oriented Programming languages like Java and C++ follow concepts of OOPS like- Encapsulation, Abstraction, Polymorphism and Inheritance etc.

Object Based Programming languages follow some features of OOPS but they do not provide support for Polymorphism and Inheritance. Egg. JavaScript, VBScript etc.

Object Based Programming languages provide support for Objects and you can build objects from constructor. They languages also support Encapsulation. These are also known as Prototype-oriented languages.

15. In Java what is the default value of an object reference defined as an instance variable in an Object?

All the instance variable object references in Java are null.

```
the default value of an object reference defined as an instance variable in an Object: Null
```

16. Why do we need constructor in Java?

Java is an object-oriented language, in which we create and use objects. A constructor is a piece of code similar to a method. It is used to create an object and set the initial state of the object.

A constructor is a special function that has same name as class name.

Without a constructor, there is no other way to create an object.

By default, Java provides a default constructor for every object. If we overload a constructor then we have to implement default constructor.

17. Why do we need default constructor in Java classes?

Default constructor is the no-argument constructor that is automatically generated by Java if no other constructor is defined.

Java specification says that it will provide a default constructor if there is no overloaded constructor in a class. But it does not say anything about the scenario in which we write an overloaded constructor in a class.

We need at least one constructor to create an object, that's why Java provides a default constructor.

When we have overloaded constructor, then Java assumes that we want some custom treatment in our code. Due to which it does not provide default constructor. But it needs default constructor as per the specification. So it gives error.

18. What is the value returned by Constructor in Java?

When we call a constructor in Java, it returns the object created by it. That is how we create new objects in Java.

19. Can we inherit a Constructor?

No, Java does not support inheritance of constructor.

20. Why constructors cannot be final, static, or abstract in Java?

If we set a method as final it means we do not want any class to override it. But the constructor (as per Java Language Specification) cannot be overridden. So there is no use of marking it final.

If we set a method as abstract it means that it has no body and it should be implemented in a child class. But the constructor is called implicitly when the new keyword is used. Therefore it needs a body.

If we set a method as static it means that it belongs to the class, but not a particular object. The constructor is always called to initialize an object. Therefore, there is no use of marking constructor static.

Inheritance

21. What is the purpose of ‘this’ keyword in java?

In Java, ‘this’ keyword refers to current instance of the object.

It is useful for differentiating between instance variables and local variables.

It can be used to call constructors. Or it can be used to refer to the instance.

In case of method overriding, this is used for falling the method of current class.

22. Explain the concept of Inheritance?

Inheritance is an important concept in Object Oriented Programming. Some objects share certain characteristics and behavior. By using Inheritance, we can put the common behavior and characteristics in a base class which also known as super class. And then all the objects with common behavior inherit from this base class.

It is also represented by IS-A relationship.

Inheritance promotes, code reuse, method overriding and poly-morphism.

23. Which class in Java is superclass of every other class?

Java is an object oriented programming language. In Java, Object class is the superclass of every other class.

24. Why Java does not support multiple inheritance?

Multiple Inheritance means that a class can inherit behavior from two or more parent classes.

The issue with Multiple Inheritance is that both the parent classes may have different implementation for the same method. So they have different ways of doing the same thing. Now which implementation should the child class choose?

This leads to ambiguity in Multiple Inheritance. This is the main reason for Java not supporting Multiple Inheritance in implementation.

Lets say you have a class TV and another class AtomBomb. Both have method switchOn() but only TV has switchOff() method. If your class inherits from both these classes then you have an issue that you can switchOn() both parents, but switchOff will only switchOff() TV.

But you can implement multiple interfaces in Java.

25. In OOPS, what is meant by composition?

Composition is also known as “has-a” relationship. In composition, “has-a” relation relates two classes. E.g. Class Car has a steering wheel.

If a class holds the instance of another class, then it is called composition.

26. How aggregation and composition are different concepts?

In OOPS, Aggregation and Composition are the types of association relations. A composition is a strong relationship. If the composite object is destroyed, then all its parts are destroyed. E.g. A Car has a Steering Wheel. If Car object is destroyed, then there is no meaning of Steering Wheel.

In Aggregation, the relationship is weaker than Composition.

E.g. A Library has students. If a Library is destroyed, Students still exist. So Library and Student are related by Aggregation. A Library has Books. If Library is destroyed, the Books are also destroyed. Books of a Library cannot exist without the Library. So Book and Library are related by Composition.

27. Why there are no pointers in Java?

In Java there are references instead of pointers. These references point to objects in memory. But there is no direct access to these memory locations. JVM is free to move the objects within VM memory.

The absence of pointers helps Java in managing memory and garbage collection effectively. Also it provides developers with convenience of not getting worried about memory allocation and de-allocation.

28. If there are no pointers in Java, then why do we get NullPointerException?

In Java, the pointer equivalent is Object reference. When we use a . it points to object reference. So JVM uses pointers but programmers only see object references.

In case an object reference points to null object, and we try to access a method or member variable on it, then we get NullPointerException.

29. What is the purpose of ‘super’ keyword in java?

‘super’ keyword is used in the methods or constructor of a child class. It refers to immediate parent class of an object.

By using ‘super’ we can call a method of parent class from the method of a child class.

We can also call the constructor of a parent class from the constructor of a child class by using ‘super’ keyword.

30. Is it possible to use this() and super() both in same constructor?

No, Java does not allow using both super() and this() in same constructor. As per Java specification, super() or this() must be the first statement in a constructor.

31. What is the meaning of object cloning in Java?

Object.clone() method is used for creating an exact copy of the object in Java. It acts like a copy constructor. It creates and returns a copy of the object, with the same class and with all the fields having same values as of the original object.

One disadvantage of cloning is that the return type is an Object. It has to be explicitly cast to actual type.

Static

32. In Java, why do we use static variable?

Whenever we want to have a common property for all objects of a class, we use a class level variable i.e. a static variable.

This variable is loaded in memory only once at the time of class loading. So it saves memory, since it is not defined per object in Java.

33. Why it is not a good practice to create static variables in Java?

Static variables are common to all the objects of a class. If a new object is created, there is no need to test the value of static variable. Any code that uses static variable can be in any state. It can be within a new object or at a class level. So the scope of static variable is open ended in a Java class.

If we want tighter control on scope, then variables should be created at the object creation level.

Also defining static variables is not a good practice because they go against the principles of Object Oriented Programming.

34. What is the purpose of static method in Java?

Java provides the feature of static method to create behavior at the class level. The static method is common to all the objects of a class. We do not need to create any object of a class to call a static method. So it provides convenience of not creating an object for calling it.

Also a static method can access and modify static data members. This also helps in keeping the behavior as well as state at the class level.

35. Why do we mark main method as static in Java?

If main method is not static, Java process may not know which method of a class to call to start the program

The main method in Java is marked as static, so that JVM can call it to start the program. If main method is not static, then which constructor will be called by Java process?

As such it is a known as convention to mark main method static in Java. But if we remove the static, then there will be ambiguity. Java process may not know which method of a class to call to start the program.

So this convention helps in Java process to identify the starting code for a program in class that is passed as an argument to java process.

36. In what scenario do we use a static block?

At times, there is a class that has static member variables. These variables need some complicated initialization. At this time static block helps as a tool to initialize complex static member variable initialization.

The static block is executed even before the execution of main.

Sometimes, we can also replace static block with a static method of class.

37. Is it possible to execute a program without defining a main() method?

No, with Java 7 onwards, you need a main() method to execute a program. In earlier versions of Java, there was a workaround available to use static blocks for execution. But now this gap has been closed.

38. What happens when static modifier is not mentioned in the signature of main method?

As per Java specification, main method has to be marked as static. It needs only one argument that is an array of String.

A program can compile with a non-static method. But on execution it will give NoSuchMethodError.

39. What is the difference between static method and instance method in Java?

Often, there is a need to define a behavior for a class that is not dependent on member variables of an object. Such behavior is captured in a static method. If there is a behavior dependent upon the member variables of an object, then we do not mark it static, it remains as instance method.

To call as static method, we do not need to create an object. We just call it with class name. But to call an instance method, we need to create/get an object first.

Instance member variables cannot be accessed by a static method. But an instance method can call both instance variables and static variables.

Method Overloading and Overriding

40. What is the other name of Method Overloading?

Method Overloading is also known as Static Polymorphism.

41. How will you implement method overloading in Java?

In Java, a class can have multiple methods with same name but different arguments. It is called Method Overloading. To implement method overloading we have to create two methods with same name in a class and do one/more of the following:

1. Different number of parameters
2. Different data type of parameters
3. Different sequence of data type of parameters

42. What kinds of argument variations are allowed in Method Overloading?

Method Overloading allows two methods with same name to differ in:

1. Number of parameters
2. Data type of parameters
3. Sequence of data type of parameters

43. Why it is not possible to do method overloading by changing return type of method in java?

If we change the return type of overloaded methods then it will lead to ambiguous behavior. How will clients know which method will return what type. Due to this different return type are not allowed in overloaded methods.

44. Is it allowed to overload main() method in Java?

Yes, Java allows users to create many methods with same name ‘main’. But only public static void main(String[] args) method is used for execution.

45. How do we implement method overriding in Java?

To override a method, we just provide a new implementation of a method with same name in subclass. So there will be at least two implementations of the method with same name. One implementation is in parent class. And another implementation is in child class.

46. Are we allowed to override a static method in Java?

No. Java does not allow overriding a static method. If you create a static method with same name in subclass, then it is a new method, not an overridden method.

47. Why Java does not allow overriding a static method?

To override a method, you need an instance of a class. Static method is not associated with any instance of the class. So the concept of overriding does not apply here.

Therefore, Java does not allow overriding a static method.

48. Is it allowed to override an overloaded method?

Yes. You can override an overloaded method in Java.

49. What is the difference between method overloading and method overriding in Java?

Differences between method overloading and overriding are:

1. Method overloading is static polymorphism. Method overriding is runtime polymorphism.
2. Method overloading occurs within the same class. Method overriding happens in two classes with hierarchy relationship.
3. Parameters must be different in method overloading. Parameters must be same in method overriding.
4. Method overloading is a compile time concept. Method overriding is a runtime concept.

50. Does Java allow virtual functions?

Yes. All instance methods in Java are virtual functions by default. Only class methods and private instance methods are not virtual methods in Java.

51. What is meant by covariant return type in Java?

A covariant return type of a method is one that can be replaced by a "narrower" type when the method is overridden in a subclass.

Let say class B is child of class A. There is a get() method in class A as well as class B. get() method of class A can return an instance of A, and get() method of class B return an instance of B. Here class B overrides get() method, but the return type is different.

Before Java 5, any method that overrides the method of parent class would have same return type.

From Java 5 onwards, a child class can override a method of parent class and the child class method can return an object that is child of object return by parent class method.

Polymorphism

52. What is Runtime Polymorphism?

Runtime Polymorphism or Dynamic Polymorphism is the polymorphism that exists at runtime. In case of method overriding it is not known which method will be called at runtime. Based on the type of object, JVM decides the exact method that should be called.

So at compile time it is not known which method will be called at run time.

53. Is it possible to achieve Runtime Polymorphism by data members in Java?

No. We need to create Runtime Polymorphism by implementing methods at two levels of inheritance in Java.

54. Explain the difference between static and dynamic binding?

In Static binding references are resolved at compile time. In Dynamic binding references are resolved at Run time.

E.g.
Person p = new Person();
p.walk(); // Java compiler resolves this binding at compile time.

public void walk(Object o){
((Person) o).walk(); // this is dynamic binding.
}

Abstraction

55. What is Abstraction in Object Oriented programming?

Abstraction is the process of hiding certain implementation details of an object and showing only essential features of the object to outside world.

It is different from Abstract class in Java.

Abstraction process identifies commonalities and hides the complexity of implementation. It helps us in focusing on the interface that we share with the outside world.

56. How is Abstraction different from Encapsulation?

Abstraction happens at class level design. It results in hiding the implementation details. Encapsulation is also known as "Information Hiding". An example of encapsulation is marking the member variables private and providing getter and setter for these member variables.

```
Encapsulation: info hidding; but has getters & setters
Abstraction:   class level; implementation hiding;
```

57. What is an abstract class in Java?

An abstract class in Java has one or more abstract methods. An abstract method is just declared in the abstract class, but it is not implemented.

An abstract class has to be extended in Java and its abstract methods have to be implemented by a child class. Also Java does not allow new instance of Abstract class.

58. Is it allowed to mark a method abstract method without marking the class abstract?

No. Java specification says that if there is at least one abstract method in a class, the class has to be marked abstract.

59. Is it allowed to mark a method abstract as well as final?

No. It will be contradictory statement to mark a method abstract as well as final.

An abstract method has to be overridden by a child class. And a final method cannot be overridden. Therefore a method can be either abstract or final in Java.

60. Can we instantiate an abstract class in Java?

No. We cannot create an instance of an abstract class in Java.

61. What is an interface in Java?

An Interface in Java is an abstract type blueprint of a class. It contains the methods that a class must implement. It is like a protocol.

It has method signatures and constant declarations.

62. Is it allowed to mark an interface method as static?

Yes, from Java 8 onwards, we can define static and default methods in an interface. Prior to Java 8, it was not allowed.

63. Why an Interface cannot be marked as final in Java?

A final method cannot be overridden. But an interface method has to be implemented by another class. So the interface method cannot be marked as final.

64. What is a marker interface?

There are interfaces that do not have any data member or methods.
These interfaces are called Marker interface.
E.g. Serializable, Cloneable, Remote etc.

65. What can we use instead of Marker interface?

We can use annotations instead of Marker interface.

66. How Annotations are better than Marker Interfaces?

Annotations serve the purpose of conveying metadata about the class to its consumers without creating a separate type for it.

Annotations are more powerful than a Marker interface. They allow programmers to pass more sophisticated information to classes that "consume" it.

67. What is the difference between abstract class and interface in Java?

Differences between Abstract class and Interface are as follows:

1. An abstract class can have implemented methods with body (non-abstract methods). Interface has only abstract methods. From Java 8 onwards, interface can have static/default methods in implemented form.
2. An abstract class can have instance member variables. An interface cannot have instance variables. It can only have constants.
3. An abstract class can have a constructor. Interface cannot have constructor. It has to be implemented by another class.
4. A class can extend only one abstract class. A class can implement more than one interface.

68. Does Java allow us to use private and protected modifiers for variables in interfaces?

No. All the variables in an interface are implicitly public.

69. How can we cast to an object reference to an interface reference?

An Object that implements an Interface can be cast to the same Interface. Since An Object implementing an Interface already provides implementation for the methods of that Interface, it is allowed to do so as per the rules of Inheritance.

Final

70. How can you change the value of a final variable in Java?

Java does not allow changing the value of a final variable. Once the value is set, it cannot be changed.

71. Can a class be marked final in Java?

Yes a class can be marked final in Java. Once a class is marked final, it cannot be extended.

72. How can we create a final method in Java?

To mark a method, add modifier final to that method. A final method can not be overridden by a child class.

73. How can we prohibit inheritance in Java?

If you mark a class final, it cannot be extended. This will prohibit the inheritance of that class in Java.

74. Why Integer class in final in Java?

Integer class is a wrapper for int. If it is not marked final, then any other class can extend it and modify the behavior of Integer operations. To avoid this Integer wrapper class is marked as final.

75. What is a blank final variable in Java?

When we declare a final variable without giving any initial value, then it is called blank final variable.

76. How can we initialize a blank final variable?

A blank final instance variable can be initialized in a constructor.

A blank final static variable can be initialized in the static block of class.

77. Is it allowed to declare main method as final?

Yes, we can mark the main method as final.

Package

78. What is the purpose of package in Java?

A package is used to encapsulate a group of classes, interfaces and sub-packages. Often, it is a hierarchical structure of storing information. It is easier to organize the related classes and sub-packages in this manner.

A Package also provides access protection for classes and interfaces. A package also helps in removing naming collision.

79. What is java.lang package?

In Java, java.lang package contains the classes that are fundamental to the design of Java programming language. The most important class in this package is Object class.

It also contains wrapper classes like- Integer, Boolean, Character etc. It provides Math class for mathematical operations.

https://www.geeksforgeeks.org/java-lang-package-java/

80. Which is the most important class in Java?

It is an open-ended question with many answers. In my view, Object class is the most important class of Java programming language. It is the root of all the classes in Java. It provides some very important and fundamental methods.

81. Is it mandatory to import java.lang package every time?

No. By default, JVM loads it internally.

82. Can you import same package or class twice in your class?

If we import same package multiple times in a class, compiler includes it only once. So neither JVM nor Compiler gives any error/warning on including a package multiple times.

If you have two classes with same name, then you may get name collision on importing the class erroneously.

JVM internally loads the class only one time.

https://beginnersbook.com/2013/05/java-static-import/

83. What is a static import in Java?

Static import is similar to normal import declaration. Normal import allows us to import classes from packages without using package qualifier. Static import allows us to import static members from a class without using class qualifier.

Example 1: Without Static Imports

```
class Demo1{
   public static void main(String args[])
   {
      double var1= Math.sqrt(5.0);
      double var2= Math.tan(30);
      System.out.println("Square of 5 is:"+ var1);
      System.out.println("Tan of 30 is:"+ var2);
   }
```

When to use static imports?
If you are going to use static variables and methods a lot then it's fine to use static imports. for example if you wanna write a code with lot of mathematical calculations then you may want to use static import.
Drawbacks
It makes the code confusing and less readable so if you are going to use static members very few times in your code then probably you should avoid using it. You can also use wildcard(*) imports.

Example 2: Using Static Imports

```
import static java.lang.System.out;
import static java.lang.Math.*;
class Demo2{
   public static void main(String args[])
   {
      //instead of Math.sqrt need to use only sqrt
      double var1= sqrt(5.0);
      //instead of Math.tan need to use only tan
      double var2= tan(30);
      //need not to use System in both the below statements
      out.println("Square of 5 is:"+var1);
```

84. What is the difference between import static com.test.Fooclass and import com.test.Fooclass?

First import is a static import and the second import is normal import of a class. First import allows us to import static members of class.

Internationalization

85. What is Locale in Java?

A Locale object represents a specific geographical, political, or cultural region. It is used to locale-sensitive operations in Java.

It helps is following the local conventions of a country, native or region. These conventions can be for formatting the dates, money, numbers etc.

86. How will you use a specific Locale in Java?

To use a specific Locale, we need to load that Locale. We can use ResourceBundle.getBundle("Locale.UK") method to load a Locale.

https://www.javatpoint.com/post/java-locale

Serialization

87. What is the serialization?

Serialization is a process converting an object into a byte array. This byte array represents the class, version and internal state of the object. JVM can use this byte array to transmit/read the object over a network.

88. What is the purpose of serialization?

Some of the uses of serialization are:

1. Communication: It is used for transmitting an object over network between two machines.
2. Persistence: We can store the object's state in a database and retrieve it from database later on.
3. Caching: Serialization can be used for caching to improve performance. We may need 10 minutes to build an object, but it may take just 10 seconds to de-serialize the object.
4. Cross JVM Synchronization: It can be used in same way across multiple JVM that follow different architecture.

89. What is Deserialization?

Deserialization is the process of reconstructing the object from the serialized state. It is the reverse process of serialization.

90. What is Serialization and Deserialization conceptually?

Serialization is to convert Object data into a stream of bytes

Deserialization is to convert a stream of bytes back into a copy of the original object.

91. Why do we mark a data member transient?

Member variables of an object are marked transient to indicate that they should not be serialized.

During serialization process the transient variables are not considered part of the persistent state of an object.

92. Is it allowed to mark a method as transient?

No, Java does not allow marking a method as transient. The transient keyword is valid only for member variables.

https://www.educative.io/answers/what-is-the-transient-keyword-in-java

93. How does marking a field as transient makes it possible to serialize an object?

Let say we have a class ABC that implements Serializable interface, but it contains a member variable object of class XYZ that does not implement Serializable interface. Due to this it is not possible to Serialize the class ABC.

To solve this issue, we can mark the member variable XYZ as Transient in class ABC. This will allow us to serialize the class ABC.

94. What is Externalizable interface in Java?

Externalizable interface extends Serializable interface in Java. It is used for giving the Class control over saving and restoring the contents of its instances.

A class implements methods writeExternal() and readExternal() to store and restore the object.

95. What is the difference between Serializable and Externalizable interface?

Serializable is a marker interface but Externalizable is not a marker interface.

When we implement Serializable interface, the class is serialized automatically by default. We can override writeObject() and readObject()methods to control more complex object Serialization process.

In case of Externalizable, we use readExternal() and writeExternal() methods to give control to class for class's serialization process.

Serializable interface is based on recursive algorithm.

Serializable gives you two options. One option is to provide custom way of serialization, the other default way. In Externalizable, you have to always implement readExternal() and writeExternal() methods.

A public no-arg constructor is needed while using Externalizable interface.

In Serialization, we need to define serialVersionUID. If it is not explicitly defined it will be generated automatically based on all the fields, methods of the class.

Reflection

96. What is Reflection in Java?

Reflection is Java language's ability to inspect and dynamically call classes, methods, attributes etc. at Runtime. It helps in examining or modifying the Runtime behavior of a class at Runtime.

97. What are the uses of Reflection in Java?

Reflection is often used in Testing, Debugging and in Integrated Development Environment (IDE).

Reflection allows you to write programs that do not have to "know" everything at compile time. It makes programs more dynamic, since they can be tied together at runtime.

Many modern frameworks like Spring etc. use Reflection. Some modern languages like Python etc. also use Reflection.

JAVA API for XML Parsing (JAXP) also uses Reflection.

98. How can we access private method of a class from outside the class?

We can use Reflection to access private method of a class from outside the class. IN Java, we use getDeclaredMethod() to get instance of a private method. Then we mark this method accessible and finally invoke it.

In following sample code, we are accessing private method message() of class Foo by Reflection.

FileName: Foo.java

```
public class Foo {
  private void message(){System.out.println("hello java"); }
}
```

FileName: FooMethodCall.java

```
import java.lang.reflect.Method;
public class FooMethodCall{
public static void main(String[] args)throws Exception{

    Class c = Class.forName("Foo");
    Object o= c.newInstance();
    Method m =c.getDeclaredMethod("message", null);
    m.setAccessible(true);
    m.invoke(o, null);
}
}
```

99. How can we create an Object dynamically at Runtime in Java?

We can use Reflection to create an Object dynamically at Runtime in Java. We can use Class.newInstance() or Constructor.newInstance() methods for creating such Objects.

Garbage Collection

100. What is Garbage Collection in Java?

Java has an internal mechanism called Garbage collection to reclaim the memory of unused projects at run time.

Garbage collection is also known as automatic memory management.

101. Why Java provides Garbage Collector?

In Java, there are no pointers. Memory management and allocation is done by JVM. Since memory allocation is automated, after some time JVM may go low on memory. At that time, JVM has to free memory from unused objects. To help with the process of reclaiming memory, Java provides an automated process called Garbage Collector.

102. What is the purpose of gc() in Java?

Java provides two methods System.gc() and Runtime.gc() to request the JVM to run the garbage collection. By using these methods, programmers can explicitly send request for Garbage Collection. But JVM process can reject this request and wait for some time before running the GC.

103. How does Garbage Collection work in Java?

Java has an automated process called Garbage Collector for Memory Management. It is a daemon in JVM that monitors the memory usage and performs memory cleanup. Once JVM is low on memory, GC process finds the unused objects that are not referenced by other objects. These unused objects are cleaned up by Garbage Collector daemon in JVM.

104. When does an object become eligible for Garbage Collection in Java?

An object can be Garbage Collected by JVM, if it is not reachable. There are two cases for deciding eligibility of objects for Garbage Collection:

1. An Object/instance that cannot be reached by a live thread.
2. A set of circularly referenced instances that cannot be reached by any other instance outside that set.

105. Why do we use finalize() method in Java?

Java provides finalize() method to perform any cleanup before Garbage Collection. This method is in Object class, and it is invoked by JVM internally. Developers are free to implement this method for any custom cleanup in case of Garbage Collection.

If an Object is not Garbage Collected, then this method may not be called.

This method is never invoked more than once by JVM.

106. What are the different types of References in Java?

In Java, there are four types of references:

1. Strong Reference
2. Soft Reference
3. Weak Reference
4. Phantom Reference

107. How can we reference an unreferenced object again?

We can provide implementation in finalize() method to reference and unreferenced object. For an unreferenced object, finalize() method is called at the time of Garbage Collection. At this time, Object can pass its reference ‘this’ to finalize() method and revive itself.

108. What kind of process is the Garbage collector thread?

Garbage Collection is a Daemon process in JVM. It is an internal process that keep checking Memory usage and cleans up the memory.

109. What is the purpose of the Runtime class?

The purpose of the Runtime class is to provide access to the Java Runtime system. This class provides certain important methods like:

1. Runtime.freeMemory() – This method returns the value of free memory in JVM

2. Runtime.maxMemory() - This method returns the value of maximum memory that JVM can use.

3. Runtime.gc() – This method can invoke garbage collection.

110. How can we invoke an external process in Java?

Java provides the method Runtime.getRuntime().exec() to invoke an external process from JVM.

111. What are the uses of Runtime class?

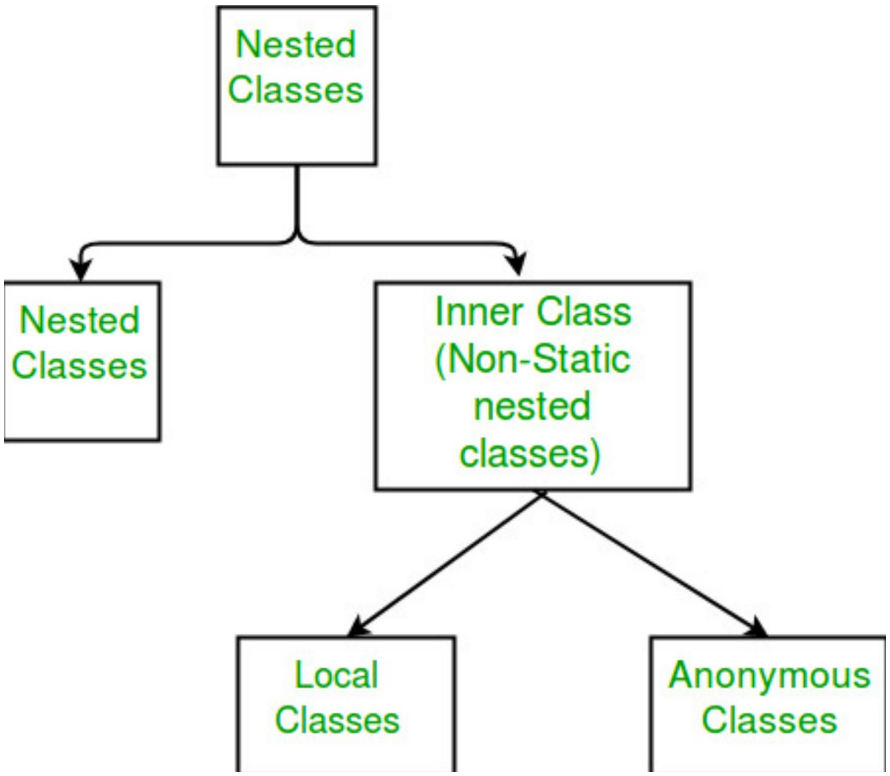
Runtime class in Java provides following benefits:

1. It allows to read data via key board
2. It can use system properties and environment variables
3. It helps in running non-java programs from within a java application.

Inner Classes

https://www.geeksforgeeks.org/local-inner-class-java/

The scope of the local inner class is restricted to the block they are defined in.
A local inner class cannot be instantiated from outside the block where it is created in.
Till JDK 7, the Local inner class can access only the final local variable of the enclosing block. However, From JDK 8, it is possible to access the non-final local variable of enclosing block in the local inner class.
A local class has access to the members of its enclosing class.
Local inner classes can extend an abstract class or implement an interface.



112. What is a Nested class?

In Java, a Nested class is a class declared inside another class. We can have more than one class declared inside a file.

113. How many types of Nested classes are in Java?

Java provides four types of Nested classes:

1. Member inner class
2. Local inner class
3. Anonymous inner class
4. Static nested class

114. Why do we use Nested Classes?

There are following reasons for using nested classes:

1. Logical Grouping: We can logically group classes in one place. If one class is useful to only one other class, then we put smaller class within the larger class and keep them in one file. This kind of nesting "helper classes" in a top-level class makes the package more streamlined.

2. Encapsulation: Nested classes increase encapsulation. Let say there are two top-level classes, Foo and Bar. Bar needs access to private members of Foo. We can hide class Bar within class Foo. In this way, private members of Foo can be accessed by class Bar. So class Foo remains encapsulated. Also, class Bar remains hidden from the outside world.

3. Code Clarity: Nested classed make the code more readable and well organized. Only Top-level classes are exposed. The helper classes are kept hidden and closer the code where it is used by a Top-level class.

115. What is the difference between a Nested class and an Inner class in Java?

An Inner class in Java is non-static class. It is a type of Nested class that is defined in another class but not qualified with a Static modifier. A Nested class is also a class can be Static Nested class or a non-Static Inner class.

An Inner class has access to other members of the enclosing class, even if they are declared private. A Static Nested class can not access the other members of the enclosing class.

https://docs.oracle.com/javase/tutorial/java/javaOO/nested.html#:~:text=A%20nested%20class%20is%20a,members%20of%20the%20enclosing%20class.

116. What is a Nested interface?

A Nested interface is declared inside another interface or a top-level class. By default it is static.

A Nested interface is also known as Static interface.

117. How can we access the non-final local variable, inside a Local Inner class?

Java allows a Local Inner class to access only Constant local members. So we have to make the non-final local variable as final constant to access it inside a Local Inner class.

118. Can an Interface be defined in a Class?

Yes, we can define a Static Nested interface within a class. Only the enclosing class can access it.

119. Do we have to explicitly mark a Nested Interface public static?

A Nested Interface is implicitly public static. So the modifiers public and static are redundant in declaration.

120. Why do we use Static Nested interface in Java?

Only the enclosing class can access a Static Nested interface. Consider following code in which interface Xyz is enclosed in class Abc.

```
public class Abc {

   public interface Xyz {
      void callback();
   }

   public static void registerCallback(Xyz xyz) {...}
}

// Client Code
Abc.registerCallback(new Abc.Xyz() {
   public void callback() {...}
});
```

Any code that cannot access Abc can not access interface Xyz also.

So the purpose of declaring an Inner interface is to restrict its access from outside world.

String

121. What is the meaning of Immutable in the context of String class in Java?

An Immutable object cannot be modified or changed in Java. String is an Immutable class in Java.

Once a String object is created, it cannot be changed. When we assign the String to a new value, a new object is created.

122. Why a String object is considered immutable in java?

Java language uses String for a variety of purposes. For this it has marked String Immutable.

There is a concept of String literal in Java.

Let say there are 2 String variables A and B that reference to a String object “TestData”. All these variables refer to same String literal. If one reference variable A changes the value of the String literal from “TestData” to “RealData”, then it will affect the other variable as well. Due to which String is considered Immutable. In this case, if one variable A changes the value to “RealData”, then a new String literal with “RealData” is created and A will point to new String literal. While B will keep pointing to “TestData”

123. How many objects does following code create?

Code:
String s1="HelloWorld";
String s2=" HelloWorld ";
String s3=" HelloWorld ";

The above code creates only one object. Since there is only one String Literal “HelloWorld” created, all the references point to same object.

124. How many ways are there in Java to create a String object?

Java provides two ways to create a String object. One is by using String Literal, the other is by using new operator.

125. How many objects does following code create?

Code:
String s = new String("HelloWorld");

The above code creates two objects. One object is created in String constant pool and the other is created on the heap in non-pool area.

126. What is String interning?

String interning refers to the concept of using only one copy of a distinct String value that is Immutable.

It provides the advantage of making String processing efficient in Time as well as Space complexity. But it introduces extra time in creation of String.

https://www.geeksforgeeks.org/interning-of-string/

127. Why Java uses String literal concept?

Java uses String literal concept to make Java more efficient in memory. If same String already exists in String constant pool, it can be reused. This saves memory usage.

128. What is the basic difference between a String and StringBuffer object?

String is an immutable object. Its value cannot change after creation. StringBuffer is a mutable object. We can keep appending or modifying the contents of a StringBuffer in Java.

129. How will you create an immutable class in Java?

In Java, we can declare a class final to make it immutable. There are following detailed steps to make it Immutable:

1. Add final modifier to class to prevent it from getting extended
2. Add private modifier to all the fields to prevent direct access
3. Do not provide any setter methods for member variables
4. Add final modifier to all the mutable fields to assign value only once
5. Use Deep Copy to initialize all the fields by a constructor
6. In clone method, return a copy of object instead of the actual object reference

130. What is the use of toString() method in java ?

In Java, Object class has toString() method. This method can be used to return the String representation of an Object. When we print an object, Java implicitly calls toString() method.

Java provides a default implementation for toString() method. But we can override this method to return the format that we want to print.

131. Arrange the three classes String, StringBuffer and StringBuilder in the order of efficiency for String processing operations?

StringBuilder is the most efficient class. It does not have the overhead of Synchronization. StringBuffer is a Synchronized class. It has better performance than String but it is slower than StringBuilder. String is the slowest for any String processing operations, since it is leads to creation of new String literal with each modification.

So the decreasing order of efficiency is: StringBuilder, StringBuffer, String

Exception Handling

132. What is Exception Handling in Java?

Java provides Exception Handling mechanism to handle Runtime errors that occur in JVM. There are checked exceptions in a program that we expect to occur in certain situations.

Exception handling mechanism catches these checked exceptions and takes relevant actions.

133. In Java, what are the differences between a Checked and Unchecked?

Checked Exceptions extend Throwable class, but they do not extend RuntimeException or Error classes. UncheckedException extend RuntimeException class.

Checked Exceptions are checked at compile time in Java. Unchecked Exceptions happen at Runtime, so they are not checked at compile time.

IOException, SQLException etc. are examples of Checked Exceptions. NullPointerException, ArithmeticException etc. are examples of Unchecked Exceptions.

134. What is the base class for Error and Exception classes in Java?

Error as well as Exception class is derived from Throwable class in Java.

135. What is a finally block in Java?

Java provides a finally block with a try block. This is an optional block. But finally block is always executed after the execution of try block.

136. What is the use of finally block in Java?

As per Java specification, a finally block is always executed, whether an error occurs or not, whether an exception is handled or not. It helps in doing the cleanup like- Rollback Transaction, Close Connection, Close a file etc.

137. Can we create a finally block without creating a catch block?

Yes. A finally block can follow a try block or catch block. So we can defined a finally block just after a try block.

138. Do we have to always put a catch block after a try block?

Java does not enforce the rule to put a catch block after try block. We can write catch block or finally block after a try block.

Any exception that we want to catch is mentioned in catch block.

139. In what scenarios, a finally block will not be executed?

There are two main scenarios in which finally block is not executed:

1. Program exits by calling system.exit() call.
2. A fatal error causes JVM to crash.

140. Can we re-throw an Exception in Java?

Yes, Java allows to re-throw an Exception.

141. What is the difference between throw and throws in Java?

Java provides throw keyword to throw an exception from a method or a static block. Java provides throws keyword to mention the probable exception thrown by a method in its declaration.

We use throw to explicitly throw an exception. We used throws to declare an exception in method definition.

We cannot propagate checked exceptions with throw only. But checked exceptions can be propagated with throws keyword.

A throw call is followed by an instance. Class or Exception follows a throws keyword.

Call to throw occurs within a method. throws is just used with method signature.

We can throw only one exception at a time. But we can mention as many exceptions in throws clause.

when an exception happens, Propagation is a process in which the exception is being dropped from to the top to the bottom of the stack. If not caught once, the exception again drops down to the previous method and so on until it gets caught or until it reach the very bottom of the call stack

142. What is the concept of Exception Propagation?

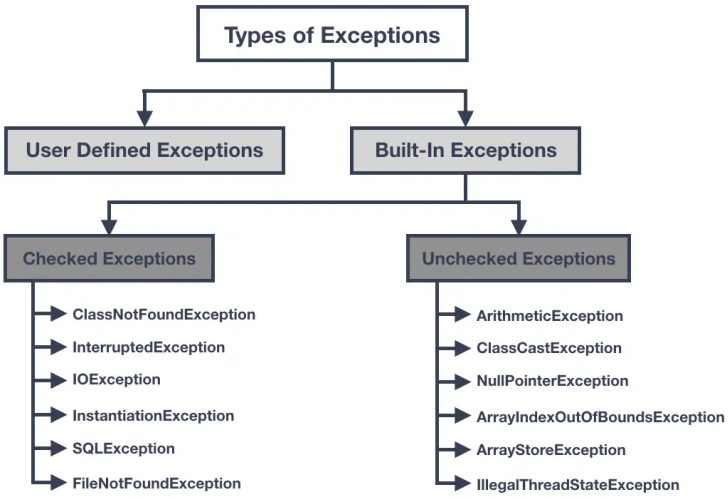
In Exception Propagation, uncaught exceptions are propagated in the call stack until stack becomes empty. This propagation is called Exception Propagation.

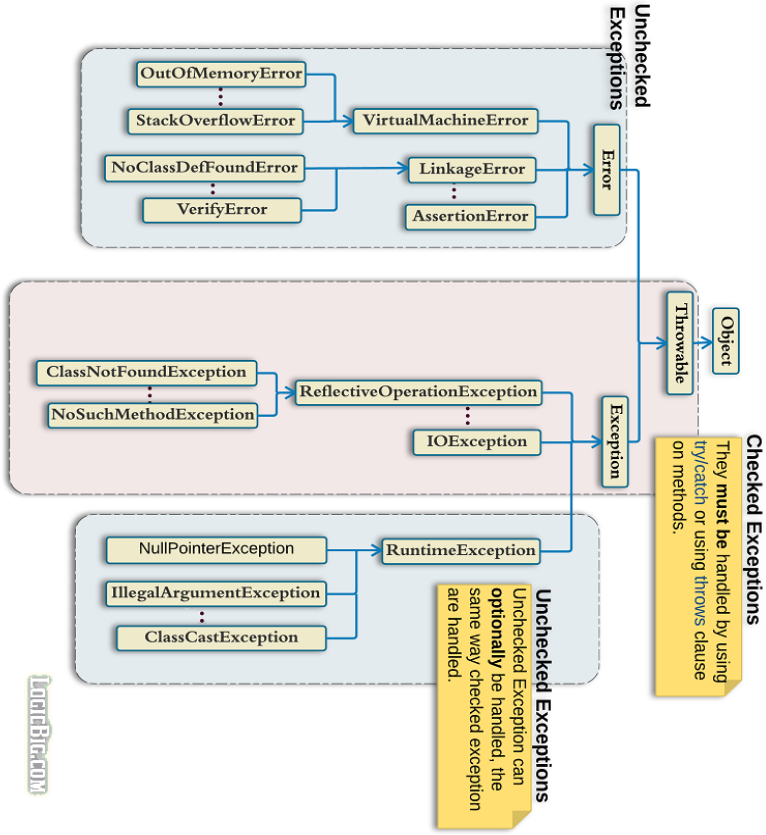
Let say an exception propagates from one method to another method. A() calls B(), which calls C(), which calls D(). And if D() throws an exception, the exception will propagate from D to C to B to A, unless one of the methods catches the exception.

Unchecked Exceptions are forwarded in calling chain (propagated)

143. When we override a method in a Child class, can we throw an additional Exception that is not thrown by the Parent class method?

Yes, Java allows us to throw additional Exception in a child class, but the additional exception should be an unchecked exception (RuntimeException).





https://rollbar.com/blog/how-to-handle-checked-unchecked-exceptions-in-java/#:~:text=An%20unchecked%20exception%20is%20a,be%20recovered%20or%20retried%20from.

https://www.logicbig.com/tutorials/core-java-tutorial/java-language/checked-unchecked-exceptions.html

Java Collection

144. What is the difference between Collection and Collections Framework in Java?

In Java, a Collection is an object that contains multiple elements of same type in a single unit. These multiple elements can be accessed through one Collection object.

In Java Collections Framework is a library that provides common architecture for creating, updating and accessing different types of collections. In Collections framework there are common methods that are frequently used by developers for working on a Collection object.

145. What are the main benefits of Collections Framework in Java?

Main benefits of Collections Framework in Java are as follows:

1. Reusability: Java Collections Framework provides common classes and utility methods than can be used with different types of collections. This promotes the reusability of the code. A developer does not have to re-invent the wheel by writing the same method again.

2. Quality: Using Java Collection Framework improves the program quality, since the code is already tested and used by thousands of developers.

3. Speed: Most of programmers report that their development speed increased since they can focus on core logic and use the generic collections provided by Java framework.

4. Maintenance: Since most of the Java Collections framework code is open source and API documents is widely available, it is easy to maintain the code written with the help of Java Collections framework. One developer can easily pick the code of previous developer.

146. What is the root interface of Collection hierarchy in Java?

The root interface of Collection hierarchy in Java is Collection interface.

But the Collection interface extends Iterable interface. Due to this some people consider Iterable interface as the root interface.

Iterable interface is present in java.lang package but Collection interface is present in java.util package. Oracle Java API docs mention that Collection interface is a member of the Java Collections framework.

Whereas, Iterable interface is not stated as a part of Java Collections framework in Java docs.

Due to this Collection interface is the root of Collections Framework.

147. What are the main differences between Collection and Collections?

Main differences between Collection and Collections are as follows:

1. Collection is an interface in Java. But Collections is a class in Java.

2. Collection is a base interface. Collections is a utility class in Java.

3. Collection defines methods that are used for data structures that contain the objects. Collections defines the methods that are used for operations like access, find etc. on a Collection.

148. What are the Thread-safe classes in Java Collections framework?

The Thread-safe classes in Java Collections framework are:

- Stack
- Properties
- Vector
- Hashtable
- BlockingQueue
- ConcurrentMap
- ConcurrentNavigableMap

149. How will you efficiently remove elements while iterating a Collection?

The right way to remove elements from a collection while iterating is by using ListIterator.remove() method.

E.g.

```
ListIterator<Integer> iter = myList.iterator();
while(iter.hasNext()) {
itr.remove();
}
```

Some developers use following code to remove an element which is incorrect:

```
Iterator<Integer> iter = myList.iterator();
while(iter.hasNext()) {
itr.remove();
}
```

By doing so we get ConcurrentModificationException.

An iterator is first created to traverse the list. But at the same time the list is changed by remove() method.

In Java, it is not allowed for a thread to modify a collection while another thread is iterating it. ListIterator provides the capability of removing an object during traversal.

150. How will you convert a List into an array of integers like- int[]?

We can use ArrayUtils class in Apache Commons Lang library.

Sample code is:
int[]intArray = ArrayUtils.toPrimitive(myList.toArray(new Integer[0]));

If we use List.toArray(), it will convert List to Integer[].

Another option is:

```
int[] intArray = new int[myList.size()];
for (int i=0; i < myList.size(); i++) {
  intArray [i] = myList.get(i);
}
```

151. How will you convert an array of primitive integers int[] to a List collection?

We can use ArrayUtils in Apache Commons Lang library for this purpose.

Sample code is:

```
List intList = Arrays.asList(ArrayUtils.toObject(intArray));
```

The other option would be to use a for loop and explicitly adding integers to a List.

Sample code is:

```
int[]intArray = {10,20,30};
List<Integer> intList = new ArrayList<Integer>();
for (int i: intArray) {
  intList.add(i);
}
```

152. How will you run a filter on a Collection?

We can use CollectionUtils of Apache for this purpose. We will have to create a Predicate that will define the condition for our filter. Then we can apply this Predicate in filter() method.

Sample code is:

In this example we filter any names that are less than 5 characters long.

```
List<String> namesList = asList( "Red", "Blue", "Green" );

List<String> shortNamesList = new ArrayList<String>();
shortNamesList.addAll( namesList );

    CollectionUtils.filter( shortNamesList, new Predicate(){
       public boolean evaluate( Object input ) {
          return ((String) input).length() < 5;
       }
    } );
```

We can also use Google Guava library for this.

In Java 8, we can use Predicate to filter a Collection through Stream.

153. How will you convert a List to a Set?

There are two ways to convert a List to a Set in Java.

Option 1: Use HashSet

```
Set<Integer> mySet = new HashSet<Integer>(myList);
```

In this case we put a list into a HashSet. Internally hashCode() method is used to identify duplicate elements.

Option 2: Use TreeSet
In this case we use our own comparator to find duplicate objects.

```
Set<Integer> mySet = new TreeSet<Integer>(myComparator);
mySet.addAll(myList);
```

154. How will you remove duplicate elements from an ArrayList?

The trick in this question is to use a collection that does not allow duplicate elements. So we use a Set for this purpose.

Option 1: Use Set
If ordering of elements is not important then we just put the elements of ArrayList in a HashSet and then add them back to the ArrayList.

Sample Code is:

```
ArrayList  myList = // ArrayList with duplicate elements
Set<Integer> mySet = new HashSet<Integer>(myList);
myList.clear();
myList.addAll(mySet);
```

Option 2: Use LinkedHashSet
If ordering of elements is important then we put the elements of ArrayList in a LinkedHashSet and then add them back to the ArrayList.

Sample Code is:

```
ArrayList  myList = // ArrayList with duplicate elements
Set<Integer> mySet = new LinkedHashSet<Integer>(myList);
myList.clear();
myList.addAll(mySet);
```

155. How can you maintain a Collection with elements in Sorted order?

In Java, there are many ways to maintain a Collection with elements in sorted order.

Some collections like TreeSet store elements in the natural ordering. In case of natural ordering we have to implement Comparable interface for comparing the elements.

We can also maintain custom ordering by providing a custom Comparator to a Collection.

Another option is to use the utility method Collections.sort() to sort a List. This sorting gives nlog(n) order of performance. But if we have to use this method multiple times then it will be costly on performance.

Another option is to use a PriorityQueue that provides an ordered queue. The main difference between PriorityQueue and Collections.sort() is that PriorityQueue maintains a queue in Order all the time, but we can only retrieve head element from queue. We cannot access the elements of PriorityQueue in Random order.

We can use TreeSet to maintain sorted order of elements in collection if there are no duplicate elements in collection.

156. What are the differences between the two data structures: a Vector and an ArrayList?

An ArrayList is a newer class than a Vector. A Vector is considered a legacy class in Java. The differences are:

1. Synchronization: Vector is synchronized, but the ArrayList is not synchronized. So an ArrayList has faster operations than a Vector.
2. Data Growth: Internally both an ArrayList and Vector use an array to store data. When an ArrayList is almost full it increases its size by 50% of the array size. Whereas a Vector increases it by doubling the underlying array size.

157. What are the differences between Collection and Collections in Java?

Main differences between Collection and Collections are:

1. Type: Collection is an interface in Java. Collections is a class.
2. Features: Collection interface provides basic features of data structure to List, Set and Queue interfaces. Collections is a utility class to sort and synchronize collection elements. It has polymorphic algorithms to operate on collections.
3. Method Type: Most of the methods in Collection are at instance level. Collections class has mainly static methods that can work on an instance of Collection.

158. In which scenario, LinkedList is better than ArrayList in Java?

ArrayList is more popular than LinkedList in Java due to its ease of use and random access to elements feature.

But LinkedList is better in the scenario when we do not need random access to elements or there are a lot of insertion, deletion of elements.

159. What are the differences between a List and Set collection in Java?

Main differences between a List and a Set are:

1. Order: List collection is an ordered sequence of elements. A Set is just a distinct collection of elements that is unordered.

2. Positional Access: When we use a List, we can specify where exactly we want to insert an element. In a Set there is no order, so we can insert element anywhere without worrying about order.

3. Duplicate: In a List we can store duplicate elements. A Set can hold only unique elements.

160. What are the differences between a HashSet and TreeSet collection in Java?

Main differences between a HashSet and TreeSet are:

1. Ordering: In a HashSet elements are stored in a random order. In a TreeSet, elements are stored according to natural ordering.

2. Null Value Element: We can store null value object in a HashSet. A TreeSet does not allow to add a null value object.

3. Performance: HashSet performs basic operations like add(), remove(), contains(), size() etc in a constant size time. A TreeSet performs these operations at the order of log(n) time.

4. Speed: A HashSet is better than a TreeSet in performance for most of operations like add(), remove(), contains(), size() etc .

5. Internal Structure: a HashMap in Java internally backs a HashSet. A NavigableMap backs a TreeSet internally.

6. Features: A TreeSet has more features compared to a HashSet. It has methods like pollFirst(), pollLast(), first(), last(), ceiling(), lower() etc.

7. Element Comparison: A HashSet uses equals() method for comparison. A TreeSet uses compareTo() method for

comparison to maintain ordering of elements.

