CRACKING JAVA INTERVIEWS

Targeted for investment banks, healthcare IT, product and service based companies

Topics covered in this ebook:

1. Core Java Concepts
2. Concurrency and Java Collections Framework
3. Algorithms, Data Structures and Puzzles
4. Object Oriented Design Problems
5. Spring, Hibernate and REST
6. Sample Interview Questions

Specifically compiled for -
RBS, UBS, Morgan Stanley, JP Morgan, Nomura, Barclays, Citibank, BlackRock, MarkIt, Sapient, Global Logic, Adobe, Goldman Sachs, BOA, hCentive, Expedia, Infosys, TCS, HCL, etc

v3.3, April 2018

About Author
Munish Chandel (मुनीश चंदेल)

Munish is Java developer having 10+ years of experience working for investment banks, healthcare, consulting and product based companies.

cancerian0684@gmail.com
http://linkedIn.com/munish.chandel

Cracking Java Interviews

This book attempts to address common questions faced by interviewee in Indian IT Industry (investment banks, product and service companies)

Topics Covered In This Book

(OOP Concepts, Core Java, Algorithms & Data Structures, Concurrency, Hibernate, Spring and REST)

Preface

This work is my sincere effort to consolidate solutions to some basic set of problems faced by my fellow mates in their day to day work. This work can be used by candidates preparing to brush up their skills for Job change.

This Book Isn't

- A research work, neither it is intended to be.
- Of much help to a fresher in IT industry as it expects some level of hands on experience. It doesn't even cover all the topics required by a newbie to start developing software from scratch.
- A reference book, one time read should be enough.

This Book Is

- Collection of excerpts discussing the common problems faced by experienced Java developers in their day to day work. The intent is not to provide with the concrete solution to a given problem, but to show the approach to get the problem solved. Possibly there could be more efficient ways to solve the given problem compared to what has mentioned in this book. The approach shown here is limited to the knowledge of the author.
- Collection of questions covering Core Java, Object Oriented Design, Concurrency, Algorithms & Data Structures, Spring, Hibernate, REST and few puzzles.

Who should read this book?

- Experienced candidates who want to brush up their skills for Java Interviews specifically in investment banking domain (having approach for enterprise level applications) and product based companies.
- Experienced Java developers who want to enhance their skills to solve their day to day software problems in a better way.

I hope this book adds value to your skills.

Munish Chandel
cancerian0684@gmail.com
https://www.linkedin.com/in/munishchandel
April 2018

How to get PDF copy of this ebook

1. Login to https://books.shunyafoundation.com
2. Show valid proof of purchase through Support Ticket
3. Download PDF copy and start receiving free minor updates to this ebook

eBook from same author

Cracking Microservices Architecture Interview Questions for Java Developers

Buy Now

Contents

Cracking Java Interviews 2

Preface 3

IT Industry and Psyche of Developers 13

Core Concepts, Spring & Hibernate 15

Q 1. What are good software practices for developing Scalable, Testable and Maintainable software? 15
Q 2. What is growth roadmap for a Java developer? 16
Q 3. What are essential skills for Investment Banking domain? 17
Q 4. What are trending skills in Information & Technology? 17
Q 5. What are good books for Java developers? 17
Q 6. Why should one choose Java for Software Development? What are pros and cons of Java? 18
Q 7. What is difference between 32 bit and 64 bit versions of Java? 18
Q 8. What is difference between JDK, JRE and JVM? 19
Q 9. What are four basic principles of Object Oriented Programming? 19
Q 10. What is Aggregation, how is it different from Composition? 20
Q 11. What is role of requirement understanding in software development process? 21
Q 12. What is a Logarithm? Why is it so relevant in algorithms & datastructures? 23
Q 13. What is 2's complement notation system? 24
Q 14. What do you understand by Big O notation, why is it important in algorithms? 25
Q 15. How would you determine Time Complexity of a given algorithm, are there any general guidelines? 26
Q 16. What is a sorting algorithm? List down sorting algorithms by their time & memory complexity in Big O notation? When do we call a sorting algorithm 'Stable'? 27
Q 17. Why Prime Numbers are so important for certain algorithms like RSA & hashcode? 33
Q 18. What is left shift <<, right shift >> and Unsigned right shift >>> operator in Java? How are these useful? 33
Q 19. How heap memory is divided in Java. How does Garbage Collector cleans up the unused Objects? Why shouldn't we use System.gc() command in production code? 35
Q 20. What is difference between Stack and Heap area of JVM Memory? What is stored inside a stack and what goes into heap? 39
Q 21. What is a Binary Tree? What are its usecases? 40
Q 22. Discuss implementation and uses of TreeSet Collection? 40
Q 23. How does Session handling works in Servlet environment? 41
Q 24. Explain Servlet Life Cycle in a Servlet Container? 42
Q 25. How can one handle relative context path while coding the web applications? For example, your web application may be deployed at a different context path in Tomcat, how will you make sure static/dynamic resources works well at custom context path? 43
Q 26. How will you write a simple Recursive Program? 44
Q 27. How many elements a complete binary tree could hold for a depth of 10? 44
Q 28. How will you swap two numbers without any temporary variable? 45

Q 29. Explain working of a hashing data structure, for example HashMap in Java. 46
Q 30. Discuss internal's of a concurrent hashmap provided by Java Collections Framework. 47
Q 31. Discuss Visitor, Template, Decorator, Strategy, Observer and Facade Design Patterns? 49
Q 32. What is a strong, soft, weak and phantom reference in Java? 51
Q 33. What is transaction Isolation level? 53
Q 34. What is difference between Primary key and Unique Key? 54
Q 35. Why do we need indexing on database table columns? 54
Q 36. How will you list all customers from customer table who have no Order(s) yet? 55
Q 37. How would you fetch Employee with nth highest Age from Employee Table using SQL? 55
Q 38. What is difference between Drop, Truncate and Delete commands in SQL? 55
Q 39. What is Inner Join, Left Outer Join and Right Outer Join? 56
Q 40. What are clustered and non-clustered indexes in database? 57
Q 41. How would you handle lazily loaded entities in web application development using hibernate? 57
Q 42. What are OneToOne, OneToMany and ManyToMany relationship mappings in database design? 58
Q 43. How would you implement ManyToMany mappings with the self entity in JPA? 59
Q 44. What are Inheritance strategies in JPA? 60
Q 45. How will you handle Concurrent updates to an database entity in JPA i.e. when two users try to update the same database entity in parallel? 60
Q 46. How to efficiently generate ID's for an Entity in Hibernate/JPA? 61
Q 47. How does a typical Hibernate Transaction look like? 61
Q 48. How will you handle batch insert in hibernate for optimal usage of memory, network and CPU? 62
Q 49. How will you operate on records of a large database table with million of entries in it using Hibernate? 63
Q 50. Do you think Hibernate's SessionFactory and Session objects are thread safe? 64
Q 51. What is difference between Hibernate's first and second level cache? 64
Q 52. What is difference between session.get() and session.load() in hibernate? 64
Q 53. What is usecase for GET, PUT, POST and DELETE method in REST API? 65
Q 54. What are different types of Http Status Codes? 65
Q 55. What is difference between HTTP Redirect and Forward? 66
Q 56. What are the best practices for handling TimeZone in database transactions? 66
Q 57. How will you check the owner information of a given domain name in web? 67
Q 58. What happens when you type www.google.com in your browser's address bar from an Indian Location? 68
Q 59. Why do we need Spring Framework? 70
Q 60. What is Inversion of Control (or Dependency Injection)? 70
Q 61. What is Bean Factory in Spring? 70
Q 62. What is Application Context? 70
Q 63. What are different types of Dependency Injection that spring support? or in other words what are the ways to initialize beans in Spring? 71
Q 64. What are different Bean Scope in Spring? 71
Q 65. What are some important Spring Modules? 71
Q 66. How will you load hierarchy of property files in Spring Context? 71
Q 67. How to handle Bean Post Initialization and Pre Destroy Tasks in Spring Framework? For example resource loading after bean construction and resource cleanup before shutdown of spring context? 72
Q 68. What is syntax of Cron Expression? 73
Q 69. How will you embed a PNG/GIF image inside a CSS file? 73
Q 70. Explain Java 8 Stream API? 74
Q 71. Most useful Code Snippets in Java 8? 76

Q 72. How will you replace tokens in a given text with properties loaded from a property file using Java Regular Expressions? 79
Q 73. How will you configure custom sized ThreadPool for Java 8 Stream API parallel operations? 80

Core Java Interview Questions 81

Q 74. What are new features added in Java 8? 81
Q 75. What is difference between method overloading, method overriding, method and variable hiding? 82
Q 76. What is order of calling constructors in case of Inheritance? 83
Q 77. What is diamond problem of multiple inheritance? 84
Q 78. How does Java 8 tackles diamond problem of multiple inheritance? 85
Q 79. When should we choose Array, ArrayList, LinkedList over one another for a given Scenario and Why? 86
Q 80. We have 3 Classes A, B an C. Class C extends Class B and Class B extends Class A. Each class has an method foo(), is there a way to call A's foo() method from Class C? 87
Q 81. Why wait is always used inside while loop as shown in the below snippet? Discuss all the probable reasons.

```
public synchronized void put(T element) throws InterruptedException {
    while(queue.size() == capacity) {
      wait();
    }
    queue.add(element);
    notify();
}
```

88
Q 82. We have a method which iterates over a Collection. We want to remove certain elements from that collection inside the loop in certain criteria is matched, How should we code this scenario? 89
Q 83. We are writing an API which will accept a Collection<Integer> as an argument and duplicate an element in the Original Collection if certain criteria in met. How would you code such an API method? 90
Q 84. If hashcode() method of an object always returns 0 then what will be the impact on the functionality of software? 90
Q 85. Iterator interface provides remove() method but no add() method. What could be the reason for such behavior? 91
Q 86. What does Collections.unmodifiableCollection() do? Is it a good idea to use it safely in multi-threading scenario without synchronization, Is it immutable? 91
Q 87. If we don't override hashcode() while using a object in hashing collection, what will be the impact? 91
Q 88. How would you detect a DeadLock in a running program? 92
Q 89. How would you avoid deadlock in a Java Program? 92
Q 90. What is difference between Vector and ArrayList, why should one prefer ArrayList over Vector? 92
Q 91. How would you simulate DeadLock condition in Java? 93
Q 92. Which data type would you choose for storing currency values like Trading Price? What's your opinion about Float, Double and BigDecimal? 94
Q 93. How would you round a double value to certain decimal Precision and Scale? 96
Q 94. How great is the Idea of synchronizing the getter methods of a shared mutable state? What if we don't? 97
Q 95. Can the keys in Hashing data structure be made Mutable? 97
Q 96. Is it safe to iterate over collection returned by Collections.synchronizedCollection() method, or should we synchronize the Iterating code? 98
Q 97. What are different type of Inner classes in Java? How to choose a type with example? 99
Q 98. When should we need a static inner class rather than creating a top level class in Java program? 99
Q 99. How does method parameter passing works in Java? Does it pass-by-reference or pass-by-value? 100
Q 100. Is it possible to write a method in Java which swaps two int/Integer? 101

Q 101. What all collections require hashcode() method? 102
Q 102. What is problem with mutable static variables in Java class? 102
Q 103. Provide a diagram for collections framework. 103
Q 104. What is Immutable Class. Why would you choose it? How would you make a class immutable? 104
Q 105. Discuss Exception class hierarchy in Java. When should we extend our custom exception from RuntimeException or Exception? 105
Q 106. How does an ArrayList expands itself when its maximum capacity is reached? 106
Q 107. How does HashMap expands itself when threshold is reached? 106
Q 108. What is StringPool In Java? 106
Q 109. What is instance level locking and class level locking? 107
Q 110. Explain threading jargons? 108
Q 111. What is float-int implicit conversion while doing calculation on mixed data type in Java? 109
Q 112. Discuss Comparable and Comparator? Which one should be used in a given scenario? 109
Q 113. How would you sort a collection of data based on two properties of an entity in Java, analogical to SQL's Order by firstField, SecondField desc? 110
Q 114. How would you convert time from One Time Zone to another in Java? 111
Q 115. Will WeakHashMap's entry be collected if the value contains the only strong reference to the key? 112
Q 116. Why HashMap's initial capacity must be power of two? 112
Q 117. Can we traverse the list and remove its elements in the same iteration loop? 112
Q 118. Do I need to override object's equals() and hashcode() method for its use in a TreeMap? 113
Q 119. Implement a thread-safe BlockingQueue using intrinsic locking mechanism. 113
Q 120. Is there a way to acquire a single lock over ConcurrentHashMap instance? 114
Q 121. How will you implement a Blocking Queue using Lock and Condition Interface provided in JDK? 114
Q 122. How would you cancel a method execution after time-out expires using Java Future? 115
Q 123. Java already had Future interface, then why did they provide Completable Future class in Java 8? 116
Q 124. What is difference between intrinsic synchronization and explicit locking using Lock? 117
Q 125. What are Stamped Locks? How they are useful in optimistic scenario where thread contention is rare? 118
Q 126. How will you find out first non-repeating character from a string? For example, String input = "aaabbbeggh", answer should be 'e' 120
Q 127. What is difference between Callable and Runnable Interface? 121
Q 128. What will happen when an exception occurs from within a synchronized code block? Will lock be retained or released? 121
Q 129. What is difference between sleep(), yield() and wait() method? 122
Q 130. What is difference between StringBuilder and StringBuffer? 122
Q 131. How does Generics work in Java? 123
Q 132. What are Upper and Lower bounds in Generics? Where to choose one? 124
Q 133. Discuss memory visibility of final fields in multi-threading scenario. 125
Q 134. Where would you use LinkedHashSet provided by Java Collections? 127
Q 135. What do you think is the reason for String Class to be Immutable? 127
Q 136. How is String concatenation implemented in Java using + operator?
for example, String name = "hello" + "world" 127
Q 137. Which data type would you choose for storing sensitive information, like passwords, and Why? 128
Q 138. What is difference between using Serializable & Externalizable Interfaces in Java? 128
Q 139. How would you design Money Class in Java? 128
Q 140. What is difference between HashMap, TreeMap and LinkedHashMap? 130

Q 141. How would you write high performing IO code in Java? Can you write a sample code for calculating checksum of a file in time efficient manner? 131
Q 142. We have an Application and we want that only Single Instance should run for that Application. If Application is already running then second instance should never be started. How would you handle this in Java? 134
Q 143. Why do we need Reader Classes when we already have Streams Classes? What are the benefit of using a Reader over a stream, in what scenario one should be preferred. 135

Concurrency in Java 136

Q 144. What is Concurrency? How will you implement Concurrency in your Java Programs? 136
Q 145. There are two Threads A and B operating on a shared resource R, A needs to inform B that some important changes has happened in R. What technique would you use in Java to achieve this? 137
Q 146. What are different states of a Thread? What does those states tells us? 138
Q 147. Question: What do you understand by Java Memory Model? What is double-checked locking? What is different about final variables in new JMM? 139
Q 148. Is i++ thread-safe (increment operation on primitive types)? 143
Q 149. What happens when wait() & notify() method are called? 143
Q 150. Discuss ThreadPoolExecutor? What different Task Queuing Strategies are possible? How will you gracefully handle rejection for Tasks? 143
Q 151. How will you write a custom ThreadPoolExecutor that can be paused and resumed on demand? You can extend the existing ThreadPoolExecutor to add this new behavior. 145
Q 152. How will you write your own custom thread pool executor from scratch? 146
Q 153. What is difference between ExecutorService's submit() and execute() method? 147
Q 154. Discuss about volatile keyword and Java Memory Model? 148
Q 155. What is a CAS? How does it help writing non-blocking scalable applications? Tell something about Atomic Package provided by Java 1.6 149
Q 156. There is a object state which is represented by two variables. How would you write a high throughput non-blocking algorithm to update the state from multiple threads? 150
Q 157. How would you implement AtomicFloat /AtomicDouble using CAS? 151
Q 158. How LongAdder and LongAccumulator are different from AtomicLong & AtomicInteger? 153
Q 159. Can we implement check & update method (similar to compare and swap) using volatile alone? 153
Q 160. What is difference between Fork/Join framework and ExecutorService? 154
Q 161. How does ForkJoinPool helps in writing concurrent applications? Please provide few examples for RecursiveTask and RecursiveAction. 154
Q 162. How will you track the largest value monitored by different threads in an non-blocking fashion (using atomic operations)? 155
Q 163. How will you calculate Fibonacci Sequence on a multi-core processor? 158
Q 164. How will you increment each element of an Integer array, utilizing all the cores of processor? 159
Q 165. You are writing a multi-threaded software piece for NSE for maintaining the volume of Trades made by its individual brokers (icici direct, reliance). It's highly concurrent scenario and we can not use lock based thread safety due to high demand of throughput. How would handle such scenario? 160
Q 166. Calculate the time spread for 10 threads - Suppose T1 started earliest and T5 finished last, then the difference between T5 and T1 will give time spread. 161
Q 167. What are fail-fast Iterator? what is fail safe? 163
Q 168. There is a stream of words which contains Anagrams. How would you print anagrams in a single bucket from that stream? 164