161. In Java, how will you decide when to use a List, Set or a Map collection?

1. If we want a Collection that does not store duplicate values, then we use a Set based collection.
2. If we want to frequently access elements operations based on an index value then we use a List based collection. E.g. ArrayList

3. If we want to maintain the insertion order of elements in a collection then we use a List based collection.

4. For fast search operation based on a key, value pair, we use a HashMap based collection.

5. If we want to maintain the elements in a sorted order, then we use a TreeSet based collection.

162. What are the differences between a HashMap and a Hashtable in Java?

Main differences between a HashMap and a Hashtable are:

1. Synchronization: HashMap is not a synchronized collection. If it is used in multi-thread environment, it may not provide thread safety. A Hashtable is a synchronized collection. Not more than one thread can access a Hashtable at a given moment of time. The thread that works on Hashtable acquires a lock on it and it makes other threads wait till its work is completed.

2. Null values: A HashMap allows only one null key and any number of null values. A Hashtable does not allow null keys and null values.

3. Ordering: A HashMap implementation by LinkedHashMap maintains the insertion order of elements. A TreeMap sorts the mappings based on the ascending order of keys. On the other hand, a Hashtable does not provide guarantee of any kind of order of elements. It does not maintain the mappings of key values in any specific order.

4. Legacy: Hashtable was not the initial part of collection framework in Java. It has been made a collection framework member, after being retrofitted to implement the Map interface. A HashMap implements Map interface and is a part of collection framework since the beginning.

5. Iterator: The Iterator of HashMap is a fail-fast and it throws ConcurrentModificationException if any other Thread modifies the map by inserting or removing any element except iterator's own remove() method.

Enumerator of the Hashtable is not fail-fast.

163. What are the differences between a HashMap and a TreeMap?

Main differences between a HashMap and a TreeMap in Java are:

1. Order: A HashMap does not maintain any order of its keys. In a HashMap there is no guarantee that the element inserted first will be retrieved first.

2. In a TreeMap elements are stored according to natural ordering of elements. A TreeMap uses compareTo() method to store elements in a natural order.

3. Internal Implementation: A HashMap uses Hashing internally. A TreeMap internally uses Red-Black tree implementation.

4. Parent Interfaces: A HashMap implements Map interface. TreeMap implements NavigableMap interface.

5. Null values: A HashMap can store one null key and multiple null values. A TreeMap can not contain null key but it may contain multiple null values.

6. Performance: A HashMap gives constant time performance for operations like get() and put(). A TreeMap gives order of log(n) time performance for get() and put() methods.

7. Comparison: A HashMap uses equals() method to compare keys. A TreeMap uses compareTo() method for maintaining natural ordering.

8. Features: A TreeMap has more features than a HashMap. It has methods like pollFirstEntry() , pollLastEntry() , tailMap() , firstKey() , lastKey() etc. that are not provided by a HashMap.

164. What are the differences between Comparable and Comparator?

Main differences between Comparable and Comparator are:

1. Type: Comparable<T> is an interface in Java where T is the type of objects that this object may be compared to.

2. Comparator<T> is also an interface where T is the type of objects that may be compared by this comparator.

3. Sorting: In Comparable, we can only create one sort sequence. In Comparator we can create multiple sort sequences.

4. Method Used: Comparator<T> interface in Java has method public int compare (Object o1, Object o2) that returns a negative integer, zero, or a positive integer when the object o1 is less than, equal to, or greater than the object o2. A Comparable<T> interface has method public int compareTo(Object o) that returns a negative integer, zero, or a positive integer when this object is less than, equal to, or greater than the object o.

5. Objects for Comparison: The Comparator compares two objects given to it as input. Comparable interface compares "this" reference with the object given as input.

6. Package location: Comparable interface in Java is defined in java.lang package. Comparator interface in Java is defined in java.util package.

165. In Java, what is the purpose of Properties file?

A Properties file in Java is a list of key-value pairs that can be parsed by java.util.Properties class.

Generally a Properties file has extension .properties e.g. myapp.properties.

Properties files are used for many purposes in all kinds of Java applications. Some of the uses are to store configuration, initial data, application options etc.
When we change the value of a key in a properties file, there is no need to recompile the Java application. So it provides benefit of changing values at runtime.

166. What is the reason for overriding equals() method?

The equals() method in Object class is used to check whether two objects are same or not. If we want a custom implementation we can override this method.

For example, a Person class has first name, last name and age. If we want two Person objects to be equal based on name and age, then we can override equals() method to compare the first name, last name and age of Person objects.

Generally in HashMap implementation, if we want to use an object as key, then we override equals() method.

167. How does hashCode() method work in Java?

Object class in Java has hashCode() method. This method returns a hash code value, which is an integer.

The hashCode() is a native method and its implementation is not pure Java.

Java doesn't generate hashCode(). However, Object generates a HashCode based on the memory address of the instance of the object.

If two objects are same then their hashCode() is also same.

168. Is it a good idea to use Generics in collections?

Yes. A collection is a group of elements put together in an order or based on a property. Often the type of element can vary. But the properties and behavior of a Collection remains same. Therefore it is good to create a Collection with Generics so that it is type-safe and it can be used with wide variety of elements.

```
List list = new ArrayList(); // before generics
list.add(10);
list.add("100");
List<Integer> list1 = new ArrayList<Integer>(); // adding generics
list1.add(10);
list1.add("100"); // compile-time error.
```

The generic collections disable the type-casting and there is no use of type-casting when it is used in generics. The generic collections are type-safe and checked at compile-time. These generic collections allow the datatypes to pass as parameters to classes

https://www.tutorialspoint.com/what-are-the-uses-of-generic-collections-in-java#:~:text=The%20generic%20collections%20are%20introduced,pass%20as%20parameters%20to%20classes.

169. What is the difference between Collections.emptyList() and creating new instance of Collection?

In both the approaches, we get an empty list. But Collections.emptyList() returns an Immutable list. We cannot add new elements to an Immutable empty list.

Collections.emptyList() works like Singleton pattern. It does not create a new instance of List. It reuses an existing empty list instance.

Therefore, Collections.emptylist() gives better performance if we need to get an emptyList multiple times.

170. How will you copy elements from a Source List to another list?

There are two options to copy a Source List to another list.

Option 1: Use ArrayList constructor

```
ArrayList<Integer> newList = new ArrayList<Integer>(sourceList);
```

Option 2: Use Collection.copy()
To use Collections.copy() destination list should be of same or larger size than source list.

```
ArrayList<Integer> newList = new ArrayList<Integer>
(sourceList.size());
Collections.copy(newList, sourceList);
```

Collections.copy() does not reallocate the capacity of destination List if it does not have enough space to contain all elements of source List. It throws IndexOutOfBoundsException.

The benefit of Collection.copy() is that it guarantees that the copy will happen in linear time. It is also good for the scenario when we want to reuse an array instead of allocating more memory in the constructor of ArrayList.

One limitation of Collections.copy() is that it can accept only List as source and destination parameters.

171. What are the Java Collection classes that implement List interface?

Java classes that implement List interface are:

- AbstractList
- AbstractSequentialList
- ArrayList
- AttributeList
- CopyOnWriteArrayList
- LinkedList
- RoleList
- RoleUnresolvedList
- Stack
- Vector

172. What are the Java Collection classes that implement Set interface?

Java classes that implement Set interface are:

- AbstractSet
- ConcurrentSkipListSet
- CopyOnWriteArraySet
- EnumSet
- HashSet
- JobStateReasons
- LinkedHashSet
- TreeSet

173. What is the difference between an Iterator and ListIterator in Java?

Iterator and ListIterator are two interfaces in Java to traverse data structures. The differences between these two are:

1. ListIterator can be used to traverse only a List. But Iterator can be used to traverse List, Set, and Queue etc.

2. An Iterator traverses the elements in one direction only. It just goes. ListIterator can traverse the elements in two directions i.e. backward as well as forward directions.

3. Iterator cannot provide us index of an element in the Data Structure. ListIterator provides us methods like nextIndex() and previousIndex() to get the index of an element during traversal.

4. Iterator does not allow us to add an element to collection while traversing it. It throws ConcurrentModificationException. ListIterator allows use to add an element at any point of time while traversing a list.

5. An existing element's value cannot be replaced by using Iterator. ListIterator provides the method set(e) to replace the value of last element returned by next() or previous() methods.

174. What is the difference between Iterator and Enumeration?

Both Iterator and Enumeration are interfaces in Java to access Data Structures. The main differences between these are:

1. Enumeration is an older interface. Iterator is a newer interface.

2. Enumeration can only traverse legacy collections. Iterator can traverse both legacy as well as newer collections.

3. Enumeration does not provide remove() method. So we cannot remove any element during traversal. Iterator provides remove() method.

4. Iterator is a fail-fast interface, it gives ConcurrentModificationException if any thread tries to modify an element in the collection being iterated. Enumeration is not fail-fast.

5. Method names in Iterator are shorter than in an Enumeration.

175. What is the difference between an ArrayList and a LinkedList data structure?

Main differences between ArrayList and LinkedList data structures are:

1. **Data Structure**: An ArrayList is an indexed based dynamic array. A LinkedList is a Doubly Linked List data structure.

2. **Insertion**: It is easier to insert new elements in a LinkedList, since there is no need to resize an array. Insertion in ArrayList is O(n), since it may require resizing of array and copying its contents to new array.

3. **Remove elements**: LinkedList has better performance in removal of elements than ArrayList.

4. **Memory Usage**: LinkedList uses more memory than ArrayList, since it has to maintain links for next and previous nodes as well.

5. **Access**: LinkedList is slower in accessing an element, since we have to traverse the list one by one to access the right location.

176. What is the difference between a Set and a Map in Java?

Main differences between a Set and a Map in Java are:

1. **Duplicate Elements**: A Set does not allow inserting duplicate elements. A Map does not allow using duplicate keys, but it allows inserting duplicate values for unique keys.

2. **Null values**: A Set allows inserting maximum one null value. In a Map we can have single null key at most and any number of null values.

3. **Ordering**: A Set does not maintain any order of elements. Some of sub-classes of a Set can sort the elements in an order like LinkedHashSet. A Map does not maintain any order of its elements. Some of its sub-classes like TreeMap store elements of the map in ascending order of keys.

177. What is the use of a Dictionary class?

The Dictionary class in Java is used to store key-value pairs. Any non-null object can be used for key or value. But we cannot insert a null key or null object in Dictionary.

Dictionary class is deprecated now. So it should not be used in newer implementations.

178. What is the default size of load factor in a HashMap collection in Java?

Default value of load factor in a HashMap is 0.75.

179. What is the significance of load factor in a HashMap in Java?

A HashMap in Java has default initial capacity 16 and the load factor is 0.75f (i.e. 75% of current map size). The load factor of a HashMap is the level at which its capacity should be doubled.

For example, in a HashMap of capacity 16 and load factor .75. The capacity will become 32 when the HashMap is 75% full. Therefore, after storing the 12th key– value pair (16 * .75 = 12) into HashMap, its capacity becomes 32.

180. What are the major differences between a HashSet and a HashMap?

The main difference between a HashSet and a HashMap are:

1. **Base class**: A HashSet class implements the Set interface. Whereas a HashMap class implements the Map interface.

2. **Storage**: A HashSet is used to store distinct objects. A HashMap is used for storing key & value pairs, so that these can be retrieved by key later on.

3. **Duplicate Elements**: A HashSet does not allow storing duplicate elements. A HashMap also does not allow duplicate keys. But we can store duplicate values in a HashMap.

4. **Null Elements**: In a HashSet we can store a single null value. In a HashMap we can store single null key, but any number of null values.

5. **Element Type**: A HashSet contains only values of objects as its elements. Whereas a HashMap contains entries(key value pairs).

6. **Iteration**: By using an Iterator we can iterate a HashSet. But a HashMap has to be converted into Set for iteration.

181. What are the similarities between a HashSet and a HashMap in Java?

As the name suggests, HashSet and HashMap are Hashing based collections. Similarities between HashSet and HashMap are:

1. **Thread Safety**: Both HashMap and HashSet are not synchronized collections. Therefore they are not good for thread-safe operations. To make these thread-safe we need to explicitly use synchronized versions.

2. **Order of Elements**: None of these classes guarantee the order of elements. These are unordered collections.

3. **Internal Implementation**: A HashMap backs up a HashSet internally. So HashSet uses a HashMap for performing its operations.

4. **Performance**: Both of these collections provide constant time performance for basic operations such as insertion and removal of elements.

182. What is the reason for overriding equals() method?

The equals() method in Object class is used to check whether two objects are same or not. If we want a custom implementation we can override this method.

For example, a Person class has first name, last name and age. If we want two Person objects to be equal based on name and age, then we can override equals() method to compare the first name, last name and age of Person objects.

Generally in HashMap implementation, if we want to use an object as key, then we override equals() method.

```
equals will take any Object as a parameter, but compareTo will only take Strings.
equals only tells you whether they're equal or not, but compareTo gives information
on how the Strings compare lexicographically.
compareTo -- Comparable;
compare -- comparator
```

183. How can we synchronize the elements of a List, a Set or a Map?

Sometimes we need to make collections Thread-safe for use in Multi-threading environment. In Java, Collections class provides useful static methods to make a List, Set or Map as synchronized collections. Some of these methods are:

static <T> Collection<T> synchronizedCollection(Collection<T> c)
Returns a synchronized (thread-safe) collection backed by the specified collection.

static <T> List<T> synchronizedList(List<T> list)
Returns a synchronized (thread-safe) list backed by the specified list.

static <K,V> Map<K,V>synchronizedMap(Map<K,V> m)
Returns a synchronized (thread-safe) map backed by the specified map.

static <T> Set<T> synchronizedSet(Set<T> s)
Returns a synchronized (thread-safe) set backed by the specified set.

static <K,V> SortedMap<K,V> synchronizedSortedMap(SortedMap<K,V> m)
Returns a synchronized (thread-safe) sorted map backed by the specified sorted map.

static <T> SortedSet<T> synchronizedSortedSet(SortedSet<T> s)
Returns a synchronized (thread-safe) sorted set backed by the specified sorted set.

184. What is Hash Collision? How Java handles hash-collision in HashMap?

In a Hashing scenario, at times two different objects may have same HashCode but they may not be equal. Therefore, Java will face issue while storing the two different objects with same HashCode in a HashMap. This kind of situation is Hash Collision.

There are different techniques of resolving or avoiding Hash Collision. But in HashMap, Java simply replaces the Object at old Key with new Object in case of Hash Collision.

185. What are the Hash Collision resolution techniques?

To resolve a Hash Collision we can use one of the following techniques:

- Separate Chaining with Linked List
- Separate Chaining with List Head Cells
- Open Addressing with Coalesced Hashing
- Open Addressing with Cuckoo Hashing
- Hopscotch Hashing
- Robinhood Hashing

186. What is the difference between Queue and Stack data structures?

Queue is a FIFO data structure. FIFO stands for First In First Out. It means the element added first will be removed first from the queue. A real world example of Queue is a line for buying tickets at a station. The person entering first in the Queue is served first.

Stack is a LIFO data structure. LIFO stands for Last In First Out. The element that is added last is removed first from the collection. In a Stack elements are added or removed from the top of stack.

A real world example of Stack is back button in browser. We can go back one by one only and it works in the reverse order of adding webpages to history .

187. What is an Iterator in Java?

Iterator is an interface in Java to access the elements in a collection. It is in java.util package.
It provides methods to iterate over a Collection class in Java.

Iterator interface in Java is based on Iterator design pattern. By using an Iterator one can traverse a container of objects and can also access the objects in the container. A container of objects is a Collection class in Java.

188. What is the difference between Iterator and Enumeration in Java?

Main differences between Iterator and Enumeration in Java are:

1. **Version**: Enumeration interface is in Java since JDK 1.0. Iterator interface was introduced in Java 1.2.

2. **remove() method**: The main difference between Enumeration and Iterator interface is remove() method. Enumeration can just traverse a Collection object. If we use Enumeration, we cannot do any modifications to a Collection while traversing the collection. Iterator interface provides remove() method to remove an element while traversing the Collection. There is not remove() method in Enumeration interface.

3. **Method names**: Names of methods in Iterator interface are hasNext(), next(), remove(). Names of methods in Enumeration interface are hasMoreElements(), nextElement().

4. **Legacy Interface**: Enumeration is considered as a legacy interface. It is used to traverse legacy classes like Vector, Stack and HashTable. Iterator is a newer interface that is used to traverse almost all of the classes in Java Collections framework.

5. **Fail-fast vs. Fail-safe**: Iterator is based on fail-fast principle. It throws ConcurrentModificationException if a collection is modified during iteration over that collection. An Enumeration is based on fail-safe principle. It doesn't throw any exception if a collection is modified during traversal.

6. **Safety**: Since Iterator is fail-fast and does not allow modification of a collection by other threads, it is considered safer than Enumeration.

189. What is the design pattern used in the implementation of Enumeration in Java?

Enumeration is based on Iterator design pattern. Iterator design pattern provides a common interface with methods to traverse the collection of objects. It hides the underlying implementation details of the collection.

190. Which methods do we need to override to use an object as key in a HashMap?

If we want to use an object as a key in a HashMap in Java, then we have to make sure that it has the implementation of equals() and hashCode() methods.

191. How will you reverse a List in Java?

In Collections class, Java provides a method reverse(List list) that can be used to reverse a List.

E.g.
Collections.reverse(myList);

192. How will you convert an array of String objects into a List?

Java provides Arrays class in java.util package. Arrays class has a method asList() that accepts an Array as input and returns a List as output.

public static <T> List<T> asList(T... a)

```
String[]  myArray =  {"George"  , "Jack" , "Ryan"};
List myList =  Arrays.asList(myArray);
```

193. What is the difference between peek(), poll() and remove() methods of Queue interface in java?

In a Java Queue, poll() and remove() methods can be used for removing the head object of Queue. The main difference arises in the case when Queue is empty().

If Queue is empty then poll() method returns null value. If Queue is empty then remove() method throws NoSuchElementException.

In a Java Queue, peek() method retrieves the head of Queue but it does not remove it. If queue is empty then peek() method returns null value.

194. What is the difference between Array and ArrayList in Java?

The main differences between Array and ArrayList in Java are:

1. **Size**: Array in Java is fixed in size. We cannot change the size of array after creating it. ArrayList is dynamic in size. When we add elements to an ArrayList, its capacity increases automatically.

2. **Performance**: In Java Array and ArrayList give different performance for different operations.

3. **add() or get():** Adding an element to or retrieving an element from an array or ArrayList object has similar performance. These are constant time operations.

4. **resize():** Automatic resize of ArrayList slows down the performance. ArrayList is internally backed by an Array. In resize() a temporary array is used to copy elements from old array to new array.

5. **Primitives**: Array can contain both primitive data types as well as objects. But ArrayList cannot contain primitive data types. It contains only objects.

6. **Iterator**: In an ArrayList we use an Iterator object to traverse the elements. We use for loop for iterating elements in an array.

7. **Type Safety**: Java helps in ensuring Type Safety of elements in an ArrayList by using Generics. An Array can

contain objects of same type of class. If we try to store a different data type object in an Array then it throws ArrayStoreException.

8. **Length**: Size of ArrayList can be obtained by using size() method. Every array object has length variable that is same as the length/size of the array.

9. **Adding elements**: In an ArrayList we can use add() method to add objects. In an Array assignment operator is used for adding elements.

10. Multi-dimension: An Array can be multi-dimensional. An ArrayList is always of single dimension.

195. How will you insert, delete and retrieve elements from a HashMap collection in Java?

We use following methods to insert, delete and retrieve elements in a HashMap.

1. **Retrieve**: We use get() method to retrieve elements from a HashMap.
   Value get(Object key)

2. **Insert**: We use put() method to insert a key value pair in a HashMap.
   Value put(Key k, Value v)

3. **Delete**: We use remove() method to delete key-value pair from the HashMap.
   Value remove(Object key)

196. What are the main differences between HashMap and ConcurrentHashMap in Java?

Main differences between HashMap and ConcurrentHashMap are:

1. **Synchronization**: A HashMap is not synchronized. But a ConcurrentHashMap is a synchronized object.
2. **Null Key**: A HashMap can have one null key and any number of null values. A ConcurrentHashMap cannot have null keys or null values.
3. **Multi-threading**: A ConcurrentHashMap works well in a multi-threading environment.

197. What is the increasing order of performance for following collection classes in Java?

The increasing order of performance is:

- Hashtable
- Collections.SynchronizedMap
- ConcurrentHashMap
- HashMap

Hashtable has the worst performance and HashMap has the best performance.

198. Why does Map interface not extend Collection interface in Java?

A Map is a collection objects. But Map interface is not compatible with Collection interface in Java.

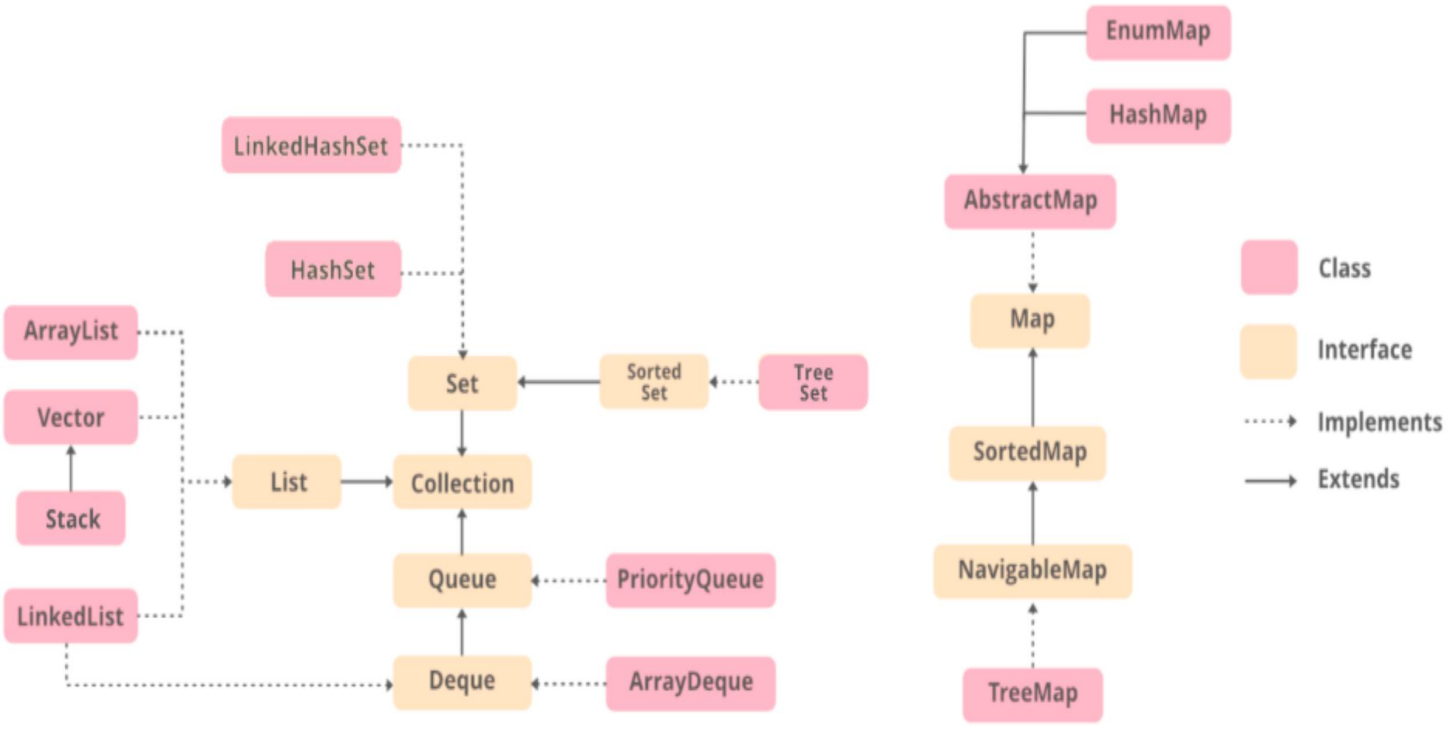
A Map requires key as well as a value. So it requires two parameters to add an element to a HashMap.

But Collection interface provides add(Object o) method with only one parameter.

Map collection has to provide methods like valueSet, keySet etc. These methods are specific to Map collection. Where as methods in Collection interface can be reused by a List, Set, Queue etc.

Collection assume elements of one value. Map assumes entries of key/value pairs. They could have been engineered to re-use the same common interface however some methods they implement are incompatible e.g.

```
Collection.remove(Object) - removes an element.
Map.remove(Object) - removes by key, not by entry.
```



199. What are the different ways to iterate elements of a list in Java?

There are mainly two ways to iterate the elements of list in Java:

1. **Iterator**: We can get an Iterator for list and use it to iterate the objects of the list.

2. **For-each loop**: We can use for-each loop to traverse all the elements of a list.

```
// List iterator
ListIterator<String> it = myList.listIterator();

// Condition check whether there is element in List
// using hasNext() which holds true till
// there is single element in List
while (it.hasNext()) {

    // Print all elements of List
    System.out.println(it.next());
}
```

https://www.geeksforgeeks.org/iterate-through-list-in-java/

200. What is CopyOnWriteArrayList? How it is different from ArrayList in Java?

CopyOnWriteArrayList was introduced in Java 5 version. It is a thread-safe collection. It is similar to an ArrayList.

In CopyOnWriteArrayList, all mutative operations (add, set etc.) are implemented by making a fresh copy of the underlying array.

Iterator of CopyOnWriteArrayList is guaranteed to not throw ConcurrentModificationException. But Iterator also does not reflect any additions, removals that happened to list after the Iterator was created.

All elements including null are permitted in CopyOnWriteArrayList.

The design of the CopyOnWriteArrayList uses an interesting technique to make it thread-safe without a need for synchronization. When we are using any of the modify methods – such as add() or remove() – the whole content of the CopyOnWriteArrayList is copied into the new internal copy.

When we're calling the iterator() method on the CopyOnWriteArrayList, we get back an Iterator backed up by the immutable snapshot of the content of the CopyOnWriteArrayList.

201. How remove() method is implemented in a HashMap?

Remove() method in HashMap uses logic similar to the one used in get() method. First we locate the correct bucket in HashMap for an entry. Then within that bucket we remove the element e. It is similar to removing a node from a single-linked list.

If e is the first element in the bucket we set the corresponding element of Hash to e.next. Else we set the next field of the element just before e to e.next.

202. What is BlockingQueue in Java Collections?

BlockingQueue was introduced in Java 1.5. It extends Queue interface in Java.

BlockingQueue supports operations that wait for the queue to become non-empty when retrieving an element. Also it supports the operations that wait for space to become available in the queue while storing an element.

Some of the features of BlockingQueue are:

- It does not accept null elements.
- Its main use is in producer-consumer problems.
- BlockingQueue implementation is thread-safe.
- It can be used in inter-thread communications.
- It does not support any kind of "close" or "shutdown" operation to indicate that no more items will be added.

203. How is TreeMap class implemented in Java?

Internally, a TreeMap class in Java uses Red-Black tree.

It is a NavigableMap. The map sorts the keys in natural order or it can use a Comparator supplied at the creation time.

The implementation of TreeMap is not synchronized in Java.

204. What is the difference between Fail-fast and Fail-safe iterator in Java?

Differences between Fail-fast and Fail-safe iterators are as follows:

Fail-fast iterator throws ConcurrentModificationException. But Fail-safe iterator does not throw this exception.

Fail-fast iterator does not clone the original collection. Fail-safe iterator creates a copy of the original collection of objects.

A Fail-fast iterator tries to immediately throw Exception when it encounters failure. A Fail-safe Iterator works on a copy of collection instead of original collection.

205. How does ConcurrentHashMap work in Java?

ConcurrentHashMap extends AbstractMap in Java. It was introduced in Java 1.5. It provides concurrency in a collection based on a HashMap.

All methods are thread-safe in ConcurrentHashMap.

Internally there is a Hashtable backing a ConcurrentHashMap. This Hashtable supports the concurrent methods for retrieval of data as well as updates on ConcurrentHashMap.

It has same functional specification as a Hashtable.

It also supports a set of sequential and bulk operations. These operations accept parallelismThreshold argument.

206. What is the importance of hashCode() and equals() methods?

In a HashMap collection it is very important for a key object to implement hashCode() method and equals() method. If hashCode() method returns same hashcode for all key objects then the hash collision will be high in HashMap. Also with same hashcode, we will get same equals method that will make our HashMap inefficient.

The problem arises when HashMap treats both outputs same instead of different. It will overwrite the most recent key-value pair with the previous key-value pair.

So it is important to implement hashCode() and equals() methods correctly for an efficient HashMap collection.

207. What is the contract of hashCode() and equals() methods in Java?

Contract of hashCode() and equals() methods is as follows in Java:

If object1.equals(object2), then object1.hashCode() == object2.hashCode() should always be true. It means if two objects are equal then their hashCode should be same.

If object1.hashCode() == object2.hashCode() is true, it does not guarantee that object1.equals(object2). It means if two objects have same hashCode, then can still have different values so that may not be equal objects.

208. What is an EnumSet in Java?

Set: EnumSet is a specialized implementation of Set.

1. **Use**: It is mainly used with enum types.

2. **Single enum type**: All the elements in an EnumSet must come from a single enum type when the set is created.

3. **Bit vector**: Internally, EnumSet is represented as bit vector.

4. **Iterator**: The iterator of EnumSet traverses the elements in their natural order. (It is the order in which the enum constants are declared).

5. **Null**: In an EnumSet, null elements are not permitted. If we try to insert a null element it throws NullPointerException.

6. **Thread-safe**: EnumSet is not a synchronized collection. For use in multi-threading scenarios, EnumSet should be synchronized.

7. **Bit flags**: EnumSet is a very good alternative to int based “bit flags” implementation.

209. What are the main Concurrent Collection classes in Java?

Java 1.5 has provided new package java.util.concurrent. This package contains thread-safe collection classed. These collection classes can be modified while iterating. The iterator of these classes is fail-safe.

Main Concurrent Collection classes in Java 8 are:

- ArrayBlockingQueue
- CopyOnWriteArrayList
- CopyOnWriteArraySet
- ConcurrentHashMap
- ConcurrentLinkedDeque
- ConcurrentLinkedQueue
- LinkedBlockingQueue
- LinkedBlockingDeque
- PriorityBlockingQueue

210. How will you convert a Collection to SynchronizedCollection in Java?

Java provides an easy method in java.utils.Collections class to create a ThreadSafe collection from a regular collection.

We can use the method synchronizedCollection() for this purpose.

For any class of type T we can use following method:

static <T> Collection<T> synchronizedCollection(Collection<T> c)

211. How IdentityHashMap is different from a regular Map in Java?

IndentityHashMap in Java implements Map interface. But it is not a general purpose implementation. It violates the general contract of Map interface by a different implementation of equals() method.

In an IdentityHashMap, two keys k1 and k2 are equal if and only if (k1==k2). (In a normal Map implementation (like HashMap) two keys k1 and k2 are considered equal if and only if (k1==null ? k2==null : k1.equals(k2)).)

It implements the Map interface with a hash table, using reference-equality in place of object-equality when comparing keys (and values).

212. What is the main use of IdentityHashMap?

Main uses of IdentityHashMap are:

1. Topology Preservation: The typical use of IdentityHashMap class is topology-preserving object graph transformations, such as serialization or deep-copying. In such a scenario, a program must maintain a "node table" to keep track of all the object references that have already been processed.

2. The node table should not considered distinct objects as equal even if they happen to be equal.

3. Proxy objects: Another use of this class is to maintain proxy objects. A debugging program has to maintain a proxy object for each object in the program being debugged.

213. How can we improve the performance of IdentityHashMap?

IdentityHashMap class has one tuning parameter for performance improvement: expectedMaxSize.

This parameter is the maximum number of key-value mappings that the map is expected to hold.

We can use this parameter is used to determine the number of buckets initially in the hash table. The precise relationship between the expected maximum size and the number of buckets is unspecified.

If the number of key-value mappings exceeds the expected maximum size, the number of buckets is increased.

Increasing the number of buckets is also known as rehashing. Rehashing may be fairly expensive. So it is better to create identity hash maps with a sufficiently large expected maximum size.

But iteration over a Map collection requires time proportional to the number of buckets in the hash table. So iteration may take extra time due to large number of buckets.

Therefore the value of expectedMaxSize should be set in consideration with both of these aspects.

214. Is IdentityHashMap thread-safe?

The implementation of IdentityHashMap is not thread-safe, since its methods are not synchronized.

The iterators returned by the iterator method of IdentityHashMap are fail-fast. But the fail-fast behavior of an iterator cannot be guaranteed.

Since the Iterator is fail-fast, it throws ConcurrentModificationException.

215. What is a WeakHashMap in Java?

WeakHashMap is a class similar to IdentityHashMap.

Internally, it is represented by a Hashtable.

It is not a synchronized class. We can make a WeakHashMap thread safe by using Collections.synchronizedMap() method.

An entry in WeakHashMap is automatically removed when it is no longer in ordinary use.

The presence of a mapping for a given key does not prevent the key from being discarded by the garbage collector.

WeakHashMap also permits null keys and null values.

216. How can you make a Collection class read Only in Java?

In Java, there are useful methods to make a Collection class read Only. We can make the Collection read Only by using one of the following methods:

- Collections.unmodifiableMap(Map m)
- Collections.unmodifiableList(List l)
- Collections.unmodifiableSet(Set s)
- Collections.unmodifiableCollection(Collection c)

```
Collections.unmodifiable........
```

217. When is UnsupportedOperationException thrown in Java?

In a Java collection UnsupportedOperationException is thrown when the requested operation is not supported by the collection.

It is an unchecked exception that is thrown on optional operations.

If there is an optional add() or remove() methods in a read only collection, then this exception can be thrown.

218. Let say there is a Customer class. We add objects of Customer class to an ArrayList. How can we sort the Customer objects in ArrayList by using customer firstName attribute of Customer class?

There are two ways to handle this scenario. We can use these options:
Comparable: Implement the Comparable interface for Customer class and compare customer objects by firstName attribute.

Comparator: Implement Comparator for comparing two Customer objects on the basis of firstName attribute. Then use this comparator object in sort method of Collections class.

219. What is the difference between Synchronized Collection and Concurrent Collection?

In Java 1.5 many Concurrent collection classes were added in SDK. These are ConcurrentHashMap, CopyOnWriteArrayList, BlockingQueue etc.

Java also provides utility methods to get a synchronized copy of collection like ArrayList, HashMap etc. by using Collections.synchronizedList(), Collections.synchronizedMap() methods.

The main difference is in performance. Concurrent collection classes have better performance than synchronized collection classes because they lock only a portion of the class to achieve concurrency and thread-safety.

220. What is the scenario to use ConcurrentHashMap in Java?

ConcurrentHashMap is more suited for scenarios where we have multiple reader threads and one writer thread. In this case map is locked only during the write operation.

If we have an equal number of reader and writer threads then ConcurrentHashMap performance is similar to a Hashtable or a synchronized HashMap.

221. How will you create an empty Map in Java?

There are two ways to create an empty Map in Java.

1. **Immutable**: If we want an immutable empty Map, we can use following code:

   myMap = Collections.emptyMap();

2. **Any map**: For all other scenarios, we can use following code by using new method:

myMap = new HashMap();

222. What is the difference between remove() method of Collection and remove() method of Iterator?

In Collection interface remove(Object o) method is used to remove objects from a Collection.

List interface also provides remove(int index) method to remove an object at a specific index.

These methods are used to remove an entry from Collection, while no thread is iterating over it.

When we are iterating over a Collection, then we have to remove() method of Iterator. This method removes current element from Iterator's point of view. If we use remove(0 method of Collection or List, then we will get ConcurrentModificationException.

Therefore, it is recommended to use remove() method of Iterator during the traversal of a Collection by an Iterator.

223. Between an Array and ArrayList, which one is the preferred collection for storing objects?

An ArrayList is backed up by array internally. There are many usability advantages of using an ArrayList over an array in Java.

Array has a fixed length at the time of creation. Once it is created we cannot change its length.

ArrayList is dynamic in size. Once it reaches a threshold, it automatically allocates a new array and copies contents of old array to new array.

Also ArrayList provides support of Generics. But Array does not support Generics.

E.g. If we store an Integer object in a String array at Runtime it will throw ArrayStoreException. Whereas, if we use ArrayList then as compile time we will get the error. This helps in preventing errors from happening at runtime.

If we know the size in advance and do not need re-sizing the collection then Array should be used in place of an ArrayList.

224. Is it possible to replace Hashtable with ConcurrentHashMap in Java?

Yes, a ConcurrentHashMap can be replaced with Hashtable in Java.

But it requires careful observation, since locking behavior of Hashtable is different than that of ConcurrentHashmap.

A Hashtable locks whole Map instead of a portion of Map. Compound operations like if(Hashtable.get(key) == null) put(key, value) work in Hashtable but not in ConcurrentHashMap.

In a ConcurrentHashMap we use putIfAbsent() method for such a scenario.

225. How CopyOnWriteArrayList class is different from ArrayList and Vector classes?

CopyOnWriteArrayList was introduced in Java 1.5. It implements List interface.

It provides better concurrent access methods than a Synchronized List.

In CopyOnWriteList, concurrency is achieved by copying ArrayList over each write and replace with original instead of locking.

CopyOnWriteArrayList also does not throw any ConcurrentModification Exception during Iteration.

It is a thread-safe list.

It is different from a Vector in terms of Concurrency. CopyOnWriteArrayList provides better Concurrency by reducing contention among readers and writers.

226. Why ListIterator has add() method but Iterator does not have?

ListIterator can iterate in the both directions of a Collection. It maintains two pointer for previous and next element. In ListIterator we can use add() method to add an element into the list immediately before the element returned by next() method.

So a subsequent call to next() method will not be affected. And the call to previous() method will return the newly added element.

In Iterator we can only traverse in one direction. So there is no purpose of add() method there.

227. Why do we sometime get ConcurrentModificationException during iteration?

When we remove an object by using remove() method of a Collection or List while an Iterator thread is traversing it, we get ConcurrentModificationException. If an Iterator detects any structural change in Collection it can throw ConcurrentModificationException.

228. How will you convert a Map to a List in Java?

In Java, a Map has three collection sets:

key set
value set
key-value set

Each of these Sets can be converted to List by using a constructor.

Sample code is as follows:

```
List keyList = new ArrayList(map.keySet());
List valueList = new ArrayList(map.values());
List entryList = new ArrayList(map.entrySet());
```

229. How can we create a Map with reverse view and lookup in Java?

In a Map we can lookup for a value by using a distinct key. In a Map with reverse view and lookup, even the values are distinct. So there is one to one mapping between keys and values and vice version.

If we enable this constraint on a Map then we can look up a key by its value. Such data structure is called bi-directional map.

There is no built data structure similar to reverse lookup Map in JDK.

But Apache Common Collections and Guava libraries provide implementation of bidirectional map. It is called BidiMap and BiMap. Both of these data structure enforce the constraint of one to one mapping between keys and values.

230. How will you create a shallow copy of a Map?

In Java, most implementations of Map interface provide a constructor to create copy of another map. But the copy method is not synchronized.

Therefore, when a thread is copying the map, another thread can modify it.

To prevent such a scenario, we should use Collections.synchronizedMap() method to first create a thread-safe map.

Another way of to create a shallow copy is by using clone() method. But it is not considered as a recommended approach.

231. Why we cannot create a generic array in Java?

Java does not allow creation of array with generics as elements.

In Java an array has to know the type information of its elements at runtime.

This information is used at runtime to throw ArrayStoreException if data type of an element to be inserted does not match the type of Array.

In case of Generics, the type information of a collection is erased at runtime by Type Erasure. Due to this array cannot use generics as elements.

https://www.javatpoint.com/generics-in-java#:~:text=The%20Java%20Generics%20programming%20is,a%20specific%20type%20of%20objects.

232. What is a PriorityQueue in Java?

A PriorityQueue is data structure based on Queue. Unlike Queue, the elements on PriorityQueue are not returned in FIFO order.

A PriorityQueue maintains the natural order of its elements or it uses a Comparator provided at initialization.

It is an unbounded queue based on a priority heap.

PriorityQueue does not allow null values. We cannot add any object that does not provide natural ordering to PriorityQueue.

PriorityQueue in Java is not thread-safe.

It gives O(log n) time for enqueing and dequeing operations.

233. What are the important points to remember while using Java Collections Framework?

Some of the important points to remember while using Java Collections Framework are:

1. **Interfaces**: For Collections, we should write code with generic interfaces instead of concrete implementation. Due to this we maintain the flexibility of changing the implementation at a later point of time.

2. **Generics**: We should use Generics for type-safety and to avoid ClassCastException at runtime.

3. **Collections**: It is recommended to use Collections utility class for algorithms and various other common methods for Collections.

4. **Right Type**: We have to choose the right type of Java collection based on our need. If size is fixed, we can use Array over ArrayList. If we do not want duplicate elements we use Set.

   If we need the ability to iterate the elements of a Map in the order of insertion then we use a TreeMap.

5. **Initial Size**: In some collection classes we can specify the initial size/capacity. Therefore we should have an estimate of number of elements in a Collection before deciding the right collection type. We can use it to avoid rehashing or resizing.

6. **Map**: We should use immutable classes provided by Java as key elements in a Map.

234. How can we pass a Collection as an argument to a method and ensure that method will not be able to modify it?

To ensure that a method is not able to modify a Collection passed as an argument, we have to make the Collection read only.

We can make a read only collection by using Collections.unmodifiableCollection(Collection c) method.

This will make sure that any operation to change the collection will throw UnsupportedOperationException.

235. Can you explain how HashMap works in Java?

In Java, a HashMap works on the concept of hashing.

A HashMap in Java stores both key and value objects, in a bucket. It is stored as an Entry object that implements Map.Entry interface.

The key object used in a HashMap has to provide implementation for hashCode() and equals() methods.

When put() method is used to store a key-value pair, the HashMap implementation calls hashCode() method on Key object to calculate a hash that is used to find a bucket where Entry object will be stored.

When get() method is used to retrieve a value stored against a key object, we first calculate a hash of Key object. Then we use this hash to find the bucket in which that particular key is stored.

Once Key object's location is found, it may happen that more than one Key is stored in same location. So now we use equals() method to find the exact Key object. Once the exact Key object is found we use it to get Value object.

236. Can you explain how HashSet is implemented in Java?

Internally, a HashSet uses a HashMap to store the elements and to maintain the uniqueness of elements.

When we create a HashSet object, a corresponding HashMap object is also created.

When we insert an element in HashSet, it inserts it into corresponding HashMap.

237. What is a NavigableMap in Java?

As the name suggests, NavigableMap provides the capability to navigate the keys of a Map in Java. A NavigableMap extends SortedMap interface.

Some of the interesting methods of a NavigableMap are descendingKeySet(), descendingMap(), headMap() and tailMap().

238. What is the difference between descendingKeySet() and descendingMap() methods of NavigableMap?

The descendingKeySet() method of NavigableMap returns a NavigableSet in which the elements are stored in reversed order as compared to the original key set.

The returned view is internally represented by the original KeySet of NavigableMap. Therefore any changes to the descending set also get reflected in the original set.

But it is not recommended to remove elements directly from the key set. We should use the Map.remove() method.

The descendingMap() method of NavigableMap returns a NavigableMap which is an inverse view of the original Map. The order of the elements in this view are in reverse order of the elements in original map. Any changes to this view are also reflected in the original map.

239. What is the advantage of NavigableMap over Map?

The main advantage of NavigableMap over Map is the Navigation capability.

It provides the capabilities of a Map, SortedMap and navigation in one collection.

It even returns the closest matches for given search targets.

Methods like lowerEntry, floorEntry, ceilingEntry, and higherEntry return Map.Entry objects associated with keys respectively less than, less than or equal, greater than or equal, and greater than a given key.

Methods like lowerKey, floorKey, ceilingKey, and higherKey return only the associated keys. All of these methods are designed for locating, not traversing entries.

240. What is the difference between headMap(), tailMap() and subMap() methods of NavigableMap?

The headMap() method returns a view of the original NavigableMap that contains the elements that are less than a given element.

```
NavigableMap original = new TreeMap();
original.put("1", "1");
original.put("2", "2");
original.put("3", "3");

//this headmap1 will contain elements "1" and "2"
SortedMap headmap1 = original.headMap("3");

//this headmap2 will contain elements "1", "2", and "3" because "inclusive"=true
NavigableMap headmap2 = original.headMap("3", true);
```

The tailMap() method works similar to headMap() method, but it returns all elements that are higher than the given input element.

The subMap() method accepts two parameters demarcating the boundaries of the view map to return.

All the three methods return a subset of the original map in a view form.

241. How will you sort objects by Natural order in a Java List?

We can use Collections.sort method to sort the elements of a List in natural order. To use this method, we have to make sure that element objects implement compareTo() method.

We can also use a Comparator to define the natural ordering for elements of a List. Then we can use this Custom Comparator in sort method of Collections class.

242. How can we get a Stream from a List in Java?

From Java 8 onwards it is a very easy to get a Stream from a List. We can just use stream() method to get a stream from a list of elements.

243. Can we get a Map from a Stream in Java?

Yes, we can create a Map from the elements of a Stream. We can use map() method to get a Map.

E.g. items.stream()
    .map(item -> item.toLowerCase())

In this example we are creating a map with each item object mapped to its LowerCase equivalent.

This is also used in Map-Reduce implementation on a Stream.

244. What are the popular implementations of Deque in Java?

The two most popular implementation of Deque interface in Java are:

1. **ArrayDeque**: It is a resizable array implementation of Deque. The capacity of ArrayDeque can increase based on the need of the program. It is not thread safe implementation. Also the iterator on ArrayDeque is fail-fast.

2. **LinkedList**: This is another popular implementation of Deque interface in Java. It is also not synchronized, so it is not thread-safe. It mainly provides functionality of a doubly linked list.

Multi-threading

245. What is a Thread in Java?

A thread in Java is a lightweight process that runs within another process or thread.

It is an independent path of execution in an application. JVM gives each thread its own method-call stack.

When we start JVM, Java starts one thread. This thread calls the main method of the class passed in argument to java call.

246. What is the priority of a Thread and how it is used in scheduling?

In Java, every Thread has a priority. This priority is specified as a number between 1 to 10.

Scheduler in Java schedules different threads based on the priority of a thread. It is also known as pre-emptive scheduling.

The thread with higher priority gets preference in execution over a thread with lower priority.

247. What is the default priority of a thread in Java?

In Java, a new thread gets the same priority as the priority of the parent thread that creates it.

Default priority of a thread is 5 (NORM_PRIORITY).

248. What are the three different priorities that can be set on a Thread in Java?

We can set following three priorities on a Thread object in Java:

1. MIN_PRIORITY: This is the minimum priority that a thread can have.

2. NORM_PRIORITY: This is the default priority that is assigned to a thread.

3. MAX_PRIORITY: This is the maximum priority that a thread can have.

Default priority of a thread is 5 NORM_PRIORITY. The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

The join() method of thread class waits for a thread to die. It is used when you want one thread to wait for completion of another. This process is like a relay race where the second runner waits until the first runner comes and hand over the flag to him.

249. What is the purpose of join() method in Thread class?

In Java, Thread Scheduler controls thread scheduling. But we can use join() method on a thread to make current thread to wait for another thread to finish.

When we use join(), the current thread stops executing. It wait for the thread on which join() is called to finish.

This makes sure that current thread will continue only after the thread it joined finished running. Consider following example:

```
Public class ThreadJoin {
 Thread importantThread = new Thread(
            new Runnable() {
               public void run () {
                  //do something
               }
            }
  );
  Thread currentThread = new Thread(
            new Runnable() {
               public void run () {
                  //do something
               }
            }
  );
  importantThread.start(); // Line 1
  importantThread.join();  // Line 2
  currentThread.start(); // Line 3
}
```

In the above example, main thread is executing. On Line 1, a new thread called importantThread is ready to run. But at Line 2, main

thread joins the importantThread. Now it lets importantTread to finish and then it moves to Line 3. So currentThread at Line 3 will not start till the importantThread has finished.

250. What is the fundamental difference between wait() and sleep() methods?

The main difference between wait() and sleep() is that wait is an Object level method, whereas sleep() is a static method in Thread class. A waiting thread can be woken up by another thread by calling notify() on the monitor which is being waited on. But a sleeping thread cannot be woken up.

A wait() and notify() has to happen within the same block that is synchronized on the monitor object.

When we call wait() the current thread releases the monitor and goes to waiting state. Then another thread calls notify() to wake it up.

In case of sleep() current thread does not release the monitor or locks. It just sleeps for some pre-defined time period.

251. Is it possible to call run() method instead of start() on a thread in Java?

Yes. We can call run() method of a thread. But it does not work as a separate thread. It will just work as a normal object in main thread and there will not be context switching between the threads.