Q 169. Describe CopyOnWriteArrayList? Where is it used in Java Applications? 166
Q 170. There are M number of Threads who work on N number of shared synchronized resources. How would you make sure that deadlock does not happen? 166
Q 171. Are there concurrent version for TreeMap and TreeSet in Java Collections Framework? 166
Q 172. Is it safe to iterate over an ArrayList and remove its elements at the same time? When do we get ConcurrentModificationException & hidden Iterator? 167
Q 173. What is ThreadLocal class, how does it help writing multi-threading code? any usage with example? 168
Q 174. How would you implement your own Transaction Handler in Core Java, using the EntityManager created in last question? 169
Q 175. What is AtomicInteger class and how is it different than using volatile or synchronized in a concurrent environment? 170
Q 176. You are writing a server application which converts microsoft word documents into pdf format. Under the hood you are launching a binary executable which does the actual conversion of document. How would you restrict the parallel launch of such binaries to 5 in Java, so as to limit the total load on the server. 171
Q 177. What are common threading issues faced by Java Developers? 173

Algorithms & Data Structures 174

Q 178. Given a collection of 1 million integers ranging from 1 to 9, how would you sort them in Big O(n) time? 174
Q 179. Given 1 million trades objects, you need to write a method that searches if the specified trade is contained in the collection or not. Which collection would you choose for storing these 1 million trades and why? 175
Q 180. I have an Integer array where every number appears even number of time except one. Find that number. 175
Q 181. how would you check if a number is even or odd using bit wise operator in Java? 176
Q 182. How would you check if the given number is power of 2? 176
Q 183. What is a PriorityQueue? How is it implemented in Java? What are its uses? 177
Q 184. What is difference between Collections.sort() and Arrays.sort()? Which one is better in terms of time efficiency? 178
Q 185. There are 1 billion cell-phone numbers each having 10 digits, all of them stored randomly in a file. How would you check if there exists any duplicate? Only 10 MB RAM is available to the system. 178
Q 186. What is a Binary Search Tree? Does Java provide implementation for BST? How do you do in-order, pre-order and post-order Traversal of its elements? 179
Q 187. What is technique to sort data that is too large to bring into memory? 180
Q 188. Check if a binary tree is a Binary Search Tree or not? 180
Q 189. How would you convert a sorted integer array to height balanced Binary Search Tree?
Input: Array {1, 2, 3}
Output: A Balanced BST
2
/ \
1 3 181
Q 190. How would you calculate depth of a binary tree? 182
Q 191. Calculate factorial using recursive method. 182
Q 192. You have a mixed pile of N nuts and N bolts and need to quickly find the corresponding pairs of nuts and bolts. Each nut matches exactly one bolt, and each bolt matches exactly one nut. By fitting a nut and bolt together, you can see which is bigger. But it is not possible to directly compare two nuts or two bolts. Given an efficient method for solving the problem. 183

Q 193. Your are give a file with 1 million numbers in it. How would you find the 20 biggest numbers out of this file? 183
Q 194. Reverse the bits of a number and check if the number is palindrome or not? 184
Q 195. How would you mirror a Binary Tree? 184
Q 196. How to calculate exponentiation of a number using squaring for performance reason? 185
Q 197. How will you implement a Queue from scratch in Java? 186
Q 198. How will you Implement a Stack using the Queue? 187
Q 199. How will you test if a given sentence is a Pangram or not? 188
Q 200. How would you implement a simple Math.random() method for a given range say (1-16)? 189
Q 201. How an elevator decides priority of a given request. Suppose you are in an elevator at 5th floor and one person presses 7th floor and then 2nd presses 8th floor. which data structure will be helpful to prioritize the requests? 189
Q 202. How would you multiply a number with 7 using bitwise hacks? 190
Q 203. What is the best way to search an element from a sorted Integer Array? What would be it's time complexity? 190
Q 204. How would you reverse a Singly linked List? 191
Q 205. How would you count word occurrence in a very large file? How to keep track of top 10 occurring words? 192
Q 206. What is difference between synchronized HashMap and a Hashtable? 196
Q 207. What is difference between Iterator and ListIterator? 196
Q 208. What do you understand by Token Bucket Algorithm. What is its use? 197
Q 209. How will you implement fibonacci series using Iterative & Recursive approach in Java 8? 199
Q 210. How will you write a multi-threaded HttpDownloader program using Java 8? 202
Q 211. How will you find first non-repeatable character from a String using Java 8? 203
Q 212. How will you find Word Frequency in sorted order for a collection of words? 203
Q 213. How will you calculate MD5 hash of a given String in Java? 205
Q 214. There is a set of integer values representing the time consumed by a job execution in seconds, how will you find average execution time ignoring the extreme run times (0 seconds or a value much above the average execution time)? 205
Q 215. There are millions of telephone numbers in a country, each state of the country has specific phone number range assigned to it. How will you determine which state a number belongs to? 205
Q 216. There is a number series in which subsequent number is either +1 or -1 of previous number. How will you determine if the range of supplied numbers are contained in this series in minimum time? 207

Object Oriented Design 208

Q 217. What are the key principles when designing a software for performance efficiency? 208
Q 218. How would you describe Producer Consumer problem in Java? What is its significance? 208
Q 219. How would you implement a Caching for HttpDownloader Task using Decorator Design Pattern? 212
Q 220. Write Object Oriented design for library management system. 213
Q 221. Design ATM machine. 215
Q 222. Design a web crawler that will crawl for links(urls). 216
Q 223. Design Phone Book for a mobile using TRIE (also known as prefix tree). 220
Q 224. How would you resolve task's inter dependency, just as in maven/ant.
Let's consider the following task dependencies.

Here first row states that task 3 is dependent on task 1 and task 5, and so on. If the two consecutive tasks have no

dependency, then they can be run in any order.
The result should look like - [1, 5, 3, 2 ,4]or [1, 5, 3, 4, 2] 222
Q 225. How would you sort 900 MB of data using 100 MB of RAM? 226
Q 226. How would you design minimum number of platforms so that the buses can be accommodated as per their schedule?
Bus Schedule for a given Platform 229
Q 227. There is a pricing service which connects to Reuters & Bloomberg and fetches the latest price for the given Instrument Tics. There could be multiple price events for the same Stock and we need to consider the latest one. Design a service to show prices for the Top 10 stocks of the Day? 231
Q 228. Design a parking lot where cars and motorcycles can be parked. What data structure to use for finding free parking spot in Parking Lot program? Assume there are million of parking slots. 234
Q 229. There is three file contains flight data, write a standalone program to search flight detail from all files depend on criteria? Write JUnit to demonstrate the working. 235
Q 230. Implement the classes to model two pieces of furniture (Desk and Chair) that can be constructed of one of two kinds of materials (Steel and Oak). The classes representing every piece of furniture must have a method getIgnitionPoint() that returns the integer temperature at which its material will combust. The design must be extensible to allow other pieces of furniture and other materials to be added later. Do not use multiple inheritance to implement the classes. 241
Q 231. How would you simulate a digital Clock in Object Oriented Programming Language? 243
Q 232. How would you design an elevator system for multi story building? Provide with request scheduling algorithm & Class diagram for the design. 246
Q 233. Given two log files, each with a billion username (each username appended to the log file), find the username existing in both documents in the most efficient manner? 246
Q 234. Design DVD renting system, database table, class and interface. 247

Puzzles & Misc 248

Q 235. Why man holes are round in shape? 248
Q 236. Solve two egg problem? 248
Q 237. There are 100 doors all closed initially. A person iterates 100 times, toggling the state of door in each iteration i.e. closed door will be opened & vice versa.
1st iteration opens all doors (1x multiplier)
2nd iteration closes 2,4,6,8 .. doors (2x multiplier)
3rd iteration opens 3,6,9,12 ... doors (3x multiplier) and so on.
In the end of 100 iterations, which all doors will be in open state? 249
Q 238. What is probability of a daat, hitting closer to centre of a circle rather than circumference? 250
Q 239. What is financial Instrument, Bond, equity, Asset, future, option, swap and stock with example each? 250
Q 240. Toolkit & Resources for a Java Developer. 251
Q 241. Unsolved Interview Questions. 252
Q 242. Sample Unsolved Puzzles? 257
Q 243. Few sample UNIX questions. 261
Q 244. Top Java Interview Questions? 262
Q 245. What is the Typical Interview Coverage for Core Java Candidate? 263
Q 246. What is the art of writing resume? 263
Q 247. Sample Questions on Swings framework. 264
Q 248. What are the Interview questions that most candidates answer wrongly? 264

IT Industry and Psyche of Developers

My opinion in Indian context

Q. What is overall picture of Indian IT Industry?

IT is outcome of Intellectual Minds from western civilization in modern era. The knowledge is still sourced from west in this domain at least at the time of this writing. The main reason for outsourcing IT related work to India is economically cheap labour, though the trend has started changing now. Typically there are two types of companies in India -

1. Product Companies (Adobe, Google, Amazon, Oracle, etc)
2. Services/Consulting/Broker Companies (TCS , Infosys, Cognizant, Wipro, HCL, Sapient, etc)

There is a big cultural and compensation differences in both these categories. Product based companies provides better culture for the employee in terms of personal satisfaction (perks, salary, other activities). Consulting companies compensate employees by short/long term onsite opportunities, which are rare in Product Based Companies.

Q. What type of work people do in India IT Industry?

We can categorize typical IT work into two types - Green Fields and maintenance work.
Most people in Indian IT industry work for services based companies where major fraction of work is of Maintenance & Support type. Even most product based companies outsource work to India once Project/ Product is stable. You could find exceptions to my words at some places.

Q. What causes a typical developer to switch his/her job so frequent, Is that bad, why that is not the case in West?

A thought to switch Job comes to one's mind when one finds blockage in growth opportunities in their current organization. I could list few reasons for the same -

1. Salary disparity and scope for higher salaries in next Job is the major reason for job switch. Most service based companies tries to maximize their profit offering lower salaries to its employees (that's beneficial for freshers who do not have any experience), but as people acquire more skills they switch for bigger roles in new Job. Demand and supply typically governs the salaries in India.
2. The quality of work is another reason for switch. Work quality directly relates to stress (more manual work more stress)
3. Shift Timings and location preference also causes people to switch their jobs

To some extent this switch is fair because we can't expect someone to work for a company few thousand dollars a year for his lifetime (As he acquires skills in parallel to take up more responsibilities). As the Industry will mature, job shift will reduce. The IT companies in west are mature, salaries are already saturated, people don't take much work stress, So western employees do not find many reasons for their Job switch.

Q. What is growth pyramid of a typical developer in Indian IT Industry?

There are two broader categories for growth - Technical and Management. Depending on one's likings, people incline to one of these paths. In Service Based companies , most employees start their carrier as a software developer/qa, but over the time they shift to Management Side. This trend has started changing now, because of the saturation on number of jobs in management side. So people will experience longer stretch working as a developer in the coming decade. Employees working for startup and product based companies, tend to remain on technical side, because Individual Contribution gives them better growth opportunities.

Q. What is typical psychology of average Indian developer? What kind of chaos pollute his mind?

Most Indian opt for IT, not by choice but for money, because of large unemployment in India. Moreover earning money in IT industry is easy and effortless compared to other parallel opportunities. Many people wants to

take IT as the jumping ground for their higher studies (MBA, MS, etc). An average fresher is polluted with the thoughts about his career growth, and is unsure about his key interests in IT field, trying various alternates in first few years.

Q. What is the problem with most Indian developers in terms of skills?
Majority of IT crowd does not have good hold over their primary skills (Technical, Presentation) which are required for the work. The underlying cause for the low skills are poor quality of education and the type of work which is fed to Indian Companies. The majority of work does not require high quality of skills on developer's part. Many people learn by their own, build their skills and fight for better quality work. One should have a very good hold over his primary skill set and look for work which is matching those skills.

Q. What are advantages of investing in skills?
1. Very good understanding the basic computer science along with the core language skills helps write very efficient/scalable/maintainable software that is capable of utilizing the available hardware effectively, with minimum bugs.
2. Skills alleviates work stress, by empowering us to design intelligently, automating the mundane tasks.
3. When you have good understanding of hardware and software then you get a deep penetration into software development which otherwise is not possible.
4. Better skills may attract better Job profile.

Q. Would it help if I memorize all the questions for cracking interviews?
No, it will not. But memorizing the most common **Patterns of software development** will definitely help crack not only the interview but also make your day to day work life easy. A single pattern resolves n number of problems emerging from that pattern, and we should always look forward finding the patterns instead of solution to a particular problem.

Q. Why do interviewers ask rocket science questions in interviews even if the new role does not require any such skills?
Hiring in IT industry is not regulated by any means, it is solely up to the interviewer to choose the topic for discussion in the interview. In today's intellectual world, people like intellectual war, and interview is a good place for that. I do not find any harm by such interview process unless interviewer hides the real picture of work that one needs to perform after joining the new role. For sure there is one plus point to such interview process that it will definitely tend to raise our skill set.

Q. Why people take so many offers at the time of Job change, doesn't it add to chaos?
The main reason for doing so, is the disparity between work and salary across the companies. People feel insecure at financial level and try their best to grab the most paying Job opportunity, and that's fair from employee perspective. On the other hand, companies tend to maximize their profit by limiting the salary offer as per individual's previous company's salary. So it is a game, where both the employer and the employee are fighting to maximize their own profit. Ultimately, the Demand and Supply equation balances the fight between employer and the employee. Saturation of salaries and work quality in coming years might hamper this.

Q. Quality work never reaches India, is that right?
Thats true for many companies, including the big MNCs. Seed for existence of Indian counterpart of a MNC is the cheap and easily scalable labour who is happy to do anything anytime for money. At present, best practices for software development exists in Europe & some parts of US. Most Indian MNCs are never setup for best talent so we can not expect quality work at the moment. Work here is mostly committed by management for fixed cost, no matter how you do it, how long you stretch. The good part is, exceptions are there and the scenario is changing at a great speed. You can be the part of that change!

Chapter 1

Core Concepts, Spring & Hibernate

Q 1. What are good software practices for developing Scalable, Testable and Maintainable software?