If you directly call run() method its body is executed in context of current thread.
When you invoke start() method a new thread is created
and run() method is executed in this new thread.

The run() method is just an ordinary method (overridden by you). As with any other ordinary method and calling it directly will cause the current thread to execute run().

252. How Multi-threading works in Java?

Java provides support for Multithreading. In a Multithreading environment, one process can execute multiple threads in parallel at the same time.

In Java, you can create process and then create multiple threads from that process. Each process can execute in parallel to perform independent tasks.

Java provides methods like- start(), notify(), wait(), sleep() etc. to maintain a multi-threading environment.

253. What are the advantages of Multithreading?

Main advantages of Multithreading are:

1. Improved performance: We can improve performance of a job by Multi-threading.
2. Simultaneous access to Multiple Applications: We can access multiple applications from a process by doing multithreading
3. Reduced number of Servers required: With Multi-threading we need lesser number of servers, since one process can spawn multiple threads.
4. Simplified Coding: In certain scenarios, it is easier to code multiple threads than managing it from same thread.

254. What are the disadvantages of Multithreading?

There are certain downsides to Multithreading. These are:

1. Difficult to Debug: Multithreading code is difficult to debug in case of an issue.
2. Difficult to manage concurrency: Due to multiple threads, we may experience different kinds of issues.
3. Difficulty of porting code: It is difficult to convert existing single threaded code into multi-threading code.
4. Deadlocks: In case of multi-threading we can experience deadlocks in threads that are waiting for same resource.

255. What is a Thread in Java?

In Java, a thread is a lightweight process that runs within another process or thread. It is an independent path of execution in an application. Each thread runs in a separate stack frame.

By default Java starts one thread when the main method of a class is called.

256. What is a Thread's priority and how it is used in scheduling?

In Java, every Thread has a priority. This priority is specified as an integer value. The priority value is used in scheduling to pick up the thread with higher priority for execution. The threads with higher priority get more preference in execution than the threads with lower priority.

The task scheduler schedules the higher priority threads first, followed by the lower priority threads.

Threads with higher priority values are generally given more CPU time and resources.
higher-priority threads have a better chance of being scheduled for execution when the CPU is available

257. What are the differences between Pre-emptive Scheduling Scheduler and Time Slicing Scheduler?

In Pre-emptive scheduling, the highest priority task will keep getting time to execute until it goes to waiting state or dead state or a task with higher priority comes into queue for scheduling.

In Time slicing scheduling, every task gets a predefined slice of time for execution, and then it goes to the pool of tasks ready for execution. The scheduler picks up the next task for execution, based on priority and various other factors.

258. Is it possible to call run() method instead of start() on a thread in Java?

Yes. We can call run() method of a thread. But it does not work as a separate thread. It will just work as a normal object in main thread and there will not be context-switching between the threads.

259. How will you make a user thread into daemon thread if it has already started?

No. We cannot make a user thread to daemon thread once it has already started.

If we do it by calling setDaemon(), it will throw IllegalThreadStateException

Daemon thread in Java is **a low-priority thread** that performs background operations such as **garbage collection**, **finalizer**, **Action Listeners**, **Signal dispatches**, etc.

260. Can we start a thread two times in Java?

No. We can call start() method only once on a thread in Java. If we call it twice, it will give us exception.

261. In what scenarios can we interrupt a thread?

We can interrupt a thread if we want to wake it up from the sleep or wait state.

262. In Java, is it possible to lock an object for exclusive use by a thread?

Yes. We can use synchronized block to lock an object. The locked object is inaccessible to any other thread. Only the thread that has locked it can access it.

263. How notify() method is different from notifyAll() method?

In Java, notify() method is used to unblock a specific thread that is in waiting stated. Whereas, notifyAll() method is used to unblock all the threads that are in waiting state.

264. What is a daemon thread in Java?

A daemon thread in Java is a low priority thread that does not prevent the JVM from exiting when the program finishes. The thread keeps running. Garbage Collection is an example of daemon thread.

265. How can we make a regular thread Daemon thread in Java?

We can call setDaemon(boolean) method to change a thread to daemon thread before the thread starts.

266. How will you make a user thread into daemon thread if it has already started?

No. We cannot make a user thread to daemon thread once it has already started. If we do it by calling setDaemon(), it will throw IllegalThreadStateException

267. Can we start a thread two times in Java?

No. We can call start() method only once on a thread in Java. If we call it twice, it will give us exception.

268. What is a Shutdown hook in Java?

The shutdown hook is a thread that is invoked implicitly by JVM just before the shut down. It can be used to clean up unused resources etc.

We can use java.lang.Runtime.addShutdownHook(Thread hook) method to register a new virtual-machine shutdown hook.

269. What is synchronization in Java?

The concept of Synchronization in Java is used in Multi-threading programming.

It is a feature in Java that helps in controlling the access of multiple threads to a shared resource.

It is used to prevent Deadlock between multiple threads.

270. What is the purpose of Synchronized block in Java?

Synchronized block has many uses in Java multi-threading environment. Some of the uses are:

It can prevent thread interference

It is also used to avoid memory inconsistency issues

In general, scope of synchronized block is smaller than the scope of a method.

271. What is static synchronization?

We can make a static method as synchronized in Java. Adding synchronized keyword to a static method can do this.

In static synchronization, the lock is on class not on object.

272. What is a Deadlock situation?

A Deadlock is a situation in which two or more threads are waiting on each other to release a resource. Each thread is waiting for a resource that is held by the other waiting thread.

At times there is a circular wait when more than two threads are waiting on each other's resources.

273. What is the meaning of concurrency?

Concurrency is the ability of a program to execute several programs simultaneously. This is achieved by distributing computations over multiple CPU cores of a machine or even over different machines within the same network.

It can increase the speed of execution of the overall program in multi-processor or multi-core system.

274. What is the main difference between process and thread?

As such both process and thread are independent sequences of execution.

The main difference is that a thread runs in a shared memory space, where as a process runs in its own memory space.

A process runs the execution in an environment provided by the operating system. A process has its own set of private resources (e.g. memory, open files, etc.).

A thread lives within a process and shares the resources like-memory, open files etc. with the other threads of the same process.

This ability to share resources between different threads makes thread more suitable for tasks where performance is a significant factor.

275. What is a process and thread in the context of Java?

In Java, a process refers to the running of Java Virtual Machine (JVM). But a thread lives within a JVM and it can be created or stopped by the Java application at runtime.

276. What is a Scheduler?

A scheduler is a program that is the implementation of a scheduling algorithm to manage access of processes and threads to limited resource like CPU or an I/O channel.

The goal of most scheduling algorithms is to provide load balancing for the available processes/threads and to guarantee that each process/thread will get a reasonable time frame to access the requested resource exclusively.

277. What is the minimum number of Threads in a Java program?

In a JVM, each Java program is executed within the main process that starts with java.exe. Therefore each Java application has at least one thread.

278. What are the properties of a Java thread?

Each Java thread has following properties:

1. **Identifier**: An identifier of type long that is unique within the JVM
2. **Name**: A name of type String
3. **Priority**: Priority of type int
4. **State**: A state of type java.lang.Thread.State
5. **Group**: A thread group the thread belongs to

279. What are the different states of a Thread in Java?

Following are the different states of a Thread in Java:

1. **New**: In the New state the thread has not yet.

2. **Runnable**: A thread executing in the JVM is in Runnable state.

3. **Blocked**: A thread waiting for a monitor lock is in Blocked state.

4. **Waiting**: A thread waiting indefinitely for another thread to perform a particular action is in Waiting state.

5. **Timed_waiting**: A thread waiting for another thread to perform an action for up to a specified waiting time is in Timed_waiting state.

6. **Terminated**: A thread that has exited is in Terminated state.

280. How will you set the priority of a thread in Java?

The priority of a thread in Java can be set by using setPriority(int priority) method.

We can use constant Thread.MAX_PRIORITY to set the maximum priority of a thread.

We can use constant Thread.MIN_PRIORITY to set the minimum priority of a thread.

Or we can use constant Thread.NORM_PRIORITY to set the default priority of a thread.

281. What is the purpose of Thread Groups in Java?

In Java, every thread belongs to a group of threads.

The JDK class java.lang.ThreadGroup provides methods to handle a whole group of Threads.

With the help of these methods we can interrupt all threads of a group or set the maximum priority of all threads of a group.

So a thread group is used for taking collective actions on a group of threads.

282. Why we should not stop a thread by calling its stop() method?

The stop() method in Thread class is a deprecated method. Its use is not recommended.

When we call stop() method, the thread unlocks all monitors that it has acquired. If any locked object was in an inconsistent state, this state gets visible to all other threads.

It can cause unexpected behavior when other threads work on this inconsistent object.

So calling stop() method to stop a thread is not advisable.

283. How will you create a Thread in Java?

There are two main ways to create a thread in Java.

1. **Extend Thread class**: We can extend java.lang.Thread class and implement run() method. On calling start() method it will start a new thread.

2. **Implement Runnable interface**: We can implement java.lang.Runnable interface and pass the implemented object to the constructor of java.lang.Thread class. On calling start() it will start a new thread.

284. How can we stop a thread in the middle of execution in Java?

We can use a volatile variable as an indicator to stop the thread.

We can create a volatile reference pointing to the current thread. This reference can be set to null by other threads to flag that the current thread should stop execution.

In following example threadStopper is the volatile reference that can be set as null in stopThread() method by other threads.

Sample code is as follows:

```
public static class MyThread extends Thread {

   private volatile Thread threadStopper;

   public void start() {
      threadStopper = new Thread(this);
      threadStopper.start();
   }

   public void stopThread() {
      threadStopper = null;
   }

   public void run() {
      Thread currThread = Thread.currentThread();
      while(currThread == threadStopper) {
         try {
            Thread.sleep(100);
         } catch (InterruptedException e) {
         }
      }
```

```
    }
}
```

285. How do you access the current thread in a Java program?

We can access the current thread in Java by calling the static method currentThread() of java.lang.Thread class.

Sample code is as follows:

```
public class MyThread {

   public static void main(String[] args) {
   // Get ID of Current Thread
      long id = Thread.currentThread().getId();

   // Get Name of Current Thread
      String name = Thread.currentThread().getName();
   }
}
```

286. What is Busy waiting in Multi-threading?

Busy waiting is also known as busy-looping or spinning. It is a multi-threading technique in which a process repeatedly checks if a condition is true.

For example, a process can keep checking if any keyboard input is available.

In general, busy waiting is considered as Anti-pattern that wastes processor time, so it should be avoided.

Sample code for busy waiting is as follows:

```
Thread thread = new Thread(new Runnable() {
  @Override
  public void run() {
    long timeToStop = System.currentTimeMillis() + 1000;
    long currentTime = System.currentTimeMillis();

    // Busy waiting
    while (timeToStop > currentTime) {
    currentTime = System.currentTimeMillis();
    }
  }
});
```

287. How can we prevent busy waiting in Java?

There is a simple way to prevent busy-waiting in Java. We can just put the current thread to sleep for given amount of time.

It can be done by calling sleep() method of java.lang.Thread class. We can pass the number of milliseconds to sleep() method as an argument.

288. Can we use Thread.sleep() method for real-time processing in Java?

Java does not guarantee that Thread.sleep() will cause the thread to sleep for exactly N number of milliseconds. Sometime the thread can sleep for than N number of milliseconds.

In real-time processing we need precise time period for which a thread should run or sleep.

Therefore the invocation of Thread.sleep() method is not recommended for use in real-time processing.

289. Can we wake up a thread that has been put to sleep by using Thread.sleep() method?

We can use interrupt() method of java.lang.Thread class to interrupt a thread that is in sleep state. It will get InterruptedException to wake up from the sleep.
Sample code is as follows:

```
public class ThreadInterrupt implements Runnable {
  public void run() {
  try {
      Thread.sleep(Long.MAX_VALUE);
    } catch (InterruptedException e) {
      SOP("Interrupted by exception!");
    }
  }
  public static void main(String[] args) throws InterruptedException
{
        Thread myThread = new Thread(new ThreadInterrupt(),
"myThread");
    myThread.start();
    SOP("Sleeping in main thread for 10 seconds");

    Thread.sleep(10000);
    SOP("Interrupting myThread");

    myThread.interrupt();
  }
}
```

290. What are the two ways to check if a Thread has been interrupted?

These are the two ways to check for thread interruption:

1. In Java, a Thread can call Thread.interrupted() method to check if it has been interrupted or not.

2. The other option is to call isInterrupted() method of Thread class to check if it has been interrupted or not.

291. How can we make sure that Parent thread waits for termination of Child thread?

We can use join() method for this purpose. On calling join() method, current thread waits for the child thread to which it joins to finish.

Sample code is as follows:

```
Thread myThread = new Thread(new Runnable() {
  public void run() {
    }
});

myThread.start();
// Join on myThread
myThread.join();
```

292. How will you handle InterruptedException in Java?

In Java we can get InterruptedException from sleep() or join() methods. Throwing InterruptedException is way to inform that another thread has interrupted this thread.

In general, the purpose of Interrupt is to ask current thread to stop its current execution and finish unexpectedly.

Therefore ignoring this exception by catching it and only logging it to the console or some log file is not the recommended approach.

The run() method of the Runnable interface does not allow that throwing any exceptions. So we cannot re-throw InterruptedException.

Therefore the correct way to handle this exception is that run() method should check and handle this exception by itself and take appropriate action.

293. Which intrinsic lock is acquired by a synchronized method in Java?

When we mark a method as synchronized and then call this method, then this method will first acquire the intrinsic lock of the object in which that method is mentioned.

Once the synchronized method returns, it releases the lock.

In case the synchronized method throws an exception, the intrinsic lock will be released.

Sample code equivalent to a synchronized method is:

```
public void myMethod() {
  synchronized(this) {
  }
}
```

294. Can we mark a constructor as synchronized in Java?

No. We cannot mark a constructor as synchronized.

This will lead to compiler error.

The reasoning behind this is that, in this case, only the constructing thread would have access to the object being constructed.

295. Can we use primitive values for intrinsic locks?

No. Java does not allow primitive values to be used for intrinsic locks.

296. Do we have re-entrant property in intrinsic locks?

Yes. An intrinsic lock can be accessed by the same thread multiple times. So an Intrinsic lock is re-entrant.

If it is not allowed then the code that acquires a lock would have to avoid acquiring the lock that it has already acquired.

297. What is an atomic operation?

An atomic operation is an operation that completes in a single step relative to other threads.

An Atomic operation is either executed completely or not at all.

There is no halfway mark in Atomic operation.

298. Can we consider the statement i++ as an atomic operation in Java?

No. The statement i++ is not an Atomic operation. It has more than one operation.

First JVM loads the current value of i in memory. Then it increments it. Finally it stores the new value back into variable i.

The current thread that executes this operation may be interrupted between any of the above-mentioned three steps. Therefore it is not an atomic operation.

299. What are the Atomic operations in Java?

Java language provides some basic Atomic operations. These operations can be used to make sure that concurrent threads always see the same value.

Some of these Atomic operations are:

1. Read operations on reference variables and primitive variables (except long and double)

2. Write operations on reference variables and primitive variables (except long and double)

3. Read operations on all variables declared as volatile

4. Write operations on all variables declared as volatile

https://www.vogella.com/tutorials/JavaConcurrency/article.html#:~:text=4.2.-,Atomic%20operation,of%20type%20long%20or%20double%20).

https://stackoverflow.com/a/34364511

300. Can you check if following code is thread-safe?

Synchronization in Java is possible by using Java keywords synchronized and volatile and locks. In Java, we can not have synchronized variable. Using synchronized keyword with a variable is illegal and will result in compilation error. Instead of using the synchronized variable in Java, you can use the java volatile variable, which will instruct JVM threads to read the value of volatile variable from main memory and don't cache it locally. If a variable is not shared between multiple threads then there is no need to use the volatile keyword.

```
public class SingletonDoubleCheck {
  private SingletonDoubleCheck instance = null;

  public SingletonDoubleCheck getInstance() {
      if (instance == null) {
      synchronized (SingletonDoubleCheck.class) {
      if (instance == null) {
      instance = new SingletonDoubleCheck();
      }
      }
      }
      return instance;
  }
}
```

The above-mentioned code is for creating a Singleton class. But this code is not thread-safe.

In this we check the value of instance second time in the synchronized block. But the JIT compiler can rearrange the Bytecode in such a way that the reference to SingletonDoubleCheck instance will be set before the execution of constructor.

Due to this the method getInstance() will return an object that may not have been initialized properly.

We can use the keyword volatile for instance to make this thread-safe code.

Any variables that is marked as volatile will be visible to other threads only after the completion of the constructor of the object.

volatile has semantics for memory visibility. Basically, the value of a volatile field becomes visible to all readers (other threads in particular) after a write operation completes on it. Without volatile, readers could see some non-updated value.

301. What are the minimum requirements for a Deadlock situation in a program?

For a deadlock to occur following are the minimum requirements:

1. **Mutual exclusion**: There has to be a resource that can be accessed by only one thread at any point of time.

2. **Resource holding**: One thread locks one resource and holds it, and at the same time it tries to acquire lock on another mutually exclusive resource.

3. **No preemption**: There is no pre-emption mechanism by which resource held by a thread can be freed after a specific period of time.

4. **Circular wait**: There can be a scenario in which two or more threads lock one resource each and they wait for each other's resource to get free. This causes circular wait among threads for same set of resources.

302. How can we prevent a Deadlock?

To prevent a Deadlock from occurring at least one requirement for a deadlock has to be removed:

1. **Mutual exclusion**: We can use optimistic locking to prevent mutual exclusion among resources.

2. **Resource holding**: A thread has to release all its exclusive locks if it does not succeed in acquiring all exclusive locks for resources required.

3. **No preemption**: We can use timeout period for an exclusive lock to get free after a given amount of time.

4. **Circular wait**: We can check and ensure that circular wait does not occur, when all exclusive locks have been acquired by all the threads in the same sequence.

303. How can we detect a Deadlock situation?

We can use ThreadMXBean.findDeadlockedThreads() method to detect deadlocks in Java program. This bean comes with JDK:

Sample code is as follows:

```
ThreadMXBean bean = ManagementFactory.getThreadMXBean();
long[] threadIds = bean.findDeadlockedThreads(); // It will return null for no deadlock
if (threadIds != null) {
   ThreadInfo[] infos = bean.getThreadInfo(threadIds);

   for (ThreadInfo info : infos) {
      StackTraceElement[] stack = info.getStackTrace();
      // Log or store stack trace information.
   }
}
```

304. What is a Livelock?

Livelock is a scenario in which two or more block each other by responding to an action caused by another thread.

In a deadlock situation two or more threads wait in one specific state.

In a Livelock scenario, two more threads change their state in such a way that it prevents progress on their regular work.

E.g. Consider scenario in which two threads try to acquire two locks. They release a lock that they have acquired, when they cannot acquire the second lock.

In a Livelock situation, both threads concurrently try to acquire the locks. Only one thread would succeed, the second thread may succeed in acquiring the second lock.

Now both threads hold two different locks. And both threads want to have both locks. So they release their lock and try again from the beginning. This situation keeps repeating multiple times..

305. What is Thread starvation?

In a priority based scheduling, Threads with lower priority get lesser time for execution than higher priority threads.

If a lower priority thread performs a long running computation, it may happen that this thread does not get enough time to finish its computations just in time. In such a scenario, the tread with lower priority would starve. It will remain away from the threads with higher priority.

306. How can a synchronized block cause Thread starvation in Java?

It is not defined for synchronization that which thread will enter a synchronized block. It may happen that if many threads are waiting for the entry to a synchronized block, some threads may have to wait longer than other threads.

Hence these threads with lower priority will not get enough time to finish their work in time.

307. What is a Race condition?

A race condition is an unwanted situation in which a program attempts to perform two or more operations at the same time, but because of the logic of the program, the operations have to be performed in proper sequence to run the program correctly.

Since it is an undesirable behavior, it is considered as a bug in code.

Most of the time race condition occurs in “check then act” scenario. Both threads check and act on same value. But one of the threads acts in between check and act. See this example to understand race condition.

```
if (x == 3) // Check
{
  y = x * 5; // Act

  // If another thread changes x
  // between "if (x == 3)” and "y = x * 5”,
  // then y will not be equal to 15.
}
```

308. What is a Fair lock in multi-threading?

In Java there is a class ReentrantLock that is used for implementing Fair lock. This class accepts an optional parameter fairness. When fairness is set to true, the RenentrantLock will give access to the longest waiting thread.

The most popular use of Fair lock is in avoiding thread starvation. Since longest waiting threads are always given priority in case of contention, no thread can starve.

Downside of Fair lock is the low throughput of the program. Since low priority or slow threads are getting locks multiple time, it leads to slower execution of a program.

The only exception to a Fair lock is tryLock() method of ReentrantLock. This method does not honor the value of fairness parameter.

309. Which two methods of Object class can be used to implement a Producer Consumer scenario?

In a Producer Consumer scenario, one thread is a Producer and another thread is a Consumer.

For this scenario to start working, a Consumer has to know when the Producer has produced. In Object class, there is a wait() method. A Consumer calls wait method to wait on Producer. The Producer used notify() method of Object class to inform Consumer that it has produced.

In this way the processor time between produce and consume operations is freed due to the use of wait() and notify() methods.

The Producer-Consumer problem is a classic problem this is used for multi-process synchronization i.e. synchronization between more than one processes.

In the producer-consumer problem, there is one Producer that is producing something and there is one Consumer that is consuming the products produced by the Producer. The producers and consumers share the same memory buffer that is of fixed-size.

https://www.baeldung.com/java-producer-consumer-problem
Producer Consumer Problem

310. How JVM determines which thread should wake up on notify()?

If multiple threads are waiting on an object's monitor, JVM awakens one of them. As per Java specification the choice of this thread is arbitrary and it is at the discretion of the implementation. So there is no guarantee of rule that a specific thread will be awakened by JVM on notify() method call.

311. Check if following code is thread-safe for retrieving an integer value from a Queue?

```
public class QueueCheck {
Queue queue;

public Integer getNextInt() {
  Integer retVal = null;
  synchronized (queue) {
       try {
       while (queue.isEmpty()) {
       queue.wait();
       }
       } catch (InterruptedException e) {
       e.printStackTrace();
       }
  }
  synchronized (queue) {
       retVal = queue.poll();
       if (retVal == null) {
       System.err.println("retVal is null");
       throw new IllegalStateException();                    }
  }
  return retVal;
}
}
```

In the above code Queue is used as object monitor to handle concurrency issues. But it may not behave correctly in a multi-threading scenario.

There are two separate synchronized blocks in above code. In case two threads are woken up simultaneously by another thread, both

threads will enter one after in the second synchronized block.

Only one of the two threads will get new value from the queue and make it empty. The second thread will poll on an empty queue and it will not get any non-null return value.

312. How can we check if a thread has a monitor lock on a given object?

In Java, Thread class has a static method holdsLock(Object objToCheck) to check whether thread has a lock on objToLock object.

This method will return true if current thread holds the lock on the objToLock object that was passed as an argument to this method.

313. What is the use of yield() method in Thread class?

The yield() method of Thread class is used to give a hint to scheduler that the current thread wants to free the processor.

The scheduler can either use this hint or just ignore this hint. Since the scheduler behavior is not guaranteed, it may happen that the current thread again gets the processor time.

It can be used for debugging or testing purposes. But there is rarely any concrete use of this method.

314. What is an important point to consider while passing an object from one thread to another thread?

This is a multi-threading scenario. In a multi-threading scenario, the most important point is to check whether two threads can update same object at the same time.

If it is possible for two threads to update the same object at the same time, it can cause issues like race condition.

So it is recommended to make the object Immutable. This will help in avoiding any concurrency issues on this object.

315. What are the rules for creating Immutable Objects?

As per Java specification, following are the rules for creating an Immutable object:

Do not provide "setter" methods that modify fields or objects referred to by fields.

Make all fields final and private.

Do not allow subclasses to override methods. The simplest way to do this is to declare the class as final. A more sophisticated approach is to make the constructor private and construct instances in factory methods.

If the instance fields include references to mutable objects, do not allow those objects to be changed.

Do not provide methods that modify the mutable objects.

Do not share references to the mutable objects. Never store references to external, mutable objects passed to the constructor; if necessary, create copies, and store references to the copies. Similarly, create copies of your internal mutable objects when necessary to avoid returning the originals in your methods.

316. What is the use of ThreadLocal class?

ThreadLocal class provides thread-local variables. Each thread accesses only its own local variables. It has its own copy of the variable.

By using ThreadLocal, if thread X stores a variable with value x and another thread Y stores same variable with the value y, then X gets x from its ThreadLocal instance and Y gets y from its ThreadLocal instance.

Typically, ThreadLocal instances are private static fields that are associated with the state of a thread.

317. What are the scenarios suitable for using ThreadLocal class?

We can use instance of ThreadLocal class to transport information within an application.

One use case is to transport security or login information within an instance of ThreadLocal so that every method can access it.

Another use case is to transport transaction information across an application, without using the method-to-method communication.

318. How will you improve the performance of an application by multi-threading?

In an environment with more than one CPU, we can parallelize the computation tasks on multiple CPUs. This leads to parallel processing of a bigger task that takes lesser time due to multiple threads dividing the work among themselves.

One example is that if we have to process 1000 data files and calculate the sum of numbers in each file. If each file takes 5 minutes, then 1000 files will take 5000 minutes for processing.

But by using multi-threading we can process these files in 10 parallel threads. So each thread will take 100 files each. Since now work is happening in 10 parallel threads, the time taken will be around 500 minutes.

319. What is scalability in a Software program?

Scalability is the capability of a program to handle growing amount of work or its potential to be enlarged in order to accommodate growth.

A program is considered scalable, if it is suitable to handle a large amount of input data or a large number of users or a large number of nodes.

When we say a program does not scale, it means that program fails on increasing the size of task.

320. How will you calculate the maximum speed up of an application by using multiple processors?

Amdahl's law gives the theoretical speedup in latency of the execution of a task at fixed workload.

It gives the formula to compute the theoretical maximum speed up that can be achieved by providing multiple processors to an application.

If S is the theoretical speedup then the formula is:
$S(n) = 1 / (B + (1-B)/n)$

where n is the number of processors
B is the fraction of the program that cannot be executed in parallel.

When n converges against infinity, the term (1-B)/n converges against zero. Therefore, the formula can be reduced in this special case to 1/B.

In general, the theoretical maximum speedup behaves in inverse proportion to the fraction that has to be executed serially. This means the lower this fraction is, the more theoretical speedup can be achieved.

321. What is Lock contention in multi-threading?

Lock contention is the situation when one thread is waiting for a lock/object that being held by another thread. The waiting thread cannot use this object until the other thread releases the lock on that object.

It is also known as Thread contention.

Ideally locks reduce the thread contention. Without locks, multiple threads can operate on same object and cause undesirable behavior. If locking is implemented correctly it reduces the occurrence of contention between multiple threads.

322. What are the techniques to reduce Lock contention?

There are following main techniques to reduce Lock contention:

1. Reduce the scope of lock.
2. Reduce object pooling.
3. Reduce the number of times a certain lock can be acquired.
4. Avoid synchronization at unnecessary places.
5. Implement hardware supported Optimistic locking in place of synchronization.

323. What technique can be used in following code to reduce Lock contention?

```
synchronized (map) {
  Random r = new Random();
  Integer value = Integer.valueOf(42);
  String key = r.nextString(5);
  map.put(key, value);
}
```

The code uses Random() to get a random string and it also used Integer to convert 42 in an object. Since these lines of code are specific to this thread, these can be moved out of Synchronization block.

```
  Random r = new Random();
  Integer value = Integer.valueOf(42);
  String key = r.nextString(5);

synchronized (map) {
  map.put(key, value);
}
```

324. What is Lock splitting technique?

Lock splitting is a technique to reduce Lock contention in multi-threading. It is applicable in scenario when one lock is used to synchronize access to different aspects of the same application.

Sometimes we put one lock to protect the whole array. There can be multiple threads trying to get the lock for same array. This single lock on array can cause Lock contention among threads. To resolve this we can give one lock to each element of the array. Or we can use modulus function to assign different locks to a small group of array elements. In this way we can reduced the chance of Lock contention. This is Lock splitting technique.

325. Which technique is used in ReadWriteLock class for reducing Lock contention?

ReadWriteLock uses two locks. One lock for read-only operations, another lock for write operations.

Its implementation is based on the premise that concurrent threads do not need a lock when they want to read a value while no other thread is trying to write.

In this implementation, read-only lock can be obtained by multiple threads. And the implementation guarantees that all read operation will see only the latest updated value as soon as the write lock is released.

326. What is Lock striping?

In Lock splitting we use different locks for different parts of the application. In Lock striping we use multiple locks to protect different parts of the same data structure.

ConcurrentHashMap class of Java internally uses different buckets to store its values. Each bucket is chosen based on the value of key. ConcurrentHashMap uses different locks to guard different buckets. When one thread that tries to access a hash bucket, it can acquire the lock for that bucket. While another thread can simultaneously acquire lock for another bucket and access it. In a synchronized version of HashMap, the whole map is has one lock.

Lock striping technique gives better performance than Synchronizing the whole data structure.

327. What is a CAS operation?

CAS is also known a Compare-And-Swap operation.

In a CAS operation, the processor provides a separate instruction that can update the value of a register only if the provided value is equal to the current value.

CAS operation can be used as an alternate to synchronization.

Let say thread T1 can update a value by passing its current value and the new value to be updated to the CAS operation. In case another thread T2 has updated the current value of previous thread, the previous thread T1's current value is not equal to the current value of T2. Hence the update operation fails.

In this case, thread T1 will read the current value again and try to update it.

This is an example of optimistic locking.

328. Which Java classes use CAS operation?

Java classes like AtomicInteger or AtomicBoolean internally use CAS operations to support multi-threading.

These classes are in package java.util.concurrent.atomic.

329. Is it always possible to improve performance by object pooling in a multi-threading application?

By using Object pools in an application we limit the number of new objects to be created for a class. In a single thread operation, it can improve the performance by reusing an already created object from a pool.

In a multi-threading application an object pool has to provide synchronized access to multiple threads. Due to this only one thread can access the pool at a time. Also there is additional cost due to Lock contention on pool. These additional costs can outweigh the cost saved by reuse of an object from the pool.

Therefore using an Object pool may not always improve the performance in a multi-threading application.

330. How can techniques used for performance improvement in a single thread application may degrade the performance in a multi-threading application?

In a single thread applications we can use Object pool for performance optimization. Where as in multi-threading environment, it may not be a good idea to use an Object pool. Increased overhead of synchronization and lock contention can degrade the performance gained by using Object pool in a multi-threading application.

Another example is the implementation in which a List keeps a separate variable to hold the number of elements. This technique is useful in single thread application where size() method can return the value from this variable, without the need to count all the elements of list.

But in a multi-threading application, this separate variable can rather degrade the performance. This variable has to be access controlled by a lock since multiple concurrent threads can insert an element in a list. The additional cost of lock on this variable can outweigh the benefit gained by it in a multi-threading application.

331. What is the relation between Executor and ExecutorService interface?

Executor interface has only execute(Runnable) method. The implementing class of this interface has to execute the given Runnable instance passed to execute() method at some time in the future.

ExecutorService interface extends Executor interface. It provides additional methods like- invokeAny(), invokeAll(), shutdown(), awaitTermination(). These method provide the ability to shutdown the thread so that further requests can be rejected. Also it provides ability to invoke a collection of Callable tasks.

332. What will happen on calling submit() method of an ExecutorService instance whose queue is already full?

The implementation of ExecutorService will throw RejectedExecutionException, when its queue is already full and a new task is submitted by calling submit() method.

333. What is a ScheduledExecutorService?

ScheduledExecutorService interface extends the interface ExecutorService. It provides various schedule() methods that can be used to submit new tasks to be executed at a given point of time.

One of the schedule() method provides the ability to schedule a one-shot task that can be executed after given delay.

Another version of schedule() method provides the ability to execute ScheduleFuture after a given amount of delay.

In addition there are scheduleAtFixedRate() and scheduleWithFixedDelay() methods that can execute an action at a periodic interval of time.

334. How will you create a Thread pool in Java?

In Java, Executors framework provides a method newFixedThreadPool(int nThreads) that can be used to create a Thread pool with a fixed number of threads.

Sample code is as follows:

```
public static void main(String[] args) throws InterruptedException, ExecutionException
{

  ExecutorService myService = Executors.newFixedThreadPool(5);
  Future<Integer>[] futureList = new Future[5];
  for (int i = 0; i < futureList.length; i++) {
       futureList[i] = myService.submit(new MyCallable());
  }
  for (int i = 0; i < futureList.length; i++) {
       Integer retVal = futureList[i].get();
       println(retVal);
  }

  myService.shutdown();
}
```

335. What is the main difference between Runnable and Callable interface?

Runnable interface defines run() method that does not return any value.

Callable interface allows call() method to return a value to its caller. A Callable interface can also throw an exception in case of an error. Also Callable is a newer addition to Java since version 1.5.

336. What are the uses of Future interface in Java?

We can use Future interface to represent the result of an asynchronous computation.

These are the operations whose result is not immediately available.

Therefore Future interface provides isDone() method to check if the asynchronous computation has finished or not.

We can also check if the task was cancelled by calling isCancelled() method.

Future also provides cancel() method to attempt the cancellation of a task.

337. What is the difference in concurrency in HashMap and in Hashtable?

In a Hashtable class all methods are synchronized.

In a HashMap implementation all the methods are not synchronized.

Therefore Hashtable is a thread-safe collection. HashMap is not a thread-safe collection.

In a multi-threading it is not advisable to use regular HashMap. We can use ConcurrentHashMap class in multi-threading applications.

338. How will you create synchronized instance of List or Map Collection?

In Java, Collections class provides methods to synchronize any collection.

It also provides synchronizedList(List) and synchronizedMap(Map) methods that can be used to convert a List or Map to a synchronized instance.

339. What is a Semaphore in Java?

Semaphore class in Java is used to implement a counting semaphore. It is used to restrict the number of threads that can access a physical or logical resource.

A Semaphore maintains a set of permits that should be acquired by competing threads.

We can also use it to control how many threads can access the critical section of a program or a resource concurrently.

The first argument in Semaphore constructor is the total number of permits available. Each invocation of acquire() method tries to obtain one of the available permits.

The acquire() method is used to acquire a permit from the semaphore. If we pass number of permits required to acquire() method, then it blocks the thread until that number of permits are available.

Once a thread has finished its work, we can use release() method to release the permits.

340. What is a CountDownLatch in Java?

CountDownLatch class helps in implementing synchronization in Java. It is used to implement the scenarios in which one or more threads have to wait until other threads have reached the same state such that all thread can start.

There is a synchronized counter that is decremented until it reaches the value zero. Once it reaches zero, it means that all waiting threads can proceed now.

It is a versatile tool that can be used for other Synchronization scenarios as well. It can also work as on/off latch or gate. All threads invoking await() method wait at the gate until it is opened by a thread invoking countdown() method.

341. What is the difference between CountDownLatch and CyclicBarrier?

CyclicBarrier takes an optional Runnable task that is run once the common barrier condition is achieved.

CountDownLatch is used in simple use cases where a simple start stop is required. A CyclicBarrier is useful in complex scenarios where more coordination is required. E.g. MapReduce algorithm implementation.

CyclicBarrier resets the internal value to the initial value once the value reaches zero. CyclicBarrier can be used to implement the scenarios in which threads have to wait for each other multiple times.

342. What are the scenarios suitable for using Fork/Join framework?

ForkJoinPool class is in the center of Fork/Join framework. It is a thread pool that can execute instances of ForkJoinTask.

ForkJoinTask class provides the fork() and join() methods. The fork() method is used to start the asynchronous execution of a task. The join() method is used to await the result of the computation.

Therefore, divide-and-conquer algorithms can be easily implemented with Fork/Join framework.

343. What is the difference between RecursiveTask and RecursiveAction class?

RecursiveAction class has compute() method that does not have to return a value.

RecursiveAction can be used when the action has to directly operate on a Data structure. It does not need to return any computed value.

In RecursiveTask class has compute() method that always returns a value.

Both RecursiveTask and RecursiveAction classes are used in ForkJoinTask implementations.

344. In Java 8, can we process stream operations with a Thread pool?

In Java 8, Collections provide parallelStream() method to create a stream that can be processed by a Thread pool.

We can also call the intermediate method parallel() on a given stream to convert it into a sequential stream of parallel tasks.

345. What are the scenarios to use parallel stream in Java 8?

A parallel stream in Java 8 has a much higher overhead compared to a sequential one.

It takes a significant amount of time to coordinate the threads.

We can use parallel stream in following scenarios:

When there are a large number of items to process and the processing of each item takes time and is parallelizable.
When there is a performance problem in the sequential processing.
When current implementation is not already running in a multi-thread environment. If there is already a multi-threading environment, adding parallel stream can degrade the performance.

346. How Stack and Heap work in Java multi-threading environment?

In Java, Stack and heap are memory areas available to an application. Every thread has its own stack. It is used to store local variables, method parameters and call stack.

Local variables stored in Stack of one Thread are not visible to another Thread.

Where as, Heap is a common memory area in JVM. Heap is shared by all threads. All objects are created inside heap.

To improve performance thread can cache the values from heap into their stack. This can create problem if the same variable is modified by more than one thread.

In such a scenario we should used volatile keyword to mark a variable volatile. For a volatile variable the thread always reads the value from main memory.

347. How can we take Thread dump in Java?

The steps to take Thread dump of Java process depends on the operating system.

On taking Thread dump, Java writes the state of all threads in log files or standard error console.

We can press Ctrl + Break key together to take thread dump in Windows.
We can execute kill -3 command for taking Thread dump on Linux.
Another option to take Thread dump is jstack tool. We can pass process id of java process to this tool for taking Thread dump.

This is the simple one, -Xss parameter is used to control stack size of Thread in Java. You can see this list of JVM options to learn more about this parameter.

348. Which parameter can be used to control stack size of a thread in Java?

We use –Xss parameter to control the stack size of a thread in Java.

If we set it as 1 MB, then every thread will get 1MB of stack size.

349. There are two threads T1 and T2? How will you ensure that these threads run in sequence T1, T2 in Java?

In Java there are multiple ways to execute threads in a sequence.

One of the simplest way for sequencing is join() method of Thread class.

We can call join() method to start a thread when another thread has finished.

We start with the last thread to execute first. And make this thread join on the next thread.

In this case we start thread T2 first. And then call T1.join() so that thread T2 waits for thread T1 to finish execution.

Once T1 completes execution, T2 thread starts executing.

https://www.java67.com/2015/07/how-to-join-two-threads-in-java-example.html

Java 8

350. What are the new features released in Java 8?

The new features released in Java 8 are:

1. Lambda Expression
2. Stream API
3. Date and Time API
4. Functional Interface
5. Interface Default and Static Methods
6. Optional
7. Base64 Encoding and Decoding
8. Nashorn JavaScript Engine
9. Collections API Enhancements
10. Concurrency Enhancements
11. Fork/Join Framework Enhancements
12. Spliterator
13. Internal Iteration
14. Type Annotations and Repeatable Annotations
15. Method Parameter Reflection
16. JVM Parameter Changes

351. What are the main benefits of new features introduced in Java 8?

The main benefits of Java 8 features are:

1. Support for functional programming by Lambda and Streams
2. Ease of high volume data processing by Streams
3. Ease of use by getting Parameter names through Reflection
4. Reusable code with enhanced Collection APIs
5. Smart exception handling with Optional
6. Control on JVM with new Parameters
7. Enhanced encryption support with Base 64
8. Faster execution with Nashorn JavaScript engine support

352. What is a Lambda expression in Java 8?

Lambda expression is an anonymous function. It is like a method that does not need any access modifiers, name or return value declaration. It accepts a set of input parameters and returns result.

Lambda expression can be passed as a parameter in a method. So we can treat code in Lambda expression as data. This piece of code can be passed to other objects and methods.

353. What are the three main parts of a Lambda expression in Java?

Three main parts of a Lambda expression are:

1. Parameter list: A Lambda expression can have zero or more parameters. Parameter list is optional to Lambda.

2. Lambda arrow operator: “->” is known as Lambda arrow operator. It separates the list of parameters and the body of Lambda.

3. Lambda expression body: The piece of code that we want to execute is written in Lambda expression body.

E.g. In following example:

```
Arrays.asList( "a", "b", "d" ).forEach( e -> System.out.println( e ) );
```

Parameter list = e
Arrow = ->
Body = System.out.println(e)

354. What is the data type of a Lambda expression?

A Lambda expression fulfills the purpose of passing code as data.

The data type of a Lambda expression is a Functional interface.

In most of the cases this is java.lang.Runnable interface.

355. What is the meaning of following lambda expression?

```
( e -> System.out.println( e ) );
```

This Lambda expression takes a parameter e and prints it via System.out.

356. Why did Oracle release a new version of Java like Java 8?

The main theme of Java 8 is support for functional programming. With increase in Database size and growth of multi-code CPU servers, there is need for Java to support such large-scale systems.

With new features of Java 8, it is possible to create functional programs to interact efficiently with Big Data systems. Support for Streams is very helpful in this regard.

Lambda expressions are very useful for cloud computing where we can pass code as data and run the same code on multiple servers.

Optional is a best practice that is borrowed from Google Guava library for handling the exceptional cases. This has made programs more robust with support for edge cases.

357. What are the advantages of a lambda expression?

We can pass a lambda expression as an object to a method. This reduces the overhead involved in passing an anonymous class.

We can also pass a method as a parameter to another method using lambda expressions.

358. What is a Functional interface in Java 8?

A Functional interface in Java is an interface that has exactly one abstract method.

It can have default methods with implementation. A default method is not abstract.

In Java 8, java.lang.Runnable and java.util.concurrent.Callable are two very popular Functional interfaces.

359. What is a Single Abstract Method (SAM) interface in Java 8?

A Functional interface is also known as Single Abstract Method Interface, since it has exactly one abstract method.

360. How can we define a Functional interface in Java 8?

To define a Functional interface in Java 8, we can create an Interface with exactly one abstract method.

Another way is to mark an Interface with annotation @FunctionalInterface. Even with the annotation we have to follow the rule of exactly one abstract method.

The only exception to this rule is that if we override java.lang.Object class's method as an abstract method, then it does not count as an abstract method.

361. Why do we need Functional interface in Java?

Functional Interfaces are mainly used in Lambda expressions, Method reference and constructor references.

In functional programming, code can be treated as data. For this purpose Lambda expressions are introduced. They can be used to pass a block of code to another method or object.

Functional Interface serves as a data type for Lambda expressions. Since a Functional interface contains only one abstract method, the implementation of that method becomes the code that gets passed as an argument to another method.

362. Is it mandatory to use @FunctionalInterface annotation to define a Functional interface in Java 8?

No, it is not mandatory to mark a Functional interface with @FunctionalInterface annotation.

Java does not impose this rule.

But, if we mark an interface with @FunctionalInterface annotation then Java Compiler will give us error in case we define more than one abstract method inside that interface.

363. What are the differences between Collection and Stream API in Java 8?

Main differences between Collection and Stream API in Java 8 are:

1. Version: Collection API is in use since Java 1.2. Stream API is recent addition to Java in version 8.
2. Usage: Collection API is used for storing data in different kinds of data structures. Stream API is used for computation of data on a large set of Objects.
3. Finite: With Collection API we can store a finite number of elements in a data structure. With Stream API, we can handle streams of data that can contain infinite number of elements.
4. Eager vs. Lazy: Collection API constructs objects in an eager manner. Stream API creates objects in a lazy manner.
5. Multiple consumption: Most of the Collection APIs support iteration and consumption of elements multiple times. With Stream API we can consume or iterate elements only once.

364. What are the main uses of Stream API in Java 8?

Main uses of Stream API in Java 8 are:

1. It helps in using data in a declarative way. We can make use of Database functions like Max, Min etc., without running a full iteration.
2. It makes good use of multi-core architectures without worrying about multi-threading code.
3. We can create a pipeline of data operations with Java Stream that can run in a sequence or in parallel.
4. It provides support for group by, order by etc. operations.
5. It supports writing for code in Functional programming style.
6. It provides parallel processing of data.

365. What are the differences between Intermediate and Terminal Operations in Java 8 Streams?

Main differences between Intermediate and Terminal Stream operations are as follows:

1. Evaluation: Intermediate operations are not evaluated until we chain it with a Terminal Operation of Stream. Terminal Operations can be independently evaluated.
2. Output: The output of Intermediate Operations is another Stream. The output of Terminal Operations is not a Stream.
3. Lazy: Intermediate Operations are evaluated in lazy manner. Terminal Operations are evaluated in eager manner.
4. Chaining: We can chain multiple Intermediate Operations in a Stream. Terminal Operations cannot be chained multiple times.
5. Multiple: There can be multiple Intermediate operations in a Stream operation. There can be only one Terminal operation in Stream processing statement.

366. What is a Spliterator in Java 8?

A Spliterator is a special type of Iterator to traverse and partition the elements of a source in Java. A source can be a collection, an IO channel or a generator function.

A Spliterator may traverse elements individually or sequentially in bulk.

367. What are the differences between Iterator and Spliterator in Java 8?

Main differences between Iterator and Spliterator are as follows:

1. Spliterator can be used with Streams in Java 8. Where as, Iterator is just used with Collection.
2. Spliterator uses Internal Iteration to iterate Streams. Iterator uses External Iteration to iterate Collections.
3. Spliterator can iterate Streams in Parallel as well as Sequential manner. Iterator only iterates in Sequential manner.
4. Spliterator can traverse elements individually as well as in bulk. Iterator only iterates elements individually.

368. What is Type Inference in Java 8?

A Java compiler can see each method's invocation and it declaration to determine what are type arguments required for invocation.

By Type Inference, Java can determine the types of the arguments as well as the type of the result being returned.

Type inference algorithm also tries to find the most specific type that can work with all types of arguments.

369. Does Java 7 support Type Inference?

Yes, Java 7 supports Type Inference. In Java 8, Oracle has enhanced the Type Inference concept. Now it can be used to define Lambda expressions, functions and Method references.

370. How does Internal Iteration work in Java 8?

In an Iterator, the fundamental question is that which party controls the iteration. Is it Iterator or the Collection on which iterator runs.

When a Collection controls the iterator, then it is called External Iteration. When the Iterator controls the iteration then it is called Internal Iteration.

In case of Internal Iteration, the client hands over an operation to Iterator and the Iterator applies the operation to all the elements in aggregate.

Internal Iteration is easier to implement, since the Iterator does not have to store the state of the collection.

371. What are the main differences between Internal and External Iterator?

Main differences between Internal and External Iterator are as follows:

1. An Internal Iterator controls the iteration itself. In an External Iterator collection controls the iteration.
2. Internal Iterator can iterate elements in individually as well as in
3. Bulk (like forEach). External iterator iterates element one by one.
4. Internal Iterator does not have to iterate elements only sequentially. External Iterator always iterates sequentially.
5. Internal Iterator supports declarative programming style that goes well with functional programming. External Iterator follows imperative style OOPS programming.
6. Some people consider Internal Iterator code more readable than that of External Iterator.

372. What are the main advantages of Internal Iterator over External Iterator in Java 8?

Some of the main advantages of Internal Iterator are:

1. Internal Iterator is based on Functional programming, therefore it can work on declarative style code.
2. There is no need to sequentially iterate elements in Internal Iterator.
3. Code is more readable and concise in Internal Iterator.
4. Internal Iterator supports concurrency and parallel processing.

373. What are the applications in which we should use Internal Iteration?

We need Internal Iterator in applications that require high performance, parallel processing, fast iteration and bulk operations support.

Also in Internal Iteration applications, we do not have much control over iteration. The other features like parallel processing etc. become more important.

374. What is the main disadvantage of Internal Iteration over External Iteration?

Internal Iteration has many advantages over External Iteration. But it has one big disadvantage. Since Java API is responsible for iterating in Internal iterator, developer does not get any control over iteration.

375. Can we provide implementation of a method in a Java Interface?

Before Java 8, it was not allowed to provide implementation of a method in an Interface.

Java 8 has introduced the flexibility of providing implementation of a method in an interface. There are two options for that:

1. Default Method: We can give default implementation of a method.
2. Static Method: We can create a static method in an interface and provide implementation.

376. What is a Default Method in an Interface?

In Java 8, we can provide implementation of a method in an Interface and mark this method with Default keyword.

In this way, this implementation of the method becomes default behavior for any class implementing the interface.

377. Why do we need Default method in a Java 8 Interface?

Default methods in an Interface provide backward compatibility feature in Java 8.

Let say there is an interface Car that is implemented by BMW, Chevrolet and Toyota classes. Now a Car needs to add capability for flying. It will require change in Car interface. Some of the car classes that do not have flying capability may fail. Therefore a Default Implementation of flying methods is added in Car interface so that cars with no flying capability can continue to implement the original Car interface.