1. Proper understand of the domain knowledge is first step towards software development.
2. **Microservices Architecture** is the current industry trend for developing scalable distributed systems. Spring Boot and Spring Cloud provide the necessary boiler plate code for developing cloud native applications in Java.
3. Make system resilent to failures by using circuit breaker & bulk head design patterns. During peak load time, its better to turn off a single failed component of a big system rather than putting down the entire system.
4. Make distributed systems more reliable and eventually consistent by using messaging queues (Advanced Message Queuing Protocol, Kafka, JMS, etc). This also allows for asynchronous communication between services. Asynchronous communication brings many benefits (loosely coupled, increased throughput, reliable, etc.) to the distributed computing.
5. Follow good software development practices like Agile with Test Driven Development. Agile development is all about incorporating changes in the software without much pain. TDD helps achieving agility in your software. A good test coverage (End to End and Unit Tests) keeps a developer away from last minute stress during production deployment.
6. Automate all non-productive mundane tasks related to deployment, using Gradle, Jenkins, Salt Stack etc.
7. Keep refactoring your code base time to time, don't leave any duplicate code inside code base. Follow DRY (don't repeat yourself) strictly. Every object must have a single authoritative representation in the system. Software development is like the art of gardening where refactoring takes it to a next level.
8. Add an automated test case for every new bug found. Appropriate test assertions are equally important otherwise it will just reflect on the code coverage without much help.
9. Use Contract Tests to make sure that your REST consumers are not affected by your changes.
10. Use Profiling Tools to identify bottlenecks of your application. One can use jvisualVM tool bundled in JDK to know the JVM profile of an application, though there are some commercially available easy to use tools available in market, e.g. JProfiler
11. Use tools to find the duplicates and then refactor to reuse the existing code with better design. IntelliJ is one of the good tools that will take care of boilerplate stuff (but soon you will become it's luxury addict)
12. Work in small steps with frequent feedback loop to avoid the last minute surprises (Agile). Github [1]is well known for its aggressive engineering practices, deploying code into production on an average 60 times a day. Amazon is on record[2] as making changes to production every 11.6 seconds on average in May of 2011.
13. Continuous Integration environment is must for rapid bug free, coordinated development. Tools like TeamCity, Hudson, Jenkins, etc can be leveraged for Continuous Integration.
14. Software development without **Art, Fun and Creativity** is boring and will bring depression, so be aware of this warning sign. Don't leave learning, be a student for lifetime!!!

1 https://githubengineering.com/move-fast/

2 https://www.thoughtworks.com/insights/blog/case-continuous-delivery

Q 2. What is growth roadmap for a Java developer?

A deep understanding of basic software components is must for a developer who wants to craft Scalable Distributed Systems from scratch. Below is the matrix showing a hierarchy of skills that are required for a senior software developer who is looking for a long term carrier in software development.

Priority	Category	Topics
7	**Practices**	Agile Development Methodologies (Test Driven Development, Continuous Integration & Continuous Deployment)
6	**Architecture**	Microservices Architecture based distributed systems, REST, Cloud Native Applications using Spring Boot and Spring Cloud
5	**Database**	SQL, Database Indexes, Query Optimization, Inner and Outer Joins, JPA/Hibernate QL, Table to Entity Mapping, Inheritance in JPA (Table per class, joined, single table), Transaction Isolation Level, embeddable, Mapped Super Classes, entity relationships - OneToOne, OneToMany, ManyToMany. NoSql - Amazon DynamoDB, MongoDB, Apache Cassandra
4	**Core Java**	Inheritance, Generics, Garbage Collector, good hold over Concurrency (synchronizer, non-blocking algorithms using atomic package, executor service, Java Memory Model, Immutability, volatile, Fork/Join), Internal of Java Collections Framework (HashMap, LinkedList, ConcurrentHashMap, PriorityQueue, TreeMap)
3	**Algorithms**	Hashing Algorithms, Tree (traversal, insertion, balancing using left & right rotation), Sorting (quick, merge, external merge, bucket, counting), Binary Search, BFS, DFS, Topological Sorting using Graph, Dijkstra's algorithm, Recursion & Dynamic Programing
2	**Data Structures**	ArrayList, LinkedList, Stack, Queue, Tree (Binary Search Tree, Red Black Tree, Binary Heap, PriorityQueue), Set, Hashtable, Prefix Tree (Trie), Suffix Tree, Graph Traversal, etc.
1	**Concepts**	Object Oriented Programming & Design Test Driven Development (JUnit, Mockito, etc) Version Control System (GIT), Continuous Integration, Design Patterns (Singleton, Observer, Template, Strategy, Decorator, Facade, Flyweight, etc), Domain Driven Design, SOLID, Memory Management (Heap, Stack, Memory Generations in Java) Logarithm, Big-O notation, Bitwise Manipulation & Number System (2's complement, binary, hexadecimal, etc)

Skills Matrix for a Java Developer

Q 3. What are essential skills for Investment Banking domain?

A good hold over the following skills will give you edge if you are preparing for investment banking domain:

1. Good understanding of distributed system (Service Oriented Architecture or Microservices Architecture)
2. Spring Framework (DI, MVC, Batch, Security etc.)
3. Concurrency, Collections, Algorithms & Data Structures
4. Good understanding of messaging system (JMS, TIBCO MQ, ActiveMQ, RabbitMQ, Kafka etc). JMS is used to keep multiple systems eventually consistent. It also allows asynchronous communication using events.
5. Test Driven Development
6. Good understanding of Database (Query, Joins, PLSQL, Indexes, Query Optimization, etc.)
7. Design Patterns (Singleton, Observer, Strategy, Visitor, Proxy, MVC, SOLID, etc.)
8. VCS (Git, branching models, different workflow, rebase vs merge)

Q 4. What are trending skills in Information & Technology?

Following skills are trending in the IT Market:

1. Microservices based Architecture (must-have)
2. Spring Boot and Spring Cloud (must-have)
3. Angular (front-end)
4. Big Data Analysis - Apache Spark, Elastic Search
5. Python R - data analytics
6. Machine Learning and Natural Language Processing
7. Android Programming
8. Reactive Programming (Java 9, Spring 5, Spring Boot 2) (should-have)
9. Docker for light weight container (should-have)
10. Amazon Web Services (EC2, S3, Cloud front, Cloud formation, Amazon DynamoDB, etc) (should-have)
11. Continuous Integration and Continuous Deployment (Jenkins, Kubernetes etc.) (should-have)

Its not possible & necessary to have hold over all the above technologies.

Q 5. What are good books for Java developers?

Core Java	Frameworks	Design Patterns, Algorithms & Misc.
1. Java 8 for the Really Impatient, Cay S. Horstmann 2. Java Concurrency in Practice by Brian Goetz 3. Head First Java 4. Effective Java, 2nd Edition	5. Spring in Action by Craig Walls, 4th Edition 6. Java Persistence with Hibernate 7. Rest In Action, Manning 8. Spring Boot and Spring Cloud documentation.	9. Head First Design Patterns, Kathy Sierra 10. Algorithms, Robert Sedgewick, Kevin 11. Cracking The Coding Interviews, 150 Programming Questions and Solutions by Gayle, Careercup 12. Domain Driven Design by Eric J. Evans. This book is helpful for understanding concept of bounded context that is central to microservices architecture.

Q 6. Why should one choose Java for Software Development? What are pros and cons of Java?

Java Pros

1. Java is free, download it and start creating your own applications. Linux, MySQL and Java have been a favorite choice for startups as well as big MNCs.
2. Spring and Hibernate makes development a fun activity. Most of the boiler plate code for Security, Object Relational Mapping, MVC framework, etc is handled by these frameworks. Teams using Java along with these frameworks can concentrate on the real business rather than wasting time on rediscovering the wheels time and again.
3. Plenty of third party libraries, frameworks & IDE for faster development (Eclipse, Spring Suite, IntelliJ, etc)
4. Platform independent, write once run on most modern platforms (Linux, Unix, Windows, Mac, 32/64 bit Hardware)
5. It supports Object Oriented Programming, thus making it easy to model real life scenarios into object model
6. In built support for multi-threading & concurrency, Its easy to write scalable applications in Java that can utilize multi-core processors, clusters of machine, distributed RAM, etc. There is in built support for Threads, ForkJoinTask (which deploys Work Stealing Algorithm), non-blocking algorithm using CAS (Compare And Swap offers better scalability under moderate thread contention compared to traditional locking mechanism), Java 8 Stream API, Parallel Streams, CompletableFuture, Parallel Array Operations, Atomic Values, LongAccumulator, etc.
7. Very good support for Internationalization & Security.
8. Memory management is automatic by use of garbage collector (G1, Concurrent Mark Sweep, parallel scavenge garbage collector, etc)
9. Pure Java byte code running on 32 bit JVM works perfectly fine on a 64 bit platform
10. Functional interfaces & lambda expressions introduced in Java 8 makes code writing an easy affair. Specifically, dealing with Collections is fun in Java 8. For example, if you want to sort a collection of people with last name, first name and e-mail (ignoring the case for e-mail), then the following code will do it all

```
Stream<Person> people = Stream.of(new Person(), ...);
people.sorted(Comparator.comparing(Person::getLastName)
        .thenComparing(Person::getFirstName)
        .thenComparing(Person::getEmail, Comparator.nullsLast(String.CASE_INSENSITIVE_ORDER)))
        .forEach(System.out::println);
```

Java Cons

1. Is not a good fit for desktop applications because of heavy memory footprint and huge VM startup time compared to a application written in low level language (C/C++, OS dependent native language). Two tools written in Java namely Eclipse & Intellij IDEA are the exception.
2. Normal Java is not a good fit for real time systems because of its "stop the world garbage collector pauses". Whenever Full GC triggers, it halts processing across the application for its duration of run.

Q 7. What is difference between 32 bit and 64 bit versions of Java?

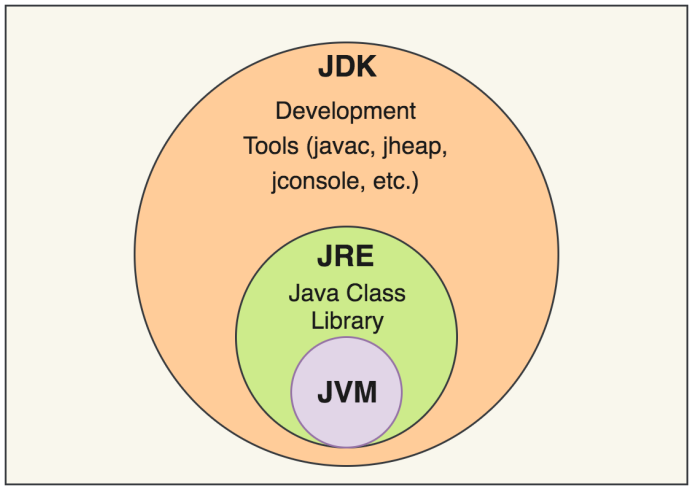
The Java language specifications are same for both the platform i.e. int will remain to be 4 bytes signed two's complement, char will remain single 16-bit Unicode, long will remain 64-bit signed two's complement, and so on. Hence any Pure Java code will not see any difference provided external native calls are not used. All that changes is the amount of addressable memory (good) and the amount of memory per Object (not that good). The size of the reference variables doubles from 32 bit to 64 bit, thus all the reference variable will take double the size when running on 64 bit JVM.
Theoretically, there are no class file differences between code compiled with the 32 bit and 64 bit versions of the same revision of Java.

For 32 bit JVM, the maximum memory is limited to 4GB, the memory limit for 64 bit JVM is very high. But more JVM memory may cause larger System wide GC pauses, so the size of JVM should be decided keeping this factor into account.
Please also note that 64 bit JVM requires more memory compared to 32 JVM for the same application because now each reference starts consuming 64 bit instead of 32 bit i.e. management cost in 64 bit version is higher than the 32 bit version. *However, newer JVMs offer object pointer compression[1] techniques which can significantly reduce the space required by 64 bit JVM.*

Q 8. What is difference between JDK, JRE and JVM?

JRE is subset of JDK while JVM is an abstract computing machine

Java Virtual Machine (JVM)
It is an abstract computing machine that enables a computer to run a Java program. An instance of a JVM is an implementation running in a process that executes a computer program compiled into Java bytecode. JVM performs tasks like loading byte code, code verification, code execution, etc.



Java Runtime Environment (JRE)
It is a software package (a physical entity) that contains necessary artifacts required to run a Java program. It includes JVM implementation together with an implementation of Java Class Library (rt.jar). Hotspot is the JVM implementation for Oracle Java.

Java Development Kit (JDK)
It is a superset of a JRE and contains tools for Java programmers, e.g. javac compiler, jconsole, jheap, jps, jvisualvm etc.

Oracle releases server JRE also, that contains normal JRE along with few tools (e.g. javac compiler) that are required by Tomcat like servlet containers.

Q 9. What are four basic principles of Object Oriented Programming?

There are 4 major principles that make an language Object Oriented. These are Encapsulation, Data Abstraction, Polymorphism and Inheritance.

Encapsulation
Encapsulation is the mechanism of hiding of data implementation by restricting access to public methods.

Abstraction
Abstract means a concept or an Idea which is not associated with any particular instance. Using abstract class/ Interface we express the intent of the class rather than the actual implementation. In a way, one class should not know the inner details of another in order to use it, just knowing the interfaces should be good enough.

Inheritance
Inheritances expresses "is a" relationship between two objects. Using Inheritance, In derived classes we can

reuse the code of existing super classes.

Polymorphism

It means one name many forms. It is further of two types - static and dynamic. Static polymorphism is achieved using method overloading and dynamic polymorphism using method overriding.

Q 10. What is Aggregation, how is it different from Composition?

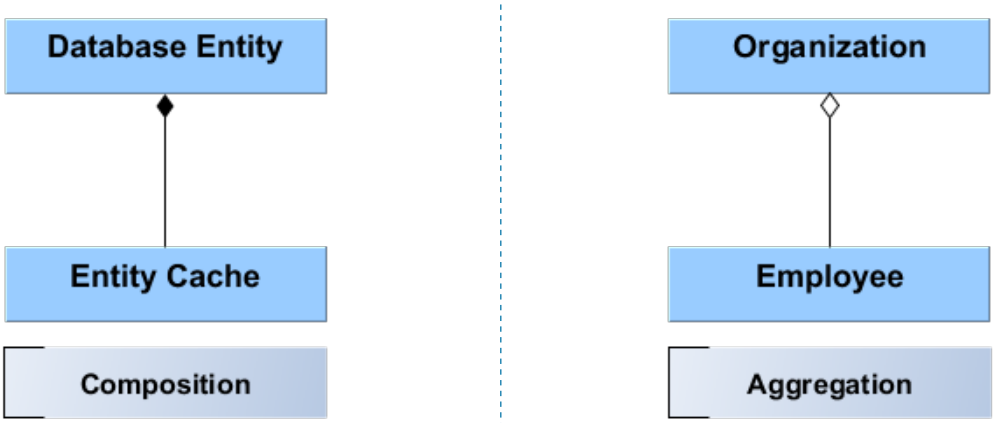
Both of these are special type of association and differ only in weight of relationship in a computer program. This term should not be confused with real world relationship that are generally softer and fuzzier.

Composition

Composition is stronger form of "has a" relationship where the composite object has sole responsibility for disposition of its component parts i.e. in Composition, component parts share a lifespan with the composite object - if the composite object is destroyed all its component parts must be destroyed. A perfect example would be an Organization (composite object) having relationship with its Departments (component parts - Finance, HR, Technical). Departments themselves do not have any independent existence outside the lifespan of an organization.

Aggregation

This represents a weaker form of "has a" relationship between whole (aggregate) and the parts (component parts). In this case the component parts may survive outside the lifespan of aggregate object and they may be accessed without going through the aggregate object. Example could be Team and its Members, or department and its members. Members may continue to exist outside the Team and even each member could be part of multiple teams. Also the members can be accessed without accessing the team.



UML Representation of Composition and Aggregation

There is also a difference in the way these two relationships are represented in UML (UML is Object Oriented Design Language) diagrams - Aggregation (weak relation) is represented by empty diamond while Composition (strong relation) is represented by filled diamond as shown in the figure.
Other example for Composition and Aggregation -

1. Caching of database objects inside your program is a perfect candidate for Composition, an object in cache shall be destroyed (evicted) as soon as the related entity is deleted from database.
2. On the other hand, Ducks(component part) swimming in a Pond(aggregate object) is an example of Aggregation where there is a weaker relationship among the two.

Compositions are a critical building block in many data structures, including Binary Tree and Linked List. However even if "Composition" completely satisfies a particular given scenario (say, Caching), you may still want to implement your design as an "Aggregation" due to technical challenges.

Implementation Level Differences in Java

Composition is implemented using component part object Initialization inside constructor (or sometimes post construction). All the component parts are destroyed as soon as composite object is destroyed, for example -

```
public class Organization {
   private final List<Department> departments;
   public Organization() {
      departments = new ArrayList<>();
      departments.add(new Department("Finance"));
      departments.add(new Department("HR"));
      departments.add(new Department("Technical"));
   }
}

class Department {
   private final String name;
   Department(String name) {
      this.name = name;
   }
}
```

Other good examples are LinkedList composite object consisting of Nodes (component parts)

On the other hands, Aggregation relationship is expressed in code using the reference of component parts inside the aggregate object, for example -

```
public class Team {
   private final List<Member> members;

   public Team(List<Member> members) {
      this.members = members;
   }
}

class Member {
   private final String name;
   Member(String name) {
      this.name = name;
   }
}
```

Here in this example, members are not destroyed when Team object is destroyed.
Other good examples could be a Pond object (aggregate object) and Duck objects (component parts), where ducks may come for sometime in the pond for swimming.