378. What is the purpose of a Static method in an Interface in Java 8?

A Static method in an Interface is utility or helper method. This is not an object level instance method. Some of the uses of Static method in an Interface are:

1. Single Class: There is no need to create a separate Utils class for storing utility or helper methods. We can keep these methods in same interface.
2. Encapsulation: With Static methods, complete behavior of a Class is encapsulated in same class. There is no need to maintain multiple classes.
3. Extension: It is easier to extend a Class/API. If we extend a collection ArrayList, we get all the methods. We need not extend Collections class also.

379. What are the core ideas behind the Date/Time API of Java 8?

There are three core ideas behind the Date/Time API of Java 8:

1. Immutable-value classes: The new API avoids thread-safety and concurrency issues by ensuring that all the core classes are immutable and represent well-defined values.

2. Domain-driven design: The new API is modeled on precise domain with classes that represent different use cases for Date and Time.

3. The emphasis on domain-driven design offers benefits like clarity and understandability.

4. Separation of chronologies: The new API allows people to work with different calendar systems. It supports the needs of users in different areas of the world likes Japan or Thailand that don't follow ISO-8601.

380. What are the advantages of new Date and Time API in Java 8 over old Date API?

Some of the advantages of Java 8 Date Time API over existing Date API are:

Concurrency: Existing Date Time classes (such as java.util.Date and SimpleDateFormatter) are not thread-safe. This does not work well in concurrent applications. In new Date Time API, developer does not have to deal with concurrency issues while writing date-handling code.

Better Design: Date/Time classes prior to Java 8 have poor API design. For example, years in java.util.Date start at 1900, months start at 1, and days start at 0. It is not very intuitive. Java 8 Date Time API handles it very well.

No need for 3rd Party Libraries: With the popularity of third-party Date/Time libraries like Joda Time, Java has to make its native Date/Time API comparable. Now we can use the Java API instead of using 3rd party libraries.

381. What are the main differences between legacy Date/Time API in Java and Date/Time API of Java 8?

Main difference between legacy Date/Time API and Java 8 Date/Time API are:

1. Old API is not Thread safe. Java 8 API is Thread safe.
2. Old API has many mutable objects. New Java 8 API is based on Immutable objects.
3. Performance of old API is not good. New Java 8 Date/Time API gives better performance.
4. Old API is less readable and maintainable. New Java 8 API is very well designed and is more readable.
5. Old API has month values from 0 to 11. New API has months from 1 to 12.

382. How can we get duration between two dates or time in Java 8?

In Java8, we have a new class Duration that provides the utility of computing duration between two dates.

We can call the static method Duration.between(date1, date2) to get the time period in hours, mins, days etc. between date1 and date2.

383. What is the new method family introduced in Java 8 for processing of Arrays on multi core machines?

Java 8 has enhanced the Arrays class with methods that can run efficiently on multi core machines.

These methods start with keyword parallel.
Egg. Arrays.parallelSetAll(), Arrays.parallelSort() etc.

This parallel set of methods provides parallel processing of Arrays that can run Java code very fast on a multi core machine.

384. How does Java 8 solve Diamond problem of Multiple Inheritance?

In Multiple Inheritance if a class extends more than one classes with two different implementation of same method then it causes Diamond problem.

Consider following example to see problem and solution for Diamond problem in Java 8:

```
public interface BaseInterface{
  default void display() { //code goes here }
}
public interface BaseOne extends BaseInterface { }
public interface BaseTwo extends BaseInterface { }
public class ChildClass implements BaseOne, BaseTwo { }
```

In the above code, class ChildClass gives compile time error. Java Compiler cannot decide which display method should it invoke in ChildClass.

To solve this problem, Java SE 8 has given the following remedy:

```
public interface A{
  default void display() { //code goes here }
}
public interface B extends A{ }
public interface C extends A{ }
public class D implements B,C{
  default void display() {
    B.super.display();
  }
}
```

```
public interface BaseInterface{
  default void display() { //code goes here }
}
public interface BaseOne extends BaseInterface { }
public interface BaseTwo extends BaseInterface { }
public class ChildClass implements BaseOne, BaseTwo {
  default void display(){
  BaseOne.super.display();
}
}
```

The method invocation at BaseOne.super.display(); solves the Diamond problem as it resolves the confusion for compiler.

385. What are the differences between Predicate, Supplier and Consumer in Java 8?

The subtle difference between Predicate, Supplier and Consumer in Java 8 is as follows:

Predicate is an anonymous function that accepts one argument and returns a result.

Supplier is an anonymous function that accepts no argument and returns a result.

Consumer is an anonymous function that accepts one argument and returns no result.

386. Is it possible to have default method definition in an interface without marking it with default keyword?

No, we have to always mark a default method in interface with default keyword.

If we create a method with implementation in an interface, but do not mark it as default, then we will get compile time error.

387. Can we create a class that implements two Interfaces with default methods of same name and signature?

No, it is not allowed to create a class that implements interfaces with same name default methods.

It will give us compile time error for duplicate default methods.

388. How Java 8 supports Multiple Inheritance?

In Multiple Inheritance a class can inherit behavior from more than one parent classes.
Prior to Java 8, a class can implement multiple interfaces but extend only one class.

In Java 8, we can have method implementation within an interface. So an interface behaves like an Abstract class.

Now if we implement more than one interface with method implementation in a class, it means we are inheriting behavior from multiple abstract classes. That is how we get Multiple Inheritance in Java 8.

389. In case we create a class that extends a base class and implements an interface. If both base class and interface have a default method with same name and arguments, then which definition will be picked by JVM?

In such a scenario, JVM will pick the definition in base class.

390. If we create same method and define it in a class , in its parent class and in an interface implemented by the class, then definition will be invoked if we access it using the reference of Interface and the object of class?

In all the cases, method defined in the class will be invoked.

391. Can we access a static method of an interface by using reference of the interface?

No, a static method of interface has to be invoked by using the name of the interface.

392. How can you get the name of Parameter in Java by using reflection?

Java 8 has introduced a method Parameter.getName() to get the name of a parameter by using reflection.

Before using this feature, we need to turn on this feature in Java compiler.

To turn on this feature, just run javac with –parameters argument.

To verify the availability of this feature, we can use Parameter. isNamePresent() method.

393. What is Optional in Java 8?

Optional is a container object that may have a null or non-null value. If it has a value then isPresent() method returns true.

It a value is present, we can call get() method to get the value. Else we will get nothing.

It is very useful in handling data that has null values.

394. What are the uses of Optional?

Some of the uses of Optional in Java are:

We can use Optional to avoid NullPointerException in an application.
Optional performs Null check at compile time, so we do not get run time exception for a null value.
Optional reduces the codebase pollution by removing unnecessary null checks.
Optional can also be used to handle default case for data when a value is null.

395. Which method in Optional provides the fallback mechanism in case of null value?

In case, an Optional has null value, we can use orElseGet() method as fallback mechanism. If we implement orElseGet() method, it will be invoked when the value of Optional is null.

396. How can we get current time by using Date/Time API of Java 8?

In Java 8 we can use Clock class to get the current time. Instead of using old method System.currentTimeMillis(), we can create a Clock object and call millis() method to get the current time in milliseconds.

We can also call instant() method on Clock object to get the current time in a readable format.

397. Is it possible to define a static method in an Interface?

Yes, from Java 8, an Interface can also has a static method.

398. How can we analyze the dependencies in Java classes and packages?

Java 8 comes with a new command line tool jdeps that can help in analyzing the package-level and class-level dependencies.

We can pass a jar file name or a class name as an argument to this tool. It will list all the dependencies of that jar or class.

399. What are the new JVM arguments introduced by Java 8?

In Java 8, PermGen space of ClassLoader is removed. It has been replaced with MetaSpace.

Now we can set the initial and maximum size of MetaSpace.

The JVM options -XX:PermSize and –XX:MaxPermSize are replaced by -XX:MetaSpaceSize and -XX:MaxMetaspaceSize respectively in Java 8.

400. What are the popular annotations introduced in Java 8?

Some of the popular annotations introduced in Java 8 are:

@FunctionalInterface: This annotation is used to mark an interface as Functional Interface. As mentioned earlier, A FunctionalInterface can be used for lambda expressions.

@Repeatable: This annotation is used for marking another annotation. It indicates that the marked annotation can be applied multiple times on a type.

401. What is a StringJoiner in Java 8?

StringJoiner is a new class in Java 8 that can be used to create a String. It can construct a sequence of characters separated by a delimiter. It can also optionally add a prefix and suffix to this sequence. We can use this sequence to get a String.

E.g.
The String "[One:Two:Three]" may be constructed as follows:

```
StringJoiner sj = new StringJoiner(":", "[", "]");
sj.add("One").add("Two").add("Three");
String desiredString = sj.toString();
```

402. What is the type of a Lambda expression in Java 8?

The type of a lambda expression depends on the context it is being used.

A lambda is like a method reference. It does not have a type of its own.

Generally, a Lambda is an instance of a Functional Interface.

403. What is the target type of a lambda expression ?

The target type of a lambda expression represents a type to which the expression can be converted.

The target type for a lambda expression is a functional interface.

The lambda expression must have same parameter type as the parameter in the function of the interface. It must also return a type compatible with the return type of function.

404. What are the main differences between an interface with default method and an abstract class in Java 8?

An interface with a default method appears same as an Abstract class in Java. But there are subtle differences between two.

1. Instance variable: An interface cannot have instance variables. An abstract class can have instance variables.
2. Constructor: An interface cannot have a constructor. An abstract class can have constructor.
3. Concrete Method: An interface cannot have concrete methods other than default method. An abstract class is allowed to define concrete methods with implementation.
4. Lambda: An interface with exactly one default method can be used for lambda expression. An abstract class cannot be used for lambda expression.

Java Tricky Questions

405. Is there any difference between a = a + b and a += b expressions?

When we add two integral variables e.g. variables of type byte, short, or int in Java, then they are first promoted to int type, and then addition happens.

The += operator implicitly casts the result of addition into the type of variable used to hold the result.
What happens when you put return statement or System.exit () on try or catch block? Will finally block execute?

It is a popular tricky Java interview question. Most of the programmers think that no matter what the finally block will always execute. This question challenges that concept by putting a return statement in the try or catch block or calling System.exit() from try or catch block.

You can answer by saying that finally block executes even if we put a return statement in the try block or catch block. But finally block does not execute if you call System.exit() from try or catch block.

406. What does the expression 1.0 / 0.0 return? Will there be any compilation error?

Double class is the source of many tricky interview questions. You may know about the double primitive type and Double class. But while doing floating point arithmetic some people don't pay enough attention to Double.INFINITY, NaN, and -0.0. There are rules that govern the floating point arithmetic calculations involving Double.

The answer to this question is that 1.0 / 0.0 will compile successfully. And it will not throw ArithmeticException. It will just return Double.INFINITY.

.

407. Can we use multiple main methods in multiple classes?

Yes. When we start an application in Java, we just mention the class name to be run to java command. The JVM looks for the main method only in the class whose name is passed to java command. Therefore, there is no conflict amongst the multiple classes having main method.

408. Does Java allow you to override a private or static method?

The question is tricky but the answer is very simple. You cannot override a private or static method in Java. If we create a similar method with same return type and same method arguments in child class, then it will hide the superclass method. This is known as method hiding.

Also, you cannot override a private method in sub class because Private method is not visible even in a subclass. Therefore, what you can do is to create another private method with the same name in the child class.

So in both the cases, it is not method overriding. It is either method hiding or a new method.

409. What happens when you put a key object in a HashMap that is already present?

In a HashMap there are buckets in which objects are stored. Key objects with same HashCode go to same bucket.

If you put the same key again in a HashMap, then it will replace the old mapping because HashMap doesn't allow duplicate keys. The same key will have same HashCode as previous key object. Due to same HashCode, it will be stored at the same position in the bucket.

410. How can you make sure that N threads can access N resources without deadlock?

This question checks your knowledge of writing multi-threading code. If you have experience with deadlock and race conditions, you can easily answer this.

The answer is that by resource ordering you can prevent deadlock. If in our program we always acquire resources in a particular order and release resources in the reverse order, then we can prevent the deadlock.

So a thread waiting for same resource can not get into deadlock while the other thread is trying to get it and holding the resource required by first thread. If both of them release the resources in right order, one of them can acquire it to finish the work.

411. How can you determine if JVM is 32-bit or 64-bit from Java Program?

We can find JVM bit size 32 bit or 64 bit by running java command from the command prompt.

Or we can get it from Java program.

Sun has a Java System property to determine the bit size of the JVM: 32 or 64:

sun.arch.data.model=32 // 32 bit JVM
sun.arch.data.model=64 // 64 bit JVM

We can use System.getProperty("sun.arch.data.model") to determine if it is 32/64 bit from Java program.

412. What is the right data type to represent Money (like Dollar/Pound) in Java?

To represent money you need decimal points in the numbers like $1.99.

BigDecimal class provides good methods to represent Money. Using BigDecimal, we can do the calculation with decimal points and correct rounding. But using BigDecimal is a little bit high on memory usage.

We can also use double with predefined precision. But calculation on double can give erroneous results.

413. How can you do multiple inheritances in Java?

This is a question to trick people coming from C++ and Scala background to Java. There are many Object Oriented languages that support multiple inheritances. But Java is not one of them.

Answer of this question can be that, Java does support multiple inheritances of by allowing an interface to extend other interfaces. You can implement more than one interface. But you cannot extend multiple classes. So Java doesn't support multiple inheritances of implementation.

But in Java 8, the default method breaks the rule of multiple inheritances behavior.

414. Is ++ operation thread-safe in Java?

No, ++ operator is not a thread safe operation. It involves multiple instructions like- reading a value, incrementing it and storing it back into memory. These instructions can overlap between multiple threads. So it can cause issues in multi-threading.

415. How can you access a non-static variable from the static context?

We cannot access a non-static variable from the static context in Java. If you write a code like that, then you will get compile time error. It is one of the most common problems for beginner Java programmers, when they try to access instance variable inside the main method in a class.

Since main method is static in Java, and instance variables are non-static, we cannot access instance variable inside main. The solution is to create an instance of the object and then access the instance variables.

416. Let say there is a method that throws NullPointerException in the superclass. Can we override it with a method that throws RuntimeException?

This question is checking your understanding of the concepts of method overloading and overriding in Java.

We can throw superclass of RuntimeException in an overridden method, but we cannot do the same if it is a checked Exception.

417. How can you mark an array volatile in Java?

If you know multi-threading well then you can easily answer it.

We can mark an array volatile in Java. But it makes only the reference to array volatile, not the whole array.

If one thread changes the reference variable to point to another array, then it will provide a volatile guarantee. But if multiple threads are changing individual array elements, they won't be having same reference due to the reference itself being volatile.

418. What is a thread local variable in Java?

Thread-local variable is a variable restricted to a specific thread. It is like thread's own copy of variable that is not shared among multiple threads.
Java provides ThreadLocal class to support thread-local variables. To achieve thread-safety, you can use it. To avoid any memory leak, it is always good to remove a thread-local variable, once its work is done.

419. What is the difference between sleep() and wait() methods in Java?

In Java, we use these methods to pause currently running thread. There is a simple difference between these.

sleep() is actually meant for short pause because it doesn't release lock.

wait() is meant for conditional wait and it can release a lock that can be acquired by another thread to change the condition on which it is waiting.

420. Can you create an Immutable object that contains a mutable object?

In Java, it is possible to create an Immutable object that contains a mutable object.

We should not share the reference of the mutable object, since it is inside an immutable object. Instead, we can return a copy of it to other methods.

421. How can you convert an Array of bytes to String?

You can convert an Array of bytes to String object by using the String constructor that accepts byte[]. We need to make sure that right character encoding is used. Else we may get different results after conversion.

422. What is difference between CyclicBarrier and CountDownLatch class?

CyclicBarrier and CountDownLatch classes were introduced from Java 5.

We can reuse CyclicBarrier even if it is broken, but we cannot reuse CountDownLatch in Java.

423. What is the difference between StringBuffer and StringBuilder?

StringBuilder was introduced in Java 5. The main difference between both of them is that StringBuffer methods e.g. length(), capacity(), append() are synchronized. But corresponding methods in StringBuilder are not synchronized.

Due to this difference, concatenation of String using StringBuilder is faster than StringBuffer. Now it is considered bad practice to use StringBuffer, because, in most of the scenarios, we perform string concatenation in the same thread.

424. Which class contains clone method? Cloneable or Object class?

It is a very basic trick question. clone() method is defined in Object class. Cloneable is a marker interface that doesn't contain any method.

425. How will you take thread dump in Java?

There are platform specific commands to take thread dump in Java.

In Linux/Unix, just use kill -3 PID, where PID is the process id of Java process. It will give the thread dump of Java process.

In Windows, press Ctrl + Break. This will instruct JVM to print thread dump in standard out or err. It can also go to console or log file depending upon your application configuration.

426. Can you cast an int variable into a byte variable? What happens if the value of int is larger than byte?

An int is 32 bit in Java. But a byte is just 8 bit in Java. We can cast an int to byte. But we will lose higher 24 bits of int while casting. Because a byte can hold only first 8 bits of int. Remaining 24 bits (32-8 = 24) will be lost.

427. In Java, can we store a double value in a long variable without explicit casting?

No, we cannot store a double value into a long variable without casting it to long. The range of double is more than that of long. So we need to type cast.

To answer this question, just remember which one is bigger between double and long in Java.

428. What will this return 5*0.1 == 0.5? true or false?

The answer is false because floating point numbers can not be represented exactly in Java, so 5*0.1 is not same as 0.5.

429. Out of an int and Integer, which one takes more memory?

An Integer object takes more memory than an int in Java. An Integer is an object and it stores meta-data overhead about the object. An int is a primitive type so its takes less memory and there is no meta-data overhead.

430. Can we use String in the switch case statement in Java?

Yes. From Java 7 onwards, String can be used in switch case statement. This gives convenience to programmer. But internally hash code of String is used for the switch statement.

431. Can we use multiple main methods in same class?

Yes. You can have multiple methods with name main in the same class. But there should be only one main method with the signature public static void main(String[] args). JVM looks for main with this signature only. Other methods with name main in same class are just ignored.

432. When creating an abstract class, is it a good idea to call abstract methods inside its constructor?

No, we should avoid calling abstract methods in the constructor of an abstract class. Because, it can restrict how these abstract methods can be implemented by child classes.

Many IDE give “Overridable method call in constructor” warning for such implementation.

This is a problem of object initialization order. The superclass constructor will run before the child class constructor. It means child class is not yet initialized. But due to presence of overridden method in superclass, the overridden method of subclass is called when the subclass is not fully initialized.

433. How can you do constructor chaining in Java?

When we call one constructor from another constructor of the same class, then it is known as constructor chaining in Java. When you have multiple overloaded constructors in a class, you can do constructor chaining.

434. How can we find the memory usage of JVM from Java code?

We can use memory management related methods provided in java.lang.Runtime class to get the free memory, total memory and maximum heap memory in Java.

By using these methods, you can find out how much of the heap is used and how much heap space still remains.

Runtime.freeMemory() returns amount of free memory in bytes.
Runtime.totalMemory() returns total memory in bytes.
Runtime.maxMemory() returns maximum memory in bytes.

435. What is the difference between x == y and x.equals(y) expressions in Java?

The x == y expression does object reference matching if both a and b are an object and only returns true if both are pointing to the same object in the heap space.

The x.equals(y) expression is used for logical mapping and it is expected from an object to override this method to provide logical equality.

Eg. A Book object may be logically equal to another copy of same Book, but it is a different object which will be false while doing x == y.

436. How can you guarantee that the garbage collection takes place?

No. We cannot guarantee the garbage collection in Java. Java documentation explicitly says that GarbageCollection is not guaranteed.

You can call System.gc() to request garbage collection, however, that's what it is - a request. It is upto GC's discretion to run.

437. What is the relation between x.hashCode() method and x.equals(y) method of Object class?

x.hashCode() method returns an int hash value corresponding to an object instance.

It is used in hashCode based collection classes like Hashtable, HashMap, LinkedHashMap etc.

hashCode() method is also related to equals() method.

As per Java specification, two objects which are equal to each other using equals() method must have same hash code.

Therefore, two objects with same hashCode may or may not be equal to each other. But two equal objects should have same hash code.

438. What is a compile time constant in Java?

A compile time constant is public static final variable. The public modifier is optional here. At compile time, they are replaced with actual values because compiler knows their value up-front and it also knows that it cannot be changed during run-time. So they are constants.

439. Explain the difference between fail-fast and fail-safe iterators?

The main difference between fail-fast and fail-safe iterators is whether or not the collection can be modified while it is being iterated.

Fail-safe iterators allow modification of collection in an iteration task. But fail-fast iterators do not allow any modification to collection during iteration.

During iteration, fail-fast iterators fail as soon as they realize that the collection has been modified. Modification can be addition, removal or update of a member. And it will throw a ConcurrentModificationException.
Eg. ArrayList, HashSet, and HashMap are fail-fast.

Fail-safe iterators operate on a copy of the collection. Therefore they do not throw an exception if the collection is modified during iteration.
Eg. ConcurrentHashMap, CopyOnWriteArrayList are fail-safe.

440. You have a character array and a String. Which one is more secure to store sensitive data (like password, date of birth, etc.)?

Short answer is, it is safe to store sensitive information in character array.

In Java, String is immutable and it is stored in the String pool. Once a String is created, it stays in the pool in memory until it is garbage collected. You have no control on garbage collection. Therefore, anyone having access to a memory dump can potentially extract the sensitive data and use it.

Whereas, if you use a mutable object like a character array, to store the value, you can set it to blank once you are done with it. Once it is made blank it cannot be used by anyone else.

441. Why do you use volatile keyword in Java?

The volatile keyword guarantees global ordering on reads and writes to a variable. This implies that every thread accessing a volatile field will read the variable's current value instead of using a cached value.

By marking the variable volatile, the value of a variable is never cached thread-locally. All reads and writes will go straight to main memory of Java.

442. What is the difference between poll() and remove() methods of Queue in Java?

It is a basic question to know the understanding of Queue data structure. Both poll() and remove() methods remove and return the head of the Queue.

When Queue is empty, poll() method fails and it returns null, but remove() method fails and throws Exception.

443. Can you catch an exception thrown by another thread in Java?

Yes, it can be done by using Thread.UncaughtExceptionHandler.

Java Documentation says “When a thread is about to terminate due to an uncaught exception the Java Virtual Machine will query the thread for its UncaughtExceptionHandler usingThread.getUncaughtExceptionHa will invoke the handler's uncaughtException method, passing the thread and the exception as arguments.”

444. How do you decide which type of Inner Class – Static or Non-Static to use in Java?

An inner class has full access to the fields and methods of the enclosing class. This is convenient for event handlers, but comes at a cost. Every instance of an inner class retains and requires a reference to its enclosing class.

Due to this cost, there are many situations where static nested classes are preferred over inner classes. When instances of the nested class outlive instances of the enclosing class, the nested class should be static to prevent memory leaks.

At times, due to their “hidden” reference to enclosing class, Inner classes are harder to construct via reflection.

445. What are the different types of Classloaders in Java?

Java Classloader is the part of the Java Runtime Environment (JRE) that loads classes on demand into Java Virtual Machine (JVM).

When the JVM is started, three types of class loaders are used:

1. Bootstrap Classloader: It loads core java API file rt.jar classes from folder.
2. Extension Classloader: It loads jar files from lib/ext folder.
3. System/Application Classloader: It loads jar files from path specified in the CLASSPATH environment variable.

Classes may be loaded from the local file system, a remote file system, or even the web.

446. What are the situations in which you choose HashSet or TreeSet?

HashSet is better than TressSet in almost every way. It gives O(1) for add(), remove() and contains() operations. Whereas, TressSet gives O(log(N)) for these operations.

Still, TreeSet is useful when you wish to maintain order over the inserted elements or query for a range of elements within the set.

We should use TreeSet when we want to maintain order. Or when there are enough read operations to offset the increased cost of write operations.

447. What is the use of method references in Java?

Java 8 has introduced Method references. It allows constructors and methods to be used as lambdas.

The main uses of Method reference are to improve code organization, clarity and terseness.

448. Do you think Java Enums are more powerful than integer constants?

Yes. Java Enums provide many features that integer constants cannot. Enums can be considered as final classes with a fixed number of instances. Enums can implement interfaces but cannot extend another class.

While implementing the strategy pattern, we can use this feature of Enums. Especially, when the number of strategies is fixed.

You can also attach meta-data to enum values in Java. Also enum values are typesafe, where as integer constants are not.

You can also define custom behavior in enum values.

449. Why do we use static initializers in Java?

In Java, a static initializer can run code during the initial loading of a class and it guarantees that this code will only run once. Also the static code will finish running before a class can be accessed in any way.

Initializing static members from constructors is more work. You have to make sure that every constructor does this. You need to maintain a flag to mark the static work when it is done. You may have to think about synchronization or races conditions for work in static block not initialized from static context.

450. Your client is complaining that your code is throwing NoClassDefFoundError or NoSuchMethodError, even though you are able to compile your code without error and method exists in your code. What could be the reason behind this?

Sometimes we upgrade our libraries even with same method name. But we forget to let the client know about the new version. Due this different in version, we get NoClassDefFoundError or NoSuchMethodError at runtime when one library was not compatible with such an upgrade.

Java build tools and IDEs can also produce dependency reports that tell you which libraries depend on that JAR. Mostly, identifying and upgrading the library that depends on the older JAR resolve the issue.

451. How can you check if a String is a number by using regular expression?

Regex is a powerful tool for matching patterns and searching patterns.

A numeric String can only contain digits i.e. 0 to 9. It can also contain + and - sign at start of the String. We can create a regular expression for these two rules. One simple example is as follows:

Pattern pattern = Pattern.compile(".*\\D.*");

452. What is the difference between the expressions String s = "Temporary" and String s = new String("Temporary ")? Which one is better and more efficient?

In general, String s = " Temporary " is more efficient to use than String s = new String("Temporary ").

In case of String s = " Temporary ", a String with the value “Temporary” is created in String pool. If another String with the same value is created (e.g., String s2 = " Temporary "), it will reference the same object in the String pool.

But, when you use String s = new String("Temporary "), Java creates a String with the value “Temporary” in the String pool. Also, that String object is then passed to the constructor of the String Object i.e. new String("Temporary "). And this call creates another String object (not in the String pool) with that value.

Therefore, each such call creates an additional String object. E.g. String s2 = new String("Temporary ") creates an extra String object, rather than just reusing the same String object from the String pool.

So String s = “Temporary” is always an efficient way.

453. In Java, can two equal objects have the different hash code?

No. It is not possible for two equal objects to have different hashcode. But two objects with same hashcode may or may not be equal.

454. How can we print an Array in Java?

We can print an array by using methods of Arrays class. We can either use Arrays.toString() method or we can use Arrays.deepToString() method.

Since array doesn't implement toString() method by itself, just passing an array to System.out.println() will not print its contents. But we can use Arrays.toString() to print each element of an array.

455. Is it ok to use random numbers in the implementation of hashcode() method in Java?

No. The hashcode of an object should be always same. If you use random number in hashcode() method, then you may get a different value of hashcode for same object. This will break the hashcode contract.

456. Between two types of dependency injections, constructor injection and setter dependency injection, which one is better?

Constructor injection guarantees that a class will be initialized with all its dependencies during creation. But setter injection provides flexibility to set an optional dependency.

If we are using an XML file to describe dependencies, the setter injection is more readable.

In general, it is a good practice to use constructor injection for mandatory dependencies and use setter injection for optional dependencies.

457. What is the difference between DOM and SAX parser in Java?

In Java, Document Object Model (DOM) parser loads the whole XML into memory and creates a tree based on DOM model. This helps it in quickly locating the nodes, and making a change in the structure of XML.

On the other hand, Simple API for XML (SAX) parser is an event based parser. It doesn't load the whole XML into memory. Due to this reason DOM is faster than SAX but require more memory and is not suitable to parse large XML files.

458. Between Enumeration and Iterator, which one has better performance in Java?

Enumeration interface is a read-only interface. It has better performance than Iterator. It is almost twice as fast as compared to an Iterator. It also uses very less memory. Also Enumeration does not have remove() method.

On the other hand, Iterator interface is safer than Enumeration, since it can check whether a collection is modified or not during iteration. If a collection is altered while an Iterator is iterating, then it throws ConcurrentModificationException.

459. What is the difference between pass by reference and pass by value?

Whenever an object is passed by value, it means that a copy of the object is passed. Even if changes are made to that object, it doesn't affect the original value.

Whenever an object is passed by reference, it means that the actual object is not passed, rather a reference of the object is passed. Therefore, any changes made by an external method, are also reflected in the actual object and its reference.

460. What are the different ways to sort a collection in Java?

The most popular way to sort a collection in Java is by calling Collections.sort() method. You can provide your custom Comparator to sort() method for sorting the data in your custom way.

The other way is to use a Sorted collection like TreeSet or TreeMap that stores the information in a sorted order and then you can convert it to a List.

461. Why Collection interface doesn't extend Cloneable and Serializable interfaces?

Collection interface just specifies groups of objects known as elements. Each concrete implementation of a Collection can choose its own way of how to maintain and order its elements.

Some collections may allow duplicate keys, while other collections may not.

A lot of collection implementations have clone method. But many do not. It is not worthwhile to include it in all, since Collection is an abstract representation. What matters is the concrete implementation.

Cloning and serialization come into picture while doing concrete implementation. Therefore, the concrete implementations of collections should decide how they can be cloned or serialized.

462. What is the difference between a process and a thread in Java?

A process is simply an execution of a program.

A Thread is a single execution sequence within a process.

A process may contain multiple threads. A Thread is also called as a lightweight process.

463. What are the benefits of using an unordered array over an ordered array?

In an ordered array the search time has time complexity of O(log n). Whereas, in an unordered array, search time complexity is O (n).

In an ordered array, the insert operation has a time complexity of O(n). Whereas, the insertion operation for an unordered array takes constant time of O(1).

Therefore, when we have more writes than reads, it is preferable to use an unordered array.

464. Between HashSet and TreeSet collections in Java, which one is better?

A HashSet is Implemented using a HashTable. Therefore, its elements are stored in a random order. The add(), remove(), and contains() methods of a HashSet have constant time complexity O(1).

A TreeSet is implemented using a tree data structure. The elements in a TreeSet are sorted in a natural order. Therefore, add(), remove(), and contains() methods have time complexity of O(logn).

So from performance perspective, HashSet has better performance than TreeSet. But if you want to store elements in a natural sorting order, then TreeSet is a better collection.

465. When does JVM call the finalize() method?

JVM instructs the Garbage Collector to call the finalize method, just before releasing an object from the memory. A programmer can implement finalize() method to explicitly release the resources held by the object. This will help in better memory management and avoid any memory leaks.

466. When would you use Serial Garabage collector or Throughput Garbage collector in Java?

The Serial Garbage collector is used for small applications that require heap memory upto 100 MB.

The Throughput Garbage collector is used in medium to large size Java applications.

467. In Java, if you set an object reference to null, will the Garbage Collector immediately free the memory held by that object?

No. JVM decides to run the Garbage Collector whenever it is low on memory. When Garbage Collector runs, it looks for objects that are available for garbage collection and then frees the memory associated with this object.

So just setting an Object reference null makes it eligible for Garbage Collection, but it does not immediately free the memory.

468. How can you make an Object eligible for Garbage collection in Java?

To make an Object eligible for Garbage collection, just make sure that it is unreachable to the program in which it is currently defined / created / used. You can set the object reference to null and make sure no other object refers it. Once the object cannot be reached, Garbage Collection can clean it during the next run.

469. When do you use Exception or Error in Java? What is the difference between these two?

Throwable class is the superclass of Exception and Error classes in Java.

When you want to catch the exceptional conditions that your program can create or encounter, then use the Exception class or subclass of Exception.

When you come across situations that are unexpected then use Error class in Java. Also recovering from Error is not possible in most of cases. So it is better to terminate the program.

470. What is the advantage of PreparedStatement over Statement class in Java?

PreparedStatements are precompiled statements for database queries. Due to this their performance is much better. Also, we can reuse PreparedStatement objects with different input values to the same query.

Where as, Statement class does not provide these features.

471. In Java, what is the difference between throw and throws keywords?

When we want to raise an exception in our code, we use the throw keyword with the name of the exception to be raised.

Where as, throws keyword is used in method declaration. Throws keyword tells us the Exception that can be thrown by this method. Any caller of this method should be prepared to expect this Exception.

Another minor difference is that throw is used only with one exception, but throws can be used with comma-separated list of multiple exceptions.

472. What happens to the Exception object after the exception handling is done?

Once the exception handling is complete, the Exception object is not reachable. Then it is garbage collected in the next run of Garbage Collector.

473. How do you find which client machine is sending request to your servlet in Java?

We can use the ServletRequest class to find the IP address or host name of the client machine.

There are methods getRemoteAddr() to get the IP address of the client machine and getRemoteHost() to get the host name of the client machine.

474. What is the difference between a Cookie and a Session object in Java?

Both Cookie and Session are used during communication between Client and Server. The Client can disable a Cookie. Due to which the Web server cannot send a cookie. But a client cannot disable a session. So a Session always works irrespective of any setting at the client side.

Also a Session can store any Java object. But the Cookie can only store small information in a String object.

475. Which protocol does Browser and Servlet use to communicate with each other?

HTTP protocol. The Browser and Servlet communicate with each other by using the HTTP protocol.

476. What is HTTP Tunneling?

There are many network communication protocols on the Internet. But HTTP is the most popular among them. HTTP Tunneling is a technique in which HTTP or HTTPS protocol encapsulated the communication done by any other type of protocol. The masking of other protocol requests as HTTP requests is known as HTTP Tunneling.

477. Why do we use JSP instead of Servlet in Java?

Since JSP pages are dynamically compiled into servlets, the programmers can easily make updates to the presentation layer code.

For better performance, JSP pages can be pre-compiled.

Also JSP pages provide flexibility to combine static templates like HTML or XML snippets.

In addition, programmers can make logic changes at the class level, without editing the JSP pages that use the class logic.

478. Is empty ‘.java’ file name a valid source file name in Java?

Yes. You can create a class and store it in a file with name .java. You can try it yourself, by creating, compiling and running such a file. It will run correctly.

479. How do you implement Servlet Chaining in Java?

To implement, Servlet Chaining, there has to be more than one servlet. The output of one servlet has to be sent to a second servlet. The output of the second servlet can be sent to a third servlet, and so on. In this way, a chain of servlets is formed to complete a task.

The last servlet in the chain will be responsible for sending final response to client.

480. Can you instantiate this class?

```
public class A
{
   A a = new A();
}
```

No, this class cannot be instantiated, since it will result in recursively calling its constructor.

481. Why Java does not support operator overloading?

Java supports Method overloading but does not support operator overloading. It would make the design more complex by adding operator loading. Also it will make more complex compiler.

One more reason is that, it will reduce the performance of JVM by operator overloading, since JCM has to do extra work to find the real meaning of overloaded operators at run time.

482. Why String class is Immutable or Final in Java?

Since String objects are cached in a String pool, it makes sense to make the String immutable. The cached String literals are shared between multiple clients. And there is a possibility that one client's action may affect another client’s access to String pool.

String is also used as a parameter in many Java classes. Eg. You can pass hostname, port number as String while opening a network connection. If any one can modify your copy of the String, it can change the hostname. Due to this reason, it makes sense to make String final as soon as it is created.

483. What is the difference between sendRedirect and forward methods?

When you use sendRedirect method, it creates a new request. When you use the forward method, it just forwards a request to a new target.

In case of sendRedirect, the previous request scope objects are not available, because it creates a new request.

In case of forward method, the previous request scope objects are available after forwarding.

Also the sendRedirect method is considered slower than the forward method.

484. How do you fix your Serializable class, if it contains a member that is not serializable?

If you want to make a class Serializable, but find that this class contains members that are not Serializable, then you have to mark those members as transient. This will ensure that this member is not persisted to a stream of bytes during Serialization.

Therefore, Transient keyword of Java comes to help in this scenario.

485. What is the use of run time polymorphism in Java?

During the run time the behavior of an Object can change based on its run time state. Due to this run time polymorphism is introduced in Java. If you override a method in a child class, then you are providing run time polymorphism. Nothing will happen at the compile time. But at the run time, JVM decides which method will be called based on the class of the Object.

486. What are the rules of method overloading and method overriding in Java?

When we want to overload a method, we need to make sure that the method name remains same. But method signature can vary in the number or datatype of arguments or in the order of arguments.

When we want to override a method, we ensure that the method is not throwing checked exceptions that are new or higher than those declared by the overridden method. Also we make sure that the method name, arguments and return type remain the same.

Also we cannot override Static and Final methods in Java.

487. What is the difference between a class and an object in Java?

A Class is a template or a blue print of an Object to be created. An Object is an instance of a Class. A Class defines the methods and member variables. But an Object populates the values of the member variables.

Therefore a class is a blueprint that you use to create objects. An object is an instance of a class – it is a concrete 'thing' that you made using a specific class.

Most of the OOPS concepts are valid only when an Object is created.

488. Can we create an abstract class that extends another abstract class?

Yes. An abstract class can extend another abstract class. It does not need to define the methods of parent abstract class. Only the last non-abstract class has to define the abstract methods of a parent abstract class.

489. Why do you use Upcasting or Downcasting in Java ?

When we want to cast a Sub class to Super class, we use Upcasting. It is also known as widening. Upcasting is always allowed in Java.

When we want to cast a Super class to Sub class, we use Downcasting. It is also known as narrowing.

At times, Downcasting can throw the ClassCastException if it fails the type check.

490. What is the reason to organize classes and interfaces in a package in Java?

As the name suggests, a package contains a collection of classes. It helps in setting the category of a file. Like- whether it is a Data Access Object (DAO) or an API.

It helps in preventing the collision of Name space.

Also we can introduce access restriction by using package and the right modifiers on a class and its methods.

491. What is information hiding in Java?

Information hiding is OOPS concept. In Java you can use encapsulation to do Information hiding. An object can use the access modifiers like-public, private, protected to hide its internal details from another object. This helps in decoupling the internal logic of an object from outside world.

By using Information hiding, an object can change its internal implementation without impacting the outside calling client's code.

492. Why does Java provide default constructor?

In Java all the interaction takes place between Object instances. To create an Object instance, JVM needs a constructor. Java does not enforce the rule on a programmer to define a default constructor for every class.

Whenever an object has to be created and programmer has not provided a constructor, Java uses default constructor to create the object. Default constructor also initializes member variables with their default values.

493. What is the difference between super and this keywords in Java?

We use super keyword to access the methods of the super class from child class.

We use this keyword to access methods of the same class.

494. What is the advantage of using Unicode characters in Java?

Unicode characters have much larger number of characters in the specification.

They also contain Asian and non-western European characters.

Most of the modern technologies, websites and browsers support these Unicode characters.

495. Can you override an overloaded method in Java?

Yes. Java allows to override an overloaded method, if that method is not a static or final method.

496. How can we change the heap size of a JVM?

Java provides the command line parameters to set the heap size for JVM.
You can specify the values in –Xms and –Xmx parameters. These parameters stand for initial and maximum heap size of JVM.

497. Why should you define a default constructor in Java?

In general, Java provides a default constructor with each class. But there are certain cases when we want to define our own version of default constructor.

When we want to construct an object with default values, we create our default constructor.

At times, we can mark the default constructor private. So that any other class cannot create an instance of our class. This technique is generally used in Singleton design pattern.

498. How will you make an Object Immutable in Java?

To make an object immutable follow these two rules. One, do not use any setter methods that can change the fields of your class. Two, make the fields final. By following these rules, the member variables cannot be changed after initialization. This will ensure that member variables of an Object do not change. And thus the Object will be considered Immutable.

499. How can you prevent SQL Injection in Java Code?

In Java, you can use PreparedStatement to prevent SQL injection. In a PreparedStatement you can pass the precompiled SQL queries with pre-defined parameters. This helps in checking the type of parameters to SQL queries. So it protects your code from SQL injection attacks.

500. Which two methods should be always implemented by HashMap key Object?

Any object that we want to use as key for HashMap or in any other hash based collection data structure e.g. Hashtable, or ConcurrentHashMap must implement equals() and hashCode() method.

501. Why an Object used as Key in HashMap should be Immutable?

The Key object should be immutable so that hashCode() method always return the same value for that object.

The Hashcode returned by hashCode() method depends on values of member variables of an object. If an object is mutable, then the member variables can change. Once the member variables change, the Hashcode changes. If the same object returns different hash code at different times, then it is not reliable to be used in the HashMap.

Let say, when you insert the object, the Hashcode is X, the HashMap will store it in bucket X. But when you search for it the Hashcode is Y, then HashMap will look for the object in bucket Y. So you are not getting what you stored.

To solve this, a key object should be immutable.

Although, the compiler does not enforce this rule, a good programmer always remembers this rule.

502. How can we share an object between multiple threads?

There are many ways to share same object between multiple threads. You can use a BlockingQueue to pass an object from one thread to another thread.

You can also use Exchanger class for this purpose. An Exchanger is a bidirectional form of a SynchronousQueue in Java. You can use it to swap the objects as well.

503. How can you determine if your program has a deadlock?

If we suspect that our application is stuck due to a Deadlock, then we just take a thread dump by using the command specific to environment in which your application is running. Eg. In Linux you can use command kill -3.

In case of deadlock, you will see in thread dump the current status and stack trace of threads in the JVM, and one or more of them will be stuck with message deadlock.

Also you can do this programmatically by using the ThreadMXBean class that ships with the JDK.

If you don't need programmatic detection you can do this via JConsole. On the thread tab there is a "detect deadlock" button.

Mixed Questions

1. What are Wrapper classes in Java?

Java has concept of Wrapper classes to allow primitive types to be accessed as objects. Primitive types like boolean, int, double, float etc. have corresponding Wrappers classes – Boolean, Integer, Double, Float etc.

Many of these Wrapper classes are in java.lang package.

Java 5.0 has launched the concept of Autoboxing and Unboxing in Java for Wrapper classes.

E.g.

```
public class WrapperTest{
public static void main(String args[]){
//Converting int into Integer
int count=50;
Integer i=Integer.valueOf(count);//converting int into Integer
Integer j=a;//autoboxing, now compiler will write Integer.valueOf(count) internally

System.out.println(count+" "+i+" "+j);
}}
```

2. What is the purpose of native method in Java?

The native keyword is used for applying to a method to indicate that the method is implemented in native code using JNI(Java Native Interface).

Therefore, native methods allow Java Developer to directly access platform specific APIs.

Often, native methods are linked to native library.

3. What is System class?

System.class is a final class provided by java.lang package. It contains several useful class fields and methods.

The purpose of System class is to provide access to system resources.

4. What is System, out and println in System.out.println method call?

System is a final class provided by java.lang package.

out refers to PrintStream class and a static member of System class.

println is a method of PrintStream class.

5. What is the other name of Shallow Copy in Java?

Object Cloning. A Shallow Copy just copies the values of references in a Class.

6. What is the difference between Shallow Copy and Deep Copy in Java?

A Shallow copy just copies the values of the references in the class.
A Deep copy copies the values of the objects as well.

7. What is a Singleton class?

A Singleton class in Java has maximum one instance of the class present in JVM, all the time. The constructor of this class is written in such a way that it never creates more than one object of same class.

8. What is the difference between Singleton class and Static class?

A static class in Java has only static methods. It is a container of functions. It is created based on procedural programming design.

Singleton class is a pattern in Object Oriented Design. A Singleton class has only one instance of an object in JVM. This pattern is implemented in such a way that there is always only one instance of that class present in JVM.

JSP

9. What are the implicit objects in JSP?

JSP has following implicit objects:

1. Request
2. Response
3. Application
4. Exception
5. Page
6. Config
7. Session

10. How will you extend JSP code?

We can extend JSP code by using Tag libraries and Custom actions.

11. How will you handle runtime exceptions in JSP?

We use Errorpage attribute in JSP to catch runtime exceptions. This attribute forwards user request to the error page automatically.

12. How will you prevent multiple submits of a page that come by clicking refresh button multiple times?

We can use Post Redirect Get (PRG) pattern to solve the issue of multiple submission of same data. It works as follows:

First time when a user submits a form to server by POST or GET method, then we update the state in application database.

Then we send a redirect response to send reply to client.

Then we load a view by using GET command. There is no data is sent in this. Since this a new JSP page, it is safe from multiple submits. The code that processes the request is idempotent. So it does not do same action twice for same request.

13. How will you implement a thread safe JSP page?

We can use SingleThreadModel Interface to implement a thread safe JSP page.

We can also add <%@page isThreadSafe=”false” %> directive in JSP page to make it thread safe.

14. How will you include a static file in a JSP page?

We can use include directive of JSP to include a Static page in JSP. In this approach, we use translation phase to include a static page. We have to specify the URL of the resource to be included as file attribute in this directive.

E.g. <%@ include file="footer.html" %>

15. What are the lifecycle methods of a JSP?

A JSP has following lifecycle methods:

1. **jspInit**(): This method is invoked when the JSP is called for the first time. We can do initial setup for servicing a request in this method.

2. **_jspService**(): This method is used to serve every request of the JSP.

3. **jspDestroy**(): Once we remove a JSP from the container, we call this method. It is used for cleanup of resources like Database connections etc.

16. What are the advantages of using JSP in web architecture?

We get following advantages by using JSP in web architecture:

1. **Performance**: JSP provides very good performance due to their design of using same code to service multiple requests.

2. **Fast**: Since JSP is pre-compiled, server can serve the pages very fast.

3. **Extendable**: JSP is based on Java Servlets. This helps in extending JSP architecture with other Java technologies like JDBC, JMS, JNDI etc.

4. **Design**: It is easier to design user interface with JSP, since it is very close to HTML. UI designers can create a JSP with mock data and developers can later provide implementation of dynamic data.

17. What is the advantage of JSP over Javascript?

In JSP we can write Java code seamlessly. It allows for writing code that can interact with the rest of the application.

Javascript code is mostly executed at client side. This limits the tasks that can be done in Javascript code. We cannot connect to database server from Javascript at the client side.

18. What is the Lifecycle of JSP?

JSP has following lifecycle stages:

1. **Compilation**: When a request is made for a JSP, the corresponding JSP is converted into Servlet and compiled. If there is already a compiled form of JSP and there is not change in JSP page since last compilation, this stage does not do anything.

2. **Initialization**: In this stage, jspInit() method is called to initialize any data or code that will be later used multiple times in _jspService() method.

3. **Service**: In this stage, with each request to JSP, _jspService() method is called to service the request. This is the core logic of JSP that generates response for request.

4. **Destroy**: In this stage, JSP is removed from the container/server. Just before removal, this stage performs the cleanup of any resources held by JSP.

19. What is a JSP expression?

A JSP expression is an element of a JSP page that is used to evaluate a Java expression and convert into a String. This String is replaced into the locations wherever the expression occurs in JSP page.

E.g. <%= expression =%>

20. What are the different types of directive tags in JSP?

JSP has following directive tags:

1. **Page**: This directive is used for page related attributes. It can be put anywhere in the JSP page. But by convention we put it on the top of the page.

   E.g.
   <%@ page attribute="value" %>

2. **Taglib**: We can create custom tags in JSP and use these by taglib directive in a JSP page.

   E.g.
   <%@ taglib uri="abc.html" prefix="tag_prefix" >

3. **Include**: We use include directive to read a file and merge its content with the JSP page. This is done during compilation stage.

   <%@ include file="relative url" >

21. What is session attribute in JSP?

Session attribute in JSP is used for HTTP session mechanism. If we do not want to use HTTP session in JSP, then we set this attribute to false. If it is set to true, we can use built in session object in JSP.

22. What are the different scopes of a JSP object?

A JSP object, implicit or explicit, can have one of the following scopes:

1. **Page**: In this scope, the object is accessible from the page where it was created. Important point here is that when a user refreshes the page, the objects of this scope also get created again.

2. **Request**: In request scope, the object is accessible to the HTTP request that created this object.

3. **Session**: In this scope, the object is available throughout the same HTTP session.

4. **Application**: This is the widest scope. The object is available throughout the application in which JSP was created.

23. What is pageContext in JSP?

In JSP, pageContext is an implicit object. This is used for storing and accessing all the page scope objects of JSP.

It is an instance of the PageContext class from javax.servlet.jsp package.

24. What is the use of jsp:useBean in JSP?

We use jsp:useBean to invoke the methods of a Java Bean class. The Java Bean class has some data and setter/getters to access the data.

With this tag, container will try to locate the bean. If bean is not already loaded then it will create an instance of a bean and load it. Later this bean can be used in expressions or JSP code.