Q 11. What is role of requirement understanding in software development process?

We should not put our precious effort in developing an application where requirement is not clear to the

developer. We must ask relevant questions from the concerned person (Business Analyst, Interviewer, User, etc) to resolve all our ambiguities. It could go really bad if we are not clear of software requirement, for example lets consider these two scenarios -

Scenario 1 (hypothetical Interview Scenario)

Business Analyst : Design an efficient Java Restful Service to sort a huge set of data.

Developer : What that data looks like and how much is the volume, where is it stored?
Business Analyst : Data consists of integers, volume could go as high as 4 GB residing in a file. User will upload the file into the server and should get a sorted file in return.

Developer : What is the range of Numbers that need to be sorted, will there be duplicates?
Business Analyst : That data will consist of small numbers, all between range of 0 < number < 1 million, but yes there will be lots of duplicates

Developer : What are the hardware specs for hosting this service?
Business Analyst : It will be 4 GB RAM, 8-core cpu, 160GB Disk machine, with a cluster of four such machines

Developer : What are the Time and Space requirements for this service?
Business Analyst : It should run in minimal time possible and there could be multiple such requests running in parallel.

So we can see how the requirement evolves after raising the relevant questions. Before asking these questions developer could have implemented any sorting algorithm for this requirement without keeping in mind the parallel processing, space requirements, etc.

So the design for such a problem would be -

1. Choose proper sorting algorithm - as data consists of integer between confined range which is smaller compared to number of data items, Counting Sort is the best fit for this scenario.
2. As there are multiple cores in the cpu, so parallel processing can be a utilized after doing some benchmark on the test data. Java offers very good support for parallel processing, checkout for Parallel Stream & Parallel Array Operations(Java 8), or Fork Join Pool, etc. Also as there are multiple machines available we need to design a distributed application.
3. As the data could be larger than the memory size of single machine, we might need to opt for poly phase sorting, where more than one pass will be required to sort the data, or even external sort could be checked out.
4. We should maintain a concurrency level in the application, incase multiple users submit the request it should not result in OutOfMemory Error.

Scenario 2 (Real time scenario in production environment)

Customer placed an enhancement request in an application to export & import a tabular data into Excel Sheet

The Dev team went ahead and provided the asked functionality without introspecting the real business cause behind this requirement, later on came the real picture -
User actually wanted to manipulate the system data using some macro's in excel sheet and then wanted to upload the same to the server, if proper questions had been asked the design would be different.

Whatever manipulation user wanted to perform, could have easily be done inside the server using Java methods itself, so there was no real need for this change request raised by the user.

Q 12. What is a Logarithm? Why is it so relevant in algorithms & datastructures?

A logarithm[2] tells what exponent (power) is needed to make a certain number. In a way it is opposite of exponentiation. For example,

$Log_2 (8) = 3$ and $2^3 = 8$
$Log_{10}(1000) = 3$ and $10^3=1000$

Number	Logarithm (base 10)
1	0
10	1
100	2
1000	3
10000	4
100000	5

Why do we need Logarithm?

Logarithm converts big number values into smaller, human readable format.

- It is easy to handle small numbers compared to very large numbers, when our motive is just to compare them. Logarithm converts big values to small numbers.
- It makes multiplication and division of large numbers easy because adding logarithms is the same as multiplying and subtracting logarithms is same as dividing.
- **Logarithm is used in Computer Science for estimating the size of a given problem**, time complexity of an algorithm, space complexity of an algorithm, etc. For example,
  i. A school may have 100-200 students which is 2 on $\log_{10}$ scale
  ii. A University may have 1000-2000 students which is 3 on $\log_{10}$ scale
  iii. A City may have population of 100, 000 which is 5 on $\log_{10}$ scale
  iv. A State may have population of 1000, 000 which is 6 on $\log_{10}$ scale
  v. A nation like India can have population of 1 Billion which is 9 on $\log_{10}$ scale

Thus its easy to deal with logarithmic scale when estimating a given problem for its size.

In pre modern era, when calculators were not there, logarithm tables were used for division and multiplication of large astronomical numbers.

Logarithm has the below mathematical properties

Sum of logs = log of product
$\log_{10}(100) + \log_{10}(1000) = \log_{10}(100,000)$
i.e. 2 + 3 = 5

Subtraction of logs = log of division
$\log_{10}(1000) - \log_{10}(10) = \log_{10}(100)$
i.e. 3 - 1 = 2

Notes

- Logarithm was used in India in ancient times around 2 BC to express astronomical units. It is known as **Laghuganak (**लघुगणक**)** in Hindi.

2 http://simple.wikipedia.org/wiki/Logarithm

- Logarithmic spirals are common in nature. Examples include the shell of a nautilus or the arrangement of seeds on a sunflower.
- The Richter scale measures earthquake intensity on a base 10 logarithmic scale.
- In astronomy, the apparent magnitude measures the brightness of stars logarithmically, since the eye also responds logarithmically to brightness.
- In Divide and Conquer algorithms (Binary Search, Data Partitioning, etc), the problem set is halved in each iteration which results in logarithmic Big O (log n) Time Complexity.

Q 13. What is 2's complement notation system?

It is a number format for storing negative numbers in Binary[3]. This system is the most common method of representing signed numbers on computers. An N-bit two's-complement numeral system can represent every integer in the range −(2N−1) to +(2N−1 − 1)

Why 2's Complement?

The two's-complement system has the advantage that the fundamental arithmetic operations of addition, subtraction, and multiplication are identical to those for unsigned binary numbers (as long as the inputs are represented in the same number of bits and any overflow beyond those bits is discarded from the result). This property makes the system both simpler to implement and capable of easily handling higher precision arithmetic. Also, zero has only a single representation, obviating the subtleties associated with negative zero, which exists in ones'-complement systems.

Calculating 2's complement

Positive numbers are represented as the ordinary binary representation in 2's complementary notation.
The most significant bit (leftmost bit) is always 0 for positive number, otherwise number is negative.
To get 2's complement of a negative numbers, the bits are inverted (using bitwise NOT operator) and then value of 1 is added to get the final result.

For example, **Let's convert 5 to -5 using 2's complement notation**

Using 8 bit system, number 5 is represented as
00000101
Now invert all the bits (1's complement)
11111010
Finally add 1 to get the result
11111011

So this is binary representation of -5 in 2's complement notation.
To convert it back to a Positive Number, calculate 2's complement of a negative number

For example, lets convert -5 to 5
11111011 represents -5

Invert all the bits
00000100

Then add 1 to get the result
00000101

That is +5.

3 http://en.wikipedia.org/wiki/Two's_complement

Q 14. What do you understand by Big O notation, why is it important in algorithms?

Big O Notation[1] is a mechanism used to measure the relative efficiencies of Algorithms in terms of **Time** and **Space (memory consumption)**. It makes us understand how execution time & memory requirements of an algorithm grow as a function of increasing input size. In this notation, O stands for the **Order** of magnitude.

Constant O(1) - a program whose running time's order of growth is constant, executes a fixed number of operations to finish the job, thus its running time does not depend on N.

Linear O(N) - a program that spends a constant amount of time processing each piece of input data and thus running time is proportional to the N.

Logarithmic O(log n) - a program where on every subsequent iteration, the problem size is cut by half, for example - Binary Search.

Following are the examples of Big O, in increasing order of their magnitude.

Big O Notation	Name	Example
O (1)	Constant-time	Searching from a HashMap, check a number for even/odd
O (log n)	Logarithmic	Find an item inside sorted array using Binary Search
O (n)	Liner	Printing all elements from an array
O (n log n)	Log Linear	Sorting using Merge Sort
O (n^2)	Quadratic	Bubble Sorting Algorithm
O (2^n)	Exponential	Shortest Path Problem Djigstraw Algorithm
O (n!)	Factorial	Solving Travelling Sales Man Problem

Importance of Big O

We should always keep time efficiencies in mind while designing an algorithm for a data structures, otherwise there could be severe performance penalties for using wrong algorithm for a given scenario.

Base of Logarithm is irrelevant in Big O Notation

The base of algorithm is not relevant with respect to the order of growth, since all logarithms with a constant base are all related by a constant proportion, so log N is used when referring to the order of growth regardless of the base of Algorithm.
Number -> 1,10,100,1000
Log_2 -> 0, 2.3, 4.6, 6.9

Time efficiency in Big O notation for few Java Collections

ArrayList (ignoring the time taken by array resize operation)
O(1) for add, size and get
O(n) for toString() method

PriorityQueue
O(1) for peek, element and size
O(log n) for offer, poll, remove() and add
O(n) for remove(Object) & contains(Object)
HashMap & ConcurrentHashMap (with no collisions)
O(1) for get operation
O(1) for put operation

1 http://en.wikipedia.org/wiki/Big_O_notation

LinkedList
O(1) for removal and O(1) for add & poll method
O(n) for toString() method

Q 15. How would you determine Time Complexity of a given algorithm, are there any general guidelines?

There are few rules which can help us in the calculation of overall running time of a given piece of code.

1. Consecutive Statements (Add the Complexity)
We should add the time complexity of each statement to calculate the total time complexity. For example if we have 3 lines of code with O(1), O(log n) and O(n) complexity respectively, then the total time complexity would be O(1)+O(log n)+O(n) = ~O(n)

In case of if-else condition, we should include the time complexity of condition and if or else part, whichever is larger.

2. Iterations and Loops - for, while and do-while (Multiply the Complexity)
Total time complexity can be calculated by multiplying the Time Complexity of individual statement with the number of iterations. for example, in the below code

```
for(int i=0;i<N;i++){ // N iterations
    PriorityQueue.offer(i);  // O(log k)
}
```

Time Complexity = O (log k) x O(n) = O(n log k)

In case of nested loops (Multiply the Complexity)
We should start analyzing from the **inner most loop first** and then start outwards. Time complexity is calculated by multiplying the running time of inner most loop with the outer loops.

```
for(int i=0;i<N;i++){ // N iterations
    for(int i=0;i<N;i++){ // N iterations
        PriorityQueue.add(i);  // Time Complexity = O (log k) where k=number of elements in priority queue
    }
}
```

Time Complexity = O (log k) x O (n) x O(n) = O (n^2 log k)

3. Logarithmic Time Complexities (Logarithmic Complexity)
In certain scenarios, the problem size is cut by 1/2 on each subsequent iteration and the resulting time complexity is O(log n), for example

```
for(int i=1;i<N;){ // N iterations
    list.add(i);  // O(1) or constant
    i=i*2; // this will cut the problem size by 1/2 on each invocation
}
```

Time Complexity = constant x O(log n) = O(log n)

Q 16. What is a sorting algorithm? List down sorting algorithms by their time & memory complexity in Big O notation? When do we call a sorting algorithm 'Stable'?

Sorting is an algorithmic technique to put all the collection elements in certain order[1] i.e. numerical order, lexicographical order (dictionary order), etc. Sorting is very helpful for solving real world problems for example, data analysis requires searching which depends upon sorted input data.

Classification of Sorting Algorithms

There can be different criteria for sorting algorithm classification, for example -
1. Comparison based vs non-comparison based algorithms

#	Comparison Based Sorting	Non Comparison (Integer Sort, etc)
Requires	Comparator is required to compare items	Faster than comparison algorithms because these algorithms utilizes integer arithmetic on keys
Examples	Quick Sort, Merge Sort, Heap Sort, Tree Sort, Selection Sort, Bubble Sort, Insertion Sort, etc	Counting Sort, Radix Sort, Bucket Sort, Polyphase Sort, etc.
Performance	Lower bound is O (n log n)	Lower bound is O (n)
Memory	O (1), In Place is possible	O (n)

2. Type of Memory - Internal Sorting algorithms and External Sorting algorithms

#	Internal Sorting	External Sorting (Distributed Algorithms)
Requires	Internal sorting takes place in the main memory, utilizing the Random Access Nature of the main memory. Sufficient Main Memory should be available for the input data.	Used when the input data is too big for the Main Memory of the machine. Interim results are stored on the external storage (Hard Drive, Main memory of other computer, etc)
Suitable Algorithms	Quick Sort, Merge Sort, Heap Sort, Tree Sort, Selection Sort, Bubble Sort, Insertion Sort, Counting Sort, etc	Merge Sort, Radix Sort, Bucket Sort, Polyphase sort, etc.
Performance	Faster due to random access of Main Memory	Slower (depends on type of external storage, seek time, number of reads and writes to memory)
Advantage	Faster	Distribution sorting algorithms i.e. individual subsets are separately sorted on different processors, then combined.
Example	Sorting 1 million integers on 4 GB RAM machine	Sorting 4 Billion integers using 10 MB RAM and a Hard Disk (or cluster of such machines)

3. Stable vs unstable sorting algorithms
An algorithms is said to be stable if it maintains the relative order of records where keys are equal. For equal Keys, Stable sort preserves the Order, so that if one element came before the other in input, it will also come before the other in output.
For example, suppose the following key-value pair need to be sorted based on key in ascending order
INPUT --> [(2,3) (1,2) (1,3) (3,1)]

Now there are two solution possible for the first two elements

1 http://en.wikipedia.org/wiki/Sorting_algorithm

OUTPUT1 --> [(1,2) (1,3) (2,3) (3,1)] --> stable sort because order is maintained
OUTPUT2 --> [(1,3) (1,2) (2,3) (3,1)] --> unstable sort because order changed from the original
Examples of Stable Sort algorithms are : Binary Tree Sort, Bubble Sort, Merge Sort, Insertion Sort, etc
Unstable Sorting Algorithms : Heap Sort, Selection Sort, Quick Sort

4. Computation complexity of swaps (In Place algorithms)
Certain algorithms allows in memory swap of elements to perform the sorting thereby offering O (1) space complexity. Example algorithms that allows In Place sorting are - bubble sort, selection sort, insertion sort, heap sort and shell sort. Quick sort is kind of in place but requires O (log n) space to keep track of recursive calls as a part of divide and conquer strategy thus it can not be called In Place algorithm.

5. Adaptive Sort
An algorithm is called adaptive if it takes advantage of existing order in its input thereby reducing the overall sorting time. Adaptive versions exists for heap and merge sort. For example, Java 8's iterative merge sort method is adaptive to an extent that it requires approximately n comparisons if the input is nearly sorted.

Algorithms Summary

Below table assumes total n items to be sorted, with keys of size k, digit size d and range of numbers r

Algorithm	Average Time Complexity	Worst Time Complexity	Space Complexity	Stable	Comparison Based?	Suitable for Memory
Quicksort	O (n log n)	n^2	log n	No	Yes	Internal
Binary Tree Sort	O (n log n)	n log n	n	Yes	Yes	Internal
Merge Sort	O (n log n)	n log n	n	Yes	Yes	External
Selection Sort	O (n^2)	n^2	1 (In Place)	No	Yes	Internal
Bubble Sort	O (n^2)	n^2	1 (In Place)	Yes	Yes	Internal
Heap Sort	O (n log n)	n log n	1 (In Place)	No	Yes	Internal
Insertion Sort	O (n^2)	n^2	1 (In Place)	Yes	Yes	Internal
Radix Sort	O n.(k/d)	n.(k/d)	$n+2^d$	No	No	External
Counting Sort	O (n+r)	n+r	n+r	Yes	No	Internal

Question: Do you know what Sorting algorithm JDK uses for Java's Collections.sort(List<E>) method?

Java 8's Collections.sort(List<E>) uses Iterative merge sort algorithm, it requires fewer than n log(n) comparisons when the input array is partially sorted (**adaptive**) and this algorithm is guaranteed to be stable in nature.

Sorting Examples in Java 8

1. Sort Array of Strings ignoring the case and print them to System out

```
public void sortStrings() {
    String[] names = {"One", "Two", "Three", "Four", "Five", "Six"};
    Stream.of(names).sorted(String::compareToIgnoreCase).forEach(System.out::println);
}
```

2. Sort String based on their length

```
private void sortStringsBasedOnLength() {
    String[] names = {"One", "Two", "Three", "Four", "Five", "Six", "Seven"};
    Stream.of(names)
            .sorted((o1, o2) -> Integer.compare(o1.length(), o2.length()))
            .forEach(System.out::println);
}
```

3. Parallel Sort employees by hire date and print them to console

```
private void sortEmployees(List<Employee> employees){
    employees.parallelStream()
            .sorted((o1, o2) -> o1.getHireDate().compareTo(o2.getHireDate()))
            .forEach(employee -> System.out.println("employee = " + employee));
}
```

4. Shortcut method for last example -

```
private void sortEmployees2(List<Employee> employees) {
    employees.parallelStream()
            .sorted(Comparator.comparing(Employee::getHireDate))
            .forEach(employee -> System.out.println("employee = " + employee));
}
```

5. Multiple Sort Criteria - Sort employees by first name and then by last name and print output to console

```
public void multiple_sort(List<Employee> employees) {
    Comparator<Employee> byFirstName = (e1, e2) -> e1.getFirstName().compareTo(e2.getFirstName());
    Comparator<Employee> byLastName = (e1, e2) -> e1.getLastName().compareTo(e2.getLastName());
    employees.stream()
            .sorted(byFirstName.thenComparing(byLastName))
            .forEach(e -> System.out.println(e));
}
```

Counting Sort Algorithm [Integer Sorting with runtime O (n)]

Counting sort is a non-comparison integer sorting algorithm that works when the minimum and maximum value of data are known. It is faster than comparison based sorting algorithms because it utilizes integer arithmetic on keys. Counting sort is efficient only if the range of input data (say 10-10000) is not significantly greater than the number of items to be sorted (10k), otherwise the comparison based sorting algorithm yield better performance.

Counting sort is Stable sorting with Time Complexity = O (n + k) where n = number of elements and k = range of elements

Pseudo Algorithm

Let us sort this data array using Counting Sort -

Input Data	1	4	1	2	6	5	2

1. Take a count array to store the count of each unique number

Index	0	1	2	3	4	5	6
Count	0	2	2	0	1	1	1

2. Modify the count array to store the prefix sum of its elements. Prefix sum is the cumulative sum of a sequence of numbers x0, x1, x2 ... is a second sequence of numbers y0, y1, y2 ..., the sum of prefixes (running totals till a given array position) of the input sequence. This gives us the position of each element in the output array i.e. the total number of elements occurring before the current item in the sorted output. This is also called histogram of counts.

Index	0	1	2	3	4	5	6
Prefix Sum	0	2	4	4	5	6	7

3. For each element in the input sequence, calculate its position from the counting array followed by decreasing the count by one in counting array. Position of 1 is 2, decrease count by 1 to place next data 1 at index 1 smaller than this, and so on position of 4 is 5 in the output array

Sorted Output	1	1	2	2	4	5	6

Java 8 example for Counting Sort

```
import java.util.Random;
public class CountingSort {
   public static void main(String[] args) {
      Random random = new Random(System.currentTimeMillis());
      int min = 0;
      int max = 1000;
      int[] input = random.ints(min, max).parallel().limit(100000000).toArray();
      CountingSort countingSort = new CountingSort();
      countingSort.sort(input, min, max);
   }

   public int[] sort(int[] input, int min, int max) {
      int counting[] = new int[max - min + 1];
      //Compute the count of each item
      for (int number : input) {
         ++counting[number - min];
      }


      //Compute the total number of items occurring before the current item in sorted output (histogram)
      for (int i = 1; i < counting.length; i++) {
         counting[i] += counting[i - 1];
      }


     //Fill the output array with correct number of zeros, ones, twos and so on.
      int[] output = new int[input.length];
      for (int i : input) {
         output[counting[i - min] - 1] = i;
         --counting[i - min];
      }

     return output;
   }
```

Layman usage in real life - A shopkeeper wants to sort the receipts at the year end from a box, he will simply take a calendar and start putting receipts against the dates.

What is Prefix Sum

Prefix sum is the cumulative sum of a sequence of numbers x0, x1, x2 ... is a second sequence of numbers y0, y1, y2 ..., the sum of prefixes (running totals till a given array position) of the input sequence -

$y_0 = x_0$
$y_1 = x_0 + x_1$
$y_2 = x_0 + x_1 + x_2$
$y_2 = x_2 + y_1$

Example of prefix sum for a given input array -

Input Sequence	1	2	3	4	5	6	7
Prefix Sums (running totals)	1	3	6	10	15	21	28

Other Variants of Source Code -

```
   public static void countingSort(int[] array, int min, int max) {
```

```
        int[] count = new int[max - min + 1];
        for (int number : array) {
            count[number - min]++;
        }
        int z = 0;
        for (int i = min; i <= max; i++) {
            while (count[i - min] > 0) {
                array[z] = i;
                z++;
                count[i - min]--;
            }
        }
    }


    public static void countingSort2(int[] a, int low, int high) {
        int[] counts = new int[high - low + 1]; // this will hold all possible values, from low to high
        for (int x : a)
            counts[x - low]++; // - low so the lowest possible value is always 0

        int current = 0;
        for (int i = 0; i < counts.length; i++) {
            Arrays.fill(a, current, current + counts[i], i + low); // fills counts[i] elements of value i + low in current
            current += counts[i]; // leap forward by counts[i] steps
        }
    }
```

Note: we know that, given an array of integers, its maximum and minimum values can be always found; but if we imagine the worst case for an array of 32 bit integers, we see that in order to hold the counts, we need an array of 2^{32} elements, i.e., we need, to hold a count value up to 2^{32}-1, more or less 4 Gbytes. So the counting sort is more practical when the range is (very) limited and minimum and maximum values are known a priori. (Anyway sparse arrays may limit the impact of the memory usage)

Bucket Sort Algorithm (Integer Sort with Time Complexity O (n) + distributed algorithm)

It is a distribution sort algorithm that works by partitioning (divide and conquer) an array into a number of buckets, with each bucket sorted individually on the same machine or another using a different sorting algorithm or by applying the same algorithm recursively.

A typical Bucket Sort program looks like -

```
    import java.util.*;

    public class BucketSort {
        public static void sort(int[] a, int maxVal) {
            int[] buckets = new int[maxVal + 1];
            for (int i = 0; i < buckets.length; i++) {
                buckets[i] = 0;
            }
            for (int i = 0; i < a.length; i++) {
                buckets[a[i]]++;
            }
            int outPos = 0;
            for (int i = 0; i < buckets.length; i++) {
                for (int j = 0; j < buckets[i]; j++) {
                    a[outPos++] = i;
                }
```

```
        }
    }

    public static void main(String[] args) {
        int maxVal = 7;
        int[] data = {7, 4, 1, 4, 1, 0, 5, 2, 3, 1, 4};
        System.out.println("Before: " + Arrays.toString(data));
        sort(data, maxVal);
        System.out.println("After:  " + Arrays.toString(data));
    }
}
```

Bucket Sort works as follows :

1. An array of buckets is created with size equals maxValue in the raw data.
2. Each of the bucket is initially set to zero.
3. A pass is made through the input array, counting the number of occurrence of each value between 0 and maxValue-1, store this count in the bucket's respective index.
4. Sorted result is produced by first placing the required number of zeros in the array, then required number of ones, followed by twos, threes and so on, up to maxVal - 1.

If the number of buckets can't be made equal to the max value in the input data, then we can use the below modified algorithm to sort the input data

```
public static int[] bucketSort(int[] array, int bucketCount) {
    if (bucketCount <= 0) throw new IllegalArgumentException("Invalid bucket count");
    if (array.length <= 1) return array; //trivially sorted

    int high = array[0];
    int low = array[0];
    for (int i = 1; i < array.length; i++) { //find the range of input elements
        if (array[i] > high) high = array[i];
        if (array[i] < low) low = array[i];
    }
    double interval = ((double) (high - low + 1)) / bucketCount; //range of one bucket

    ArrayList<Integer> buckets[] = new ArrayList[bucketCount];
    for (int i = 0; i < bucketCount; i++) { //initialize buckets
        buckets[i] = new ArrayList();
    }

    for (int i = 0; i < array.length; i++) { //partition the input array
        buckets[(int) ((array[i] - low) / interval)].add(array[i]);
    }

    int pointer = 0;
    for (int i = 0; i < buckets.length; i++) {
        Collections.sort(buckets[i]); //mergeSort
        for (int j = 0; j < buckets[i].size(); j++) { //merge the buckets
            array[pointer] = buckets[i].get(j);
            pointer++;
        }
    }
    return array;
}
```

Q 17. Why Prime Numbers are so important for certain algorithms like RSA & hashcode?

Prime numbers are always unique and can not be divided by any other number except 1. Infact all integer numbers (except 0 and 1) are made up of primes or their composites.

Prime numbers are very useful in generating hashcode, RSA algorithm and random number generators.

RSA Algorithm & Prime Numbers

RSA encryption algorithm which is commonly used in secure commerce web sites, is based on the fact that it is easy to take two big prime numbers and multiply them, while it is computationaly difficult to do the opposite (factorize a big number into two prime numbers) on the current hardware.

Hashcode Implementation

String class's hashcode method multiplies its hash value by prime number 31:

```
int hash =0;
 for (char ch : str.toCharArray()) {
    hash = 31 * hash + ch;
 }
```

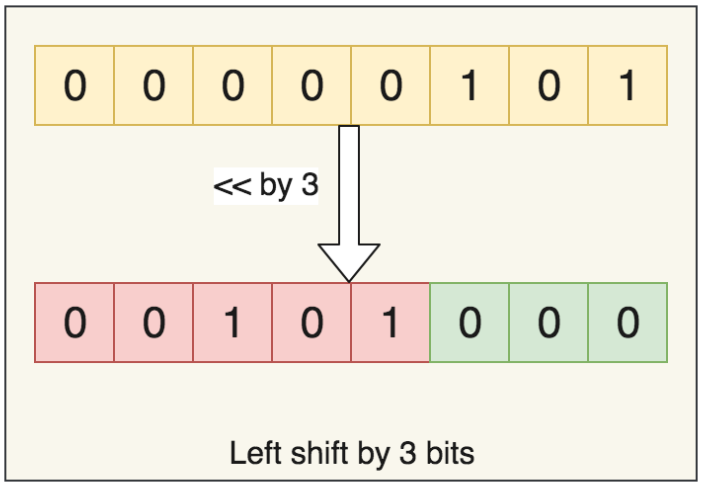
A number is either prime number or a composite number (can be factorized into prime numbers). The product of prime number with any other number has the best chances of being unique (though not as unique as Prime number itself) due to the fact that prime number is used to compose it. This property makes them very suitable for use in hashing function so as to obtain *fair distribution* in its hashcode output and thus achieving low collisions.
Multiplying by the prime number will not tend to shift information away from the low end, as it would multiplying by a power of 2, thus achieving a fair randomness.

Q 18. What is left shift <<, right shift >> and Unsigned right shift >>> operator in Java? How are these useful?

All Integer in Java are of signed type (negative numbers are represented in 2's complementary notation), hence Java provides both signed and unsigned bit shift operators to support signed and unsigned shift of bits.

Left Shift Operator << (Signed)

It shifts the underlying bits of an integer to left by the given distance filling the right most bits with zero always.
X = a << b means the same as X = a*2^b
a is given Number and b is the shift amount.
Here is an example of 8 bit representation of number 5. and when we left shift it's bit by 3 then the right most 3 bits are filled by zero.
And the number becomes
$5*2^3 = 40$.
The same thing happens for negative numbers which are represented in 2's complementary notation. for example -5 becomes -40 as follow
11111011 becomes 11011000

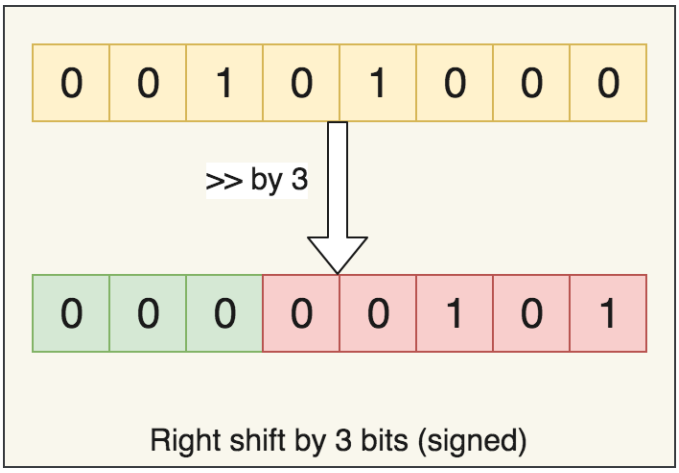


Right Shift Operator >> (Signed)

Shifts the bits to left by specified amount maintaining the sign of underlying integer i.e.

It fills the left most bits with 0 if the number is positive otherwise with bit 1.

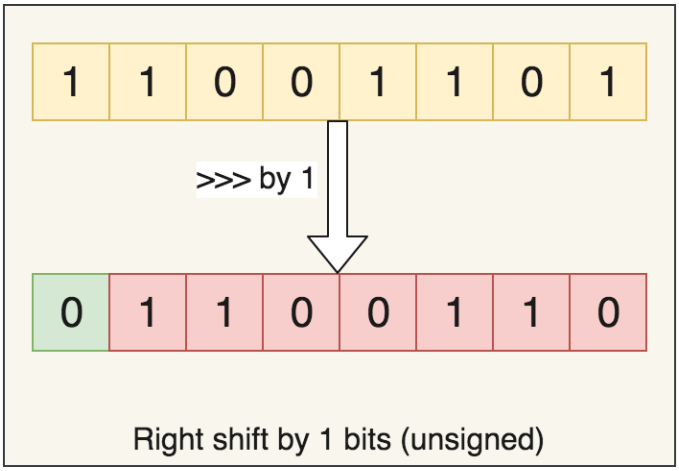
X = a >> b means same as arithmetic operation X = a / (2^b)



Unsigned right shift Operator >>> (does not respect sign of Number, does not preserve the 1st bit)

Unsigned right shift operator >>> is effectively same as >> except that it is unsigned, it fills the left most positions with bit 0 always. (Irrespective the sign of the underlying number)
For example,
1100 1100 >>> 1 becomes 0110 0110
(shown in diagram)
10000000 >>> 3 becomes 10000 in binary

256 >> 3 becomes 256 / 2^3 = 16.