25. What is difference between include Directive and include Action of JSP?

Some of the main differences between include Directive and include Action are as follows:

1. Include directive is called at translation phase to include content in JSP. Include Action is executed during runtime of JSP.

2. It is not possible to pass parameters to include directive. Include action can accept parameters by jsp:param tag.

3. Include directive is just copying of content from another file to JSP code and then it goes through compilation. Include action will dynamically process the resource being called and then include it in the JSP page.

26. How will you use other Java files of your application in JSP code?

We can use import tag to import a Java file in JSP code. Once a file is imported, it can be used by JSP code. It is a very convenient method to use Java classes in JSP code.

For better organization of Java code, we should create a package of classes that we are planning to use in JSP code.

27. How will you use an existing class and extend it to use in the JSP?

We can use extends attribute in include tag to use an existing class and extend it in the current JSP.

E.g.

<%@ include page extends=“parent_class” %>

28. Why _jspService method starts with _ symbol in JSP?

All the code that we write in a JSP goes into _jspService method during translation phase. We cannot override this method. Where as other lifecycle methods jspInit() and jspDestroy() can be overridden.

It appears that container uses _ symbol to distinguish the method that cannot be overridden by client code.

29. Why do we use tag library in JSP?

At times we want to create a UI framework with custom tags. In such a scenario, taglib is a very good feature of JSP. With taglib we can create tags that can provide custom features.

Taglib is also a nice way to communicate with UI designers who can use custom tags in the html without going into the details of how the code is implemented.

Another benefit of taglib is reusability of the code. This promotes writing code only once and using is multiple times.

30. What is the different type of tag library groups in JSTL?

JSTL stands for JavaServer Pages Standard Tag Library. In JSTL, we have a collection of JSP tags that can be used in different scenarios. There are following main groups of tags in JSTL:

1. Core tags
2. SQL tags
3. Formatting tags
4. XML tags
5. JSTL Functions

31. How will you pass information from one JSP to another JSP?

We can pass information from one JSP to another by using implicit objects. If different JSP are called in same session, we can use session object to pass information from one JSP to another.

If we want to pass information from one JSP to another JSP included in the main JSP, then we can use jsp:param to pass this information.

32. How will you call a stored procedure from JSP?

JSP allows running Java code from a .jsp file. We can call a stored procedure by using JDBC code.

We can call a CallableStatement from JSP code to invoke a stored procedure.

If we are using Spring framework, then we can use JdbcTemplate class to invoke stored procedure from a JSP.

33. Can we override _jspService() method in JSP?

No, JSP specification does not allow overriding of _jspService method in JSP. We can override other methods like jspInit() and jspDestroy().

34. What is a directive in JSP?

JSP directive is a mechanism to pass message to JSP container. JSP directive does not produce an output to the page. But it communicates with JSP container.

E.g. <%@include ..%> directive is used for telling JSP container to include the content of another file during translation of JSP.

There can be zero or more attributes in a directive to pass additional information to JSP container.

Some of the important directives in JSP are: page, include and taglib.

35. How will you implement Session tracking in JSP?

We can use different mechanisms to implement Session tracking JSP. Some these mechanisms are as follows:

1. **Cookies**: We can use cookie to set session information and pass it to web client. In subsequent requests we can use the information in cookie to track session.

2. **Hidden Form Field**: We can send session id in a hidden field in HTML form. By using this we can track session.

3. **Session object**: We can use the built in session object to track session in JSP.

4. **URL Rewriting**: We can also add session id at the end of a URL.

   Like- www.abcserver.com?sessionid=1234

36. How do you debug code in JSP?

In simplest form we can write logger statements or System.out.println() statements to write messages to log files. When we call a JSP, the log messages get written to logs. With useful information getting logged we can easily debug the code.

Another option in debugging is to link JSP container with an IDE. Once we link IDE debugger to JSP Engine, we can use standard operations of debugging like breakpoint, step through etc.

37. How will you implement error page in JSP?

To implement an error-handling page in JSP, we first create a JSP with error page handling information. In most of the cases we gracefully handle error by giving a user-friendly message like “Sorry! There is system error. Please try again by refreshing page.”

In this error page, we show user-friendly message to user, but we also log important information like stack trace to our application log file.

We have to add parameter isErrorPage=true in page directive of this page. This tells to JSP container that this is our error page.

<%@page isErrorPage=”true” %>

Now we can use this error page in other JSP where we want to handle error. In case of an error or exception, these JSP will direct it to errorPage.

<% page errorPage=”ErrorPage.jsp” %>

38. How will you send XML data from a JSP?

In general, JSP is used to pass HTML data to web browser. If we want to send data in XML format, we can easily do it by setting contentType=”text/xml” in page directive.

E.g. <%@page contentType=”text/xml” %>

39. What happens when we request for a JSP page from web browser?

When a user calls JSP page from web browser, the request first comes to web server. Web server checks for .jsp extension of page and passes the request to JSP container like Tomcat.

The JSP container checks whether it has precompiled JSP class or not. If this is the first time this JSP is called, then JSP container will translate JSP into a servlet and compiles it.

After compiling, JSP code if loaded in memory and JSP container will call jspInit() method and _jspService() methods.

The _jspService() method will create the output that will be sent by JSP container to client browser.

40. How will you implement Auto Refresh of page in JSP?

We can use setIntHeader() method to set the refresh frequency with which we want to auto-refresh a JSP page.

We can send key “Refresh” with the time in seconds for auto refresh of the JSP page.

E.g. response.setIntHeader(“Refresh”,10)

41. What are the important status codes in HTTP?

Every HTTP request comes back with a status code from the server. The important status codes in HTTP are as follows:

1. 200: It means the request is successful.
2. 400: It means the request was bad.
3. 401: It means request was not authorized.
4. 404: It means the resource requested was not found.
5. 503: It means the service is not available.

42. What is the meaning of Accept attribute in HTTP header?

In HTTP header, Accept attribute is used to specify the MIME types that a HTTP client or browser can handle. MIME type is the identifier for specifying the type of file/data that we are planning to pass over the internet.

43. What is the difference between Expression and Scriptlet in JSP?

We use Expression in a JSP to return a value and display it at a specific location. It is generally used for dynamically print information like- time, counter etc in a HTML code.

Scriptlet is for writing Java code in a JSP. We can define variable, methods etc in a Scriptlet. A Scriptlet can handle much more complex code and can be also reused.

44. How will you delete a Cookie in JSP?

We can use following options to delete a Cookie in JSP:

1. **setMaxAge**(): we can set the maximum age of a cookie. After this time period, Cookie will expire and will be deleted.

2. **Header**: We can also set the expiry time in header of response. Respone.setHeader(). This will also expire the cookie after specified time period.

45. How will you use a Cookie in JSP?

We can use a Cookie in JSP by performing following steps:

First we create a Cookie object. We set the name and value of the cookie to be created.

We set the expiry time of the Cookie by setting the maximum age. We can use setMaxAge() method for this.

Finally, we can send the cookie in a HTTP Response by sending it in HTTP header. In this way cookie goes to client browser and gets stored there till the maximum age is not achieved.

Once a Cookie is set in the client browser, we can call getCookies() method to get the list of all the cookies set in Client. We iterate through the list of all the cookies and get the value of the cookie that was set in earlier request.

In this way we can use Cookie to set some information at client side and retrieve its value.

46. What is the main difference between a Session and Cookie in JSP?

A Session is always stored at the Server side. In JSP, session is a built-in object in JSP container.

A Cookie is always stored at the client side.

We can use both the methods for Session tracking. But Cookie method needs permission from user for storing cookie at the client location.

47. How will you prevent creation of session in JSP?

We can simply set the session attribute as false in page directive to prevent creation of session object.

E.g. <% @page session=”false” %>

48. What is an output comment in JSP?

We can write output in JSP in such a way that it becomes a comment in HTML code. This comment will not be visible in the web browser. But when we view page source to see HTML, we can see output comment.

An HTML comment is of following format:

<!-- comment -->

If we output comment in above format, it will be visible to client.

49. How will you prevent caching of HTML output by web browser in JSP?

We can use set the header in response object for Cache-Control to specify no caching.

Sample code is as follows:

```
response.setHeader(“Cache-Control”, “no-store”);
response.setDateHeader(“Expires”,”0”);
```

50. How will you redirect request to another page in browser in JSP code?

We can use sendRedirect() method in JSP to redirect the request to another location or page.

In this case the request will not come back to server. It will redirect in the browser itself.

Sample code is as follows:

<% response.sendRedirect(URL); %>

51. What is the difference between sendRedirect and forward in a JSP?

Both forward and sendRedirect are mechanisms of sending a client to another page. The main difference between these two are as follows:

1. In forward, the processing takes place at server side. In case of sendRedirect() the processing takes place the client side.

2. In forward, the request is transferred to another resource within same server. In case of sendRedirect the request can be transferred to resource on some other server.

3. In forward only one request call is consumed. In case of sendRedirect two request response calls are created and consumed.

4. The forward is declared in RequestDispatcher interface. Where as sendRedirect is declared in HttpServletResponse object.

52. What is the use of config implicit object in JSP?

In JSP, config object is of type ServletConfig. This object is created by Servlet Container for each JSP page. It is used for setting initialization parameters for a specific JSP page.

53. What is the difference between init-param and context-param?

We can specify both init-param and context-param in web.xml file.

We use init-param to specify the parameters that are specific to a servlet or jsp. This information is confined to the scope of that JSP.

We use context-param to specify the parameters for overall application scope. This information does not change easily. It can be used by all the JSP/Servlet in that Container.

54. What is the purpose of RequestDispatcher?

We use RequestDispatcher interface to forward requests to other resources like HTML, JSP etc.

It can also be used to include the content of another page in a JSP.

It has two methods: forward and include.

We have to first get the RequestDispatcher object from the container and then we can call include or forward method on this object.

55. How can be read data from a Form in a JSP?

There is a built-in request object in a JSP that provides methods to read Form data. Some of the methods are as follows::

1. **getParameterNames():** This method returns the list of all the parameters in the Form.

2. **getParameter():** We call this method to get the value of parameter set in the Form. It returns null if the parameter is not found.

3. **getParameterValues():** If a Parameter is mentioned multiple times in a Form, we use request.getParameterValues() method to get all the values. This method returns an array of String values.

4. **getParameterMap():** This method returns the map of all the Parameters in Form.

56. What is a filter in JSP?

We can define filters in JSP to intercept requests from a client or to change response from a server.

Filter is a Java class that is defined in the deployment descriptor of web.xml of an application. The JSP container reads filter from web.xml and applies a filter as per the URL pattern associated with the filter.

JSP Engine loads all the filters in when we start the server.

57. How can you upload a large file in JSP?

To upload a file by JSP we can use <input type=”file”> in the Form data being passed from HTML.

If the file is very large in size, we can set enctype=multipart/form-data.

We have to use POST method in the Form to send a file.

Once the request is received, we can implement the logic to read mulitpart data in doPost() method of JSP. There are methods in JSP framework to read large files via this method.

58. In which scenario, Container initializes multiple JSP/Servlet objects?

To initialize multiple JSP objects, we have to specify same Servlet object multiple times in web.xml.

This indicates to JSP container to initialize separate JSP/Servlet object for each element. Each of the Servlet instance will have its own ServletConfig object and parameters.

Java Design Patterns

59. When will you use Strategy Design Pattern in Java?

Strategy pattern is very useful for implementing a family of algorithms. It is a behavioral design pattern.

With Strategy pattern we can select the algorithm at runtime. We can use it to select the sorting strategy for data. We can use it to save files in different formats like- .txt, .csv, .jpg etc.

In Strategy pattern we create an abstraction, which is an interface through which clients interact with our system. Behind the abstraction we create multiple implementation of same interface with different algorithms.

For a client, at runtime we can vary the algorithm based on the type of request we have received.

So we use Strategy pattern to hide the algorithm implementation details from client.

In Java Collections.sort() method uses strategy design pattern.

60. What is Observer design pattern?

In Observer design pattern, there is a Subject that maintains the list of Observers that are waiting for any update on the Subject. Once there is an update in Subject it notifies all the observers for the change.

E.g. In real life, students are waiting for the result of their test. Here students are the observers and test is the subject. Once the result of test is known, testing organization notifies all the students about their result.

The most popular use of Observer pattern is in Model View Controller (MVC) architectural pattern.

Main issue with Observer pattern is that it can cause memory leaks. The subject holds a strong reference to observers. If observers are not de-registered in time, it can lead to memory leak.

61. What are the examples of Observer design pattern in JDK?

In JDK there are many places where Observer design pattern is used. Some of these are as follows:

1. java.util.Observer, java.util.Observable
2. javax.servlet.http.HttpSessionAttributeListener
3. javax.servlet.http.HttpSessionBindingListener
4. All implementations of java.util.EventListener, and also in Swing packages
5. javax.faces.event.PhaseListener

62. How Strategy design pattern is different from State design pattern in Java?

State design pattern is a behavioral design pattern that is use for defining the state machine for an object. Each state of an object is defined in a child class of State class. When different actions are taken on an Object, it can change its state.

Strategy pattern is also a behavioral pattern, but it is mainly used for defining multiple algorithms. With same action of a client, the algorithm to be used can change.

Some people consider State pattern similar to Strategy pattern, since an Object changes its Strategy with different method invocations. But the main difference is that in State pattern internal state of an Object is one of the determining factors for selecting the Strategy for change of state.

Where as in Strategy pattern, client can pass some external parameter in input during method invocation that determines the strategy to be used at run time.

Therefore State pattern is based on the Object's internal state, where as Strategy pattern is based on Client's invocation.

State pattern is very useful in increasing the maintainability of the code in a large code-base.

63. Can you explain Decorator design pattern with an example in Java?

Some people call Decorator pattern as Wrapper pattern as well. It is used to add the behavior to an object, without changing the behavior of other objects of same class.

One of the very good uses of Decorator pattern is in java.io package. We can have a FileInputStream to handle a File. To add Buffering behavior we can decorate FileInputStream with BufferedInputStream. To add the gzip behavior BufferedInputStream we can decorate it with GzipInputStream. To add serialization behavior to GzipInputStream, we can decorate it with ObjectInputStream.

E.g.
Open a FileInputStream:

FileInputStream fis = new FileInputStream("/myfile.gz");

Add buffering:

BufferedInputStream bis = new BufferedInputStream(fis);

Add Gzip:

GzipInputStream gis = new GzipInputStream(bis);

Add Serialization:

ObjectInputStream ois = new ObjectInputStream(gis);

So with each step we have decorated the FileInputStream with additional behavior.

64. What is a good scenario for using Composite design Pattern in Java?

Some of the good scenarios where Composite design pattern can be used are as follows:

Tree Structure: The most common use of Composite design pattern is Tree structure. If you want to represent data in a Tree data structure, Composite pattern can be used.

E.g. In an Organization, to a Manager has Employees. But Manager is also an Employee. If we start from CEO level, there is one big tree for the whole organization structure. Under that big tree there are many sub-trees. This can be easily represented with Composite design pattern.

Recursion: Another use of Composite design pattern is Recursion. If we have a Recursion based algorithm, we need data to be passed to algorithm in a data structure that treats individual objects and compositions at each level of recursion uniformly.
E.g. To implement a recursive Polynomial Solving algorithm, we can use Composite design pattern to store the intermediate results.

Graphics: Another good use of Composite design pattern is in Graphics. We can group shapes inside a composite and make higher-level groups of smaller groups of shapes to complete the graphics to be displayed on screen.

65. Have you used Singleton design pattern in your Java project?

Yes. Singleton is one of the most popular design patterns in enterprise level Java applications. Almost in every project we see some implementation of Singleton.

With Singleton pattern we can be sure that there is only one instance of a class at any time in the application.

This helps in storing properties that have to be used in the application in a unique location.

66. What are the main uses of Singleton design pattern in Java project?

Some of the main uses of Singleton design pattern in Java are as follows:

1. Runtime: In JDK, java.lang.Runtime is a singleton-based class. There is only one instance of Runtime in an application. This is the only class that interfaces with the environment/machine in which Java process is running.

2. Enum: In Java, enum construct is also based on Singleton pattern. Enum values can be accessed globally in same way by all classes.

3. Properties: In an application it makes sense to keep only one copy of the properties that all classes can access. This can be achieved by making properties class Singleton so that every class gets same copy of properties.

4. Spring: In Spring framework, all the beans are by default Singleton per container. So there is only one instance of bean in a Spring IoC container. But Spring also provides options to make the scope of a bean prototype in a container.

67. Why java.lang.Runtime is a Singleton in Java?

In Java, java.lang.Runtime is implemented on Singleton design pattern.

Runtime is the class that acts as an interface with the environment in which Java process is running. Runtime contains methods that can interact with the environment.

Like- totalmemory() method gives the total memory in JVM. maxMemory() method gives the maximum memory that JVM can use.

There is an exit() method to exit the Java process. We do not want multiple objects in JVM to have exit() method.

Similarly there is gc() method that can run the Garbage Collector. With only one copy of gc() method, we can ensure that no other object can run the Garbage Collector when one instance of GC is already running.

Due to all these reasons there is only one copy of Runtime in Java. To ensure single copy of Runtime, it is implemented as a Singleton in Java.

68. What is the way to implement a thread-safe Singleton design pattern in Java?

In Java there are many options to implement a thread-safe Singleton pattern. Some of these are as follows:

1. Double Checked Locking: This is the most popular method to implement Singleton in Java. It is based on Lazy Initialization. In this we first check the criteria for locking before acquiring a lock to create an object. In Java we use it with volatile keyword.

Sample code:

```
class DoubleCheckSingleton {
   private volatile HelloSingleton helloSingleton; // Use Volatile

   public HelloSingleton getHelloSingleton() {
      HelloSingleton result = helloSingleton;
      if (result == null) {
         synchronized(this) { // Synchronize for thread safety
            result = helloSingleton;
            if (result == null) {
   result = new HelloSingleton();
               helloSingleton = result;
            }
         }
      }
      return result;
   }

}
```

2. Bill Pugh Singleton: We can also use the method by Bill Pugh for implementing Singleton in Java. In this we use an Inner Static class to create the Singleton instance.

Sample code:

```
public class SingletonBillPugh {

   // Inner class that holds instance
   private static class InnerSingleton{
         private static final SingletonBillPugh INSTANCE = new
SingletonBillPugh();
   }

    // Private constructor
   private SingletonBillPugh(){}

public static SingletonBillPugh getInstance(){
      return InnerSingleton.INSTANCE;
   }
}
```

When first time SingletonBillPugh is loaded in memory, InnerSingleton is not loaded. Only when getInstance() method is called, InnerSingleton class is loaded and an Instance is created.

3. Enum: We can also use Java enum to create thread-safe implementation. Java enum values are accessible globally so these can be used as a Singleton.

Sample Code:

```
public enum SingletonEnum {

   INSTANCE;
```

```
    public static void doImplementation(){
     …..
    }
}
```

69. What are the examples of Singleton design pattern in JDK?

In JDK there are many places where Singleton design pattern is used. Some of these are as follows:

1. java.lang.Runtime.getRuntime(): This method gives Runtime class that has only one instance in a JVM.

   java.lang.System.getSecurityManager(): This method returns a SecurityManager for the current platform.

   java.awt.Desktop.getDesktop()

70. What is Template Method design pattern in Java?

It is a behavioral design pattern. We can use it to create an outline for an algorithm or a complex operation. We first create the skeleton of a program. Then we delegate the steps of the operation to subclasses. The subclasses can redefine the inner implementation of each step.

E.g. While designing a Game in Java, we can implement it as an algorithm with Template Method pattern. Each step in the game can be deferred to subclasses responsible for handling that step.

Let say we implement Monopoly game in Java. We can create methods like initializeGame(), makeMove(), endGame() etc. Each of these methods can be handled in subclasses in an independent manner.

We can use same algorithm for Chess game with same set of abstract methods. The subclass for Chess game can provide the concrete implementation of methods like initializeGame(), makeMove(), endGame() etc.

Template Method pattern is very useful in providing customizable class to users. We can create the core class with a high level implementation. And our users can customize our core class in their custom subclasses.

71. What are the examples of Template method design pattern in JDK?

In JDK there are many places where Template method design pattern is used. Some of these are as follows:

1. In Java Abstract Collection classes like java.util.AbstractList, java.util.AbstractSet and java.util.AbstractMap implement a template for their corresponding Collection.

2. javax.servlet.http.HttpServlet: In the HttpServlet class all the doGet(), doPost() etc. methods send a HTTP 405 "Method Not Allowed" error to the response. This error response is like a Template that can be further customized for each of these methods.

3. In java.io package there are Stream and Writer classes like java.io.InputStream, java.io.OutputStream, java.io.Reader and java.io.Writer that provide non-abstract methods. These methods are implementation of Template method design pattern.

72. Can you tell some examples of Factory Method design pattern implementation in Java?

Factory Method pattern is a creational design pattern. A Factory is an object that is used to create more objects.

In general, a Factory object has methods that can be used to create a type of objects. Some people call it Factory Method design pattern as well.

Some of the examples of Factory Method pattern in JDK are:

- Java.lang.Class.forName()
- java.net.URLStreamHandlerFactory.createURLStreamHan
- java.util.Calendar.getInstance()
- java.util.ResourceBundle.getBundle()
- java.text.NumberFormat.getInstance()
- java.nio.charset.Charset.forName()
- java.util.EnumSet.of()
- javax.xml.bind.JAXBContext.createMarshaller()

73. What is the benefit we get by using static factory method to create object?

By using Static Factory Method we encapsulate the creation process of an object. We can use new() to create an Object from its constructor. Instead we use static method of a Factory to create the object. One main advantage of using Factory is that Factory can choose the correct implementation at runtime and create the right object. The caller of method can specify the desired behavior.

E.g. If we have a ShapeFactory with createShape(String type) method. Client can call ShapeFactory.createShape("Circle") to get a circular shape. ShapeFactory.createShape("Square") will return square shape. In this way, ShapeFactory knows how to create different shapes based on the input by caller.

Another use of Factory is in providing access to limited resources to a large set of users.

E.g. In ConnectionPool, we can limit the total number of connections that can be created as well as we can hide the implementation details of creating connection. Here ConnectionPool is the factory. Clients call static method ConnectionPool.getConnection().

74. What are the examples of Builder design pattern in JDK?

In JDK there are many places where Builder design pattern is used. Some of these are as follows:

1. java.lang.StringBuilder.append(): StringBuilder is based on Builder pattern.

2. java.nio.IntBuffer.put(): Invocation of put() method return IntBuffer. Also there are many variants of this method to build the IntBuffer.

3. javax.swing.GroupLayout.Group.addComponent(): We can use addComponent() method to build a UI that can contain multiple levels of components.

4. java.lang.Appendable

5. java.lang.StringBuffer.append(): StringBuffer is similar to StringBuilder and it is also based on Builder design pattern.

75. What are the examples of Abstract Factory design pattern in JDK?

In JDK there are many places where Abstract Factory design pattern is used. Some of these are as follows:

- javax.xml.xpath.XPathFactory.newInstance()
- javax.xml.parsers.DocumentBuilderFactory.newInstance()
- javax.xml.transform.TransformerFactory.newInstance()

76. What are the examples of Decorator design pattern in JDK?

In JDK there are many places where Decorator design pattern is used. Some of these are as follows:

1. In java.io package many classes use Decorator pattern. Subclasses of java.io.InputStream, OutputStream, Reader and Writer have a constructor that can take the instance of same type and decorate it with additional behavior.

2. In java.util.Collections, there are methods like checkedCollection(), checkedList(), checkedMap(), synchronizedList(), synchronizedMap(), synchronizedSet(), unmodifiableSet(), unmodifiableMap() and unmodifiableList() methods that can decorate an object and return the same type.

3. In javax.servlet package, there are classes like javax.servlet.http.HttpServletRequestWrapper and HttpServletResponseWrapper that are based on Decorator design pattern.

77. What are the examples of Proxy design pattern in JDK?

Proxy design pattern provides an extra level of indirection for providing access to another object. It can also protect a real object from any extra level of complexity.

In JDK there are many places where Proxy design pattern is used. Some of these are as follows:

- java.lang.reflect.Proxy
- java.rmi.*
- javax.inject.Inject
- javax.ejb.EJB
- javax.persistence.PersistenceContext

78. What are the examples of Chain of Responsibility design pattern in JDK?

In JDK there are many places where Chain of Responsibility design pattern is used. Some of these are as follows:

1. java.util.logging.Logger.log(): In this case Logger class provides multiple variations of log() method that can take the responsibility of logging from client in different scenarios. The client has to just call the appropriate log() method and Logger will take care of these commands.

2. javax.servlet.Filter.doFilter(): In the Filter class, the Container calls the doFilter method when a request/response pair is passed through the chain. With filter the request reaches to the appropriate resource at the end of the chain. We can pass FilterChain in doFilter() method to allow the Filter to pass on the request and response to the next level in the chain.

79. What are the main uses of Command design pattern?

Command design pattern is a behavioral design pattern. We use it to encapsulate all the information required to trigger an event. Some of the main uses of Command pattern are:

1. Graphic User Interface (GUI): In GUI and menu items, we use command pattern. By clicking a button we can read the current information of GUI and take an action.

2. Macro Recording: If each of user action is implemented as a separate Command, we can record all the user actions in a Macro as a series of Commands. We can use this series to implement the "Playback" feature. In this way, Macro can keep on doing same set of actions with each replay.

3. Multi-step Undo: When each step is recorded as a Command, we can use it to implement Undo feature in which each step can by undo. It is used in text editors like MS-Word.

4. Networking: We can also send a complete Command over the network to a remote machine where all the actions encapsulated within a Command are executed.

5. Progress Bar: We can implement an installation routine as a series of Commands. Each Command provides the estimate time. When we execute the installation routine, with each command we can display the progress bar.

6. Wizard: In a wizard flow we can implement steps as Commands. Each step may have complex task that is just implemented within one command.

7. Transactions: In a transactional behavior code there are multiple tasks/updates. When all the tasks are done then only transaction is committed. Else we have to rollback the transaction. In such a scenario each step is implemented as separate Command.

80. What are the examples of Command design pattern in JDK?

In JDK there are many places where Command design pattern is used. Some of these are as follows:

- All implementations of java.lang.Runnable
- All implementations of javax.swing.Action

81. What are the examples of Interpreter design pattern in JDK?

Interpreter design pattern is used to evaluate sentences in a language. E.g. In SQL we can use it to evaluate a query by evaluating each keyword like SELECT, FROM, WHERE clause.

In an Interpreter implementation there is a class for each keyword/symbol. A sentence is just a composite of these keywords. But the sentence is represented by Syntax tree that can be interpreted.

In JDK there are many places where Interpreter design pattern is used. Some of these are as follows:

- java.util.Pattern
- java.text.Normalizer
- Subclasses of java.text.Format: DateFormat, MessageFormat, NumberFormat
- Subclasses of javax.el.ELResolver: ArrayELResolver, MapELResolver, CompositeELResolver etc.

82. What are the examples of Mediator design pattern in JDK?

By using Mediator pattern we can decouple the multiple objects that interact with each other. With a Mediator object we can create many-to-many relationships in multiple objects.

In JDK there are many places where Mediator design pattern is used. Some of these are as follows:

- java.util.Timer: schedule() methods in Timer class act as Mediator between the clients and the TimerTask to be scheduled.
- java.util.concurrent.Executor.execute(): The execute() method in an Executor class acts as a Mediator to execute the different tasks.
- java.util.concurrent.ExecutorService
- java.lang.reflect.Method.invoke(): In Method class of reflection package, invoke() method acts as a Mediator.
- java.util.concurrent.ScheduledExecutorService: Here also schedule() method and its variants are Mediator pattern implementations.

83. What are the examples of Strategy design pattern in JDK?

In JDK there are many places where Strategy design pattern is used. Some of these are as follows:

1. java.util.Comparator: In a Comparator we can use compare() method to change the strategy used by Collections.sort() method.

2. javax.servlet.http.HttpServlet: In a HttpServlet class service() and doGet(), doPost() etc. methods take HttpServletRequest and HttpServletResponse and the implementor of Servlet processes it based on the strategy it selects.

84. What are the examples of Visitor design pattern in JDK?

By using Visitor design pattern we can add new virtual methods to existing classes without modifying their core structure.

In JDK there are many places where Visitor design pattern is used. Some of these are as follows:

- javax.lang.model.element.AnnotationValue and AnnotationValueVisitor
- java.nio.file.FileVisitor and SimpleFileVisitor
- javax.lang.model.type.TypeMirror and TypeVisitor
- javax.lang.model.element.Element and ElementVisitor
- javax.faces.component.visit.VisitContext and VisitCallback

85. How Decorator design pattern is different from Proxy pattern?

Main differences between Decorator and Proxy design pattern are:

- Decorator provides an enhanced interface after decorating it with additional features. Proxy provides same interface since it is just acting as a proxy to another object.

- Decorator is a type of Composite pattern with only one component. But each decorator can add additional features. Since it is one component in Decorator, there is no object aggregation.

- Proxy can also provide performance improvement by lazy loading. There is nothing like this available in Decorator.

- Decorator follows recursive composition. Proxy is just one object to another object access.

- Decorator is mostly used for building a variety of objects. Proxy is mainly used for access to another object.

86. What are the different scenarios to use Setter and Constructor based injection in Dependency Injection (DI) design pattern?

We use Setter injection to provide optional dependencies of an object. Constructor injection is used to provide mandatory dependency of an object.

In Spring IoC, Dependency Injection is heavily used. There we have to differentiate between the scenario suitable for Setter based and Constructor based dependency injection.

87. What are the different scenarios for using Proxy design pattern?

Proxy design pattern can be used in a wide variety of scenario in Java. Some of these are as follows:

1. Virtual Proxy: This is a virtual object that acts as a proxy for objects that are very expensive to create. It is used in Lazy Loading. When client makes the first request, the real object is created.

2. Remote Proxy: This is a local object that provides access to a remote object. It is generally used in Remote Method Invocation (RMI) and Remote Procedure Call (RPC). It is also known as a Stub.

3. Protective Proxy: This is an object that control the access to a Master object. It can authenticate and authorize the client for accessing the Master object. If client has right permissions, it allows client to access the main object.

4. Smart Proxy: It is an object that can add additional information to the main object. It can track the number of other objects accessing the main object. It can track the different clients from where request is coming. It can even deny access to an object if the number of requests is greater than a threshold.

88. What is the main difference between Adapter and Proxy design pattern?

Adapter pattern provides a different interface to an object. But the Proxy always provides same interface to the object.

Adapter is like providing an interface suitable to client's use. But Proxy is same interface that has additional feature or check.

E.g. In electrical appliances we use Adapter to convert from one type of socket to another type of socket. In case of proxy, we have a plug with built-in surge protector. The interface for plug and the original device remains same.

89. When will you use Adapter design pattern in Java?

If we have two classes with incompatible interfaces, we use Adapter pattern to make it work. We create an Adapter object that can adapt the interface of one class to another class.

It is generally used for working with third party libraries. We create an Adapter class between third party code and our class. In case of any change in third party code we have to just change the Adapter code. Rest of our code can remain same and just take to Adapter.

90. What are the examples of Adapter design pattern in JDK?

In JDK there are many places where Adapter design pattern is used. Some of these are as follows:

- java.util.Arrays.asList(): This method can adapt an Array to work as a List.
- java.util.Collections.list(): This method can adapt any collection to provide List behavior.
- java.util.Collections.enumeration(): This method returns an enumeration over the collection.
- java.io.InputStreamReader(InputStream): This method adapts a Stream to Reader class.
- java.io.OutputStreamWriter(OutputStream): This method adapts an OutputStream to Writer class.
- javax.xml.bind.annotation.adapters.XmlAdapter.marshal()

91. What is the difference between Factory and Abstract Factory design pattern?

With Factory design pattern we can create concrete products of a type that Factory can manufacture. E.g. If it is CarFactory, we can produce, Ford, Toyota, Honda, Maserati etc.

With Abstract Factory design pattern we create a concrete implementation of a Factory. E.g. DeviceFactory can be Abstract and it can give us GoogleDeviceFactory, AppleDeviceFactory etc. With AppleDeviceFactory we will get products like- iPhone, iPad, Mac etc. With GoogleDeviceFactory we will get products like- Nexus phone, Google Nexus tablet, Google ChromeBook etc.

So it is a subtle difference between Factory and Abstract Factory design pattern. One way to remember is that within Abstract Factory pattern, Factory pattern is already implemented.

92. What is Open/closed design principle in Software engineering?

Open/closed design principle states “software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification”.

Open/closed principle term was originated by Bertrand Meyer in his book Object Oriented Software Construction.

As per this principle, if a module is available for extension then it is considered open. If a module is available for use by other modules then it is considered closed.

Further Robert C. Martin has mentioned it as O in SOLID principles of Object Oriented design.

It is used in State and Strategy design patterns. Context class is closed for modification. But new functionality can be added by writing new strategy code.

93. What is SOLID design principle?

SOLID word in SOLID design principle is an acronym for:

1. S: Single responsibility. A Class should have a single responsibility.
2. O: Open-closed. Software entities should be open for extension but closed for modification.
3. L: Liskov substitution. Objects in a program should be replaceable by subclasses of same type without any adverse impact.
4. I: Interface segregation. Multiple client specific interfaces are preferable over single generic interface.
5. D: Dependency inversion. Program should depend on abstract entities. It should not depend on concrete implementation of an interface.

This principle was mentioned by Robert C. Martin. These are considered five basic principles of Object Oriented design.

If we follow these principles, then we can create a stable program that is easy to maintain and can be extended over time.

94. What is Builder design pattern?

Builder design pattern is a creational design pattern. We can use Builder pattern to create complex objects with multiple options.

E.g. when we have to create a Meal in a restaurant we can use Builder pattern. We can keep adding options like- Starter, Drink, Main Course, and Dessert etc. to create complete meal. When a user selects other options of Starter, Drink Main Course, Dessert another type of meal is created.

Main feature of Builder pattern is step-by-step building of a complex object with multiple options.

95. What are the different categories of Design Patterns used in Object Oriented Design?

In Object Oriented design mainly three categories of design patterns are used. These categories are:

- Creational Design Patterns:
- Builder
- Factory Method
- Abstract Factory
- Object Pool
- Singleton
- Prototype

- Structural Design Patterns:
- Adapter
- Bridge
- Façade
- Decorator
- Composite
- Flyweight
- Proxy

- Behavioral Design Patterns:
- Command
- Iterator
- Chain of Responsibility
- Observer
- State
- Strategy
- Mediator
- Interpreter

96. What is the design pattern suitable to access elements of a Collection?

We can use Iterator design pattern to access the individual elements of a Collection. In case of an ordered collection we can get Iterator that returns the elements in an order.

In Java there are many implementation of Iterator in Collections package. We have iterators like- Spliterator, ListIterator etc. that implement Iterator pattern.

97. How can we implement Producer Consumer design pattern in Java?

We can use BlockingQueue in Java to implement Producer Consumer design pattern.

It is a concurrent design pattern.

98. What design pattern is suitable to add new features to an existing object?

We can use Decorator design pattern to add new features to an existing object. With a Decorator we work on same object and return the same object with more features. But the structure of the object remains same since all the decorated versions of object implement same interface.

99. Which design pattern can be used when to decouple abstraction from the implementation?

We can use Bridge design pattern to detach the implementation from the abstraction.

Bridge is mainly used for separation of concern in design. We can create an implementation and store it in the interface, which is an abstraction. Where as specific implementation of other features can be done in concrete classes that implement the interface.

Often Bridge design pattern is implemented by using Adapter pattern.

E.g. we have Shape interface. We want to make Square and Circle shapes. But further we want to make RedSquare, BlackSquare shapes and GreenCircle, WhiteCircle shapes. In this case rather than creating one hierarchy of all the shapes, we separate the Color concern from Shape hierarchy.

So we create two hierarchies. One is Shape to Square and Shape to Circle hierarchy. Another one is Color to Red, Black, Green, White hierarchy. In this way we can create multiple types of shapes with multiple colors with Bridge design pattern.

100. Which is the design pattern used in Android applications?

Android applications predominantly use Model View Presenter design pattern.

1. Model: This is the domain model of the Android application. It contains the business logic and business rules.

2. View: These are the UI components in your application. These are part of the view. Also any events on UI components are part of view module.

3. Presenter: This is the bridge between Model and View to control the communication. Presenter can query the model and return data to view to update it.

4. E.g. If we have a Model with large news article data, and view needs only headline, then presenter can query the data from model and only give headline to view. In this way view remains very light in this design pattern.

101. How can we prevent users from creating more than one instance of singleton object by using clone() method?

First we should not implement the Cloneable interface by the object that is a Singleton.

Second, if we have to implement Cloneable interface then we can throw exception in clone() method.

This will ensure that no one can use clone() method or Cloneable interface to create more than one instance of Singleton object.

102. What is the use of Interceptor design pattern?

Interceptor design pattern is used for intercepting a request. Primary use of this pattern is in Security policy implementation.

We can use this pattern to intercept the requests by a client to a resource. At the interception we can check for authentication and authorization of client for the resource being accessed.

In Java it is used in javax.servlet.Filter interface.

This pattern is also used in Spring framework in HandlerInterceptor and MVC interceptor.

103. What are the Architectural patterns that you have used?

Architectural patterns are used to define the architecture of a Software system. Some of the patterns are as follows:

1. MVC: Model View Controller. This pattern is extensively used in the architecture of Spring framework.

2. Publish-subscribe: This pattern is the basis of messaging architecture. In this case messages are published to a Topic. And subscribers subscribe to the topic of their interests. Once the message is published to a topic in which a Subscriber has an interest, the message is consumed by the relevant subscriber.

3. Service Locator: This design pattern is used in a service like JNDI to locate the available services. It uses as central registry to maintain the list of services.

4. n-Tier: This is a generic design pattern to divide the architecture in multiple tiers. E.g. there is 3-tier architecture with Presentation layer, Application layer and Data access layer. It is also called multi-layer design pattern.

5. Data Access Object (DAO): This pattern is used in providing access to database objects. The underlying principle is that we can change the underlying database system, without changing the business logic. Since business logic talks to DAO object, there is no impact of changing Database system on business logic.

6. Inversion of Control (IoC): This is the core of Dependency Injection in Spring framework. We use this design pattern

to increase the modularity of an application. We keep the objects loosely coupled with Dependency Injection.

104. What are the popular uses of Façade design pattern?

Some of the popular uses of Façade design pattern are as follows:

1. A Façade provides convenient methods for common tasks that are used more often.
2. A Façade can make the software library more readable.
3. A Façade can reduce the external dependencies on the working of inner code.
4. A Façade can act as a single well-designed API by wrapping a collection of poorly designed APIs.
5. A Façade pattern can be used when a System is very complex and difficult to use. It can simplify the usage of complex system.

105. What is the difference between Builder design pattern and Factory design pattern?

Both Factory and Builder patterns are creational design patterns. They are similar in nature but Factory pattern is a simplified generic version of Builder pattern.

We use Factory pattern to create different concrete subtypes of an Object. The client of a Factory may not know the exact subtype. E.g. If we call createDrink() of a Factory, we may get Tea or Coffee drinks.

We can also use Builder pattern to create different concrete subtypes of an object. But in the Builder pattern the composition of the object can be more complex. E.g. If we call createDrink() for Builder, we can getCappuccino Coffee with Vanilla Cream and Sugar, or we can get Latte Coffee with Splenda and milk cream.

So a Builder can support creation of a large number of variants of an object. But a Factory can create a broader range of known subtypes of an object.

106. What is Memento design pattern?

- Memento design pattern is used to implement rollback feature in an object. In a Memento pattern there are three objects:

- Originator: This is the object that has an internal state.

- Caretaker: This is the object that can change the state of Originator. But it wants to have control over rolling back the change.

- Memento: This is the object that Caretaker gets from Originator, before making and change. If Caretaker wants to Rollback the change it gives Memento back to Originator. Originator can use Memento to restore its own state to the original state.

E.g. One good use of memento is in online Forms. If we want to show to user a form pre-populated with some data, we keep this copy in memento. Now user can update the form. But at any time when user wants to reset the form, we use memento to make the form in its original pre-populated state. If user wants to just save the form we save the form and update the memento. Now onwards any new changes to the form can be rolled back to the last saved Memento object.

107. What is an AntiPattern?

An AntiPattern is opposite of a Design Pattern. It is a common practice in an organization that is used to deal with a recurring problem but it has more bad consequences than good ones.

AntiPattern can be found in an Organization, Architecture or Software Engineering.

Some of the AntiPatterns in Software Engineering are:

1. Gold Plating: Keep on adding extra things on a working solution even though these extra things do not add any additional value.

2. Spaghetti Code: Program that are written in a very complex way and are hard to understand due to misuse of data structures.

3. Coding By Exception: Adding new code just to handle exception cases and corner case scenarios.

4. Copy Paste Programming: Just copying the same code multiple times rather than writing generic code that can be parameterized.

108. What is a Data Access Object (DAO) design pattern?

DAO design pattern is used in the data persistent layer of a Java application. It mainly uses OOPS principle of Encapsulation.

By using DAO pattern it makes the application loosely coupled and less dependent on actual database.

We can even implement some in-memory database like H2 with DAO to handle the unit-testing.

In short, DAO hides the underlying database implementation from the class that accesses the data via DAO object.

Recently we can combine DAO with Spring framework to inject any DB implementation.

Spring

109. What is Spring framework?

Spring is development framework for Java programming. It is an open source development framework for Enterprise Java.

The core features of Spring Framework can be used in developing a Java Enterprise application.

It has many extensions and jars for developing web applications on top of Java EE platform.

With Spring we can develop large-scale complex Java applications very easily. It is also based on good design patterns like Dependency Injection, Aspect oriented programming for developing extensible feature rich software.

110. What are the benefits of Spring framework in software development?

Many benefits of Spring framework are:

Lightweight Framework: Basic Spring framework is very small in size. It is easy to use and does not add a lot of overhead on software. It just has 2 MB in basic version.

Container:

The container will create the objects, wire them together, configure them, and manage their complete life cycle from creation till destruction

Container: Spring framework provides the basic container that creates and manages the life cycle of application objects like Plain old Java objects (POJO). It also stores the configuration files of application objects to be created.

Dependency Injection (DI): Spring provided loose coupling is application by Dependency Injection. It uses Inversion of Control technique by which objects specify their dependencies to Spring container instead of creating new objects themselves.

Dependency Injection (DI) is a design pattern used to implement IoC. It allows the creation of dependent objects outside of a class and provides those objects to a class through different ways. Using DI, we move the creation and binding of the dependent objects outside of the class that depends on them.

Aspect Oriented Programming (AOP): Spring framework promotes and provides support for Aspect oriented programming in Java. This helps in separating application business logic from system services that are common across all the business logic. E.g. Logging can be a cross cutting concern in an Application.

Transaction Management: Spring provides a framework for transaction management. So a developer does not have to implement it from scratch. Spring Transaction Management is so powerful that we can scale it from one local transaction to global transactions in a cluster.

MVC Framework: For Web applications, Spring provides MVC framework. This framework is based on MVC design pattern and

has better features compared to other web frameworks.

Exception Handling: Spring also gives support for a common API to handle exceptions in various technologies like- Hibernate, JDBC etc.

111. What are the modules in Core Container of Spring framework?

Spring framework has a Core Container. Modules in Core Container are:

Core module
Bean module
Context module
Spring Expression Language module

The Core and Beans modules provide the fundamental parts of the framework, including the IoC and Dependency Injection features

112. What are the modules in Data Access/Integration layer of Spring framework?

Modules in Data Access/Integration Layer of Spring framework are:

JDBC module: An abstraction layer to remove tedious JDBC coding.
ORM module Integration layers for Object Relational Mapping
OXM module: An abstraction layer to support Object XML mapping.
Java Messaging Service (JMS) module: Module for producing and consuming messages.
Transactions module: Transaction Management for POJO classes

113. What are the modules in Web layer of Spring framework?

Modules in Web Layer of Spring framework are:

Web module: This provides basic web-oriented integration features.
Servlet module: Support for Servlet Listeners.
WebSocket module: Support for Web Socket style messaging.
Portlet module: MVC implementation for Portlet environment.

114. What is the main use of Core Container module in Spring framework?

As the name suggests, Spring Core Container is the core of Spring framework. It gives the basic functionality of the Spring. All the parts of Spring Framework are built on top of Core Container.

Its main use is to provide Dependency Injection (DI) and Inversion of control (IOC) features.

115. What kind of testing can be done in Spring Test Module?

Spring Test Module provides support for Unit testing as well as Integration testing of Spring components. It allows using JUnit or TestNG testing frameworks. It also gives ability to mock objects to use the test code.

116. What is the use of BeanFactory in Spring framework?

BeanFactory is the main class that helps in implementing Inversion of Control pattern in Spring. It is based on the factory design pattern. It separates the configuration and dependencies of an application from the rest of application code.

Implementations of BeanFactory like XmlBeanFactory class are used by applications built with Spring.

117. Which is the most popular implementation of BeanFactory in Spring?

XMLBeanFactory is the most popular implementation of BeanFactory in Spring.

118. What is XMLBeanFactory in Spring framework?

XMLBeanFactory is one of the most useful implementation of BeanFactory in Spring. This factory loads its beans based on the definitions mentioned in an XML file.

Spring container reads bean configuration metadata from an XML file and creates a fully configured application with the help of XMLBeanFactory class.

119. What are the uses of AOP module in Spring framework?

AOP module is also known as Aspect Oriented Programming module. Its uses are:

Development of aspects in a Spring based application
Provides interoperability between Spring and other AOP frameworks
Supports metadata programming to Spring

120. What are the benefits of JDBC abstraction layer module in Spring framework?

Spring provides JDBC abstraction layer module. Main benefits of this module are:

Helps in keeping the database code clean and simple.
Prevents problems that result from a failure to close database resources.
Provides a layer of useful exceptions on top of the error messages given by different database servers.
Based on Spring's AOP module
Provides transaction management services for objects in a Spring application

121. How does Spring support Object Relational Mapping (ORM) integration?

Spring supports Object Relational Mapping (ORM) by providing ORM Module. This module helps in integrating with popular ORM framework like Hibernate, JDO, and iBATIS SQL Maps etc.

Transaction Management module of Spring framework supports all of these ORM frameworks as well as JDBC.

122. How does Web module work in Spring framework?

Spring provides support for developing web application by using Web module. This module is built on application context module that provides context for web-based applications.

This module also supports web-oriented integration features like- transparently handling multipart requests for uploading files, programmatically binding request parameters to business objects etc.

This module also supports integration with popular web frameworks like Jakarta Struts, JSF, and Tapestry etc.

123. What are the main uses of Spring MVC module?

Spring-webmvc module is also known as Web-servlet module. It is based on Web Model View Controller pattern.

Main uses of this module are:

Integration of Spring with other MVC frameworks
Supports IoC to provide clean separation of controller logic from business objects
Provides clean separation between domain model code and web forms
Allows developers to declaratively bind request parameters to business objects

124. What is the purpose of Spring configuration file?

Spring application can be configured by an XML file. This file contains information of classes and how these classes are configured and introduced to each other.

Spring IoC container uses some kind of configuration metadata. This configuration metadata represents how an application developer tells the Spring container to instantiate, configure, and assemble the objects in your application. This configuration metadata is stored in Spring configuration file.

The other ways of specifying configuration metadata are Java based configuration and Annotation based configuration.

125. What is the purpose of Spring IoC container?

The Spring IoC Container is responsible for:

Creating the objects
Configuring the objects
Managing dependency between objects (with dependency injection (DI))
Wiring the objects together
Managing complete lifecycle of objects

126. What is the main benefit of Inversion of Control (IOC) principle?

Inversion of Control (IOC) principle is the base of Spring framework. It supports dependency injection in an application. With Dependency Injection, a programmer has to write minimal code. It also makes easier to test an application.

Most important benefit is that it leads to loose coupling within objects. With loose coupling it is easier to change the application with new requirements.

127. Does IOC containers support Eager Instantiation or Lazy loading of beans?

IOC Container in Spring supports both the approaches. Eager instantiation as well as lazy loading of beans.

128. What are the benefits of ApplicationContext in Spring?

ApplicationContext in Spring provides following benefits:

Bean factory methods: These are used to access application components
Load File Resources: It helps in loading file resources in a generic fashion
Publish Events: It enables publishing events to registered listeners
Internationalization Support: Ability to resolve messages to support internationalization
Parent Context: Ability to inherit from a parent context

129. How will you implement ApplicationContext in Spring framework?

ApplicationContext in Spring can be implemented in one of the following three ways:

FileSystemXmlApplicationContext: If we want to load the definitions of beans from an XML file then FileSystemXmlApplicationContext is used. The full path of XML bean configuration file is provided to the constructor.
ClassPathXmlApplicationContext: To loads the definitions of beans from an XML file in the CLASSPATH, we use ClassPathXmlApplicationContext. It is used for application context embedded in jars.

WebXmlApplicationContext: To provide configuration for a web application WebXmlApplicationContext is used. While the application is running, it is read only. But it can be reloaded if underlying application supports it.

130. Explain the difference between ApplicationContext and BeanFactory in Spring?

Main differences between ApplicationContext and BeanFactory are:

Automatic BeanPostProcessor registration: BeanFactory does not support BeanPostProcessor registration. Whereas ApplicationContext support this.

Automatic BeanFactoryPostProcessor registration: BeanFactory also does not allow Automatic BeanFactoryPostProcessor registration. Whereas ApplicationContext allows this.

MessageSource access: BeanFactory is not convenient for MessageSource access. ApplicationContext is quite convenient for MessageSource access.

ApplicationEvent: We cannot publish ApplicationEvent with BeanFactory. But ApplicationContext provides ability to publish ApplicationEvent.

131. Between ApplicationContext and BeanFactory which one is preferable to use in Spring?

Spring documentation recommends using ApplicationContext in almost all the cases. ApplicationContext has all the functionality of BeanFactory.

132. What are the main components of a typical Spring based application?

In a Spring based application, main components are:
Spring configuration XML file: This is used to configure Spring application
API Interfaces: Definition of API interfaces for functions provided by application
Implementation: Application code with implementation of APIs
Aspects: Spring Aspects implemented by application
Client: Application at client side that is used for accessing functions

133. Explain Dependency Injection (DI) concept in Spring framework?

Dependency Injection is a software design pattern. It is used to implement Inversion of Control (IOC) in Spring framework. As per this pattern, we do not create objects in an application by calling new. Rather, we describe how an object should be created. In this way creation of an object is not tightly coupled with another object.

A container is responsible for creating and wiring the objects. The container can call injecting code and wire the objects as per the configuration at runtime.

134. What are the different roles in Dependency Injection (DI)?

There are four roles in Dependency Injection:

Service object(s) to be used
Client object that depends on the service
Interface that defines how client uses services
Injector responsible for constructing services and injecting them into client

135. Spring framework provides what kinds of Dependency Injection mechanism?

Spring framework provides two types of Dependency Injection mechanism:

Constructor-based Dependency Injection: Spring container can invoke a class constructor with a number of arguments. This represents a dependency on other class.

Setter-based Dependency Injection: Spring container can call setter method on a bean after creating it with a no-argument constructor or no-argument static factory method to instantiate another bean.

136. In Spring framework, which Dependency Injection is better? Constructor-based DI or Setter-based DI?

Spring framework provides support for both Constructor-based and Setter-based Dependency Injection. There are different scenarios in which these options can be used.

It is recommended to use Constructor-based DI for mandatory dependencies. Whereas Setter-based DI is used for optional dependencies.

137. What are the advantages of Dependency Injection (DI)?

Dependency Injection (DI) pattern has following advantages:

Dependency Injection reduces coupling between a class and its dependencies.

With Dependency Injection (DI), we can do concurrent or independent software development. Two teams can work parallel on classes that will be used by each other.

In Dependency Injection (DI), the client can be configured in multiple ways. It needs to just work with the given interface. Rest of the implementation can be changed and configured for different features.

Dependency injection is also used to export a system's configuration details into configuration files. So we can configure same application run in different environments based on configuration. E.g. Run in Test environment, UAT environment, and Production environment.

Dependency Injection (DI) applications provide more ease and flexibility of testing. These can be tested in isolation in Unit Test.

Dependency injection (DI) isolates client from the impact of design and implementation changes. Therefore, it promotes reusability, testability and maintainability.

138. What are the disadvantages of Dependency Injection (DI)?

Dependency Injection (DI) pattern has following disadvantages:

Most of the time Dependency Injection forces developers to use an injection framework like Spring. This causes dependency on a framework.

With Dependency Injection, clients are dependent on the configuration data. This becomes extra task for developers when the application does not need so many custom configuration values.

Code is difficult to trace and read in Dependency Injection. DI separates behavior from construction of objects.

Dependency injection increases complexity in the linkages between classes. It may become harder to manage such complexity outside the implementation of a class.

139. What is a Spring Bean?

A Spring Bean is a plain old Java object (POJO) that is created and managed by a Spring container.

There can be more than one bean in a Spring application. But all these Beans are instantiated and assembled by Spring container.

Developer provides configuration metadata to Spring container for creating and managing the lifecycle of Spring Bean.

In general a Spring Bean is singleton. Evert bean has an attribute named "singleton". If its value is true then bean is a singleton. If its value is false then bean is a prototype bean.

By default the value of this attribute is true. Therefore, by default all the beans in spring framework are singleton in nature.

140. What does the definition of a Spring Bean contain?

A Spring Bean definition contains configuration metadata for bean. This configuration metadata is used by Spring container to:

Create the bean
Manage its lifecycle
Resolve its dependencies

141. What are the different ways to provide configuration metadata to a Spring Container?

Spring supports three ways to provide configuration metadata to Spring Container:

XML based configuration: We can specify configuration data in an XML file.

Annotation-based configuration: We can use Annotations to specify configuration. This was introduced in Spring 2.5.

Java-based configuration: This is introduced from Spring 3.0. We can embed annotations like @Bean, @Import, @Configuration in Java code to specify configuration metadata.

142. What are the different scopes of a Bean supported by Spring?

Spring framework support seven types of scopes for a Bean. Out of these only five scopes are available for a web-aware ApplicationContext application:

singleton: This is the default scope of a bean. Under this scope, there is a single object instance of bean per Spring IoC container.

prototype: Under this scope a single bean definition can have multiple object instances.

request: In this scope, a single bean definition remains tied to the lifecycle of a single HTTP request. Each HTTP request will have its own instance of a bean for a single bean definition. It is only valid in the context of a web-aware Spring ApplicationContext.

session: Under this scope, a single bean definition is tied to the lifecycle of an HTTP Session. Each HTTP Session will have one instance of bean. It is also valid in the context of a web-aware Spring ApplicationContext.

globalSession: This scope, ties a single bean definition to the lifecycle of a global HTTP Session. It is generally valid in a Portlet context. It is also valid in the context of a web-aware Spring ApplicationContext.

application: This scope, limits a single bean definition to the lifecycle of a ServletContext. It is also valid in the context of a web-aware Spring ApplicationContext.

websocket: In this scope, a single bean definition is tied to the lifecycle of a WebSocket. It is also valid in the context of a web-aware Spring ApplicationContext.

143. How will you define the scope of a bean in Spring?

In configuration xml, we can specify the scope of bean in its definition. This is used by container to decide the scope of bean in Spring.

E.g. <bean id="userService" class="com.um.UserService" scope="prototype"/>

This is an example of userService bean with prototype scope.

144. Is it safe to assume that a Singleton bean is thread safe in Spring Framework?

No, Spring framework does not guarantee anything related to multi-threaded behavior of a singleton bean. Developer is responsible for dealing with concurrency issues and maintaining thread safety of a singleton bean.

145. What are the design-patterns used in Spring framework?

Spring framework uses many Design patterns. Some of these patterns are:

Singleton – By default beans defined in spring config files are singleton. These are based on Singleton pattern.

Template – This pattern is used in many classes like- JdbcTemplate, RestTemplate, JmsTemplate, JpaTemplate etc.

Dependency Injection – This pattern is the core behind the design of BeanFactory and ApplicationContext.

Proxy – Aspect Oriented Programming (AOP) heavily uses proxy design pattern.

Front Controller – DispatcherServlet in Spring is based on Front Controller pattern to ensure that incoming requests are dispatched to other controllers.

Factory pattern – To create an instance of an object, BeanFactory is used. This is based on Factory pattern.

View Helper – Spring has multiple options to separating core code from presentation in views. Like- Custom JSP tags, Velocity macros etc.

146. What is the lifecycle of a Bean in Spring framework?

A Bean in Spring framework goes through following phases in its lifecycle.

Initialization and creation: Spring container gets the definition of Bean from XML file and instantiates the Bean. It populates all the properties of Bean as mentioned in the bean definition.

Setting the Behavior of Bean: In case a Bean implements BeanNameAware interface, Spring uses setBeanName() method to pass the bean's id. In case a Bean implements BeanFactoryAware interface, Spring uses setBeanFactory() to pass the BeanFactory to bean.

Post Processing: Spring container uses postProcesserBeforeInitialization() method to call BeanPostProcessors associated with the bean. Spring calls afterPropertySet() method to call the specific initialization methods. In case there are any BeanPostProcessors of a bean, the postProcessAfterInitialization() method is called.

Destruction: During the destruction of a bean, if bean implements DisposableBean, Spring calls destroy() method.

147. What are the two main groups of methods in a Bean's lifecycle?

A Bean in Spring has two main groups of lifecycle methods.

Initialization Callbacks: Once all the necessary properties of a Bean are set by the container, Initialization Callback methods are used for performing initialization work. A developer can implement method afterPropertiesSet() for this work.

Destruction Callbacks: When the Container of a Bean is destroyed, it calls the methods in DisposableBean to do any cleanup work. There is a method called destroy() that can be used for this purpose to make Destruction Callbacks.

Recent recommendation from Spring is to not use these methods, since it can strongly couple your code to Spring code.

148. Can we override main lifecycle methods of a Bean in Spring?

Yes, Spring framework allows developers to override the lifecycle methods of a Bean. This is used for writing any custom behavior for Bean.

149. What are Inner beans in Spring?

A bean that is used as a property of another bean is known as Inner bean. It can be defined as a <bean/> element in <property/> or <constructor-arg/> tags.

It is not mandatory for an Inner bean to have id or a name. These are always anonymous.

Inner bean does not need a scope. By default it is of prototype scope.

150. How can we inject a Java Collection in Spring framework?

Spring promotes Dependency Injection (DI) in code. It gives support for injecting not only objects but also collection of objects.

We can inject collections like- list, set, map etc. in Spring. Following tags can be used for this purpose:

<list> : This type is used for injecting a list of values. In a <list> duplicates are allowed.

<set> : This type is used for injecting a set of values. As per set property, duplicates are not allowed.

<map> : This type is used for injecting name-value pairs in form of map. Name and value can be of any type that is allowed for a map.

<props> : This type is used to inject a collection of String based name-value. It is like a properties file.

151. What is Bean wiring in Spring?

A Spring container is responsible for injecting dependencies between beans. This process of connecting beans is called wiring.

Developer mentions in configuration file, the dependencies between beans. And Spring container reads these dependencies and wires the beans on creation.

152. What is Autowiring in Spring?

Autowiring is a feature of Spring in which container can automatically wire/connect the beans by reading the configuration file.

Developer has to just define “autowire” attribute in a bean.

Spring resolves the dependencies automatically by looking at this attribute of beans that are autowired.

153. What are the different modes of Autowiring supported by Spring?

There are five modes of Autowiring supported by Spring framework:

no: This is default setting for Autowiring. In this case, we use “ref” mode to mention the explicit bean that is being referred for wiring.

E.g. In this example Employee bean refers Manager bean.

```
<bean id="employee" class="com.dept.Employee">
        <property name="manager" ref="manager" />
</bean>
<bean id="manager" class="com.dept.Manager" />
```

byName: In this case, Spring container tries to match beans by name during Autowiring. If the name of a bean is same as the name of bean referred in autowire byname, then it automatically wires it.

E.g. In following example, Manager bean is wired to Employee bean by Name.

```
<bean id="employee" class="com.dept.Employee" autowire="byName" />
<bean id="manager" class="com.dept.Manager" />
```

byType: In this case, Spring container check the properties of beans referred with attribute byType. Then it matches the type of bean and wires. If it finds more than one such bean of that type, it throws a fatal exception.

E.g. In following example, Manager bean is wired by type to

Employee bean.

```
<bean id="employee" class="com.dept.Employee" autowire="byType" />
<bean id="manager" class="com.dept.Manager" />
```

constructor: In this case, Spring container looks for byType attribute in constructor argument. It tries to find the bean with exact name. If it finds more than one bean of same name, it throws fatal exception. This case is similar to byType case.

E.g. In following example “constructor” mode is used for autowiring.

```
<bean id="employee" class="com.dept.Employee" autowire="constructor" />
<bean id="manager" class="com.dept.Manager" />
```

autodetect: This is an advanced mode for autowiring. In this case, by default Spring tries to find a constructor match. If it does not find constructor then it uses autowire by Type.

E.g. This is an example of autodetect Autowiring.

```
<bean id="employee" class="com.dept.Employee" autowire="autodetect" />
<bean id="manager" class="com.dept.Manager" />
```

154. What are the cases in which Autowiring may not work in Spring framework?

Autowiring is a great feature in Spring. It can be used in most of the cases. But there are certain scenarios in which Autowiring may not work.

Explicit wiring: Since Autowiring is done by Spring, developer does not have full control on specifying the exact class to be used. It is preferable to use Explicit wiring in case of full control over wiring.

Primitive Data types: Autowiring does not allow wiring of properties that are based on primitive data types like- int, float etc.

155. Is it allowed to inject null or empty String values in Spring?

Yes, Spring allows injecting null or empty String values.

156. What is a Java-based Configuration in Spring?

Spring allows for Java-based configuration in which a developer can specify configuration by using Java-based annotations. This feature was introduced in Spring 3.0.

You can use annotations like- @Configuration, @Bean, @Import and @DependsOn in Java classes for specifying the configuration.

157. What is the purpose of @Configuration annotation?

This annotation is used in a class to indicate that this is class is the primary source of bean definitions. This class can also contain inter-bean dependencies that are annotated by @Bean annotation.

158. What is the difference between Full @Configuration and 'lite' @Beans mode?

Spring allows for using @Bean annotation on methods that are declared in classes not annotated with @Configuration. This is known as “lite” mode. In this mode, bean methods can be declared in a @Component or a plain java class without any annotation.

In the “lite” mode, @Bean methods cannot declare inter-bean dependencies.

It is recommended that one @Bean method should not invoke another @Bean method in 'lite' mode.

Spring recommends that @Bean methods declared within @Configuration classes should be used for full configuration. This kind of full mode can prevent many bugs.

159. In Spring framework, what is Annotation-based container configuration?

From Spring 2.5 version it is possible to provide configuration by using annotation.

To turn this configuration on, we need to mention <context:annotation-config/> in spring XML file.

Now developer can use annotations like @Required, @Autowired, @Qualifier etc. in a class file to specify the configuration for beans. Spring container can use this information from annotation for creating and wiring the beans.

160. How will you switch on Annotation based wiring in Spring?

To use Annotation based wiring, we need to turn on Annotation based configuration in Spring.

By default, Annotation based configuration is switched off in Spring. To turn it is we can specify <context:annotation-config/> element in Spring config file.

Once it is turned on, we can use @Autowired annotation or @Required annotation in a Java class for wiring in Spring.

161. What is @Autowired annotation?

We can use @Autowired annotation to auto wire a bean on a setter method, constructor or a field. @Autowired auto wiring is done by matching the data type.

Before using @Autowired annotation we have to register AutowiredAnnotationBeanPostProcessor. This can be done by including <context:annotation-config /> in bean configuration file.

162. What is @Required annotation?

We use @Required annotation to a property to check whether the property has been set or not.

Spring container throws BeanInitializationException if the @Required annotated property is not set.

When we use @Required annotation, we have to register RequiredAnnotationBeanPostProcessor in Spring config file.

163. What are the two ways to enable RequiredAnnotationBeanPostProcessc in Spring?

RequiredAnnotationBeanPostProcessor can be enabled in two ways in Spring:

Include <context:annotation-config />
Add Spring context and <context:annotation-config /> in bean configuration file.
E.g.

```
<beans
  ...
  xmlns:context="http://www.springframework.org/schema/context"
  ...
  http://www.springframework.org/schema/context
  http://www.springframework.org/schema/context/spring-context-
2.5.xsd">
  ...
  <context:annotation-config />
  ...
</beans>
```

Include RequiredAnnotationBeanPostProcessor in bean configuration file

E.g.

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/bea
  http://www.springframework.org/schema/beans/spring-beans-
2.5.xsd">

<bean
```

```
class="org.springframework.beans.factory.annotation.RequiredAnno

  <bean id="BookBean" class="com.foo.Book">
      <property name="action" value="price" />
      <property name="type" value="1" />
  </bean>

  <bean id="AuthorBean" class="com.foo.Author">
      <property name="name" value="Rowling" />
  </bean>

</beans>
```

164. What is @Qualifier annotation in Spring?

We use @Qualifier annotation to mark a bean as ready for auto wiring. This annotation is used along with @Autowired annotation to specify the exact bean for auto wiring by Spring container.

165. How Spring framework makes JDBC coding easier for developers?

Spring provides a mature JDBC framework to provide support for JDBC coding. Spring JDBC handled resource management as well as error handling in a generic way. This reduces the work of software developers.

They just have to write queries and related statements to fetch the data or to store the data in database.

166. What is the purpose of JdbcTemplate?

Spring framework provides JdbcTemplate class that contains many convenient methods for regular tasks like- converting data into primitives or objects, executing prepared or callable statements etc.

This class makes it very easy to work with database in our Application and it also provides good support for custom error handling in database access code.

167. What are the benefits of using Spring DAO?

Some of the benefits of using Spring DAO are:

It makes it easier to work on different data access methods like- JDBC, Hibernate etc.
It provides a consistent and common way to deal with different data access methods.
Spring DAO makes it easier to switch between different data persistence frameworks.
No need for catching framework specific exceptions.

168. What are the different ways to use Hibernate in Spring?

Spring provides two ways to use Hibernate:

We can extend HibernateDAOSupport and apply an AOP interceptor node to use Hibernate.
We can also use HibernateTemplate and Callback to access Hibernate. This is based on Inversion of Control.

169. What types of Object Relational Mapping (ORM) are supported by Spring?

Spring supports following Object Relational Mapping (ORM) frameworks:

Hibernate
Java Persistence API (JPA)
TopLink
Java Data Objects (JDO)
Apache Object Relational Bridge (ORB)

170. How will you integrate Spring and Hibernate by using HibernateDaoSupport?

We can use following steps for integrating Spring and Hibernate:

Add dependencies for Spring and Hibernate in pom.xml
Implement DAO from HibernateDaoSupport
Use Hibernate functions via getHibernateTemplate() method

171. What are the different types of the Transaction Management supported by Spring framework?

Spring framework provides support for two types of Transaction Management:

Programmatic: In this method, we have to manage Transaction by programming explicitly. It provides flexibility to a developer, but it is not easier to maintain.

Declarative: In this approach, we can separate Transaction Management from the Application Business code. We can use annotations or XML based configuration to manage the transactions in declarative approach.

172. What are the benefits provided by Spring Framework's Transaction Management?

Main benefits provided by Spring Transaction Management are:

Consistent: By using Spring Transaction management, we can use consistent programming model across different transaction APIs like- JPA, JDBC, JTA, Hibernate, JPA, JDO etc.

Simplicity: Spring TM provides simple API for managing the transaction programmatically.

Declarative: Spring also supports annotation or xml based declarative transaction management.

Integration: Spring Transaction management is easier to integrate with other data access abstractions of Spring.

173. Given a choice between declarative and programmatic Transaction Management, which method will you choose?

In Spring, Declarative Transaction Management is the preferred choice. This method is very less invasive and it has very less impact in Application Business Logic.

Although Declarative method gives less flexibility than Programmatic method, it is simpler to use and easier to maintain in long run.

174. What is Aspect Oriented Programming (AOP)

Aspect Oriented Programming (AOP) is a programming paradigm that promotes programmers to develop code in different modules that can be parallel or in crosscutting concerns.

E.g. To develop banking software, one team can work on business logic for Money withdrawal, Money deposit, Money Transfer etc. The other team can work on Transaction Management for committing the transaction across multiple accounts.

In an Auto company, one team can work on software to integrate with different components of car. The other team can work on how all the components will send signal and current information to a common dashboard.

175. What is an Aspect in Spring?

An Aspect is the core construct of AOP. It encapsulates the behavior that affects multiple classes in a reusable module.

An Aspect can have a group of APIs that provide cross-cutting features.

E.g. A logging module can be an Aspect in an Application.

An application can have multiple of Aspects based on the different requirements.

An Aspect can be implemented by using annotation @Aspect on a class.

176. In Spring AOP, what is the main difference between a Concern and a Cross cutting concern?

A Concern in Spring is the behavior or expectation from an application. It can be the main feature that we want to implement in the application.

A Cross cutting concern is also a type of Concern. It is the feature or functionality that is spread throughout the application in a thin way.

E.g. Security, Logging, Transaction Management etc. are cross cutting concerns in an application.

177. What is a Joinpoint in Spring AOP?

In Spring AOP, Joinpoint refers to a candidate point in application where we can plug in an Aspect.

Joinpoint can be a method or an exception or a field getting modified.

This is the place where the code of an Aspect is inserted to add new behavior in the existing execution flow.

178. What is an Advice in Spring AOP?

An Advice in Spring AOP, is an object containing the actual action that an Aspect introduces.

An Advice is the code of cross cutting concern that gets executed.

There are multiple types of Advice in Spring AOP.

179. What are the different types of Advice in Spring AOP?

Spring AOP provides five kinds of Advice:

1. Before Advice: This type of advice runs just before a method executes. We can use @Before annotation for this.

2. After (finally) Advice: This type of advice runs just after a method executes. Even if the method fails, this advice will run. We can use @After annotation here.

3. After Returning Advice: This type of advice runs after a method executes successfully. @AfterReturning annotation can be used here.

4. After Throwing Advice: This type of advice runs after a method executes and throws an exception. The annotation to be used is @AfterThrowing.

5. Around Advice: This type of advice runs before and after the method is invoked. We use @Around annotation for this.

180. What is a Pointcut in Spring AOP?

A Pointcut in Spring AOP refers to the group of one or more Joinpoints where an advice can be applied.

We can apply Advice to any Joinpoint. But we want to limit the places where a specific type of Advice should be applied. To achieve this we use Pointcut.

We can use class names, method names or regular expressions to specify the Pointcuts for an Advice.

181. What is an Introduction in Spring AOP?

In Spring AOP we can declare additional methods or fields on behalf of a type. To do this we use an Introduction. It is also known as inter-type declaration.

E.g. We can use an Introduction for making a bean implement IsModified interface.

182. What is a Target object in Spring AOP?

A Target object is the object that gets Advice from one or more Aspects.

This is also known as advised object.

In most cases it is a proxy object.

183. What is a Proxy in Spring AOP?

In Spring AOP, a Proxy is an object created by the AOP framework to implement Aspect contracts. It is generally a JDK dynamic proxy or CGLIB proxy.

184. What are the different types of AutoProxy creators in Spring?

Spring AOP provides following standard types of Autoproxy creators:

1. BeanNameAutoProxyCreator: This is a BeanPostProcessor that creates AOP proxies for beans automatically by matching names.

2. DefaultAdvisorAutoProxyCreator: This creator is more powerful that other Proxy Creators. This also applies eligible advisors automatically to bean in the current context.

3. AbstractAdvisorAutoProxyCreator: This is the parent class of DefaultAdvisorAutoProxyCreator. We can create our own auto-proxy creators by extending this class.

185. What is Weaving in Spring AOP?

In Aspect oriented programming, linking Aspects with the other application types creates an Advised object. This process is known as Weaving.

Without Weaving, we just have definition of Aspects. Weaving makes use realize full potential of the AOP.

Weaving can be done at compile time, load time or at run time.

186. In Spring AOP, Weaving is done at compile time or run time?

Spring container performs Weaving at run time.

187. What is XML Schema-based Aspect implementation?

Spring allows for implementing Aspect by using regular classes and XML based configurations. This is different from Annotation based Aspect implementation. But it achieves the same goal of AOP.

We can use elements like <aop:aspect id=”testAspect" ref="testBean" /> and <aop:pointcut id="testPointcut" /> in Spring XML config file.

To use this we need to import Spring AOP schema as follows:

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"

xsi:schemaLocation="http://www.springframework.org/schema/bea
        http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
">
```

188. What is Annotation-based aspect implementation in Spring AOP?

This is a declarative style AOP implementation. In this case, we use annotations like @Aspect, @Pointcut, @Joinpoint etc. to annotate code with different types of AOP elements.

This can be used Java 5 onwards, when the support for Annotations was introduced.

189. How does Spring MVC framework work?

Spring provides its own Model View Controller (MVC) framework for developing web applications.

Spring MVC framework is based on Inversion of Control (IOC) principle. It separates the business objects from controller.

It is designed around the DispatcherServlet that is responsible for dispatching requests to relevant handlers.

Spring MVC framework also supports annotation based binding of request parameters.

190. What is DispatcherServlet?

In Spring MVC, DispatcherServlet is the core servlet that is responsible for handling all the requests and dispatching these to handlers.

Dispatcher servlet knows the mapping between the method to be called and the browser request. It calls the specific method and combines the results with the matching JSP to create an html document, and then sends it back to browser.

In case of RMI invocation, it sends back response to the client application.

191. Can we have more than one DispatcherServlet in Spring MVC?

Yes, a Spring MVC web application can have more than one DispatcherServlets.
Each DispatcherServlet has to operate in its own namespace. It has to load its own ApplicationContext with mappings, handlers, etc.

Only the root application context will be shared among these Servlets.

192. What is WebApplicationContext in Spring MVC?

WebApplicationContext is the child of plain ApplicationContext. It is used in web applications. It provides features to deal with web-related components like- controllers, view resolvers etc.

A Web Application can have multiple WebApplicationContext to handle requests.

Each DispatcherServlet is associated with one WebApplicationContext.

193. What is Controller in Spring MVC framework?

Controller is an interface in Spring MVC. It receives HttpServletRequest and HttpServletResponse in web app just like an HttpServlet, but it is able to participate in an MVC flow.

Controllers are similar to a Struts Action in a Struts based Web application.

Spring recommends that the implementation of Controller interface should be a reusable, thread-safe class, capable of handling multiple HTTP requests throughout the lifecycle of an application.

It is preferable to implement Controller by using a JavaBean.

Controller interprets user input and transforms it into a model. The model is represented to the user by a view.

Spring implements a controller in a very generic way. This enables us to create a wide variety of controllers.
What is @Controller annotation in Spring MVC?

We use @ Controller annotation to indicate that a class is a Controller in Spring MVC.

The dispatcher in Spring scans for @Controller annotated classes for mapped methods and detects @RequestMapping.

194. What is @RequestMapping annotation in Spring?

In Spring MVC, we use @RequestMapping annotation to map a web request to either a class or a handler method.

In @RequestMapping we can specify the path of URL as well as HTTP method like- GET, PUT, POST etc.

@RequestMapping also supports specifying HTTP Headers as attributes.

We can also map different media types produced by a controller in @RequestMapping. We use HTTP Header Accepts for this purpose.

E.g. @RequestMapping(
  value = "/test/mapping",
  method = GET,
  headers = "Accept=application/json")

195. What are the main features of Spring MVC?

Spring MVC has following main features:

1. Clear separation of role: In Spring MVC, each role like- controller, validator, command object, form object, model object, DispatcherServlet, handler mapping, view resolver etc. is fulfilled by a specialized object.

2. Reusability: Spring MVC promotes reusable business code that reduces the need for duplication. We can use existing business objects as command or form objects instead of copying them to extend a particular framework base class.

3. Flexible Model Transfer: Spring MVC Model transfer supports easy integration with other view technologies as well.

4. Customizable binding and validation: In Spring MVC, we can to custom binding between Requests and Controllers. Even validation can be done on non-String values as well.

5. JSP form tag library: From Spring 2.0, there is a powerful JSP form tag library that makes writing forms in JSP pages much easier.

6. Customizable locale, time zone and theme resolution: Spring MVC supports customization in locale, timezone etc.

196. What is the difference between a Singleton and Prototype bean in Spring?

Every bean in Spring has a scope that defines its existence timeframe in the application.

Singleton scope for bean limits a bean to a single object instance per Spring IOC container.

This single instance is limited to a specific ApplicationContext. If there are multiple ApplicationContext then we can have more than one instance of bean.

By default all the beans in Spring framework are Singleton scope beans.

With Prototype scope a single bean definition can have multiple object instances in a Spring container.

In prototype scope bean, the Spring IoC container creates new bean instance of the object every time a request for that specific bean is made.

197. How will you decide which scope- Prototype or Singleton to use for a bean in Spring?

In general, we use prototype scope for all stateful beans and singleton scope for stateless beans.

Since a stateless bean does not maintain any state, we can use the same object instance again and again. Singleton scope bean serves the same purpose.

In a stateful bean, there is a need to maintain the state in each request, it is necessary to use a new instance of object with each call. A Prototype scope bean ensures that we get a new instance each time we request for the object.

198. What is the difference between Setter and Constructor based Dependency Injection (DI) in Spring framework?

Main differences between Setter and Constructor based Dependency Injection (DI) in Spring are:

Priority: Setter based injection has higher priority than a constructor based injection in Spring. If an application uses Setter as well as Constructor injection, Spring container uses the Setter injection.

Partial dependency: We can inject partial dependency by using Setter injection. In Constructor injection, it is not possible to do just a partial dependency injection.

E.g. If there are two properties in a class, we can use Setter method to inject just one property in the class.

Flexibility: Setter injection gives more flexibility in introducing changes. One can easily change the value by Setter injection. In case of Constructor injection a new bean instance has to be created always.

Readability: Setter injection is more readable than Constructor injection. Generally Setter method name is similar to dependency class being used in setter method.

199. What are the drawbacks of Setter based Dependency Injection (DI) in Spring?

Although Setter based Dependency Injection has higher priority than Constructor based DI, there are some disadvantages of it.

No Guarantee: In Setter based DI, there is no guarantee that a certain dependency is injected or not. We may have an object with partial or no dependency. Whereas in Constructor based DI, an object in not created till the time all the dependencies are ready.

Security: One can use Setter based DI to override another dependency. This can cause Security breach in a Spring application.

Circular Dependency: Setter based DI can cause circular dependency between objects. Where as Constructor based DI will throw ObjectCurrentlyInCreationException if there is a circular dependency during the creation of an object.

200. What are the differences between Dependency Injection (DI) and Factory Pattern?

Main differences between Dependency Injection (DI) and Factory Pattern are:

Coupling: Factory pattern adds tight coupling between an object, factory and dependency. In case of DI, there is no coupling between objects. We just mention the dependencies on different objects and container resolves and introduces these dependencies.

Easier Testing: DI is easier to test, since we can inject the mock objects as dependency in Test environment. In case of Factory pattern, we need to create actual objects for testing.

Flexibility: DI allows for switching between different DI frameworks easily. It gives flexibility in the choice of DI framework.

Container: DI always needs a container for injecting the dependencies. This leads to extra overhead as well as extra code in your application. In factory pattern, you can just use POJO classes to implement the application without any container.

Cleaner Code: DI code is much cleaner than Factory pattern based code. In DI, we do not need to add extra code for factory methods.

201. In Spring framework, what is the difference between FileSystemResource and ClassPathResource?

In Spring we can specify configuration by using a file or classpath.

In FileSystemResource we have to give absolute path / relative path of Spring Configuration file spring-config.xml file.

In ClassPathResource Spring looks for Spring Configuration file spring-config.xml in ClassPath. Therefore, developer has to include spring-config.xml in classpath.

ClassPathResource looks for configuration file in CLASSPATH, whereas FileSystemResource looks for configuration file in file system.

202. Name some popular Spring framework annotations that you use in your project?

Spring has many Annotations to serve different purposes. For regular use we refer following popular Spring annotations:

@Controller: This annotation is for creating controller classes in a Spring MVC project.

@RequestMapping: This annotation maps the URI to a controller handler method in Spring MVC.

@ResponseBody: For sending an Object as response we use this annotation.

@PathVariable: To map dynamic values from a URI to handler method arguments, we use this annotation.

@Autowired: This annotation indicates to Spring for auto-wiring dependencies in beans.

@Service: This annotation marks the service classes in Spring.

@Scope: We can define the scope of Spring bean by this annotation.

@Configuration: This an annotation for Java based Spring configuration.

@Aspect, @Before, @After, @Around, @Joinpoint, @Pointcut: These are the annotations in Spring for AspectJ AOP.

203. How can you upload a file in Spring MVC Application?

In Spring MVC framework we can use MultipartResolver interface to upload a file. We need to make configuration changes to make it work. After uploading the file, we have to create Controller handler method to process the uploaded file in application.

204. What are the different types of events provided by Spring framework?

Spring framework provides following five events for Context:

ContextRefreshedEvent: Whenever ApplicationContext is initialized or refreshed, Spring publishes this event. We can also raise it by using refresh() method on ConfigurableApplicationContext interface.

ContextStartedEvent: When ApplicationContext is started using start() method on ConfigurableApplicationContext interface, ContextStartedEvent is published. We can poll database or restart any stopped application after receiving this event.

ContextStoppedEvent: Spring publishes this event when ApplicationContext is stopped using stop() method on ConfigurableApplicationContext interface. This is used for doing any cleanup work.

ContextClosedEvent: Once the ApplicationContext is closed using close() method, ContextClosedEvent is published. Once a context is closed, it is the last stage of its lifecycle. After this it cannot be refreshed or restarted.

RequestHandledEvent: This is a web specific event that informs to all beans that an HTTP request has been serviced.

205. What is the difference between DispatcherServlet and ContextLoaderListener in Spring?

DispatcherServlet is the core of Spring MVC application. It loads Spring bean configuration file and initialize all the beans mentioned in config file.

In case we have enabled annotations in Spring config file, it also scans the packages and configures any bean annotated with @Component, @Controller, @Repository or @Service annotations.

ContextLoaderListener is a listener to start up and shut down Spring's root WebApplicationContext. ContextLoaderListener links the lifecycle of ApplicationContext to the lifecycle of the ServletContext. It automates the creation of ApplicationContext. It can also be used to define shared beans used across different spring contexts.

206. How will you handle exceptions in Spring MVC Framework?

Spring MVC Framework provides following mechanisms to help us achieve exception handling:

Controller Based: A developer can define exception handler methods in a Controller class. To do so, they have to annotate the methods with @ExceptionHandler annotation.

Global Exception Handler: Spring provides @ControllerAdvice annotation for exception handling as cross-cutting concern. We can mark any class as global exception handler by using this annotation.

HandlerExceptionResolver implementation: Spring Framework provides HandlerExceptionResolver interface that can be implemented to create a global exception handler.

207. What are the best practices of Spring Framework?

In Spring Framework, following are some of the best practices:

We can Divide spring bean configurations based on their concerns such as spring-jdbc.xml, spring-security.xml.

It is better to avoid version numbers in schema reference. This makes sure that we have the latest config files.

It is a good practice to configure bean dependencies as much as possible. Unless there is a good reason, we try to avoid autowiring.

For spring beans that are used in multiple contexts in Spring MVC, we can create them in root context and initialize with listener.

Spring framework provides many features and modules. We should just use what we need for our application. An extra dependency has to be removed

For application properties, it is good to create a property file and read it in Spring configuration file.

Annotations are useful for smaller applications, but for larger applications annotations can become an overhead. It is easier to maintain if all the configurations are in xml files.

When we are doing AOP, we have to make sure to keep the Joinpoint as narrow as possible to avoid Advice on unwanted methods.

We should use right annotation for components or services. For services use @Service and for DAO beans use @Repository.

Dependency Injection (DI) has to be used when there is real benefit. It should not be used just for the sake of loose coupling.

208. What is Spring Boot?

Spring Boot is a ready made solution to create Spring applications with production grade features. It favors convention over configuration.

We can embed Tomcat or Jetty in in an application created with Spring Boot. Spring Boot automatically configures Spring in an application.

It does not require any code generation or xml configuration. It is an easy solution to create applications that can run stand-alone.

Hibernate

209. What is Hibernate framework?

Hibernate is a popular Object Relational Mapping (ORM) framework of Java. It helps in mapping the Object Oriented Domain model to Relational Database tables.

Hibernate is a free software distributed under GNU license.

Hibernate also provides implementation of Java Persistence API (JPA).

In simple words, it is a framework to retrieve and store data from database tables from Java.

210. What is an Object Relational Mapping (ORM)?

Object Relational Mapping (ORM) is a programming technique to map data from a relational database to Object oriented domain model. This is the core of Hibernate framework.

In case of Java, most of the software is based on OOPS design. But the data stored in Database is based on Relation Database Management System (RDBMS).

ORM helps in data retrieval in an Object Oriented way from an RDBMS. It reduces the effort of developers in writing queries to access and insert data.

211. What is the purpose of Configuration Interface in Hibernate?

Configuration interface can be implemented in an application to specify the properties and mapping documents for creating a SessionFactory in Hibernate.

By default, a new instance of Configuration uses properties mentioned in hibernate.properties file.

Configuration is mainly an initialization time object that loads the properties in helps in creating SessionFactory with these properties.

In short, Configuration interface is used for configuring Hibernate framework in an application.

212. What is Object Relational Impedance Mismatch?

Object Relational Impedance Mismatch (ORIM) is also known as paradigm mismatch. It means that Object model and Relational model do not work well with each other.

Relational model or a RDBMS represents data in tabular format like a spreadsheet. Object model or OOPS represents the data as an inter-connected graph of objects.

Mixing these two models leads to various problems. The common name for these issues is Object Relational Impedance Mismatch.

213. What are the main problems of Object Relational Impedance Mismatch?

Object model and Relational models (RDBMS) have following problems that are part of Object Relational Impedance Mismatch:

Granularity: Object model is more granular than Relational model. There are more classes in object model than the corresponding tables in relational model.

Inheritance: Object model supports inheritance. But Relational model does not have any concept of inheritance.

Identity: Relational model has just one criteria for sameness of data. It is based on primary key. In object model like Java we can have equals as well as == for sameness of objects.

Associations: In Object model associations are uni-directional. In RDBMS, there is a concept of foreign key for association. Also multiplicity of a relationship is hard to judge by looking at object model.

Data navigation: In Object model, you can move from one object to another object for getting data. Egg. you can retrieve and Employee object, then go to its department object and then get the employees in the department object. In RDBMS, we try to minimize the SQL calls, so we get all the data by using joins.

214. What are the key characteristics of Hibernate?

Hibernate has following key characteristics:

Object/Relational Mapping (ORM): Hibernate provides ORM capabilities to developers. So then can write code in Object model for connecting with data in Relational model.

JPA Provider: Hibernate provides an excellent implementation of Java Persistence API (JPA) specification.

Idiomatic persistence: Hibernate provides persistence based on natural Object-oriented idioms with full support for inheritance, polymorphism, association, composition, and the Java collections framework. It can work with any data for persistence.

High Performance: Hibernate provides high level of performance supporting features like- lazy initialization, multiple fetching strategies, optimistic locking etc. Hibernate does not need its own database tables or fields. It can generate SQL at system initialization to provide better performance at runtime.

Scalability: Hibernate works well in multi server clusters. It has built in scalability support. It can work well for small projects as well as for large business software.

Reliable: Hibernate very reliable and stable framework. This is the reason for its worldwide acceptance and popularity among developer community.

Extensible: Hibernate is quite generic in nature. It can be configured and extended as per the use case of application.

215. Can you tell us about the core interfaces of Hibernate framework?

The core interfaces of Hibernate framework are as follows:

Configuration: Configuration interface can be implemented in an application to specify the properties and mapping documents for creating a SessionFactory in Hibernate. Hibernate application bootstraps by using this interface.

SessionFactory: In Hibernate, SessionFactory is used to create and manage Sessions. Generally, there is one SessionFactory created for one database. It is a thread-safe interface that works well in multi-threaded applications.

Session: Session is a lightweight object that is used at runtime between a Java application and Hibernate. It contains methods to create, read and delete operations for entity classes. It is a basic class that abstracts the concept of persistence.

Transaction: This is an optional interface. It is a short lived object that is used for encapsulating the overall work based on unit of work design pattern. A Session can have multiple Transactions.

Query: This interface encapsulates the behavior of an object-oriented query in Hibernate. It can accept parameters and execute the queries to fetch results. Same query can be executed multiple times.

Criteria: This is a simplified API to retrieve objects by creating Criterion objects. It is very easy to use for creating Search like features.

216. How will you map the columns of a DB table to the properties of a Java class in Hibernate?

We can map the class properties and table columns by using one of the two ways:

XML: We can map the column of a table to the property of a class in XML file. It is generally with extension hbm.xml

Annotation: We can also use annotations @Entity and @Table to map a column to the property of a class.

217. Does Hibernate make it mandatory for a mapping file to have .hbm.xml extension?

No. It is a convention to have.hbm.xml extension in the name of a mapping file. It is not a requirement enforced by Hibernate. We can use any other extension of our convenience for this.

218. What are the steps for creating a SessionFactory in Hibernate?

Steps to create a SessionFactory in Hibernate are:

Configuration: First create a Configuration object. This will refer to the path of configuration file.
Resource: Add config file resource to Configuration object.
Properties: Set properties in the Configuration object.
SessionFactory: Use Configuration object to build SessionFactory.

Egg.

```
Configuration config = new Configuration();
config.addResource("testInstance/configuration.hbm.xml");
config.setProperties( System.getProperties() );
SessionFactory sessions = config.buildSessionFactory();
```

219. Why do we use POJO in Hibernate?

POJO stands for Plain Old Java Objects. A POJO is java bean with getter and setter methods for each property of the bean.

It is a simple class that encapsulates an object's properties and provides access through setters and getters.

Some of the reasons for using POJO in Hibernate are:

POJO emphasizes the fact that this class is a simple Java class, not a heavy class like EJB.

POJO is a well-constructed class, so it works well with Hibernate proxies.

POJO also comes with a default constructor that makes it easier to persist with a default constructor.

220. What is Hibernate Query Language (HQL)?

Hibernate Query Language is also known as HQL. It is an Object Oriented language. But it is similar to SQL.

HQL works well with persistent objects and their properties. HQL does not work on database tables.

HQL queries are translated into native SQL queries specific to a database.

HQL supports direct running of native SQL queries also. But it creates an issue in Database portability.

221. How will you call a stored procedure in Hibernate?

Hibernate supports executing not only simple queries but also stored procedure of database. There are three ways to call a stored procedure in Hibernate:

XML mapping file:
We can declare the store procedure inside XML Mapping file.

```
<!-- Employee.hbm.xml -->
...
<hibernate-mapping>
                    <class      name="com.testHibernate.util.Employee"
table="employee" ...>
      <id name="employeeId" type="java.lang.Integer">
         <column name="EMPLOYEE_ID" />
         <generator class="identity" />
      </id>
      <property name="employeeId" type="string">
                 <column  name="EMPLOYEE_ID"  length="10"  not-
null="true" unique="true" />
      </property>
</class>

   <sql-query name="callEmployeeStoreProcedure">
 <return                                              alias="employee"
class="com.testHibernate.util.Employee"/>
 <![CDATA[CALL GetEmployees(:employeeId)]]>
   </sql-query>

</hibernate-mapping>
```

We can call it with getNamedQuery().

```
Query                              query                              =
```

```
session.getNamedQuery("callEmployeeStoreProcedure")
  .setParameter("employeeId", “1234”);
List result = query.list();
for(int i=0; i<result.size(); i++){
  Employee employee = (Employee)result.get(i);
  System.out.println(employee.getEmployeeCode());
}
```

Native SQL: We can use Native SQL to call a store procedure query directly. In this example GetEmployees() stored procedure is being called.

```
Query query = session.createSQLQuery(
  "CALL GetEmployees(:employeeId)")
  .addEntity(Employee.class)
  .setParameter("employeeId", “1234);

List result = query.list();
for(int i=0; i<result.size(); i++){
  Employee employee = (Employee) result.get(i);
  System.out.println(employee.getEmployeeCode());
}
```

Use annotation:
We can also mark out stored procedure with @NamedNativeQueries annotation.

```
//Employee.java

@NamedNativeQueries({
  @NamedNativeQuery(
  name = "callEmployeeStoreProcedure",
  query = "CALL GetEmployees(:employeeId)",
  resultClass = Employee.class
  )
})
@Entity
```

```
@Table(name = "employee")
public class Employee implements java.io.Serializable {
...
```

Call it with getNamedQuery().

```
Query query =
session.getNamedQuery("callEmployeeStoreProcedure")
  .setParameter("employeeId", “1234”);
List result = query.list();
for(int i=0; i<result.size(); i++){
  Employee employee = (Employee)result.get(i);
  System.out.println(employee.getEmployeeCode());
}
```

222. What is Criteria API in Hibernate?

Criteria is a simplified API in Hibernate to get entities from database by creating Criterion objects.

It is a very intuitive and convenient approach for search features. Users can specify different criteria for searching entities and Criteria API can handle these.

Criterion instances are obtained through factory methods on Restrictions.

223. Why do we use HibernateTemplate?

This is a trap question. HibernateTemplate has been deprecated. There were earlier good reasons to use HibernateTemplate. But now the trend has changed towards not using it anymore.

224. How can you see SQL code generated by Hibernate on console?

To display the SQL generated by Hibernate, we have to turn on the show_sql flag.

This can be done in Hibernate configuration as follows:

<property name=”show_sql”>true</property>

225. What are the different types of collections supported by Hibernate?

Hibernate supports following two types of collections:

Indexed Collections: List and Maps
Sorted Collections: java.util.SortedMap and java.util.SortedSet

226. What is the difference between session.save() and session.saveOrUpdate() methods in Hibernate?

Save method first stores an object in the database. Then it persists the given transient instance by assigning a generated identifier. Finally, it returns the id of the entity that is just created.

SaveOrUpdate() method calls either save() or update() method. It selects one of these methods based on the existence of identifier.

If an identifier exists for the entity then update() method is called. If there is no identifier for the entity then save() method is called as mentioned earlier.

227. What are the advantages of Hibernate framework over JDBC?

Main advantages of Hibernate over JDBC are as follows:

Database Portability: Hibernate can be used with multiple types of database with easy portability. In JDBC, developer has to write database specific native queries. These native queries can reduce the database portability of the code.

Connection Pool: Hibernate handles connection pooling very well. JDBC requires connection pooling to be defined by developer.

Complexity: Hibernate handles complex query scenarios very well with its internal API like Criteria. So developer need not gain expertise in writing complex SQL queries. In JDBC application developer writes most of the queries.

228. How can we get statistics of a SessionFactory in Hibernate?

In Hibernate we can get the statistics of a SessionFactory by using Statistics interface. We can get information like Close Statement count, Collection Fetch count, Collection Load count, Entity insert count etc.

229. What is the Transient state of an object in Hibernate?

When an object is just instantiated using the new operator but is not associated with a Hibernate Session, then the object is in Transient state.

In Transient state, object does not have a persistent representation in database. Also there is no identifier assigned to an object in Transient state.

An object in Transient state can be garbage collected if there is no reference pointing to it.

230. What is the Detached state of an object in Hibernate?

An object is in detached state if it was persistent earlier but its Session is closed now.

Any reference to this object is still valid. We can even update this object. Later on we can even attach an object in detached state to a new session and make it persistent.

Detached state is very useful in application transactions where a user takes some time to finish the work.

231. What is the use of Dirty Checking in Hibernate?

Dirty Checking is very useful feature of Hibernate for write to database operations. Hibernate monitors all the persistent objects for any changes. It can detect if an object has been modified or not.

By Dirty Checking, only those fields of an object are updated that require any change in them. It reduces the time-consuming database write operations.

232. What is the purpose of Callback interface in Hibernate?

Callback interface in Hibernate is mainly used for receiving notifications of different events from an object.

Egg. We can use Callback to get the notification when an object is loaded into or removed from database.

233. What are the different ORM levels in Hibernate?

There are following four different ORM levels in Hibernate:

Pure Relational ORM: At this level entire application is designed around the relational model. All the operations are SQL based at this level.

Light Object Mapping: At this level entity classes are mapped manually to relational tables. Business logic code is hidden from data access code. Applications with less number of entities use this level.

Medium Object Mapping: In this case, application is designed around an object model. Most of the SQL code is generated at compile time. Associations between objects are supported by the persistence mechanism. Object-oriented expression language is used to specify queries.

Full Object Mapping: This is one of the most sophisticated object modeling level. It supports composition, inheritance, polymorphism and persistence. The persistent classes do not inherit any special base class at this level. There are efficient fetching and caching strategies implemented transparently to the application.

234. What are the different ways to configure a Hibernate application?

There are mainly two ways to configure Hibernate application:

XML based: We can define the Hibernate configuration in an XML file like ibernate.cfg.xml file

Programming based: We can also use code logic to configure Hibernate in our application.

235. What is Query Cache in Hibernate?

Hibernate provides Query Cache to improve the performance of queries that run multiple times with same parameters.

At times Query Caching can reduce the performance of Transactional processing. By default Query Cache is disabled in Hibernate.

It has to be used based on the benefits gained by it in performance of the queries in an application.