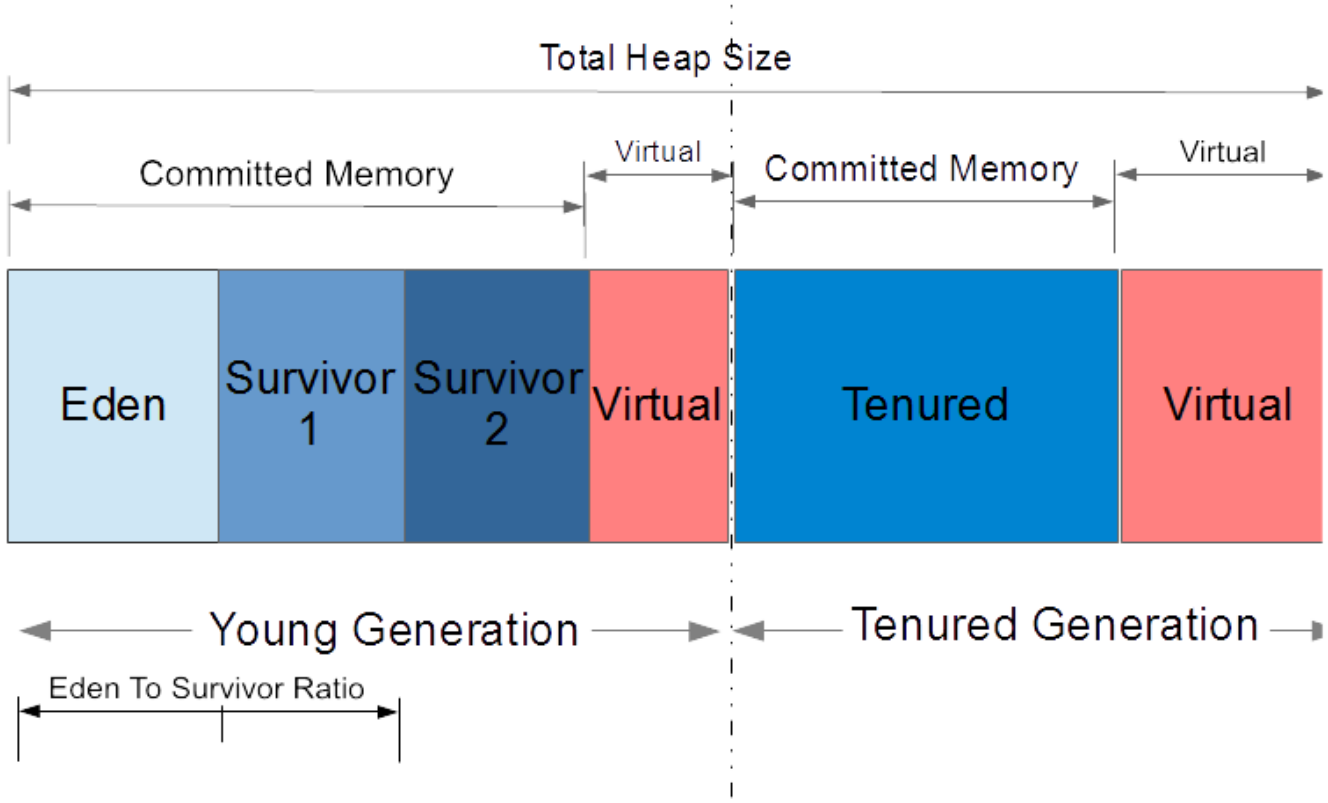
Notes

- Eight-bit type ***byte*** is promoted to ***int*** in shift-expressions. To mitigate such effects we can use bit masking to get the result as byte for example, (b & 0xFF) >>> 2. Casting can also help achieving the same.
- **Why there is no need of unsigned left shift?**
  Because there is no need to have that. Sign bit is the right most bit of an integer and shifting bits to right only require the decision of sign. Logical and arithmetic left-shift operations are identical so << signed solves the purpose of unsigned left shift as well.
- **Uses of bitwise operators:** bitwise operators are used for few very efficient mathematical calculations in Big O(1). Bloom Filter, fast mathematical calculations, hashing functions of HashMap are some of applications.

Q 19. How heap memory is divided in Java. How does Garbage Collector cleans up the unused Objects? Why shouldn't we use System.gc() command in production code?

Memory taken up by the JVM is divided into Stack, Heap and Non Heap memory areas. Stacks are taken up by individual threads for running the method code while heap is used to hold all class instances and arrays created using new operation. Non-heap memory includes a method area shared among all threads and is logically part of the heap but, depending upon the implementation, a Java VM may not invoke GC on this part.

Java HotSpot VM Heap Memory is divided into Generations[1]



The Young generation - This further consists of one Eden Space and ***two*** survivor spaces. The VM initially assigns all objects to Eden space, and most objects die there. When VM performs a minor GC, it moves any remaining objects from the Eden space to one of the survivor spaces.

Tenured/Old Generation - VM moves objects that live long enough in the survivor spaces to the "tenured" space in the old generation. When the tenured generation fills up, there is a full GC that is often much slower because it involves all live objects.

Metaspace - The metaspace holds all the reflective data of the virtual machine itself, such as class metadata, classloader related data. Garbage collection of the dead classes and classloaders is triggered once the class metadata usage reaches the “MaxMetaspaceSize”.

Never invoke GC programmatically from within your code

Invoking System.gc() may have significant performance side effects on the application. GC by its design is intelligent piece of code and it knows when to invoke partial or full collection. Whenever there is a need it tries

1 http://docs.oracle.com/javase/7/docs/technotes/guides/management/jconsole.html
http://www.oracle.com/technetwork/java/javase/gc-tuning-6-140523.html

continued on 36

to collect the space from Young Generation First (very low performance overhead), but when we force our JVM to invoke System.gc(), JVM will do a Full GC which might pause your application for certain amount of time, isn't that a bad approach then? Let GC decide its timing.

Memory Spaces

Eden Space (heap): The pool from which memory is initially allocated for most objects.

Survivor Space (heap): The pool containing objects that have survived the garbage collection of the Eden space.

Tenured/Old Generation (heap): The pool containing objects that have existed for some time in the survivor space.

Metaspace (non-heap): The pool containing all the reflective data of the virtual machine itself, such as meta-data of classes, objects (e.g pointers into the heap where objects are allocated) and method objects, classloader related data.

Code Cache (non-heap): The HotSpot Java VM also includes a code cache, containing memory that is used for compilation and storage of native code.

Performance Tuning GC[2]

Set appropriate heap using -Xms and -Xmx VM parameter. Unless you have GC pause problem, you should give as much memory as possible to the virtual machine.

-XX:+DisableExplicitGC
Disable Sysytem.gc() which cause the Full GC to run and thus causing the JVM pauses.

-verbose:gc
-XX:+PrintGC
-XX:+PrintGCDetails
-XX:+PrintGCTimeStamps
This will print every GC details

-XX:NewRatio
The ratio between the young space and the old is set by this parameter. For example, -XX:NewRatio=2, would make old generation 2 times bigger than the young generation (ratio between the young and tenured generation is 1:2), or we can say that the young generation is 1/3rd the size of total heap size(young + old)

-XX:SurvivorRatio
This command line parameter sets the ratio between each survivor space and eden. For example, -XX:SurvivorRatio=6 will make each survivor space one eighth of the young generation. (there are two survivor space and 6 eden spaces in this case, hence 1/8)

-XX:NewSize=n
Sets the initial size of young generation, it should typically be 1/4th of total heap size. The bigger the young generation, the less frequent the minor collection happens. (though for a bounded heap size, it may cause more frequent major collections)

-XX:MaxMetaspaceSize=128m

2 http://www.oracle.com/technetwork/java/javase/gc-tuning-6-140523.html#generation_sizing.young_gen.survivors

Sets the maximum metaspace size (non-heap) to 128 MB which stores Classes, methods and other metadata.

We should carefully design the object pool because they fool the garbage collector by keeping the live reference to the unused objects, thus causing application to demand more memory.
Default Values as of JDK 1.6 on server VM

New Ratio = 2 (old generation is 2 times bigger than young generation)
New Size = 2228K
Max New Size = Not limited
Survivor ratio = 32 (survivor space will be 1/34th of young generation)

The rules of thumb for server applications are[3]

- First decide the maximum heap size you can afford to give the virtual machine. Then plot your performance metric against young generation sizes to find the best setting.
  - Note that the maximum heap size should always be smaller than the amount of memory installed on the machine, to avoid excessive page faults and thrashing.
- If the total heap size is fixed, increasing the young generation size requires reducing the tenured generation size. Keep the tenured generation large enough to hold all the live data used by the application at any given time, plus some amount of slack space (10-20% or more).
- Subject to the above constraint on the tenured generation:
  - Grant plenty of memory to the young generation.
  - Increase the young generation size as you increase the number of processors, since allocation can be parallelized.

Notes

Question: We have a application which creates millions of temporary large StringBuilder Objects from multiple threads. But none of such object is really required after extracting useful information from them. Somehow we started facing frequent gc pauses. What could be the problem, and how would you approach it?
Solution
Performance tuning GC may solve this problem to some extent. Let's first understand memory requirements of this application. This application create lots of short lived objects - thus we would require a large young generation for lowering the frequency of minor garbage collection. If our young generation is small, then the short lived objects will be promoted to Tenured Generation and thus causing frequent major collection. This can be addressed by setting appropriate value for -XX:NewSize parameter at the JVM startup.
We also need to adjust the survivor ratio so that the eden space is large compared to survivor space, large value of Survivor ratio should help solve this problem.
We can also try increasing the Heap size if we have sufficient memory installed on our computer.

Sample Settings for increasing Eden Space and New Generation
java -client -XX:SurvivorRatio=12 -XX:NewRatio=2 -XX:NewSize=50m -Xmx256m -Xms64m
-XX:MaxMetaspaceSize=128m -XX:+PrintGCDetails -jar dli-downloader-4.2-jar-with-dependencies.jar

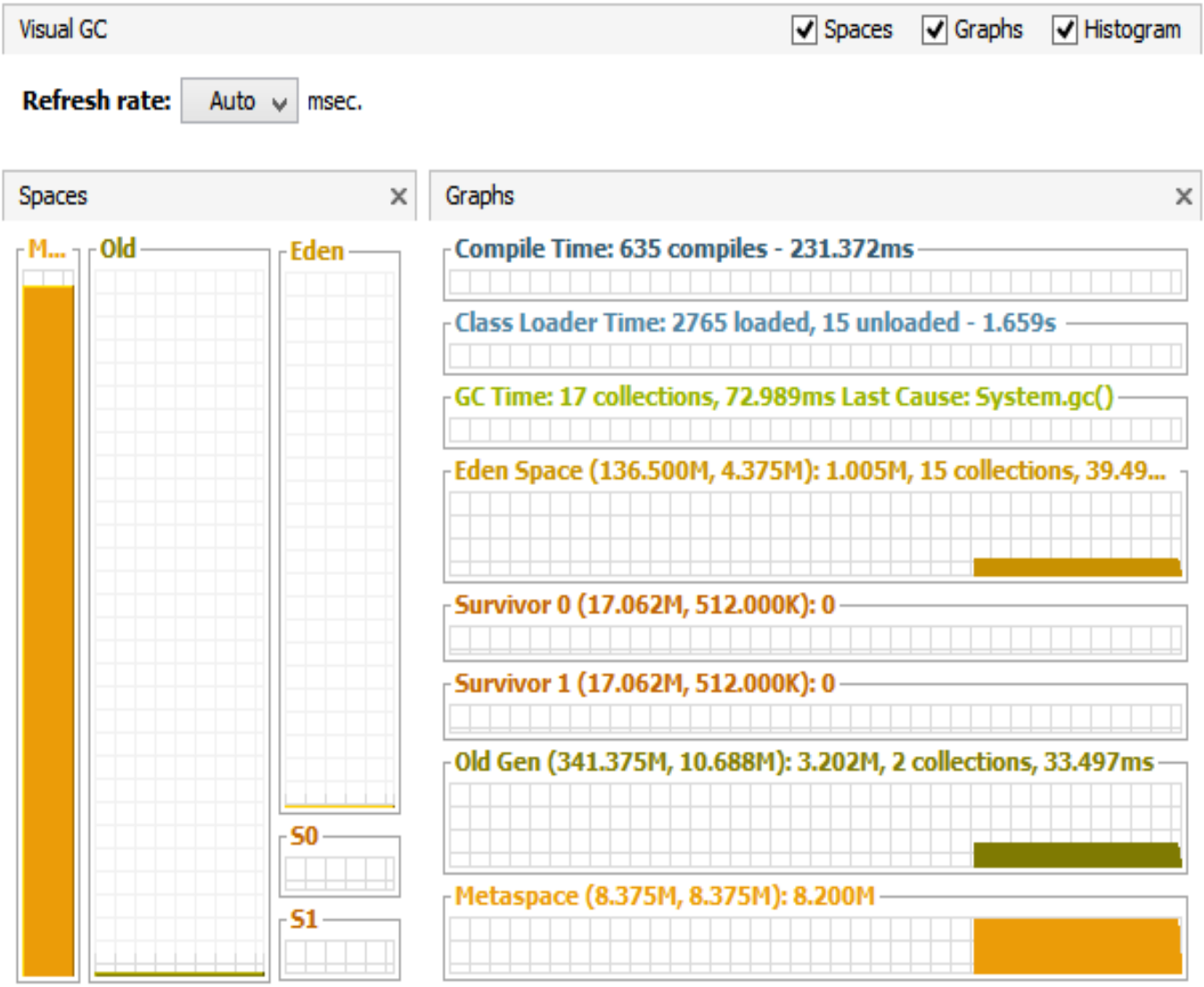
Question: Does GC collects memory from Perm Gen Space?
Solution : The PermGen space is garbage collected like the other parts of the heap. PermGen contains meta-data of classes and objects (pointers to heap memory allocation). It also includes ClassLoaders that need to be manually destroyed at the end of their use.

Question : What are the available tools to give the visual view of the different memory spaces in a

3 http://www.oracle.com/technetwork/java/javase/gc-tuning-6-140523.html#generation_sizing.young_gen.survivors

running JVM?

There are lot of free tools available for troubleshooting memory related problem in a JVM. JConsole and JVisualVM are two of them that come shipped with every JDK. Below is the screenshot of JVisualVM (with Visual GC plugin) showing the visual representation of the different memory segments for a running JVM.



VisualVM snapshot with VisualGC plugin

You can always profile an application and see the memory trends and customize the memory allocations accordingly. That can significantly reduce GC overhead and thus improve the application performance.

How do you interpret GC log message , like the one shown below?

8109.128: [GC [PSYoungGen: 109881K->14201K(139914K)] 691015K->595352K(1119040K), 0.0454530 secs]

- 107Mb used before GC, 14Mb used after GC, max young generation size 137Mb
- 675Mb heap used before GC, 581Mb heap used after GC, 1Gb max heap size
- minor GC occurred 8109.128 seconds since the start of the JVM and took 0.04 seconds

Q 20. What is difference between Stack and Heap area of JVM Memory? What is stored inside a stack and what goes into heap?

The biggest difference between Heap and Stack section of memory is the lifecycle of the objects that reside in these two memory locations

Memory of Stack Section is bound to a method context and is destroyed once a thread returns from the function i.e. the Stack objects exists within the scope of the function they are created in.

On the other hand Heap objects exists outside the method scope and are available till GC recollects the memory.

Java stores all objects in Heap weather they are created from within a method or class. Escape analysis can be enabled in compiler to hint JVM to create method local objects in stack if the objects does not escape the method context. All class level variables and references are also stored in heap so that they can be accessed from anywhere. Metadata of classes, methods, etc also reside in Heap's PermGen space.

The Stack section of memory contains methods, local variables and reference variables and all os these are cleared when a thread returns from the method call.

Question: An ArrayList is created inside a method, will it be allocated in Stack section or Heap section of JVM Memory?

```
public void foo(){
   ArrayList<String> myList = new ArrayList<>();
}
```

Answer : All Java Objects are created in Heap memory section, so the ArrayList will be created on the heap. But the local reference (myList) will be created in the Stack section of memory. Once the method call is finished and if myList variable is not escaped from this method then GC will collect the ArrayList object from heap.

As of JDK 1.6_14, escape analysis[1] can be enabled by setting the appropriate JVM flag (java -XX:+DoEscapeAnalysis) which hints the compiler to convert heap allocations to stack allocations if the method local objects do not escape the method scope.

In the following code, if we enable the escape analysis, then the Object Foo may be created on Stack, resulting in significant performance gain due to lesser GC activity.

```
public static void main(String[] args) {
   System.out.println("start");
   for (int i = 0; i < 1000 * 1000 * 1000; ++i) {
      Foo foo = new Foo();
   }
   System.out.println(Foo.counter);
}
```

1 http://docs.oracle.com/javase/7/docs/technotes/guides/vm/performance-enhancements-7.html

Q 21. What is a Binary Tree? What are its usecases?

Binary Tree is a tree data structure made up of nodes. Each node has utmost two children.

Why to prefer Binary Tree over any other linear data structure?

Binary trees are a very good candidate (not the best) for storing data when faster search/retrieval is required based on certain criteria. It does so by storing its elements in sorted order offering low time complexity for retrieval operations compared to any other linear data structure. Any un-sorted collection can be inserted into Binary Search Tree in O (n log n) time complexity. Though the insertion time is increased per element from O(1) in Random Access array to O(log n) in Binary Search Tree, but we get a major advantage when we want to search/retrieve a particular element from the tree data structure.

Worst-case Search time complexity is logarithmic in a balanced Binary Search Tree i.e. Binary tree cuts down the problem size by half upon every subsequent iteration.