236. What are the different types of Association mappings supported by Hibernate?

Hibernate supports following four types of Association mappings:

Unidirectional association: This kind of association works in only one direction.
Unidirectional association with join tables
Bidirectional association: This kind of association works in both directions.
Bidirectional association with join tables

237. What are the different types of Unidirectional Association mappings in Hibernate?

In Hibernate there can be following three types of Unidirectional Association mappings:

Many to one
One to one
One to many

238. What is Unit of Work design pattern?

Unit of Work is a design pattern to define business transactions.

A Unit of Work is a list of ordered operations that we want to run on a database together. Either all of these go together or none of these goes.

Most of the time, we use term business transaction in place of Unit of Work.

Egg. In case of money transfer from account A to B, the unit of work can be two operation Debit account A and Credit account B in a sequence. Both these operations should happen together and in right sequence.

239. In Hibernate, how can an object go in Detached state?

Once the session attached to an Object is closed, the object goes into Detached state. An Object in Detached state can be attached to another session at a later point of time.

This state is quite useful in concurrent applications that have long unit of work.

240. How will you order the results returned by a Criteria in Hibernate?

Hibernate provides an Order criterion that can be used to order the results. This can be order objects based on their property in ascending or descending order.

Class is org.hibernate.criterion.Order.

One example is as follows:

Egg.

```
List employees = session.createCriteria(Employee.class)
    .add( Restrictions.like("name", "F%")
    .addOrder( Order.asc("name") )
    .addOrder( Order.desc("age") )
    .setMaxResults(10)
    .list();
```

241. How does Example criterion work in Hibernate?

In Hibernate, we can create an object with desired properties. Then we can use this object to search for objects with similar object. For this we can use org.hibernate.criterion.Example criterion.

Egg. First we create a sample book object of author Richard and category mystery. Then we search for similar books.

```
Book book = new Book();
book.setAuthor('Richard');
book.setCategory(Category.MYSTERY);
List results = session.createCriteria(Book.class)
    .add( Example.create(book) )
    .list();
```

242. How does Transaction management work in Hibernate?

In Hibernate we use Session interface to get a new transaction. Once we get the transaction we can run business operations in that transaction. At the end of successful business operations, we commit the transaction. In case of failure, we rollback the transaction.

Sample code is a follows:

```
Session s = null;
Transaction trans = null;
try {
s = sessionFactory.openSession();
trans = s.beginTransaction();
doTheAction(s);
trans.commit();
} catch (RuntimeException exc) {
trans.rollback();
} finally {
s.close();
}
```

243. How can we mark an entity/collection as immutable in Hibernate?

In Hibernate, by default an entity or collection is mutable. We can add, delete or update an entity/collection.

To mark an entity/collection as immutable, we can use one of the following:

@Immutable: We can use the annotation @Immutable to mark an entity/collection immutable.

XML file: We can also set the property mutable=false in the XML file for an entity to make it immutable.

244. What are the different options to retrieve an object from database in Hibernate?

In Hibernate, we can use one of the following options to retrieve objects from database:

Identifier: We can use load() or get() method and pass the identifier like primary key to fetch an object from database.

HQL: We can create a HQL query and get the object after executing the query.

Criteria API: We can use Criteria API to create the search conditions for getting the objects from database.

Native SQL: We can write native SQL query for a database and just execute it to get the data we want and convert it into desired object.

245. How can we auto-generate primary key in Hibernate?

We can use the primary key generation strategy of type GenerationType.AUTO to auto-generate primary key while persisting an object in Hibernate.

Egg.
@Id
@GeneratedValue(strategy=GenerationType.AUTO)
private int id;
We can leave it null/0 while persisting and Hibernate automatically generates a primary key for us.

Sometimes, AUTO strategy refers to a SEQUENCE instead of an IDENTITY .

246. How will you re-attach an object in Detached state in Hibernate?

We can call one of the methods Session.update(), Session.saveOrUpdate(), or Session.merge() to re-attach an object in detached state with another session in Hibernate.

247. What is the first level of cache in Hibernate?

A Hibernate Session is the first level of cache for persistent data in a transaction.

The second level of cache is at JVM or SessionFactory level.

248. What are the different second level caches available in Hibernate?

In Hibernate, we can use different cache providers for implementing second level cache at JVM/SessionFactory level.

Some of these are:

Hashtable
EHCache
OSCache
SwarmCache
JBoss Cache 1.x
JBoss Cache 2

249. Which is the default transaction factory in Hibernate?

In Hibernate, default transaction factory is JDBCTransactionFactory. But we can change it by setting the property hibernate.transaction.factory_class.

250. What are the options to disable second level cache in Hibernate?

This is a trick question. By default Second level cache is already disabled in Hibernate.

In case, your project is using a second level cache you can use one of the following options to disable second level cache in Hibernate:

We can set hibernate.cache.use_second_level_cache to false.
We can use CacheMode.IGNORE to stop interaction between the session and second-level cache. Session will interact with cache only to invalidate cache items when updates occur

251. What are the different fetching strategies in Hibernate?

Hibernate 3 onwards there are following fetching strategies to retrieve associated objects:

Join fetching: In Join strategy Hibernate uses OUTER join to retrieve the associated instance or collection in the same SELECT.

Select fetching: In Select strategy, Hibernate uses a second SELECT to retrieve the associated entity or collection. We can explicitly disable lazy fetching by specifying lazy="false". By default lazy fetching is true.

Subselect fetching: In Subselect strategy, Hibernate uses a second SELECT to retrieve the associated collections for all entities retrieved in a previous query or fetch.

Batch fetching: In Batch strategy, Hibernate uses a single SELECT to retrieve a batch of entity instances or collections by specifying a list of primary or foreign keys. This is a very good performance optimization strategy for select fetching.

252. What is the difference between Immediate fetching and Lazy collection fetching?

In Immediate fetching an association, collection or attribute is retrieved at the same time when the owner is loaded.
But in Lazy collection fetching, a collection is fetched only when an operation is invoked on that collection by client application.

This is the default fetching strategy for collections in Hibernate.

Lazy fetching is better from performance perspective.

253. What is ‘Extra lazy fetching’ in Hibernate?

In Extra lazy fetching, only individual elements of a collection are fetched from the database when they are required.

In this strategy, Hibernate does not fetch the whole collection into memory unless it is essential.

It is a good fetching strategy for large collections of objects.

254. How can we check is a collection is initialized or not under Lazy Initialization strategy?

Hibernate provides two convenient methods, Hibernate.initialize() and Hibernate.isInitialized() to check whether a collection is initialized or not.

By using Hibernate.initialize() we can force the initialization of a collection in Hibernate.

255. What are the different strategies for cache mapping in Hibernate?

Hibernate provides following strategies for cache mapping:

Read only: If an application requires caching only for read but not for write operations, then we can use this strategy. It is very simple to use and give very good performance benefit.

It is also safe to use in a cluster environment.

Read/Write: If an application also needs caching for write operations, then we use Read/Write strategy.

Read/write cache strategy should not be used if there is requirement for serializable transaction isolation level.

If we want to use it in a cluster environment, we need to implement locking mechanism.

Nonstrict Read/Write: If an application only occasionally updates the data, then we can use this strategy. It cannot be used in systems with serializable transaction isolation level requirement.

Transactional: This strategy supports full transactional cache providers like JBoss TreeCache.

256. What is the difference between a Set and a Bag in Hibernate?

A Bag in Hibernate is an unordered collection. It can have duplicate elements. When we persist an object in a bag, there is no guarantee that bag will maintain any order.

A Set in Hibernate can only store unique objects. If we add the same element to set second time, it just replaces the old one. By default a Set is unordered collection in Hibernate.

257. How can we monitor the performance of Hibernate in an application?

We can use following ways to monitor Hibernate performance:

Monitoring SessionFactory: Since there is one SessionFactory in an application, we can collect the statistics of a SessionFactory to monitor the performance. Hibernate provides sessionFactory.getStatistics() method to get the statistics of SessionFactory.

Hibernate can also use JMX to publish metrics.

Metrics: In Hibernate we can also collect other metrics like- number of open sessions, retrieved JDBC connections, cache hit, miss etc.

These metrics give great insight into the performance of Hibernate. We can tune Hibernate settings and strategies based on these metrics.

258. How can we check if an Object is in Persistent, Detached or Transient state in Hibernate?

We can use following methods to check the state of an object in Hibernate:

Persistent State: If call to EntityManager.contains(object) returns true, the object is in Persistent state.

Detached State: If the call to PersistenceUnitUtil.getIdentifier(object) returns identifier property then the object is in detached state.

Transient State: If call to PersistenceUnitUtil.getIdentifier(object) returns null then object is in Transient state.

We can get access to PersistenceUnitUtil from the EntityManagerFactory in Hibernate.

259. What is ‘the inverse side of association’ in a mapping?

Let us consider an example in which a customer can have multiple orders and for every order there has to be a customer.

In OO world, customer is the owner of order. In SQL world, an Order has reference to customer id.

It is a bi-directional one to many mapping from customer to order.

The inverse side in this mapping is the owner of object. In this case customer is the owner or order. Since an order cannot exist without a customer. But a customer can exist without an order.

Also customer has no column to save order data. But an Order table can store customer id, which is used for mapping.

260. What is ORM metadata?

ORM uses metadata for its internal work. ORM maintains metadata to generate code used for accessing columns and tables.

ORM maps classes to tables and stores this information in Metadata. It maps fields in classes to columns in tables. These kinds of mappings are also part of Metadata.

Application developers can also access Hibernate Metadata by using ClassMetadata and CollectionMetadata interfaces and Type hierarchy.

261. What is the difference between load() and get() method in Hibernate?

In Hibernate, load() and get() methods are quite similar in functionality.

The main difference is that load() method will throw an ObjectNotFoundException if row corresponding to an object is not found in the database.

On the other hand, get() method returns null value when an object is not found in the database.

It is recommended that we should use load() method only when we are sure that object exists in database.

262. When should we use get() method or load() method in Hibernate?

As a thumb rule we can follow these guidelines:

We should use get() method when we want to load an object.
We should use load() method when we need a reference to an object without running extra SQL queries.

263. What is a derived property in Hibernate?

In Hibernate, a derived property is not mapped to any column of a database table.

A derived property is computed at runtime by evaluation of an expression.

These are read only properties.

Egg. In this example profitMargin is derived from salePrice and buyPrice.

<property name="profitMargin" formula="(SELECT (i.salePrice – i.buyPrice) FROM item i WHERE i.Id = Id)"/>

264. How can we use Named Query in Hibernate?

A Named SQL query is the HQL query that is associated with a string name and can be referenced in the application by name.

It can be used in following ways:

XML Mapping File: We can define it in XML mapping file.

```
Egg.    <query name="findBookByAuthor”>
          <![CDATA[from Book s where s.author = :author]]>
      </query>
```

Annotation: We can also mark Named SQL with annotation.

```
@NamedQueries({
  @NamedQuery(
  name = "findBookByAuthor”,
  query = "from Book s where s.author = :author”
  )
})
```

265. What are the two locking strategies in Hibernate?

There are two popular locking strategies that can be used in Hibernate:

Optimistic: In Optimistic locking we assume that multiple transactions can complete without affecting each other. So we let the transactions do their work without locking the resources initially.

Just before the commit, we check if any of the resource has changed by another transaction, then we throw exception and rollback the transaction.

Pessimistic: In Pessimistic locking we assume that concurrent transactions will conflict while working with same resources. So a transaction has to first obtain lock on the resources it wants to update.

The other transaction can proceed with same resource only after the lock has been released by previous transaction.

266. What is the use of version number in Hibernate?

Version number is used in optimistic locking in Hibernate. When a transaction modifies an object, it increments its version. Based on version number, second transaction can determine if the object it has read earlier has changed or not.

If the version number at the time of write is different than the version number at the time of read, then we should not commit the transaction.

267. What is the use of session.lock() method in Hibernate?

Session.lock() is a deprecated method in Hibernate. We should not use it.

Instead we should call buildLockRequest(LockMode).lock(entityName, object) method in Hibernate.

268. What inheritance mapping strategies are supported by Hibernate?

Hibernate supports following inheritance mapping strategies between classes and tables:

Table per class hierarchy: In case of multiple types of books, we can have one book class and one book table. We can store all child classes of book like- HardCoverBook, PaperBackBook etc in same table book. But we can identify the subclasses by a BookType column in Book table.

Table per subclass: In this case we can have separate table for each kind of book. HardCoverBook table for HardCoverBook book class. PaperBackBook table for PaperBackBook book class. And there will be a parent table, Book for Book class.

Table per concrete class: In this case also we have separate table for each kind of book. But in this case we have even inherited properties defined inside each table. There is no parent table Book for Book class, since it is not a concrete class.

Maven

269. What is Maven?

Maven is a software project management tool. It is open source software from Apache software foundation.

It is used for building, reporting and documenting a Software project. It is mainly based on POM (Project Object Model).

270. What are the main features of Maven?

Some of the main features of Maven are:

1. **Simple**: Maven provides simple project setup that is based on best practices.

2. **Fast**: You can get a new project or module started in a few seconds in Maven.

3. **Easy to learn**: Maven usage and commands are easy to learn across all projects. Therefore ramp up time for new developers coming onto a project is very less.

4. **Dependency management**: Maven provides superior dependency management including automatic updates and transitive dependencies.

5. **Multiple Projects**: You can easily work with multiple projects at the same time by using Maven.

6. **Large Library**: Maven has a large and growing repository of libraries and metadata to use out of the box.

7. **Extensible**: Maven supports the ability to easily write plugins in Java or scripting languages for extending its core functionality.

8. **Instant**: Maven is online and it provides instant access to new features with very less configuration.

271. What areas of a Project can you manage by using Maven?

Maven can help us manage following areas of a project:

1. Build
2. Testing
3. Release
4. Reporting
5. Software Change Management (SCM)
6. Documentation
7. Distribution

272. What are the main advantages of Maven?

Maven has a long list of advantages for Software development. Some of the main advantages are:

1. **Common Project Structure**: By using Maven, every developer has a common project structure that helps in understanding the code as well as developing new features in a new project.

2. **Modular Design**: Maven promotes modular design that divides a complex project into multiple modules that are easier to manage. By using Maven, it is easier to manage multiple modules for build, test, release etc.

3. **Centralized Dependency Management**: With Maven, each developer does not have to include the jars separately in each project or module. Maven provides a centralized dependency management that can help improve efficiency of software development.

4. **Fewer Decisions**: With Maven a developer has to make fewer decisions about things unrelated to software development work. The project structure comes ready with Maven, dependency management is a uniform approach and build/release are handled by Maven. So a developer can focus on core work of developing software.

273. Why do we say “Maven uses convention over configuration”?

Convention over configuration is a Software Design Paradigm that decreases the number of decisions made by a software developer, without losing flexibility.

In Maven, there are many conventions for setting up the project, building the artifacts, running unit tests and releasing the code. These conventions lead to common process for Software development.

In case of other tools, there are a lot of configuration options are present. But most of the time, a developer uses same set of configuration options. So it is better to make these as a default options. Maven uses default options from best practices and provides right conventions for Software development.

274. What are the responsibilities of a Build tool like Maven?

A Build tool like Maven helps us with following tasks:

1. **Source Code**: A Build tool can generate source code based on templates.

2. **Documentation**: We can get documentation files from source code by using a build tool. E.g. Javadoc

3. **Compilation**: Primary responsibility of a Build tool is to compile source code into executable code.

4. **Packaging**: A Build tool packages compiled code into a deployable file like- jar, zip war etc.

5. **Deployment**: We can deploy the packaged code on server by using a Build tool.

275. What are the differences between Ant and Maven?

Key differences between Ant and Maven are:

1. Ant is a Java library and command line toolbox for build process. Maven is a framework for many aspects of software development like- project setup, compile, build, documentation etc.

2. Ant does not have any conventions for project structure or build processes. Maven has conventions for setting up project structure as well as for build processes.

3. Ant is based on procedural programming. We have to write code for compilation build, copy etc. Maven is based on declarative programming. We have to just configure it for our project setup and programming.

4. Ant does not impose any lifecycle. We need to create the sequence of tasks manually. Maven has a lifecycle for software build processes. There are well-defined phases that we can use in Maven.

5. Ant scripts are not reusable in multiple projects. Maven has plugins that are reusable across multiple projects.

276. What is MOJO in Maven?

MOJO stands for Maven plain Old Java Object.

Every MOJO is an executable goal in Maven. It is like an annotated Java class. It specifies metadata about a goal like- goal name, phase of lifecycle for goal and parameters required by goal.

A Maven plugin can contain multiple MOJOs.

277. What is a Repository in Maven?

A repository is a location on file system where build artifacts, jars, dependencies and pom.xml files are stored.

278. What are the different types of repositories in Maven?

There are mainly two types of repositories in Maven:

1. **Local Repository**: This is your local folder in which a copy of your installation and dependencies is stored.

2. **Remote Repository**: This is a remote folder in which jars and other build artifacts are stored. These can be located on servers within your organization.

3. **Central Remote Repository**: This is the central Maven repository that is located on repo.maven.apache.org or uk.maven.org or any other third party location. This where we can find artifacts from different providers that are available for download and use. Like- Hibernate, Spring libraries etc.

279. What is a local repository in Maven?

Maven local repository is a folder in your local files system in which your project's installation, dependency jars, plugins etc. are stored.

Default location of Maven local repository is .m2 folder. It can be located under following location on file system:

Windows – C:\Documents and Settings\{ username}\.m2
Unix/Linux/Mac – ~/.m2

280. What is a central repository in Maven?

Maven central repository is a truly remote repository that is located on repo.maven.apache.org or uk.maven.org or any other third party location.

This contains the jars and artifacts provided by various software providers.

Central repository contains a large amount of data. Therefore it is not allowed to scrape the whole site. But you can use the relevant jars that you want for download and use in your Maven project.

281. What is a Remote repository in Maven?

A Remote repository is a remote location on the internet where the jars and dependencies from different vendors are stored.

These files can be accessed by protocols like- file:// or http:// etc.

These can be truly remote repositories set up by third party vendors or locations inside your organization that contains the relevant jars required by your project.

282. Why we should not store jars in CVS or any other version control system instead of Maven repository?

Maven recommends storing jars in local repository instead of CVS or any other version control system. There are following advantages of storing it in Maven repo vs. CVS:

Less Storage: A repository is very large, but it takes less space because each JAR is stored only in one place. E.g. If we have 10 modules dependent on Spring jar, then they all refer to same Spring jar stored in local repository.

Quicker Checkout: Project checkout is quicker from local repository, since there is not need to checkout jars if they are already present in repo.

No need for versioning: There is no need to version JARS since external dependencies do not change so often.

283. Can anyone upload JARS or artifacts to Central Repository?

No, we need special permissions to upload JARS and artifacts to Central Maven Repository?

284. What is a POM?

POM is an abbreviation for Project Object Model. This is the basic unit of work in Maven. It is an XML file with name pom.xml.

It contains details of project and project configuration that are used by Maven to build the project.

It also contains default values for many projects. E.g. target is the name of build directory for Java Maven project.

285. What is Super POM?

Super POM is Maven's default POM. All the POM files extend from Super POM.

286. What are the main required elements in POM file?

Every POM file should have following required elements:

1. project root
2. modelVersion
3. groupID: the id of the project's group.
4. artifactID: the id of the artifact (project)
5. version: the version of the artifact under the specified group

287. What are the phases in Build lifecycle in Maven?

In Maven, each build lifecycle consists of many phases. Default build lifecycle has following phases:

1. **validate**: In this phase, Maven validates that the project is correct and all necessary information is available to run next phase.

2. **compile**: Maven compiles the source code of the project in this phase.

3. **test**: This is the phase to run unit tests on the compiled source. There should not be any need to package or deploy the code to run these tests.

4. **package**: In this phase, Maven takes the compiled code and packages it in its distributable format, such as a JAR.

5. **verify**: Maven runs any checks on results of integration tests to ensure that quality criteria are met.

6. **install**: In this phase, Maven installs the package into local repository. After this it can be used as a dependency in other projects locally.

7. **deploy**: In the build environment, Maven copies the final package to the remote repository for sharing with other developers and projects.

288. What command will you use to package your Maven project?

To package a project into a distributable format we use following command:

mvn -package

289. What is the format of fully qualified artifact name of a Maven project?

A Maven project has artifact name with following format:

<groupId>:<artifactId>:<version>

Following is the convention used by some organizations:

Parent pom

groupId: org.Orgname.Projectname
artifactId: org.Orgname.Projectname
version: x.x.x

E.g. org.Orgname.Projectname:org.Orgname.Projectname-1.0.0.pom

Modules

groupId: org.Orgname.Projectname
artifactId: org.Orgname.Projectname.Modulename
version: x.x.x

E.g.
org.Orgname.Projectname:org.Orgname.Projectname.Modulename-1.0.0.jar

290. What is an Archetype in Maven?

As per official definition, an Archetype is a Maven project templating toolkit.

By using an Archetype, an author of Archetype can create a Project template. Users of this project template (archetype) can pass different parameters to this template and start using it.

Archetype promotes consistency in the process of creating and working on a project. It also helps in reducing the ramp up time for new developers to come on board on a project.

291. What is the command in Maven to generate an Archetype?

In Maven, we can use following command to generate an Archetype:

mvn archetype:generate

292. What are the three main build lifecycles of Maven?

Maven has following three build lifecycles that further contain multiple phases:

1. **clean**: In this lifecycle any files generated by previous builds are removed.

2. **default**: This lifecycle is used for validating, compiling and creating the application. It has multiple phases like-compile, test, package inside it.

3. **site**: Maven generates and deploys the documentation of a site in this phase.

293. What are the main uses of a Maven plugin?

Maven is mainly a plugin execution framework. At the code of Maven all the work is done by plugins. A Maven plugin can be used for following purposes:

1. Cleaning up the code
2. Compiling the code
3. Creating a JAR file
4. Deploying the artifacts
5. Running the unit tests
6. Documenting the project
7. Generating the site of a project
8. Generating a WAR file
9. Generate a checkstyle report

294. How will you find the version of a plugin being used?

Maven Help Plugin has a describe goal. This can be used for listing the version of a plugin. Sample command for this is:

mvn -Dplugin=install help:describe

Note: In the above command replace Dplugin with the plugin prefix as the argument. Do not use the artifact ID of plugin here.

295. What are the different types of profile in Maven? Where will you define these profiles?

In Maven, we can have following types of Profile:

Per Project
It is defined in the POM itself (pom.xml).

Per User
We can define it in the Maven-settings (%USER_HOME%/.m2/settings.xml).

Global
It is defined in the global Maven-settings (${maven.home}/conf/settings.xml).

Profile descriptor
Descriptor is located in project basedir (profiles.xml) (It is not supported in Maven 3.0)

296. What are the different setting files in Maven? Where will you find these files?

Maven is very simple to use. At the core it has a setting file names settings.xml. This file contains the setting element that is used to configure the Maven with different options.

The main locations where this file can be found are:

Maven Installation directory: ${maven.home}/conf/settings.xml

User Home directory: ${user.home}/ .m2 / settings.xml

297. What are the main elements we can find in settings.xml?

In settings.xml we can have all the configuration information for Maven. Some of the important elements are:

localRepository: The value of this element is the path of this build system's local repository. The default value is ${user.home}/.m2/repository.

It is used for a main build server to allow all logged-in users to build from a common local repository.

interactiveMode: If it is true then Maven should attempt to interact with the user for input. If it is false then Maven does not interact with the user. Default setting is true.

usePluginRegistry: If it is true Maven uses the ${user.home}/.m2/plugin-registry.xml file to manage plugin versions. By defaults it is false.

offline: If it is true this build system should be able to operate in offline mode. By default it is false. This element is used for build servers that cannot connect to a remote repository due to network setup or security reasons.

298. How will you check the version of Maven in your system?

We can use following command in console to check the version of Maven in our system.

mvn -version

299. How will you verify if Maven is installed on Windows?

To check this, type mvn –version in cmd prompt of Windows. This will give you the version of Maven installed on Windows.

300. What is a Maven artifact?

A Maven artifact is a file that gets deployed to a Maven repository. In most cases it is a JAR file.

When Maven build runs, it creates one or more artifacts. In case of Java projects, it produces a compiled jar and a sources jar.

Every artifact in Maven has a group ID, an artifact ID and a version string. These three attributes uniquely identify an artifact.

In Maven, we specify a project's dependencies as artifacts.

301. What are the different dependency scopes in Maven?

Maven supports following dependency scopes:

compile: This is the default dependency scope in Maven. The compile level dependencies are available in all classpaths of a project. These dependencies are also propagated to dependent projects.

provided: This scope is similar to compile. But in this scope we expect the JDK or a container to provide the dependency at runtime. E.g. While building a web application for the Java Enterprise Edition, we can set the dependency on the Servlet API and related Java EE APIs to scope provided. The web container will provide these classes at runtime to our application.

This scope is only available on the compilation and test classpath, and is not transitive.
runtime: The dependency in this scope is not required for compilation. It is required for execution. It is available in the runtime and test classpaths. It is not present in the compile classpath.

test: This scope is used for dependencies that are required for test compilation and execution phases. This scope is not transitive.

system: This scope is same as provided scope, except that you have to provide the JAR that contains it explicitly. In this case, he artifact is always available. There is no need to look it up in a repository.

import: This scope is only used on a dependency of type pom in the <dependencyManagement> section. In this case, the specified POM has to be replaced with the dependencies in that POM's <dependencyManagement> section. This scope is only available in

Maven 2.0.9 or later.

302. How can we exclude a dependency in Maven?

To exclude a dependency we can add the <exclusions> tag under the <dependency> section of the pom.

E.g.

```
<dependencies>
   <dependency>
    <groupId>test.ProjectX</groupId>
    <artifactId>ProjectX</artifactId>
    <version>1.0</version>
    <scope>compile</scope>
    <exclusions>
     <exclusion>  <!-- exclusion is mentioned here -->
      <groupId>test.ProjectY</groupId>
      <artifactId>ProjectY</artifactId>
     </exclusion>
    </exclusions>
   </dependency>
</dependencies>
```

303. How Maven searches for JAR corresponding to a dependency?

Maven first looks for a JAR related to a dependency in the local repository. If it finds it there then it stops.

If it does not find it in local repo, it looks for the JAR in the remote repository and downloads the corresponding version of JAR file. From remote repository it stores the JAR into local repository.

304. What is a transitive dependency in Maven?

Let say you have a Project A that depends on dependency B. The dependency B further depends on dependency C. So your dependency C is a Transitive Dependency of your project A.

In Maven, starting from 2.0, you do not have to specify transitive dependencies. You just mention your immediate dependencies in pom.xml.

Maven takes care of resolving the Transitive dependencies and includes them automatically.

305. What are Excluded dependencies in Maven?

Let say a project A depends on project B, and project B depends on project C. The developers of project A can explicitly exclude project C as a dependency. We can use the "exclusion" element to exclude it.

Such dependencies are called Excluded dependencies in Maven.

306. What are Optional dependencies in Maven?

Let say a project B depends on project C. The developers of project B can mark project C as an optional dependency by using the "optional" element.

In case project A depends on project B, A will depend only on B and not on B's optional dependency C.

The developers of project A may then explicitly add a dependency on C. The dependency of B on C is known as Optional dependency in Maven.

307. Where will you find the class files after compiling a Maven project successfully?

Once Maven completes the compilation successfully, it stores the files in target folder. The default location for class files is:

${basedir}/target/classes/

308. What are the default locations for source, test and build directories in Maven?

The default locations are as follows:

Source: src/main/java
Test: src/main/test
Build: Target

309. What is the result of jar:jar goal in Maven?

In Maven, jar:jar goal creates a jar file in the Maven build directory. Jar file is create with the name format ${project.id}-${project.currentVersion}.jar.

The id and currentVersion are mentioned in the project.xml of the project being built.

jar:jar does not recompile sources. It just creates a jar from already compiled classes.

310. How can we get the debug or error messages from the execution of Maven?

At times, project build or compile fails in Maven. At this time it is very helpful to see the debug or error messages from Maven execution.

To get the debug messages we can call Maven with -X option.

To get the error/exception messages we can call Maven with -e option.

311. What is the difference between a Release version and SNAPSHOT version in Maven?

A SNAPSHOT version in Maven is the one that has not been released.

Before every release version there is a SNAPSHOT version. Before 1.0 release there will be 1.0-SNAPSHOT.

If we download 1.0-SNAPSHOT today then we may get different set of files than the one we get on downloading it yesterday. SNAPSHOT version can keep getting changes in it since it is under development.

But release version always gives exactly same set files with each download.

312. How will you run test classes in Maven?

We need Surefire plugin to run the test classes in Maven.

To run a single test we can call following command:

mvn -Dtest=TestCaseA test

We can also use patterns to run multiple test cases:

mvn -Dtest=TestCase* test

or

mvn -Dtest=TestCaseA,TestCaseB,TestImportant* test

313. Sometimes Maven compiles the test classes but doesn't run them? What could be the reason for it?

In Maven, Surefire plugin is used for running the Tests.

We can configure it to run certain test classes. Sometimes we you may have unintentionally specified an incorrect value to ${test} in settings.xml or pom.xml.

We need to look for following in pom.xml/settings.xml and fix it:

```
<properties>
   <property>
    <name>test</name>
    <value>some-value</value>
   </property>
 </properties>
```

314. How can we skip the running of tests in Maven?

We can use the parameter -Dmaven.test.skip=true or -DskipTests=true in the command line for skipping the tests.

The parameter -Dmaven.test.skip=true skips the compilation of tests.

The parameter -DskipTests=true skips the execution of tests

Surefire plugin of Maven honors these parameters.

315. Can we create our own directory structure for a project in Maven?

Yes, Maven gives us the flexibility of creating our own directory structure. We just need to configure the elements like <sourceDirectory>, <resources> etc. in the <build> section of pom.xml.

316. What are the differences between Gradle and Maven?

Gradle is nowadays getting more popular. Google uses it for Android development and release. Companies like LinkedIn also use Gradle.

Gradle is based on Domain Specific Language (DSL). Maven is based on XML.

Gradle gives more flexibility to do custom tasks similar to ANT. Maven scripts have predefined structure. So it is less flexible.

Maven is mainly used for Java based systems. Gradle is used for a variety of languages. It is a Polyglot build tool.

317. What is the difference between Inheritance and Multi-module in Maven?

In Maven, we can create a parent project that will pass its values to its children projects.

A multi-module project is created to manage a group of other sub-projects or modules. The multi-module relationship is like a tree that starts from the topmost level to the bottom level. In a multi-module project, we specify that a project should include the specific modules for build. Multi-module builds are used to group modules together in a single build.

Whereas in Inheritance, the parent-child project relationship starts from the leaf node and goes upwards. It deals more with the definition of a specific project. In this case a child's pom is derived from its parent's pom.

318. What is Build portability in Maven?

In Maven, the portability of a build is the measure of how easy it is to take a particular project and build it in different environments.

A build that does not require any custom configuration or customization of properties files is more portable than a build that requires a lot of custom work to build it from scratch.

Open source projects from Apache Commons are one of the most portable projects. These build can work just out of the box.

GIT

319. How can we see n most recent commits in GIT?

We can use git log command to see the latest commits. To see the three most recent commits we use following command:

git log -3

320. How can we know if a branch is already merged into master in GIT?

We can use following commands for this purpose:

git branch --merged master : This prints the branches merged into master

git branch --merged lists : This prints the branches merged into HEAD (i.e. tip of current branch)

git branch --no-merged : This prints the branches that have not been merged

By default this applies only to local branches.

We can use -a flag to show both local and remote branches.

Or we can use -r flag to show only the remote branches.

321. What is the purpose of git stash drop?

In case we do not need a specific stash, we use git stash drop command to remove it from the list of stashes.

By default, this command removes to latest added stash

To remove a specific stash we specify as argument in the git stash drop <stashname> command.

322. What is the HEAD in GIT?

A HEAD is a reference to the currently checked out commit.

It is a symbolic reference to the branch that we have checked out.

At any given time, one head is selected as the ‘current head’ This head is also known as HEAD (always in uppercase).

323. What is the most popular branching strategy in GIT?

There are many ways to do branching in GIT. One of the popular ways is to maintain two branches:

master: This branch is used for production. In this branch HEAD is always in production ready state.

develop: This branch is used for development. In this branch we store the latest code developed in project. This is work in progress code.

Once the code is ready for deployment to production, it is merged into master branch from develop branch.

324. What is SubGit?

SubGit is software tool used for migrating SVN to Git. It is very easy to use. By using this we can create a writable Git mirror of a Subversion repository.

It creates a bi-directional mirror that can be used for pushing to Git as well as committing to Subversion.

SubGit also takes care of synchronization between Git and Subversion.

325. What is the use of git instaweb?

Git-instaweb is a script by which we can browse a git repository in a web browser.

It sets up the gitweb and a web-server that makes the working repository available online.

326. What are git hooks?

Git hooks are scripts that can run automatically on the occurrence of an event in a Git repository. These are used for automation of workflow in GIT.

Git hooks also help in customizing the internal behavior of GIT.

These are generally used for enforcing a GIT commit policy.

327. What is GIT?

GIT is a mature Distributed Version Control System (DVCS). It is used for Source Code Management (SCM).

It is open source software. It was developed by Linus Torvalds, the creator of Linux operating system.

GIT works well with a large number of IDEs (Integrated Development Environments) like- Eclipse, InteliJ etc.

GIT can be used to handle small and large projects.

328. What is a repository in GIT?

A repository in GIT is the place in which we store our software work.

It contains a sub-directory called .git. There is only one .git directory in the root of the project.

In .git, GIT stores all the metadata for the repository. The contents of .git directory are of internal use to GIT.

329. What are the main benefits of GIT?

There are following main benefits of GIT:

1. Distributed System: GIT is a Distributed Version Control System (DVCS). So you can keep your private work in version control but completely hidden from others. You can work offline as well.
2. Flexible Workflow: GIT allows you to create your own workflow. You can use the process that is suitable for your project. You can go for centralized or master-slave or any other workflow.
3. Fast: GIT is very fast when compared to other version control systems.
4. Data Integrity: Since GIT uses SHA1, data is not easier to corrupt.
5. Free: It is free for personal use. So many amateurs use it for their initial projects. It also works very well with large size project.
6. Collaboration: GIT is very easy to use for projects in which collaboration is required. Many popular open source software across the globe use GIT.

330. What are the disadvantages of GIT?

GIT has very few disadvantages. These are the scenarios when GIT is difficult to use. Some of these are:

1. Binary Files: If we have a lot binary files (non-text) in our project, then GIT becomes very slow. E.g. Projects with a lot of images or Word documents.
2. Steep Learning Curve: It takes some time for a newcomer to learn GIT. Some of the GIT commands are non-intuitive to a fresher.
3. Slow remote speed: Sometimes the use of remote repositories in slow due to network latency. Still GIT is better than other VCS in speed.

331. What are the main differences between GIT and SVN?

The main differences between GIT and SVN are:

1. Decentralized: GIT is decentralized. You have a local copy that is a repository in which you can commit. In SVN you have to always connect to a central repository for check-in.

2. Complex to learn: GIT is a bit difficult to learn for some developers. It has more concepts and commands to learn. SVN is much easier to learn.

3. Unable to handle Binary files: GIT becomes slow when it deals with large binary files that change frequently. SVN can handle large binary files easily.

4. Internal directory: GIT creates only .git directory. SVN creates .svn directory in each folder.

5. User Interface: GIT does not have good UI. But SVN has good user interfaces.

332. How will you start GIT for your project?

We use git init command in an existing project directory to start version control for our project.

After this we can use git add and git commit commands to add files to our GIT repository.

333. What is git clone in GIT?

In GIT, we use git clone command to create a copy of an existing GIT repository in our local.

This is the most popular way to create a copy of the repository among developers.

It is similar to svn checkout. But in this case the working copy is a full-fledged repository.

334. How will you create a repository in GIT?

To create a new repository in GIT, first we create a directory for the project. Then we run ‘git init’ command.

Now, GIT creates .git directory in our project directory. This is how our new GIT repository is created.

335. What are the different ways to start work in GIT?

We can start work in GIT in following ways:

New Project: To create a new repository we use git init command.

Existing Project: To work on an existing repository we use git clone command.

336. GIT is written in which language?

Most of the GIT distributions are written in C language with Bourne shell. Some of the commands are written in Perl language.

337. What does ‘git pull’ command in GIT do internally?

In GIT, git pull internally does a git fetch first and then does a git merge.

So pull is a combination of two commands: fetch and merge.

We use git pull command to bring our local branch up to date with its remote version.

338. What does ‘git push’ command in GIT do internally?

In GIT, git push command does following two commands:

1. **fetch**: First GIT, copies all the extra commits from server into local repo and moves origin/master branch pointer to the end of commit chain.

2. **merge**: Then it merges the origin/master branch into the master branch. Now the master branch pointer moves to the newly created commit. But the origin/master pointer remains there.

339. What is git stash?

In GIT, sometimes we do not want to commit our code but we do not want to lose also the unfinished code. In this case we use git stash command to record the current state of the working directory and index in a stash. This stores the unfinished work in a stash, and cleans the current branch from uncommitted changes.

Now we can work on a clean working directory.

Later we can use the stash and apply those changes back to our working directory.

At times we are in the middle of some work and do not want to lose the unfinished work, we use git stash command.

340. What is the meaning of ‘stage’ in GIT?

In GIT, stage is a step before commit. To stage means that the files are ready for commit.

Let say, you are working on two features in GIT. One of the features is finished and the other is not yet ready. You want to commit and leave for home in the evening. But you can commit since both of them are not fully ready. In this case you can just stage the feature that is ready and commit that part. Second feature will remain as work in progress.

341. What is the purpose of git config command?

We can set the configuration options for GIT installation by using git config command.

342. How can we see the configuration settings of GIT installation?

We can use ‘git config --list’ command to print all the GIT configuration settings in GIT installation.

343. How will you write a message with commit command in GIT?

We call following command for commit with a message:
$/> git commit –m <message>

344. What is stored inside a commit object in GIT?

GIT commit object contains following information:

SHA1 name: A 40 character string to identify a commit
Files: List of files that represent the state of a project at a specific point of time
Reference: Any reference to parent commit objects

345. How many heads can you create in a GIT repository?

There can be any number of heads in a repository.

By default there is one head known as HEAD in each repository in GIT.

346. Why do we create branches in GIT?

If we are simultaneously working on multiple tasks, projects, defects or features, we need multiple branches. In GIT we can create a separate branch for each separate purpose.

Let say we are working on a feature, we create a feature branch for that. In between we get a defect to work on then we create another branch for defect and work on it. Once the defect work is done, we merge that branch and come back to work on feature branch again.

So working on multiple tasks is the main reason for using multiple branches.

347. What are the different kinds of branches that can be created in GIT?

We can create different kinds of branches for following purposes in GIT:

Feature branches: These are used for developing a feature.

Release branches: These are used for releasing code to production.

Hotfix branches: These are used for releasing a hotfix to production for a defect or emergency fix.

348. How will you create a new branch in GIT?

We use following command to create a new branch in GIT:

$/> git checkout –b <branchname>

349. How will you add a new feature to the main branch?

We do the development work on a feature branch that is created from master branch. Once the development work is ready we use git merge command to merge it into master branch.

350. What is a pull request in GIT?

A pull request in GIT is the list of changes that have been pushed to GIT repository. Generally these changes are pushed in a feature branch or hotfix branch. After pushing these changes we create a pull request that contains the changes between master and our feature branch. This pull request is sent to reviewers for reviewing the code and then merging it into develop or release branch.

351. What is merge conflict in GIT?

A merge conflict in GIT is the result of merging two commits. Sometimes the commit to be merged and current commit have changes in same location. In this scenario, GIT is not able to decide which change is more important. Due to this GIT reports a merge conflict. It means merge is not successful. We may have to manually check and resolve the merge conflict.

352. How can we resolve a merge conflict in GIT?

When GIT reports merge conflict in a file, it marks the lines as follows:

E.g.
the business days in this week are
<<<<<<< HEAD
five
=======
six
>>>>>>> branch-feature

To resolve the merge conflict in a file, we edit the file and fix the conflicting change. In above example we can either keep five or six.

After editing the file we run git add command followed by git commit command. Since GIT is aware that it was merge conflict, it links this change to the correct commit.

353. What command will you use to delete a branch?

After the successful merge of feature branch in main branch, we do not need the feature branch.

To delete an unwanted branch we use following command:

git branch –d <branchname>

354. What command will you use to delete a branch that has unmerged changes?

To forcibly delete an unwanted branch with unmerged changes, we use following command:

git branch –D <branchname>

355. What is the alternative command to merging in GIT?

Another alternative of merging in GIT is rebasing. It is done by git rebase command.

356. What is Rebasing in GIT?

Rebasing is the process of moving a branch to a new base commit. It is like rewriting the history of a branch.

In Rebasing, we move a branch from one commit to another. By this we can maintain linear project history.

Once the commits are pushed to a public repository, it is not a good practice to use Rebasing.

357. What is the ‘Golden Rule of Rebasing’ in GIT?

The golden rule of Rebasing is that we should never use git rebase on public branches. If other people are using the same branch then they may get confused by looking at the changes in Master branch after GIT rebasing.

Therefore, it is not recommended to do rebasing on a public branch that is also used by other collaborators.

358. Why do we use Interactive Rebasing in place of Auto Rebasing?

By using Interactive rebasing we can alter the commits before moving them to a new branch.

This is more powerful than an automated rebase. It gives us complete control over the branch's commit history.

Generally, we use Interactive Rebasing to clean up the messy history of commits just before merging a feature branch into master.

359. What is the command for Rebasing in Git?

Git command for rebasing is:

git rebase <new-commit>

360. What is the main difference between git clone and git remote?

The main difference between git clone and git remote is that git clone is used to create a new local repository whereas git remote is used in an existing repository.

git remote adds a new reference to existing remote repository for tracking further changes.

git clone creates a new local repository by copying another repository from a URL.

361. What is GIT version control?

GIT version control helps us in managing the changes to source code over time by a software team. It keeps track of all the changes in a special kind of database. If we make a mistake, we can go back in time and see previous changes to fix the mistake.

GIT version control helps the team in collaborating on developing a software and work efficiently. Every one can merge the changes with confidence that everything is tracked and remains intact in GIT version control. Any bug introduced by a change can be discovered and reverted back by going back to a working version.

362. What GUI do you use for working on GIT?

There are many GUI for GIT that we can use. Some of these are:

GitHub Desktop
GITX-dev
Gitbox
Git-cola
SourceTree
Git Extensions
SmartGit
GitUp

363. What is the use of git diff command in GIT?

In GIT, git diff command is used to display the differences between 2 versions, or between working directory and an index, or between index and most recent commit.

It can also display changes between two blob objects, or between two files on disk in GIT.

It helps in finding the changes that can be used for code review for a feature or bug fix.

364. What is git rerere?

In GIT, rerere is a hidden feature. The full form of rerere is “reuse recorded resolution”.

By using rerere, GIT remembers how we’ve resolved a hunk conflict. The next time GIT sees the same conflict, it can automatically resolve it for us.

365. What are the three most popular version of git diff command?

Three most popular git diff commands are as follows:

git diff: It displays the differences between working directory and the index.

git diff –cached: It displays the differences between the index and the most recent commit.

git diff HEAD: It displays the differences between working directory and the most recent commit

366. What is the use of git status command?

In GIT, git status command mainly shows the status of working tree.

It shows following items:

1. The paths that have differences between the index file and the current HEAD commit.

2. The paths that have differences between the working tree and the index file

3. The paths in the working tree that are not tracked by GIT.

Among the above three items, first item is the one that we commit by using git commit command. Item two and three can be committed only after running git add command.

367. What is the main difference between git diff and git status?

In GIT, git diff shows the differences between different commits or between the working directory and index.

Whereas, git status command just shows the current status of working tree.

368. What is the use of git rm command in GIT?

In GIT, git rm command is used for removing a file from the working tree and the index.

We use git rm –r to recursively remove all files from a leading directory.

369. What is the command to apply a stash?

Sometimes we want to save our unfinished work. For this purpose we use git stash command. Once we want to come back and continue working from the last place where we left, we use git stash apply command to bring back the unfinished work.

So the command to apply a stash is:

git stash apply

Or we can use
git stash apply <stashname>

370. Why do we use git log command?

We use git log command to search for specific commits in project history.

We can search git history by author, date or content. It can even list the commits that were done x days before or after a specific date.

371. Why do we need git add command in GIT?

GIT gives us a very good feature of staging our changes before commit. To stage the changes we use git add command. This adds our changes from working directory to the index.

When we are working on multiple tasks and we want to just commit the finished tasks, we first add finished changes to staging area and then commit it. At this time git add command is very helpful.

372. Why do we use git reset command?

We use git reset command to reset current HEAD to a specific state. By default it reverses the action of git add command.

So we use git reset command to undo the changes of git add command.

373. What does a commit object contain?

Whenever we do a commit in GIT by using git commit command, GIT creates a new commit object. This commit objects is saved to GIT repository.

The commit object contains following information:

HASH: The SHA1 hash of the Git tree that refers to the state of index at commit time.
Commit Author: The name of person/process doing the commit and date/time.
Comment: Some text messages that contains the reason for the commit .

374. How can we convert git log messages to a different format?

We can use pretty option in git log command for this.

git log – pretty

This option converts the output format from default to other formats. There are pre-built formats available for our use.

git log –pretty=oneline

E.g. git log --pretty=format:"%h - %an, %ar : %s"
ba72a6c - Dave Adams, 3 years ago : changed the version number

375. What are the programming languages in which git hooks can be written?

Git hooks are generally written in shell and PERL scripts. But these can be written in any other language as long as it has an executable.

Git hooks can also be written in Python script.

376. What is a commit message in GIT?

A commit message is a comment that we add to a commit. We can provide meaningful information about the reason for commit by using a commit message.

In most of the organizations, it is mandatory to put a commit message along with each commit.

Often, commit messages contain JIRA ticket, bug id, defect id etc. for a project.

377. How GIT protects the code in a repository?

GIT is made very secure since it contains the source code of an organization. All the objects in a GIT repository are encrypted with a hashing algorithm called SHA1.

This algorithm is quite strong and fast. It protects source code and other contents of repository against the possible malicious attacks.

This algorithm also maintains the integrity of GIT repository by protecting the change history against accidental changes.

378. How GIT provides flexibility in version control?

GIT is very flexible version control system. It supports non-linear development workflows. It supports flows that are compatible with external protocols and existing systems.

GIT also supports both branching and tagging that promotes multiple kinds of workflows in version control.

379. How can we change a commit message in GIT?

If a commit has not been pushed to GitHub, we can use git commit --ammend command to change the commit message.

When we push the commit, a new message appears on GitHub.

380. Why is it advisable to create an additional commit instead of amending an existing commit?

Git amend internally creates a new commit and replaces the old commit. If commits have already been pushed to central repository, it should not be used to modify the previous commits.

It should be generally used for only amending the git comment.

381. What is a bare repository in GIT?

A repository created with git init –bare command is a bare repository in GIT.

The bare repository does not contain any working or checked out copy of source files. A bare repository stores git revision history in the root folder of repository instead of in a .git subfolder.

It is mainly used for sharing and collaborating with other developers.

We can create a bare repository in which all developers can push their code.

There is no working tree in bare repository, since no one directly edits files in a bare repository.

382. How do we put a local repository on GitHub server?

To put a local repository on GitHub, we first add all the files of working directory into local repository and commit the changes. After that we call git remote add <Remote Repo URL> command to add the local repository on GitHub server.

Once it is added, we use git push command to push the contents of local repository to remote GitHub server.

383. How will you delete a branch in GIT?

We use git branch –d <branchname> command to delete a branch in GIT.

In case a local branch is not fully merged, but we want to delete it by force, then we use git branch –D <branchname> command.

384. How can we set up a Git repository to run code sanity checks and UAT tests just before a commit?

We can use git hooks for this kind of purpose. We can write the code sanity checks in script. This script can be called by pre-commit hook of the repository.

If this hook passes, then only commit will be successful.

385. How can we revert a commit that was pushed earlier and is public now?

We can use git revert command for this purpose.

Internally, git revert command creates a new commit with patches that reverse the changes done in previous commits.

The other option is to checkout a previous commit version and then commit it as a new commit.

386. In GIT, how will you compress last n commits into a single commit?

Tom compress last n commits a single commit, we use git rebase command. This command compresses multiple commits and creates a new commit. It overwrites the history of commits.

It should be done carefully, since it can lead to unexpected results.

387. How will you switch from one branch to a new branch in GIT?

In GIT, we can use git checkout <new branchname> command to switch to a new branch.

388. How can we clean unwanted files from our working directory in GIT?

GIT provides git clean command to recursively clean the working tree. It removes the files that are not under version control in GIT.

If we use git clean –x, then ignored files are also removed.