Balanced Binary Tree

Binary Tree is useful only when the tree is balanced, because only in that case a Binary Tree provides O(log n) search complexity, otherwise a binary tree will behave more like a linear data structure with O(n) time complexity for searching. A tree is called balanced when the height of the tree is logarithmic compared to number of its elements.

Binary Search Tree

Left child of root is less in value than the right child. And the same is true for left and right sub tree in case of Binary Search Tree. BST is build for efficiently sorting & searching. In Order Traversal of a Binary Search Tree results in Ascending Order sorting of its elements.

Binary Tree Implementations used in Java 8

Red-black-tree (TreeMap) and binary heap (PriorityQueue) implementation of Binary Tree is provided by Java Collection Framework, both of which are thoroughly tested and easy to use.

Red-black-tree is a height balanced binary tree where root is colored black and every other element is colored either black or red with the following two rules,

1. If an element is colored red, none of its children can be colored red.
2. The number of black elements is the same in all paths from the root to the element with one child or with no children.

It is useful for maintaining the order of elements in the collection based on the given comparator. It also provide efficient mechanism to find the neighboring elements which are either big or small compared to given number, because those numbers are stored physically closer in the data structure.

Q 22. Discuss implementation and uses of TreeSet Collection?

Skill Set - Sorting Algorithm & Data Structure (Red Black binary tree)

TreeSet is a navigable set implementation based on TreeMap. All the elements are ordered using their Natural ordering or by comparator provided at TreeSet construction time.

NavigableSet provides us with methods like first(), last(), floor(), ceiling(), headSet(), tailSet() which can be used to search the neighboring elements based on element's ordering.

TreeMap is Red-Black Binary Search Tree which guarantees logarithmic time for insertion, removal and searching of an element. All the elements in this collection are stored in sorted order and the tree is height balanced using Red black algorithm. If two elements are nearby in order, then TreeSet places them closely in the data structure.

Uses

It is a best collection if we need to search the nearby elements of a given item based on their ordering.

Notes

- Note that this implementation is not synchronized. If multiple threads access a tree set concurrently, and at least one of the threads modifies the set, it must be synchronized externally. This is typically accomplished by synchronizing on some object that naturally encapsulates the set.
- If no such object exists, the set should be "wrapped" using the Collections.synchronizedSortedSet method. This is best done at creation time, to prevent accidental unsynchronized access to the set:

```
SortedSet s = Collections.synchronizedSortedSet(new TreeSet(...));
```

- If we are looking for high throughput in a multi-threaded application then we can prefer ConcurrentSkipListSet which is scalable concurrent implementation of NavigableSet.
- Iterator returned by this class are fail-fast.
- TreeSet does not allow duplicates, it just replaces the old entry with the new one if both are equal (using compareTo method)
- TreeSet does not preserve the insertion order of its elements.
- TreeSet provides guaranteed Big O (log n) time complexity for add(), remove() and contains() method.

Q 23. How does Session handling works in Servlet environment?

Skill Set - Web (HTTP protocol, Servlet)

There are multiple ways to handle session by a server framework. Most often, server uses one of the following three mechanisms to handle session on server side -

1. Storing Cookies on the client side
2. URL Rewriting
3. Hidden form fields

Servlets use cookies as the default mechanism for session tracking, but in case cookies are disabled on the client, Server can use URL re-writing for achieving the same. When server calls request.getSession(true), then server generates and sends JSESSIONID back to the client for all future session references. JSESSIONID will then be stored by the client and sent back to the server using any of the above mentioned mechanisms.
To ensure that your Servlets support servers that use URL rewriting to track sessions, you must pass all the URL's used in your servlet through the HttpServletResponse.encodeURL() method, as shown below

```
out.println("<form actionb ='"+res.encodeURL("/example/htmlpage")+"'>");
```

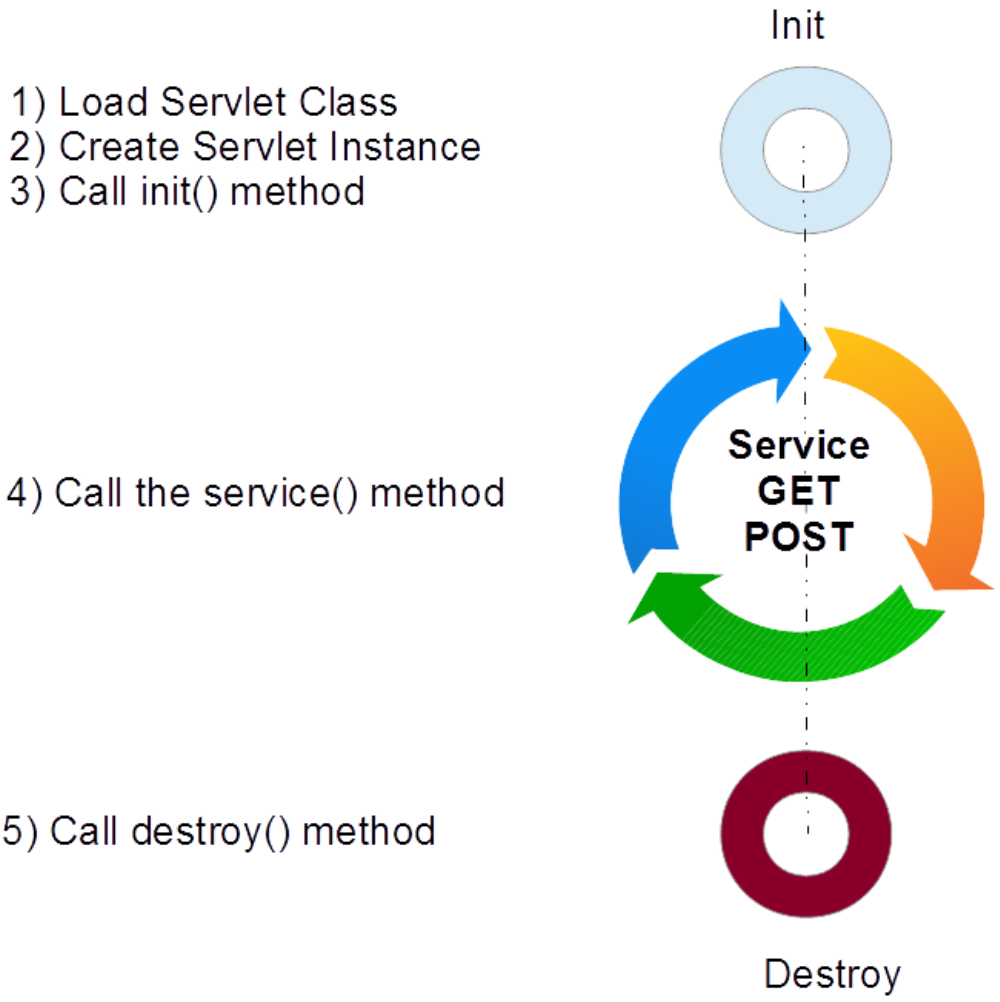
This will append the sessionID to the form's action.

Q 24. Explain Servlet Life Cycle in a Servlet Container?

Servlet Life Cycle consists of three main phases - construction, service and destroy. It is controlled by the container in which servlet has been deployed.

1. Construction - if the instance of servlet does not exist, the web container -
   i. Loads the servlet class.
   ii. Creates an instance of servlet class.
   iii. Initializes servlet instance by calling init() method
2. Service - invokes the service method, passing request and response objects.
3. Destroy - If the container needs to remove the servlet, it finalizes servlet by calling servlet's destroy method.

```
public interface Servlet {
public void init(ServletConfig config) throws ServletException;
public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException;
public void destroy();
}
```



Servlet Life Cycle

Q 25. How can one handle relative context path while coding the web applications? For example, your web application may be deployed at a different context path in Tomcat, how will you make sure static/dynamic resources works well at custom context path?

There are two main ways to handle relative Context Path -

1. Do not provide absolute context path in your dynamic/static web pages. So instead of mentioning absolute url (that starts with /), use something like this -

```
static/images/a.gif

../static/images/a.gif

<link href="resources/css/style.css" rel="stylesheet"  type="text/css" />
```

2. Use API to handle relative Context Path, for example

Spring MVC Framework

One can use spring tag to access the resources (static/dynamic)

```
<img src="<spring:url value='/static/images/a.gif'/>"/>
```

JSP EL

One can use pageContext variable available in session scope. Request contains contextPath variable which points to the actual context path assigned to web application at the time of deployment.

```
<link href="${pageContext.request.contextPath}/resources/css/style.css" rel="stylesheet"  type="text/css" />
<img src="<%=request.getContextPath()%>/static/images/a.gif"/>

${pageContext.request.contextPath}
```

In Freemarker

In your view resolver you can add the following property (mvc-dispatcher.xml)

```
<bean id="viewResolver" class="org.springframework.web.servlet.view.freemarker.FreeMarkerViewResolver">
<property name="cache" value="true"/>
<property name="prefix" value=""/>
<property name="suffix" value=".ftl"/>
<property name="requestContextAttribute" value="rc"/>
</bean>
```

Then in your freemarker template you can get the request context patch like

```
${rc.getContextPath()}
```

or, simply as

```
${rc.contextPath}
```

Q 26. How will you write a simple Recursive Program?

A general structure of any recursion program is like this :

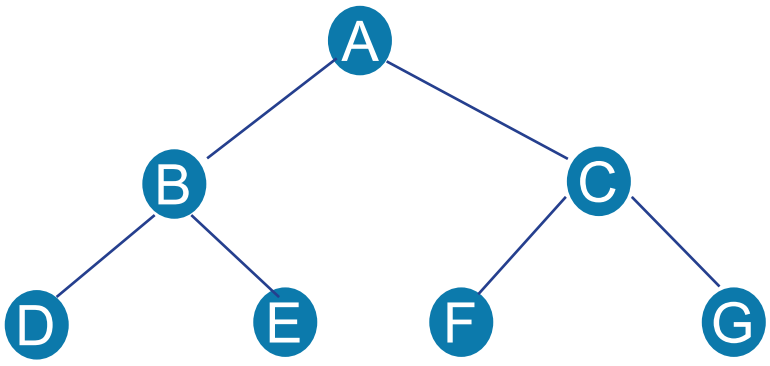
```
if (base condition...){
   // return some simple iterative expression
}
else {// recursive case
   //some work before call
   //recursive call
   //some work after call
}
```

Recursion is helpful in writing complex algorithms in easy to understand manner. But normally iterative solutions provide better efficiency compared to recursive one because of so much overhead involved in executing recursive steps.
For example, we would use the following code to calculate the **Fibonacci** series using recursion

```
public int fib(int n) {
   if (n <= 1)       //Base Condition
      return n;
   else {          //Recursive case
      return fib(n - 1) + fib(n - 2);
   }
}
```

Q 27. How many elements a complete binary tree could hold for a depth of 10?

A binary tree is said to be complete if it is fully populated, so that each node has two child except the child nodes.
From the figure shown, we can conclude that maximum
Nodes at level 0 = 1
Nodes at level 1 = 2
Nodes at level 2 = 4
Nodes at level n = 2^n



So maximum number of nodes nk_{max} at level k of a binary tree is $\mathbf{nk_{max} = 2^k}$

And the maximum number of nodes in a Tree with height (also called depth) h is
$\mathbf{N_{max} = 1+2+4+...+2^{h-1} = 2^h - 1}$

or in other words number of Levels= $\log_2(N+ 1)$

Hence, we can say

Total nodes in tree with depth 1 = 1
Total nodes in tree with depth 2 = 3
Total nodes in tree with depth 5 = 31
Total nodes in a tree with depth 10 will be = 2^{10}-1 = 1023 nodes

Q 28. How will you swap two numbers without any temporary variable?

There are two approaches for swapping numbers without using any temporary variable.

1. Using sum of numbers
2. Using XOR operation (preferred)

Lets explore both of these options.

Approach 1. Using sum of numbers

We can easily swap two numbers using summation approach, as illustrated in the below code snippet.

```
void method1() {
    int x = 10, y = 5;

    x = x + y; // x becomes 15
    y = x - y; // y becomes 15 - 5 = 10
    x = x - y; // x becomes 15 - 10 = 5
    System.out.println("After swap x = " + x + ", y = " + y);
}
```

The above code works as expected for smaller numbers, but there are chances of Integer overflow and underflow when number is too large.

Integer overflow & underflow

If it overflows, it goes back to the minimum value (Integer.MIN_VALUE = -231) and continues from there. If it underflows, it goes back to the maximum value (Integer.MAX_VALUE = 231) and continues from there. The main issue here is that overflow and underflow are silent and you will not see any exceptions if it occurs.

Approach 2. Using bitwise XOR operator

Best way to swap two numbers without falling into trap of integer overflow problem is to use XOR bitwise operator.

What is Bitwise XOR

A bitwise XOR takes two bit patterns of equal length and performs the logical exclusive OR operation on each pair of corresponding bits. The result in each position is 1 if only the first bit is 1 or only the second bit is 1, but will be 0 if both are 0 or both are 1. In this we perform the comparison of two bits, being 1 if the two bits are different, and 0 if they are the same.

Bitwise XOR Practical Use

Bitwise XOR can be used to flip the bits in a given number. Any bit may be toggled by XORing it with 1.

Java Code to swap two integer using XOR

```
void method2() {
    int x = 10;
    int y = 5;

    // Code to swap 'x' (1010) and 'y' (0101)
    x = x ^ y; // x now becomes 15 (1111)
    y = x ^ y; // y becomes 10 (1010)
    x = x ^ y; // x becomes 5 (0101)

    System.out.println("After swap: x = " + x + ", y = " + y);
}
```

Q 29. Explain working of a hashing data structure, for example HashMap in Java.

HashMap is a hashing data structure which utilizes object's hashcode to place that object inside map. It provides best case time complexity of O(1) for insertion and retrieval of an object. So it is a best suited data structure where we want to store a key-value pair which later on can retrieved in minimum time.

HashMap is not a thread safe ADT, so we should provide necessary synchronization if used in multi-threaded environment.

HashMap is basically an array of buckets where each bucket uses linked list to hold elements.

Initial Capacity

The default initial capacity of a hashmap is 16 (the number of buckets) and it is always expressed in power of two (2,4,8,16, etc) reaching maximum of 1 << 30 (2^{30})

Put Operation - Big O(1) Time Complexity

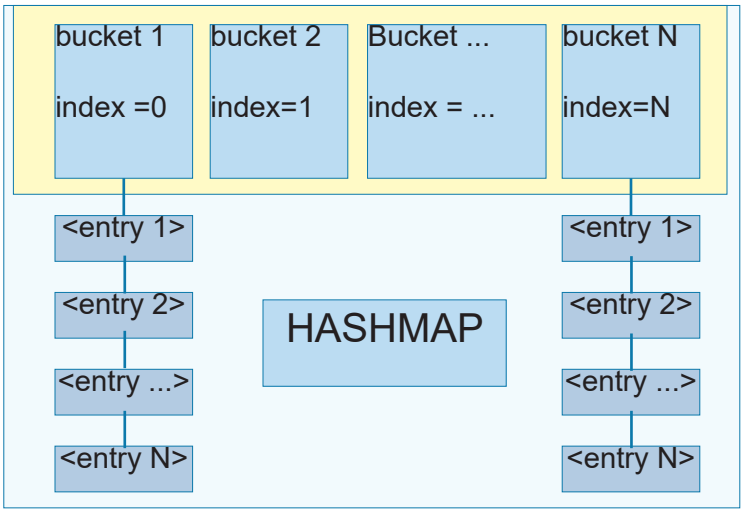
When we add a key-value pair to hashmap, it queries key's hashcode. Hashmap uses that code to calculate the bucket index in which to place the key/value. For example, if hashcode is zero then hashmap will place the key value in 0th bucket. Hashmap strips down the key's hashcode to fit into the existing count of buckets using a bitwise hack which is equivalent to the shown below,

bucket index = hashcode % (number of buckets)

The actual method is implemented as

```
int indexFor(int hashcode, int length) {
    return hashcode & (length-1);
}
```

The number of buckets in a hashmap is always power of two, i.e. 2,4,8,16, etc, otherwise the above mentioned bitwise method will not work correctly.
Once the bucket is identified, the key-value pair is added to the List lying at that bucket address.
When multiple objects map to same bucket, we call that phenomenon Collision.