389. What is the purpose of git tag command?

We use git tag command to add, delete, list or verify a tag object in GIT.

Tag objects created with options –a, -s, -u are also known as annotated tags.

Annotated tags are generally used for release.

390. What is cherry-pick in GIT?

A git cherry-pick is a very useful feature in GIT. By using this command we can selectively apply the changes done by existing commits.

In case we want to selectively release a feature, we can remove the unwanted files and apply only selected commits.

391. What is shortlog in GIT?

A shortlog in GIT is a command that summarizes the git log output.

The output of git shortlog is in a format suitable for release announcements.

392. How can you find the names of files that were changed in a specific commit?

Every commit in GIT has a hash code. This hash code uniquely represents the GIT commit object.

We can use git diff-tree command to list the name of files that were changed in a commit.

The command will be as follows:
git diff-tree -r <hash of commit>

By using -r flag, we just get the list of individual files.

393. How can we attach an automated script to run on the event of a new commit by push command?

In GIT we can use a hook to run an automated script on a specific event. We can choose between pre-receive, update or post-receive hook and attach our script on any of these hooks.

GIT will automatically run the script on the event of any of these hooks.

394. What is the difference between pre-receive, update and post-receive hooks in GIT?

Pre-receive hook is invoked when a commit is pushed to a destination repository. Any script attached to this hook is executed before updating any reference. This is mainly used to enforce development best practices and policies.

Update hook is similar to pre-receive hook. It is triggered just before any updates are done. This hook is invoked once for every commit that is pushed to a destination repository.

Post-receive hook is invoked after the updates have been done and accepted by a destination repository. This is mainly used to configure deployment scripts. It can also invoke Continuous Integration (CI) systems and send notification emails to relevant parties of a repository.

395. Do we have to store Scripts for GIT hooks within same repository?

A Hook is local to a GIT repository. But the script attached to a hook can be created either inside the hooks directory or it can be stored in a separate repository. But we have to link the script to a hook in our local repository.

In this way we can maintain versions of a script in a separate repository, but use them in our repository where hooks are stored.

Also when we store scripts in a separate common repository, we can reuse same scripts for different purposes in multiple repositories.

396. How can we determine the commit that is the source of a bug in GIT?

In GIT we can use git bisect command to find the commit that has introduced a bug in the system.

GIT bisect command internally uses binary search algorithm to find the commit that introduced a bug.

We first tell a bad commit that contains the bug and a good commit that was present before the bug was introduced.

Then git bisect picks a commit between those two endpoints and asks us whether the selected commit is good or bad.

It continues to narrow down the range until it discovers the exact commit responsible for introducing the bug.

397. How can we see differences between two commits in GIT?

We can use git diff command to see the differences between two commits. The syntax for a simple git diff command to compare two commits is:

git diff <commit#1> <commit#2>

398. What are the different ways to identify a commit in GIT?

Each commit object in GIT has a unique hash. This hash is a 40 characters checksum hash. It is based on SHA1 hashing algorithm. We can use a hash to uniquely identify a GIT commit.

Git also provides support for creating an alias for a commit. This alias is known as refs. Every tag in GIT is a ref. These refs can also be used to identify a commit. Some of the special tags in GIT are HEAD, FETCH_HEAD and MERGE_HEAD.

399. When we run git branch <branchname>, how does GIT know the SHA-1 of the last commit?

GIT uses the reference named HEAD for this purpose. The HEAD file in GIT is a symbolic reference to the current branch we are working on.

A symbolic reference is not a normal reference that contains a SHA-1 value. A symbolic reference contains a pointer to another reference.

When we open head file we see:
$ cat .git/HEAD
ref: refs/heads/master

If we run git checkout branchA, Git updates the file to look like this:

$ cat .git/HEAD
ref: refs/heads/branchA

400. What are the different types of Tags you can create in GIT?

In GIT, we can create two types of Tags.

Lightweight Tag: A lightweight tag is a reference that never moves. We can make a lightweight tag by running a command similar to following:

$ git update-ref refs/tags/v1.0 dad0dab538c970e37ea1e769cbbde608743bc96d

Annotated Tag: An annotated tag is more complex object in GIT. When we create an annotated tag, GIT creates a tag object and writes a reference to point to it rather than directly to the commit. We can create an annotated tag as follows:

$ git tag -a v1.1 1d410eabc13591cb07496601ebc7c059dd55bfe9 -m 'test tag'

401. How can we rename a remote repository?

We can use command git remote rename for changing the name of a remote repository. This changes the short name associated with a remote repository in your local. Command would look as follows:

git remote rename repoOldName repoNewName

402. Some people use git checkout and some use git co for checkout. How is that possible?

We can create aliases in GIT for commands by modifying the git configuration.

In case of calling git co instead of git checkout we can run following command:

git config --global alias.co checkout

So the people using git co have made the alias for git checkout in their own environment.

403. How can we see the last commit on each of our branch in GIT?

When we run git branch command, it lists all the branches in our local repository. To see the latest commit associated with each branch, we use option –v.

Exact command for this is as follows:

git branch –v

It lists branches as:

```
  issue75  83b576c fix issue
* master   7b96605 Merge branch 'issue75'
  testing 972ac34 add dave to the developer list
```

404. Is origin a special branch in GIT?

No, origin is not a special branch in GIT.

Branch origin is similar to branch master. It does not have any special meaning in GIT.

Master is the default name for a starting branch when we run git init command.

Origin is the default name for a remote when we run git clone command. If we run git clone -o myOrigin instead, then we will have myOrigin/master as our default remote branch.

405. How can we configure GIT to not ask for password every time?

When we use HTTPS URL to push, the GIT server asks for username and password for authentication. It prompts us on the terminal for this information.

If we don't want to type username/password with every single time push, we can set up a "credential cache".

It is kept in memory for a few minutes. We can set it by running:

git config --global credential.helper cache

406. What are the four major protocols used by GIT for data transfer?

GIT uses following major protocols for data transfer:

1. Local
2. HTTP
3. Secure Shell (SSH)
4. Git

407. What is GIT protocol?

Git protocol is a mechanism for transferring data in GIT. It is a special daemon. It comes pre-packaged with GIT. It listens on a dedicated port 9418. It provides services similar to SSH protocol.

But Git protocol does not support any authentication.

So on plus side, this is a very fast network transfer protocol. But it lacks authentication.

408. How can we work on a project where we do not have push access?

In case of projects where we do not have push access, we can just fork the repository. By running git fork command, GIT will create a personal copy of the repository in our namespace. Once our work is done, we can create a pull request to merge our changes on the real project.

409. What is git grep?

GIT is shipped along with a grep command that allows us to search for a string or regular expression in any committed tree or the working directory.

By default, it works on the files in your current working directory.

410. How can your reorder commits in GIT?

We can use git rebase command to reorder commits in GIT. It can work interactively and you can also select the ordering of commits.

411. How will you split a commit into multiple commits?

To split a commit, we have to use git rebase command in interactive mode. Once we reach the commit that needs to be split, we reset that commit and take the changes that have been reset. Now we can create multiple commits out of that.

412. What is filter-branch in GIT?

In GIT, filter-branch is another option to rewrite history. It can scrub the entire history. When we have large number of commits, we can use this tool.

It gives many options like removing the commit related changes to a specific file from history.

You can even set you name and email in the commit history by using filter-branch.

413. What are the three main trees maintained by GIT?

GIT maintains following three trees:

HEAD: This is the last commit snapshot.
Index: This is the proposed next commit snapshot.
Working Directory: This is the sandbox for doing changes.

414. What are the three main steps of working GIT?

GIT has following three main steps in a simple workflow:

1. Checkout the project from HEAD to Working Directory.
2. Stage the files from Working Directory to Index.
3. Commit the changes from Index to HEAD.

415. What are ours and theirs merge options in GIT?

In GIT, we get two simple options for resolving merge conflicts: ours and theirs

These options tell the GIT which side to favor in merge conflicts.

In ours, we run a command like git merge -Xours branchA

As the name suggests, in ours, the changes in our branch are favored over the other branch during a merge conflict.

416. How can we ignore merge conflicts due to Whitespace?

GIT provides an option ignore-space-change in git merge command to ignore the conflicts related to whitespaces.

The command to do so is as follows:

git merge -Xignore-space-change whitespace

417. What is git blame?

In GIT, git blame is a very good option to find the person who changed a specific line. When we call git blame on a file, it displays the commit and name of a person responsible for making change in that line.

Following is a sample:

```
$ git blame -L 12,19 HelloWorld.java
^1822fe2 (Dave Adams  2016-03-15 10:31:28 -0700 12)  public class HelloWorld {
^1822fe2 (Dave Adams  2016-03-15 10:31:28 -0700 13)
^1822fe2 (Dave Adams  2016-03-15 10:31:28 -0700 14)     public static void main(String[] args) {
af6560e4 (Dave Adams  2016-03-17 21:52:20 -0700 16)    // Prints "Hello, World" to the terminal window.
a9eaf55d (Dave Adams  2016-04-06 10:15:08 -0700 17)    System.out.println("Hello, World");
af6560e4 (Dave Adams  2016-03-17 21:52:20 -0700 18)    }
af6560e4 (Dave Adams  2016-03-17 21:52:20 -0700 19) }
```

418. What is a submodule in GIT?

In GIT, we can create sub modules inside a repository by using git submodule command.

By using submodule command, we can keep a Git repository as a subdirectory of another Git repository.

It allows us to keep our commits to submodule separate from the commits to main Git repository.

AWS

419. What do you know about AWS Region?

An AWS Region is a completely independent entity in a geographical area. There are two more Availability Zones in an AWS Region.

Within a region, Availability Zones are connected through low-latency links.

Since each AWS Region is isolated from another Region, it provides very high fault tolerance and stability.

For launching an EC2 instance, we have to select an AMI within the same region.

420. What are the important components of IAM?

The important components of IAM are as follows:

1. **IAM User**: An IAM User is a person or service that will interact with AWS. User can sign into AWS Management Console for performing tasks in AWS.

2. **IAM Group**: An IAM Group is a collection of IAM users. We can specify permission to an IAM Group. This helps in managing large number of IAM users. We can simply add or remove an IAM User to an IAM Group to manage the permissions.

3. **IAM Role**: An IAM Role is an identity to which we give permissions. A Role does not have any credentials (password or access keys). We can temporarily give an IAM Role to an IAM User to perform certain tasks in AWS.

4. **IAM Permission:** In IAM we can create two types of Permissions. Identity based and Resource based. We can create a Permission to access or perform an action on an AWS Resource and assign it to a User, Role or Group. We can also create Permissions on resources like S3 bucket, Glacier vault etc and specify who has access to the resource.

5. **IAM Policy**: An IAM Policy is a document in which we list permissions to specify Actions, Resources and Effects. This document is in JSON format. We can attach a Policy to an IAM User or Group.

421. What are the important points about AWS IAM?

Some of the important points about AWS IAM are as follows:

1. A new User in IAM does not have any permission.
2. AWS IAM assigns an Access Key and a Secret Access Key to a new User.
3. An Access Key cannot be used to login to AWS Console.
4. We use Access Key to access AWS via an APIs or Command Line interface.
5. IAM is a universal application. It is common across all the regions in AWS.
6. When we first setup our AWS account, we get a root account that has complete Admin access.

422. What are the important features of Amazon S3?

Some of the important features of Amazon S3 are as follows:

1. Amazon S3 provides unlimited storage for files.
2. File size in Amazon S3 can vary from 0 Bytes to 5 Terabytes.
3. We have store files in Buckets in Amazon S3.
4. In Amazon S3, names of buckets have to be unique globally.
5. Amazon S3 is Object Based storage.

423. What is the scale of durability in Amazon S3?

Amazon S3 supports durability at the scale of 99.999999999% of time. This is 9 nines after decimal.

424. What are the Consistency levels supported by Amazon S3?

Amazon S3 supports Read after Write consistency when we create a new object by PUT. It means as soon as we Write a new object, we can access it.

Amazon S3 supports Eventual Consistency when we overwrite an existing object by PUT. Eventual Consistency means that the effect of overwrite will not be immediate but will happen after some time.

For deletion of an object, Amazon S3 supports Eventual Consistency after DELETE.

425. What are the different tiers in Amazon S3 storage?

Different Storage tiers in Amazon S3 are as follows:

1. **S3 Standard**: In this tier, S3 supports durable storage of files that become immediately available. This is used for frequently used files.

2. **S3 Standard -Infrequent Access (IA):** In this tier, S3 provides durable storage that is immediately available. But in this tier files are infrequently accessed.

3. **S3 Reduced Redundancy Storage (RRS)**: In this tier, S3 provides the option to customers to store data at lower levels of redundancy. In this case data is copied to multiple locations but not on as many locations as standard S3.

426. How will you upload a file greater than 100 megabytes in Amazon S3?

Amazon S3 supports storing objects or files up to 5 terabytes. To upload a file greater than 100 megabytes, we have to use Multipart upload utility from AWS. By using Multipart upload we can upload a large file in multiple parts.

Each part will be independently uploaded. It doesn't matter in what order each part is uploaded. It even supports uploading these parts in parallel to decrease overall time. Once all the parts are uploaded, this utility makes these as one single object or file from which the parts were created.

427. What happens to an Object when we delete it from Amazon S3?

Amazon S3 provides DELETE API to delete an object.

If the bucket in which the object exists is version controlled, then we can specify the version of the object that we want to delete. The other versions of the Object still exist within the bucket.

If we do not specify the version, and just pass the key name, Amazon S3 will delete the object and return the version id. And the object will not appear on the bucket.

In case the bucket is Multi-factor authentication (MFA) enabled, then the DELETE request will fail if we do not specify a MFA token.

428. What is the use of Amazon Glacier?

Amazon Glacier is an extremely low cost cloud based storage service provided by Amazon.

We mainly use Amazon Glacier for long-term backup purpose.

Amazon Glacier can be used for storing data archives for months, years or even decades.

It can also be used for long term immutable storage based on regulatory and archiving requirements. It provides Vault Lock support for this purpose. In this option, we write once but can read many times same data.

One use case is for storing certificates that can be issued only once and only the original person keeps the main copy.

429. Can we disable versioning on a version-enabled bucket in Amazon S3?

No, we cannot disable versioning on a version-enabled bucket in Amazon S3. We can just suspend the versioning on a bucket in S3.

Once we suspend versioning, Amazon S3 will stop creating new versions of the object. It just stores the object with null version ID.

On overwriting an existing object, it just replaces the object with null version ID. So any existing versions of the object still remain in the bucket. But there will be no more new versions of the same object except for the null version ID object.

430. What are the use cases of Cross Region Replication Amazon S3?

We can use Cross Region Replication Amazon S3 to make copies of an object across buckets in different AWS Regions. This copying takes place automatically and in an asynchronous mode.

We have to add replication configuration on our source bucket in S3 to make use of Cross Region Replication. It will create exact replicas of the objects from source bucket to destination buckets in different regions.

Some of the main use cases of Cross Region Replication are as follows:

1. **Compliance**: Some times there are laws/regulatory requirements that ask for storing data at farther geographic locations. This kind of compliance can be achieved by using AWS Regions that are spread across the world.

2. **Failover**: At times, we want to minimize the probability of system failure due to complete blackout in a region. We can use Cross-Region Replication in such a scenario.
3. **Latency**: In case we are serving multiple geographies, it makes sense to replicate objects in the geographical Regions that are closer to end customer. This helps in reducing the latency.

431. Can we do Cross Region replication in Amazon S3 without enabling versioning on a bucket?

No, we have to enable versioning on a bucket to perform Cross Region Replication.

432. What are the different types of actions in Object Lifecycle Management in Amazon S3?

There are mainly two types of Object Lifecycle Management actions in Amazon S3.

1. **Transition Actions**: These actions define the state when an Object transitions from one storage class to another storage class. E.g. a new object may transition to STANDARD_IA (infrequent access) class after 60 days of creation. And it can transition to GLACIER after 180 days of creation.

2. **Expiration Actions**: These actions specify what happens when an Object expires. We can ask S3 to delete an object completely on expiration.

433. How do we get higher performance in our application by using Amazon CloudFront?

If our application is content rich and used across multiple locations, we can use Amazon CloudFront to increase its performance. Some of the techniques used by Amazon CloudFront are as follows:

Caching: Amazon CloudFront caches the copies of our application's content at locations closer to our viewers. By this caching our users get our content very fast. Also due to caching the load on our main server decreases.

Edge / Regional Locations: CloudFront uses a global network of Edge and Regional edge locations to cache our content. These locations cater to almost all of the geographical areas across the world.

Persistent Connections: In certain cases, CloudFront keeps persistent connections with the main server to fetch the content quickly.

Other Optimization: Amazon CloudFront also uses other optimization techniques like TCP initial congestion window etc to deliver high performance experience.

434. What is the mechanism behind Regional Edge Cache in Amazon CloudFront?

A Regional Edge Cache location lies between the main webserver and the global edge location. When the popularity of an object/content decreases, the global edge location may take it out from the cache.

But Regional Edge location maintains a larger cache. Due to this the object/content can stay for long time in Regional Edge location. Due to this CloudFront does not have to go back to main webserver. When it does not find any object in Global Edge location it just looks for in Regional Edge location.

This improves the performance for serving content to our users in Amazon CloudFront.

435. What are the benefits of Streaming content?

We can get following benefits by Streaming content:

1. **Control**: We can provide more control to our users for what they want to watch. In a video streaming, users can select the locations in video where they want to start watching from.

2. **Content**: With streaming our entire content does not stay at a user's device. Users gets only the part they are watching. Once the session is over, content is removed from the user's device.

3. **Cost**: With streaming there is no need to download all the content to a user's device. A user can start viewing content as soon as some part is available for viewing. This saves costs since we do not have to download a large media file before starting each viewing session.

436. What is Lambda@Edge in AWS?

In AWS, we can use Lambda@Edge utility to solve the problem of low network latency for end users.

In Lambda@Edge there is no need to provision or manage servers. We can just upload our Node.js code to AWS Lambda and create functions that will be triggered on CloudFront requests.

When a request for content is received by CloudFront edge location, the Lambda code is ready to execute.

This is a very good option for scaling up the operations in CloudFront without managing servers.

437. What are the different types of events triggered by Amazon CloudFront?

Different types of events triggered by Amazon CloudFront are as follows:

1. **Viewer Request**: When an end user or a client program makes an HTTP/HTTPS request to CloudFront, this event is triggered at the Edge Location closer to the end user.
2. **Viewer Response**: When a CloudFront server is ready to respond to a request, this event is triggered.
3. **Origin Request**: When CloudFront server does not have the requested object in its cache, the request is forwarded to Origin server. At this time this event is triggered.
4. **Origin Response**: When CloudFront server at an Edge location receives the response from Origin server, this event is triggered.

438. What is Geo Targeting in Amazon CloudFront?

In Amazon CloudFront we can detect the country from where end users are requesting our content. This information can be passed to our Origin server by Amazon CloudFront. It is sent in a new HTTP header.

Based on different countries we can generate different content for different versions of the same content. These versions can be cached at different Edge Locations that are closer to the end users of that country.

In this way we are able to target our end users based on their geographic locations.

439. What are the main features of Amazon CloudFront?

Some of the main features of Amazon CloudFront are as follows:

1. Device Detection
2. Protocol Detection
3. Geo Targeting
4. Cache Behavior
5. Cross Origin Resource Sharing
6. Multiple Origin Servers
7. HTTP Cookies
8. Query String Parameters
9. Custom SSL

440. What are the security mechanisms available in Amazon S3?

Amazon S3 is a very secure storage service. Some of the main security mechanisms available in Amazon S3 are as follows:

1. **Access**: When we create a bucket or an object, only the owner get the access to the bucket and objects.

2. **Authentication**: Amazon S3 also support user authentication to control who has access to a specific object or bucket.

3. **Access Control List**: We can create Access Control Lists (ACL) to provide selective permissions to users and groups.

4. **HTTPS**: Amazon S3 also supports HTTPS protocol to securely upload and download data from cloud.

5. **Encryption**: We can also use Server Side Encryption (SSE) in Amazon S3 to encrypt data.

Cloud Computing

441. What are the benefits of Cloud Computing?

There are ten main benefits of Cloud Computing:

Flexibility: The businesses that have fluctuating bandwidth demands need the flexibility of Cloud Computing. If you need high bandwidth, you can scale up your cloud capacity. When you do not need high bandwidth, you can just scale down. There is no need to be tied into an inflexible fixed capacity infrastructure.

Disaster Recovery: Cloud Computing provides robust backup and recovery solutions that are hosted in cloud. Due to this there is no need to spend extra resources on homegrown disaster recovery. It also saves time in setting up disaster recovery.

Automatic Software Updates: Most of the Cloud providers give automatic software updates. This reduces the extra task of installing new software version and always catching up with the latest software installs.

Low Capital Expenditure: In Cloud computing the model is Pay as you Go. This means there is very less upfront capital expenditure. There is a variable payment that is based on the usage.

Collaboration: In a cloud environment, applications can be shared between teams. This increases collaboration and communication among team members.

Remote Work: Cloud solutions provide flexibility of working remotely. There is no on site work. One can just connect from anywhere and start working.

Security: Cloud computing solutions are more secure than regular onsite work. Data stored in local servers and computers is prone to security attacks. In Cloud Computing, there are very few loose ends. Cloud providers give a secure working environment to its users.

Document Control: Once the documents are stored in a common repository, it increases the visibility and transparency among companies and their clients. Since there is one shared copy, there are fewer chances of discrepancies.

Competitive Pricing: In Cloud computing there are multiple players, so they keep competing among themselves and provide very good pricing. This comes out much cheaper compared to other options.

Environment Friendly: Cloud computing saves precious environmental resources also. By not blocking the resources and bandwidth.

442. What is On-demand computing in Cloud Computing?

On-demand Computing is the latest model in enterprise systems. It is related to Cloud computing. It means IT resources can be provided on demand by a Cloud provider.

In an enterprise system demand for computing resources varies from time to time. In such a scenario, On-demand computing makes sure that servers and IT resources are provisioned to handle the increase/decrease in demand.

A cloud provider maintains a poll of resources. The pool of resources contains networks, servers, storage, applications and services. This pool can serve the varying demand of resources and computing by various enterprise clients.

There are many concepts like- grid computing, utility computing, autonomic computing etc. that are similar to on-demand computing.

This is the most popular trend in computing model as of now.

443. What are the different layers of Cloud computing?

Three main layers of Cloud computing are as follows:

Infrastructure as a Service (IAAS): IAAS providers give low-level abstractions of physical devices. Amazon Web Services (AWS) is an example of IAAS. AWS provides EC2 for computing, S3 buckets for storage etc. Mainly the resources in this layer are hardware like memory, processor speed, network bandwidth etc.

Platform as a Service (PAAS): PAAS providers offer managed services like Rails, Django etc. One good example of PAAS is Google App Engineer. These are the environments in which developers can develop sophisticated software with ease.

Developers just focus on developing software, whereas scaling and performance is handled by PAAS provider.

Software as a Service (SAAS): SAAS provider offer an actual working software application to clients. Salesforce and Github are two good examples of SAAS. They hide the underlying details of the software and just provide an interface to work on the system. Behind the scenes the version of Software can be easily changed.

444. What resources are provided by Infrastructure as a Service (IAAS) provider?

An IAAS provider can give physical, virtual or both kinds of resources. These resources are used to build cloud.

IAAS provider handles the complexity of maintaining and deploying these services.

IAAS provider also handles security and backup recovery for these services. The main resources in IAAS are servers, storage, routers, switches and other related hardware etc.

445. What is the benefit of Platform as a Service?

Platform as a service (PaaS) is a kind of cloud computing service. A PaaS provider offers a platform on which clients can develop, run and manage applications without the need of building the infrastructure.

In PAAS clients save time by not creating and managing infrastructure environment associated with the app that they want to develop.

446. What are the main advantages of PaaS?

The advantages of PaaS are:

It allows development work on higher level programming with very less complexity.

Teams can focus on just the development of the application that makes the application very effective.

Maintenance and enhancement of the application is much easier.

It is suitable for situations in which multiple developers work on a single project but are not co-located.

447. What is the main disadvantage of PaaS?

Biggest disadvantage of PaaS is that a developer can only use the tools that PaaS provider makes available. A developer cannot use the full range of conventional tools.

Some PaaS providers lock in the clients in their platform. This also decreases the flexibility of clients using PaaS.

448. What are the different deployment models in Cloud computing?

Cloud computing supports following deployment models:

Private Cloud: Some companies build their private cloud. A private cloud is a fully functional platform that is owned, operated and used by only one organization.

Primary reason for private cloud is security. Many companies feel secure in private cloud. The other reasons for building private cloud are strategic decisions or control of operations.

There is also a concept of Virtual Private Cloud (VPC). In VPC, private cloud is built and operated by a hosting company. But it is exclusively used by one organization.

Public Cloud: There are cloud platforms by some companies that are open for general public as well as big companies for use and deployment. E.g. Google Apps, Amazon Web Services etc.

The public cloud providers focus on layers and application like- cloud application, infrastructure management etc. In this model resources are shared among different organizations.

Hybrid Cloud: The combination of public and private cloud is known as Hybrid cloud. This approach provides benefits of both the approaches- private and public cloud. So it is very robust platform.

A client gets functionalities and features of both the cloud platforms. By using Hybrid cloud an organization can create its own cloud as well as they can pass the control of their cloud to another third party.

449. What is the difference between Scalability and Elasticity?

Scalability is the ability of a system to handle the increased load on its current hardware and software resources. In a highly scalable system it is possible to increase the workload without increasing the resource capacity. Scalability supports any sudden surge in the demand/traffic with current set of resources.

Elasticity is the ability of a system to increase the workload by increasing the hardware/software resources dynamically. Highly elastic systems can handle the increased demand and traffic by dynamically commission and decommission resources. Elasticity is an important characteristic of Cloud Computing applications. Elasticity means how well your architecture is adaptable to workload in real time.

E.g. If in a system, one server can handle 100 users, 2 servers can handle 200 users and 10 servers can handle 1000 users. But in case for adding every X users, if you need 2X the amount of servers, then it is not a scalable design.

Let say, you have just one user login every hour on your site. Your one server can handle this load. But, if suddenly, 1000 users login at once, can your system quickly start new web servers on the fly to handle this load? Your design is elastic if it can handle such sudden increase in traffic so quickly.

450. What is Software as a Service?

Software as Service is a category of cloud computing in which Software is centrally hosted and it is licensed on a subscription basis. It is also known as On-demand software. Generally, clients access the software by using a thin-client like a web browser.

Many applications like Google docs, Microsoft office etc. provide SaaS model for their software.

The benefit of SaaS is that a client can add more users on the fly based on its current needs. And client does not need to install or maintain any software on its premises to use this software.

451. What are the different types of Datacenters in Cloud computing?

Cloud computing consists of different types of Datacenters linked in a grid structure. The main types of Datacenters in Cloud computing are:

Containerized Datacenter

As the name suggests, containerized datacenter provides high level of customization for an organization. These are traditional kind of datacenters. We can choose the different types of servers, memory, network and other infrastructure resources in this datacenter. Also we have to plan temperature control, network management and power management in this kind of datacenter.

Low-Density Datacenters

In a Low-density datacenter, we get high level of performance. In such a datacenter if we increase the density of servers, the issue with power comes. With high density of servers, the area gets heated. In such a scenario, effective heat and power management is done. To reach high level of performance, we have to optimize the number of servers' in the datacenter.

452. Explain the various modes of Software as a Service (SaaS) cloud environment?

Software as a Service (SaaS) is used to offer different kinds of software applications in a Cloud environment. Generally these are offered on subscription basis. Different modes of SaaS are:

Simple multi-tenancy: In this setup, each client gets its own resources. These resources are not shared with other clients. It is more secure option, since there is no sharing of resources. But it an inefficient option, since for each client more money is needed to scale it with the rising demands. Also it takes time to scale up the application in this mode.

Fine grain multi-tenancy: In this mode, the feature provided to each client is same. The resources are shared among multiple clients. It is an efficient mode of cloud service, in which data is kept private among different clients but computing resources are shared. Also it is easier and quicker to scale up the SaaS implementation for different clients.

453. What are the important things to care about in Security in a cloud environment?

In a cloud-computing environment, security is one of the most important aspects.

With growing concern of hacking, every organization wants to make its software system and data secure. Since in a cloud computing environment, Software and hardware is not on the premises of an organization, it becomes more important to implement the best security practices.

Organizations have to keep their Data most secure during the transfer between two locations. Also they have to keep data secure when it is stored at a location. Hackers can hack into application or they can get an unauthorized copy of the data. So it becomes important to encrypt the data during transit as well as during rest to protect it from unwanted hackers.

454. Why do we use API in cloud computing environment?

Application Programming Interfaces (API) is used in cloud computing environment for accessing many services. APIs are very easy to use. They provide a quick option to create different set of applications in cloud environment.

An API provides a simple interface that can be used in multiple scenarios.

There are different types of clients for cloud computing APIs. It is easier to serve different needs of multiple clients with APIs in cloud computing environment.

455. What are the different areas of Security Management in cloud?

Different areas of Security management in cloud are as follows:

Identity Management: This aspect creates different level of users, roles and their credentials to access the services in cloud.

Access Control: In this area, we create multiple levels of permissions and access areas that can be given to a user or role for accessing a service in cloud environment.

Authentication: In this area, we check the credentials of a user and confirm that it is the correct user. Generally this is done by user password and multi-factor authentication like-verification by a one-time use code on cell phone.

Authorization: In this aspect, we check for the permissions that are given to a user or role. If a user is authorized to access a service, they are allowed to use it in the cloud environment.

456. What are the main cost factors of cloud based data center?

Costs in a Cloud based data center are different from a traditional data center. Main cost factors of cloud based data center are as follows:

Labor cost: We need skilled staff that can work with the cloud-based datacenter that we have selected for our operation. Since cloud is not a very old technology, it may get difficult to get the right skill people for handling cloud based datacenter.

Power cost: In some cloud operations, power costs are borne by the client. Since it is a variable cost, it can increase with the increase in scale and usage.

Computing cost: The biggest cost in Cloud environment is the cost that we pay to Cloud provider for giving us computing resources. This cost is much higher compared to the labor or power costs.

457. How can we measure the cloud-based services?

In a cloud-computing environment we pay for the services that we use. So main criteria to measure a cloud based service its usage.

For computing resource we measure by usage in terms of time and the power of computing resource.

For a storage resource we measure by usage in terms of bytes (giga bytes) and bandwidth used in data transfer.

Another important aspect of measuring a cloud service is its availability. A cloud provider has to specify the service level agreement (SLA) for the time for which service will be available in cloud.

458. How a traditional datacenter is different from a cloud environment?

In a traditional datacenter the cost of increasing the scale of computing environment is much higher than a Cloud computing environment. Also in a traditional data center, there are not much benefits of scaling down the operation when demand decreases. Since most of the expenditure is in capital spent of buying servers etc., scaling down just saves power cost, which is very less compared to other fixed costs.

Also in a Cloud environment there is no need to higher a large number of operations staff to maintain the datacenter. Cloud provider takes care of maintaining and upgrading the resources in Cloud environment.

With a traditional datacenter, people cost is very high since we have to hire a large number of technical operation people for in-house datacenter.

459. How will you optimize availability of your application in a Cloud environment?

In a Cloud environment, it is important to optimize the availability of an application by implementing disaster recovery strategy. For disaster recovery we create a backup application in another location of cloud environment. In case of complete failure at a data center we use the disaster recovery site to run the application.

Another aspect of cloud environment is that servers often fail or go down. In such a scenario it is important to implement the application in such a way that we just kill the slow server and restart another server to handle the traffic seamlessly.

460. What are the requirements for implementing IaaS strategy in Cloud?

Main requirements to implement IAAS are as follows:

Operating System (OS): We need an OS to support hypervisor in IaaS. We can use open source OS like Linux for this purpose.

Networking: We have to define and implement networking topology for IaaS implementation. We can use public or private network for this.

Cloud Model: We have to select the right cloud model for implementing IaaS strategy. It can be SaaS, PaaS or CaaS.

DOCKER

461. What is Docker?

Docker is Open Source software. It provides the automation of Linux application deployment in a software container.

We can do operating system level virtualization on Linux with Docker.

Docker can package software in a complete file system that contains software code, runtime environment, system tools, & libraries that are required to install and run the software on a server.

462. What is the difference between Docker image and Docker container?

Docker container is simply an instance of Docker image.

A Docker image is an immutable file, which is a snapshot of container. We create an image with **build** command.

When we use run command, an Image will produce a container.

In programming language, an Image is a Class and a Container is an instance of the class.

463. How will you remove an image from Docker?

We can use docker rmi command to delete an image from our local system.

Exact command is:

% docker rmi <Image Id>

If we want to find IDs of all the Docker images in our local system, we can user docker images command.

% docker images

If we want to remove a docker container then we use docker rm command.

% docker rm <Container Id>

464. How is a Docker container different from a hypervisor?

In a Hypervisor environment we first create a Virtual Machine and then install an Operating System on it. After that we deploy the application. The virtual machine may also be installed on different hardware configurations.

In a Docker environment, we just deploy the application in Docker. There is no OS layer in this environment. We specify libraries, and rest of the kernel is provided by Docker engine.

In a way, Docker container and hypervisor are complementary to each other.

465. Can we write compose file in json file instead of yaml?

Yes. Yaml format is a superset of json format. Therefore any json file is also a valid Yaml file.

If we use a json file then we have to specify in docker command that we are using a json file as follows:

% docker-compose -f docker-compose.json up

466. Can we run multiple apps on one server with Docker?

Yes, theoretically we can run multiples apps on one Docker server. But in practice, it is better to run different components on separate containers.

With this we get cleaner environment and it can be used for multiple uses.

467. What are the common use cases of Docker?

Some of the common use cases of Docker are as follows:

1. **Setting up Development Environment**: We can use Docker to set the development environment with the applications on which our code is dependent.
2. **Testing Automation Setup**: Docker can also help in creating the Testing Automation setup. We can setup different services and apps with Docker to create the automation testing environment.
3. **Production Deployment**: Docker also helps in implementing the Production deployment for an application. We can use it to create the exact environment and process that will be used for doing the production deployment.

468. What are the main features of Docker-compose?

Some of the main features of Docker-compose are as follows:

1. **Multiple environments on same Host**: We can use it to create multiple environments on the same host server.
2. **Preserve Volume Data on Container Creation**: Docker compose also preserves the volume data when we create a container.
3. **Recreate the changed Containers**: We can also use compose to recreate the changed containers.
4. **Variables in Compose file**: Docker compose also supports variables in compose file. In this way we can create variations of our containers.

469. What is the most popular use of Docker?

The most popular use of Docker is in build pipeline. With the use of Docker it is much easier to automate the development to deployment process in build pipeline.

We use Docker for the complete build flow from development work, test run and deployment to production environment.

470. What is the role of open source development in the popularity of Docker?

Since Linux was an open source operating system, it opened new opportunities for developers who want to contribute to open source systems.

One of the very good outcomes of open source software is Docker. It has very powerful features.

Docker has wide acceptance due to its usability as well as its open source approach of integrating with different systems.

UNIX Shell

471. How will you remove all files in current directory? Including the files that are two levels down in a sub-directory.

In Unix we have rm command to remove files and sub-directories. With rm command we have –r option that stands for recursive. The –r option can delete all files in a directory recursively.

It means if we our current directory structure is as follows:

My_dir
->Level_1_dir
-> Level_1_dir ->Level_2_dir
-> Level_1_dir ->Level_2_dir->a.txt

With rm –r * command we can delete the file a.txt as well as sub-directories Level_1_dir and Level_2_dir.

Command:
rm – r *

The asterisk (*) is a wild card character that stands for all the files with any name.

472. What is the difference between the –v and –x options in Bash shell scripts?

In a BASH Unix shell we can specify the options –v and –x on top of a script as follows:

#!/bin/bash -x –v

With –x option BASH shell will echo the commands like for, select, case etc. after substituting the arguments and variables. So it will be an expanded form of the command that shows all the actions of the script. It is very useful for debugging a shell script.

With –v option BASH shell will echo every command before substituting the values of arguments and variables. In –v option Unix will print each line as it reads.

In –v option, If we run the script, the shell prints the entire file and then executes. If we run the script interactively, it shows each command after pressing enter.

473. What is a Filter in Unix command?

In Unix there are many Filter commands like- cat, awk, grep, head, tail cut etc.

A Filter is a software program that takes an input and produces an output, and it can be used in a stream operation.

E.g. cut -d : -f 2 /etc/passwd | grep abc

We can mix and match multiple filters to create a complex command that can solve a problem.

Awk and Sed are complex filters that provide fully programmable features.

Even Data scientists use Unix filters to get the overview of data stored in the files.

474. What is Kernel in Unix operating system?

Kernel is the central core component of a Unix operating system (OS).

A Kernel is the main component that can control everything within Unix OS.

It is the first program that is loaded on startup of Unix OS. Once it is loaded it will manage the rest of the startup process.

Kernel manages memory, scheduling as well as communication with peripherals like printers, keyboards etc.

But Kernel does not directly interact with a user. For a new task, Kernel will spawn a shell and user will work in a shell.

Kernel provides many system calls. A software program interacts with Kernel by using system calls.

Kernel has a protected memory area that cannot be overwritten accidentally by any process.

475. What is a Shell in Unix OS?

Shell in Unix is a user interface that is used by a user to access Unix services.

Generally a Unix Shell is a command line interface (CLI) in which users enter commands by typing or uploading a file.

We use a Shell to run different commands and programs on Unix operating system.

A Shell also has a command interpreter that can take our commands and send these to be executed by Unix operating system.

Some of the popular Shells on Unix are: Korn shell, BASH, C shell etc.

476. What are the different shells in Unix that you know about?

Unix has many flavors of Shell. Some of these are as follows:

Bourne shell: We use sh for Bourne shell.
Bourne Again shell: We use bash to run this shell.
Korn shell: We can use ksh to for Korn shell.
Z shell: The command to use this is zsh
C shell: We use csh to run C shell.
Enhanced C shell: tcsh is the command for enhanced C shell.

477.
What is the first character of the outpu in ls –l command ?

We use ls -l command to list the files and directories in a directory. With -l option we get long listing format.

In this format the first character identifies the entry type. The entry type can be one of the following:

b	Block special file
c	Character special file
d	Directory
l	Symbolic link
s	Socket link
p	FIFO
-	Regular file

In general we see d for directory and - for a regular file.

478. What is the difference between Multi-tasking and Multi-user environment?

In a Multi-tasking environment, same user can submit more than one tasks and operating system will execute them at the same time.

In a Multi-user environment, more than one user can interact with the operating system at the same time.

479. What is Command Substitution in Unix?

Command substitution is a mechanism by which Shell passes the output of a command as an argument to another command. We can even use it to set a variable or use an argument list in a for loop.

E.g. rm `cat files_to_delete`
In this example files_to_delete is a file containing the list of files to be deleted. cat command outputs this file and gives the output to rm command. rm command deletes the files.

In general Command Substitution is represented by back quotes `.

480. What is an Inode in Unix?

An Inode is a Data Structure in Unix that denotes a file or a directory on file system. It contains information about file like- location of file on the disk, access mode, ownership, file type etc.

Each Inode has a number that is used in the index table. Unix kernel uses Inode number to access the contents of an Inode.

We can use ls -i command to get the inode number of a file.

481. What is the difference between absolute path and relative path in Unix file system?

Absolute path is the complete path of a file or directory from the root directory. In general root directory is represented by / symbol. If we are in a directory and want to know the absolute path, we can use pwd command.

Relative path is the path relative the current location in directory.

E.g. In a directory structure /var/user/kevin/mail if we are in kevin directory then pwd command will give absolute path as /var/user/kevin.

Absolute path of mail folder is /var/user/kevin/mail. For mail folder ./mail is the relative path of mail directory from kevin folder.

482. What are the main responsibilities of a Unix Shell?

Some of the main responsibilities of a Unix Shell are as follows:

1. Program Execution: A shell is responsible for executing the commands and script files in Unix. User can either interactively enter the commands in Command Line Interface called terminal or they can run a script file containing a program.

2. Environment Setup: A shell can define the environment for a user. We can set many environment variables in a shell and use the value of these variables in our program.

3. Interpreter: A shell acts as an interpreter for our scripts. It has a built in programming language that can be used to implement the logic.

4. Pipeline: A shell also can hookup a pipeline of commands. When we run multiple commands separated by | pipe character, the shell takes the output of a command and passes it to next one in the pipeline.

5. I/O Redirection: Shell is also responsible for taking input from command line interface (CLI) and sending the output back to CLI. We use >, <, >> characters for this purpose.

483. What is a Shell variable?

A Unix Shell variable is an internal variable that a shell maintains. It is local to that Shell. It is not made available to the parent shell or child shell.

We generally use lower case names for shell variables in C shell.

We can set the value of a shell variable by set command.

E.g. % set max_threads=10

To delete a Shell variable we can use unset command.

To use a Shell variable in a script we use $ sign in front of the variable name.

E.g. echo $max_threads

Microservices

484. What is a Microservice?

A Microservice is a small and autonomous piece of code that does one thing very well. It is focused on doing well one specific task in a big system.
It is also an autonomous entity that can be designed, developed and deployed independently.

Generally, it is implemented as a REST service on HTTP protocol, with technology-agnostic APIs.
Ideally, it does not share database with any other service.

485. What are the benefits of Microservices architecture?

Microservices provide many benefits. Some of the key benefits are:

1. Scaling: Since there are multiple Microservices instead of one monolith, it is easier to scale up the service that is being used more. Eg. Let say, you have a Product Lookup service and Product Buy service. The frequency of Product Lookup is much higher than Product Buy service. In this case, you can just scale up the Product Lookup service to run on powerful hardware with multiple servers. Meanwhile, Product Buy service can remain on less powerful hardware.
2. Resilience: In Microservice architecture, if your one service goes down, it may not affect the rest of the system. The other parts can keep functioning, business as usual (BAU). Eg. Let say, you have Product Recommendation service and Product Buy service. If Product Recommendation service goes down, the Product Buy service can still keep running.
3. Technology Mix: With so many changes in technology everyday, you can keep using the latest technology for your new Microservices. You can adopt new technologies with less risk compared to Monolithic architecture. This is one of the best benefits of Microservices architecture.
4. Reuse: Microservices help you in reusing the lessons learnt from one service to another.
5. Easy Deployment: Microservices architecture, if done correctly, helps in making the deployment process smooth. If anything goes wrong, it can be rolled back easily and quickly in Microservices.

486. What is the role of architect in Microservices architecture?

Architects, in Microservices architecture, play the role of Town planners. They decide in broad strokes about the layout of the overall software system.

They help in deciding the zoning of the components. They make sure components are mutually cohesive but not tightly coupled. They need not worry about what is inside each zone.

Since they have to remain up to date with the new developments and problems, they have to code with developers to learn the challenges faced in day-to-day life.

They can make recommendations for certain tools and technologies, but the team developing a micro service is ultimately empowered to create and design the service. Remember, a micro service implementation can change with time.

They have to provide technical governance so that the teams in their technical development follow principles of Microservice.

At times they work as custodians of overall Microservices architecture.

487. What is the advantage of Microservices architecture over Service Oriented Architecture (SOA)?

Service Oriented Architecture (SOA) is an approach to develop software by creating multiple services. It creates small parts of services and promotes reusability of software. But SOA development can be slow due to use of things like communication protocols SOAP, middleware and lack of principles.

On the other hand, Microservices are agnostic to most of these things. You can use any technology stack, any hardware/middleware, any protocol etc. as long as you follow the principles of Microservices.

Microservices architecture also provides more flexibility, stability and speed of development over SOA architecture.

488. Is it a good idea to provide a Tailored Service Template for Microservices development in an organization?

If your organization is using similar set of technologies, then it is a good idea to provide a Service Template that can be tailored by development teams. It can make development faster. Also it can help in promoting adoption of various good practices that are already built into template.

But if your organization uses wide variety of technologies, then it may not be wise to produce and maintain a template for each service. Instead of that, it is better to introduce tools that help in maintaining same set of practices related to Microservices among all such technologies.

There are many organizations that provide tailored templates for Microservices. Eg. Dropwizard, Karyon etc. You can use these templates to make faster development of services in your organization.

Also remember that template code should not promote shared code. This can lead to tight coupling between Microservices.

489. What are the disadvantages of using Shared libraries approach to decompose a monolith application?

You can create shared libraries to increase reuse and sharing of features among teams. But there are some downsides to it.
Since shared libraries are implemented in same language, it constrains you from using multiple types of technologies.
It does not help you with scaling the parts of system that need better performance.
Deployment of shared libraries is same as deployment of Monolith application, so it comes with same deployment issues.
Shared libraries introduce shared code that can increase coupling in software.

490. What are the characteristics of a Good Microservice?

Good Microservices have these characteristics:

1. Loose coupling: A Microservice knows little about any other service. It is as much independent as possible. The change made in one Microservice does not require changes in other Microservices.
2. Highly cohesive: Microservices are highly cohesive so that each one of them can provide one set of behavior independently.
3. Bounded Context: A Microservice serves a bounded context in a domain and communicates with rest of the domain by using an interface for that Bounded context.
4. Business Capability: Microservices individually add business capability that is part of big picture in organization.

491. What is Bounded Context?

A bounded context is like a specific responsibility that is developed within a boundary. In a domain there can be multiple bounded contexts that are internally implemented. Eg. A hospital system can have bounded contexts like- Emergency Ward handling, Regular vaccination, Out patient treatment etc. Within each bounded context, each sub-system can be independently designed and implemented.

492. What are the points to remember during integration of Microservices?

Some of the important points to remember during integration of Microservices are:
Technology Agnostic APIs: Developing Microservices in a technology agnostic way helps in integration of multiple Microservices. With time, the technology implementation can change but the interface between Microservices can remain same.

Breaking Changes: Every change in Microservice should not become a breaking change for client. It is better to minimize the impact of a change on an existing client. So that existing clients' do not have to keep changing their code to adapt to changes in a Microservice.

Implementation Hiding: Each Microservice should hide its internal implementation details from another one. This helps in minimizing the coupling between Microservices that are integrated for a common solution.

Simple to use: A Microservice should be simple to use for a consumer, so that the integration points are simpler. It should allow clients to choose their own technology stack.

493. Is it a good idea for Microservices to share a common database?

Sharing a common database between multiple Microservices increases coupling between them. One service can start accessing data tables of another service. This can defeat the purpose of bounded context. So it is not a good idea to share a common database between Microservices.

494. What is the preferred type of communication between Microservices? Synchronous or Asynchronous?

Synchronous communication is a blocking call in which client blocks itself from doing anything else, till the response comes back. In Asynchronous communication, client can move ahead with its work after making an asynchronous call. Therefore client is not blocked.

In synchronous communication, a Microservice can provide instant response about success or failure. In real-time systems, synchronous service is very useful. In Asynchronous communication, a service has to react based on the response received in future.

Synchronous systems are also known as request/response based. Asynchronous systems are event-based.

Synchronous Microservices are not loosely coupled.

Depending on the need and critical nature of business domain, Microservices can choose synchronous or asynchronous form of communication.

495. What is the difference between Orchestration and Choreography in Microservices architecture?

In Orchestration, we rely on a central system to control and call various Microservices to complete a task. In Choreography, each Microservice works like a State Machine and reacts based on the input from other parts.

Orchestration is a tightly coupled approach for integrating Microservices. But Choreography introduces loose coupling. Also, Choreography based systems are more flexible and easy to change than Orchestration based systems.

Orchestration is often done by synchronous calls. But choreography is done by asynchronous calls. The synchronous calls are much simpler compared to asynchronous communication.

496. What are the issues in using REST over HTTP for Microservices?

In REST over HTTP, it is difficult to generate a client stub.
Some Web-Servers also do not support all the HTTP verbs like-GET, PUT, POST, DELETE etc.
Due to JSON or plain text in response, performance of REST over HTTP is better than SOAP. But it is not as good as plain binary communication.
There is an overhead of HTTP in each request for communication.
HTTP is not well suited for low-latency communications.
There is more work in consumption of payload. There may be overhead of serialization, deserialization in HTTP.

497. Can we create Microservices as State Machines?

Yes, Microservices are independent entities that serve a specific context. For that context, the Microservice can work as a State Machine. In a State Machine, there are lifecycle events that cause change in the state of the system.
Eg. In a Library service, there is a book that changes state based on different events like- issue a book, return a book, lose a book, late return of a book, add a new book to catalog etc. These events and book can form a state machine for Library Microservice.

THANKS

If you enjoyed this book or gained knowledge from it in any way, then I'd like to ask you for a favor. Would you be kind enough to leave a review for this book on Amazon.com?

It'd be greatly appreciated!

REFERENCES

https://aws.amazon.com

https://www.docker.com/

https://www.nagios.com

https://github.com/