A Typical Hashing Data Structure

Get Operation - Big O(1) Time Complexity

get operation takes a key and then calculates the index of bucket using the method mentioned above. Then that bucket's List is searched for the given key using key's equals() method, and finally the result is returned.

Load factor and Rehashing

Rehashing occurs automatically by the map when the number of keys in the map reaches threshold value.
threshold = capacity* (load factor of 0.75)
In this case a new array is created with more capacity and all the existing contents are copied over to it.

Q 30. Discuss internal's of a concurrent hashmap provided by Java Collections Framework.

In Java 1.8, A ConcurrentHashMap is a hashmap supporting full concurrency of retrieval via volatile reads of segments and tables without locking, and adjustable expected concurrency for updates. All the operations in this class are thread-safe, although the retrieval operations does not depend on locking mechanism (non-blocking). And there is not any support for locking the entire table, in a way that prevents all access. The allowed concurrency among update operations is guided by the optional concurrencyLevel constructor argument (default is 16), which is used as a hint for internal sizing.

ConcurrentHashMap is similar in implementation to that of HashMap, with resizable array of hash buckets, each consisting of List of HashEntry elements. Instead of a single collection lock, ConcurrentHashMap uses a fixed pool of locks that form a partition over the collection of buckets.

Here is the code snippet showing HashEntry class

```
static final class HashEntry<K,V> {
        final int hash;
        final K key;
        volatile V value;
        volatile HashEntry<K,V> next;
...
```

HashEntry class takes advantage of final and volatile variables to reflect the changes to other threads without acquiring the expensive lock for read operations.
The table inside ConcurrentHashMap is divided among **Segments** (which extends Reentrant Lock), each of which itself is a concurrently readable hash table. Each segment requires uses single lock to consistently update its elements flushing all the changes to main memory.

put() method holds the bucket lock for the duration of its execution and doesn't necessarily block other threads from calling get() operations on the map. It firstly searches the appropriate hash chain for the given key and if found, then it simply updates the volatile value field. Otherwise it creates a new HashEntry object and inserts it at the head of the list.
Iterator returned by the ConcurrentHashMap is fail-safe but weakly consistent. keySet().iterator() returns the iterator for the set of hash keys backed by the original map. The iterator is a "weakly consistent" iterator that will never throw ConcurrentModificationException, and guarantees to traverse elements as they existed upon construction of the iterator, and may (but is not guaranteed to) reflect any modifications subsequent to construction.
Re-sizing happens dynamically inside the map whenever required in order to maintain an upper bound on hash collision. Increase in number of buckets leads to rehashing the existing values. This is achieved by recursively acquiring lock over each bucket and then rehashing the elements from each bucket to new larger hash table.

Notes & Further Questions

Question: Is this possible for 2 threads to update the ConcurrentHashMap at the same moment?
Answer : Yes, its possible to have 2 parallel threads writing to the CHM at the same time, infact in the default implementation of CHM, at most 16 threads can write and read in parallel. But in worst case if the two objects lie in the same segment, then parallel write would not be possible.

Question: Can multiple threads read from a given Hashtable concurrently?
Answer : No, get() method of hash table is synchronized (even for synchronized HashMap). So only one thread can get value from it at any given point in time. Full concurrency for reads is possible only in ConcurrentHashMap via the use of volatile.

Question: Is Segment in ConcurrentHashMap similar to Bucket?
Answer : No, in fact Segment is like a mini specialized version of hashtable that contains many buckets. Each segment holds a single lock, thus no two entries in the segment can be updated by more than one thread at a time. Definition of Segment as of JDK 1.7 looks like -

```
static final class Segment<K,V> extends ReentrantLock implements Serializable {
transient volatile HashEntry<K,V>[] table;

final V put(K key, int hash, V value, boolean onlyIfAbsent) {
        HashEntry<K,V> node = tryLock()? null :
           scanAndLockForPut(key, hash, value);
...
```

Question: Can two threads read simultaneously from the same segment in ConcurrentHashMap?
Answer: Segments maintain table of entry list that are always kept in consistent state, thus many threads can read from the same Segment in parallel via volatile read access. Even the updates operations (put and remove) may overlap with the retrieval operation without any blocking happening.

Question: What enhancements were made to ConcurrentHashMap in Java 8?
Answer: few new methods related to concurrency has been added to CHM in Java 8

1. putIfAbsent (The entire method invocation is performed atomically)
2. compute (The entire method invocation is performed atomically)
3. computeIfAbsent (The entire method invocation is performed atomically)
4. computeIfPresent (The entire method invocation is performed atomically)
5. search (key, value)
6. reduce (key, value)
7. forEach

All these methods make concurrent programming a lot simpler than before, for example

- The below statement will conditionally create a new LongAdder() objects if none existed against the given word and then increment the counter by One.

```
map.putIfAbsent(word, new LongAdder());
map.get(word).increment();
```

- The blow statement will print the entire key-value pair from the Hashmap (threshold is parallelism threshold number beyond which multiple threads will execute the given operation)

```
map.forEach(threshold, (k, v) -> System.out.println(k + "->" + v));
```

- The below code snippet will increment the counter by one initializing to one if it is null

```
map.compute(word, (k, v) -> v == null ? 1: v+1);
```

- The below statement is another way of doing the same thing

```
map.computeIfAbsent(word, k -> new LongAdder()).increment();
```

- The below code snippet will search for the first match where value is greater than 100, returning null if nothing found

```
String result = map.search(threshold, (k, v) -> v > 100 ? k : null) ;
```

- The below code snippet will count entries that have value > 100

```
Long count = map.reduceValues(threshold, v -> v > 100 ? 1L : null, Long::sum);
```

For more details please refer to -
http://www.ibm.com/developerworks/java/library/j-jtp08223/
http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html

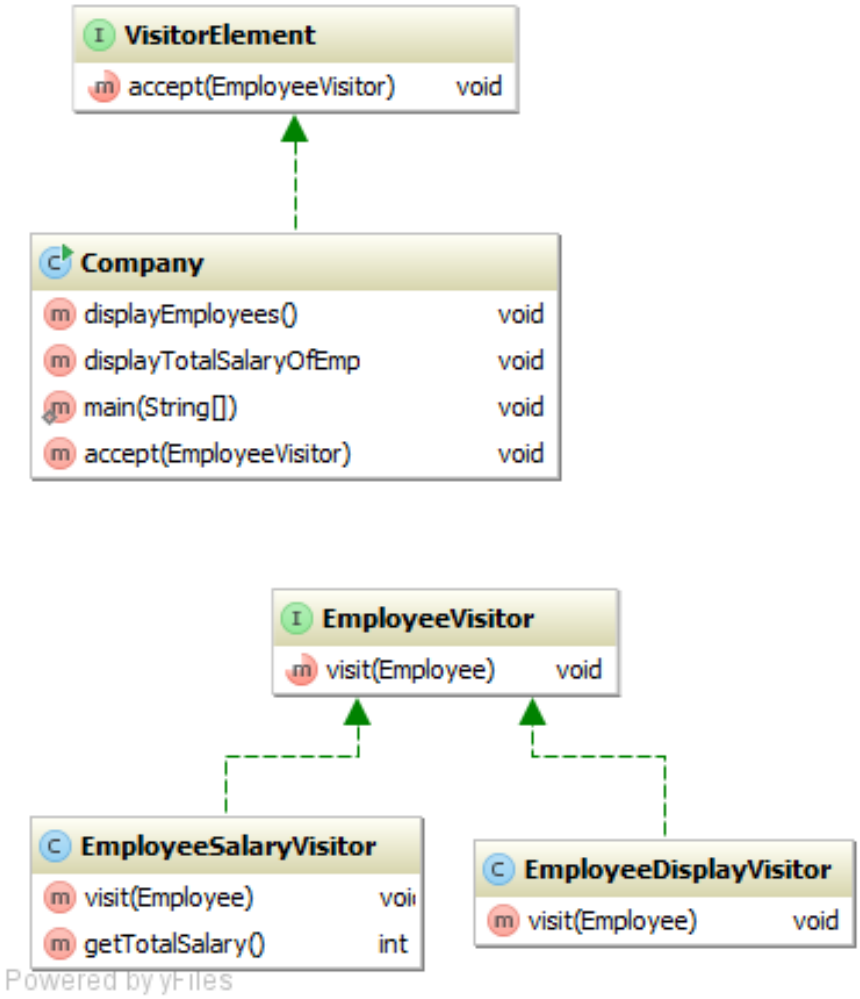
Q 31. Discuss Visitor, Template, Decorator, Strategy, Observer and Facade Design Patterns?

Visitor Design Pattern

Visitor design pattern is a way of separating an algorithm from an object structure on which it operates. This results in ability to add new operations to existing object structures without modifying those structures.
For example,
Suppose, A company wants to display the first name of all its employees. In order to achieve a separation between the data model and the algorithm to display employee, Visitor pattern can be utilized. Moreover if a new requirement comes in for calculating the total salary of an employee, then it can be easily added without making any changes to the model.

Similarly

Calculating taxes in different regions on sets of invoices would require many different variations of calculation logic. Implementing a visitor allows the logic to be de-coupled from the invoices and line items. This allows the hierarchy of items to be visited by calculation code that can then apply the proper rates for the region. Changing regions is as simple as substituting a different visitor.



continued on 50

```
public interface EmployeeVisitor {
    abstract void visit(Employee emp);
}

class EmployeeDisplayVisitor implements EmployeeVisitor{

    @Override
    public void visit(Employee emp) {
        System.out.println(emp.getName());
    }
}

public interface VisitorElement {
    void accept(EmployeeVisitor visitor);
}

public class Company implements VisitorElement {
    private final List<Employee> employees = new ArrayList<>();
    public void displayEmployeeNames(){
        EmployeeDisplayVisitor visitor = new EmployeeDisplayVisitor();
        accept(visitor);
    }

    @Override
    public void accept(EmployeeVisitor visitor){
        for (Employee employee : employees) {
            visitor.visit(employee);
        }
    }
}
```

Template Design Pattern

The template design pattern defines program skeleton of an algorithm in a method called a template method, which defers some steps to subclasses. It lets one redefine certain steps of an algorithm without changing the algorithm's structure.
For example,
A online computer buying shop (www.dell.com) provides with a template specifications of computer, like Monitor, HDD, RAM, CPU, Disk Drive, etc. An individual buyer can customize the HDD capacity, Monitor size, RAM size, CPU clock speed keeping the overall structure same.

Decorator Design Pattern

A design pattern that allows behavior to be added to an existing object dynamically.

Observer Design Pattern

A pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notify them automatically of any state changes, usually by calling one of their methods.

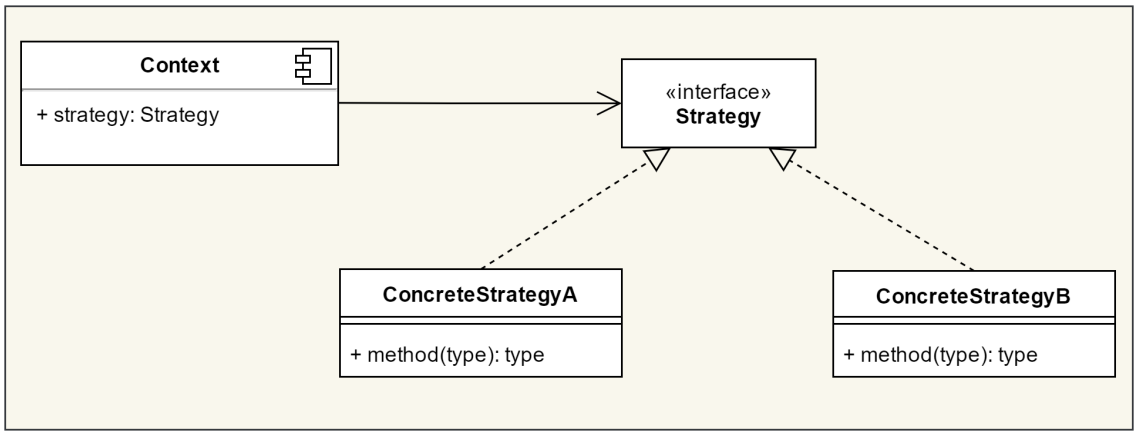
Strategy Design Pattern

Strategy is a design pattern whereby an algorithm's behavior can be selected at runtime, and making them interchangeable.
Strategy lets the algorithm vary independently from clients that use it.

continued on 51

Class Diagram For Strategy

Context uses strategy to perform a calculation, whose behavior can be changed at runtime by substituting ConcreteStrategyA with ConcreteStrategyB.



Facade Design Pattern

Facade provides a single interface to a set of interfaces in the system, like in case of DAO Facade where we hide the internal details of an ORM framework and provide the end programmer with a simple interface to find/ delete and update an Object into the database.
It should be used when a simple interface is needed to provide access to a very complex underlying system.

Another good example of facade design pattern could be : exposing a set of functionalities using web services (SOA architecture). Client does not need to worry about the complex dependencies of the underlying system after building such API.

Q 32. What is a strong, soft, weak and phantom reference in Java?

Skills Set - in depth understanding for GC, automatic memory allocation and de-allocation, LRU Cache, etc

SoftReference, WeakReference & PhantomReference are are reference-object classes, which supports limited degree of interaction with the GC. A programmer may use these classes to maintain a reference to some other object (referent) in such a way that the object may still be reclaimed by GC.

Reference Queues

Reference queue is used to track the objects claimed by GC. We can use the reference objects to check whether the objects referred by these are still active or are claimed by GC.

SoftReference

If the strongest reference to an object is a soft reference then GC will not reclaim the object until the JVM is falling short of memory, though it must be reclaimed before throwing an Out Of Memory Error. So the object will stay longer than a weakly referenced object. It is mostly used for writing memory sensitive caches.

WeakReference

Is similar to soft reference with the only difference that it will be GC'ed in the next GC cycle if the strongest reference to the object is a weak reference. When a weak reference has been created with an associated reference queue and the referent becomes a candidate for GC, the reference object (not the referent) is enqueued on the reference queue after the reference is cleared. The application can then retrieve the reference from the reference queue and learn that the referent has been *continued on 52*

collected so it can perform associated cleanup activities, such as expunging the entries for objects that have fallen out of a weak collection.

WeakHashMap

It is a HashMap that store its keys (not values) using WeakReferences. An entry in this map is automatically removed when there is no other non-weak references to keys. This collection can be used to store associative objects like transient object & its metadata, as soon as the object is claimed by the GC, the associated metadata will also be removed by the map. Other application could be in a servlet environment where as soon as the session expire's, clear all the session data/attributes.

PhantomReference

PhantomReference are garbage collected when the strongest reference to an object is a phantom. When an object is phantomly reachable, the object is already finalized but not yet reclaimed, so the GC enqueues it in a reference queue for post finalization processing. A Phantom Reference is not automatically cleared when it is enqueued., so we must remember to call its clear() method or to allow phantom reference object itself to be garbage collected. get() method always return null so as not to allow resurrect the referent object.
Phantom references are safe way to know an object has been removed from memory and could be thought of as a substitute for finalize() method.

Automatically-cleared references

Soft and weak references are automatically cleared by the collector before being added to the queues with which they are registered, if any. Therefore soft and weak references need not be registered with a queue in order to be useful, while phantom references do. An object that is reachable via phantom references will remain so until all such references are cleared or themselves become unreachable.

Reachability levels from strongest to weakest : strong, soft, weak, phantom. Java 6 docs states that -

- An object is strongly reachable if it can be reached by some thread without traversing any reference objects. A newly-created object is strongly reachable by the thread that created it.
- An object is softly reachable if it is not strongly reachable but can be reached by traversing a soft reference.
- An object is weakly reachable if it is neither strongly nor softly reachable but can be reached by traversing a weak reference. When the weak references to a weakly-reachable object are cleared, the object becomes eligible for finalization.
- An object is phantom reachable if it is neither strongly, softly, nor weakly reachable, it has been finalized, and some phantom reference refers to it.
- Finally, an object is unreachable, and therefore eligible for reclamation, when it is not reachable in any of the above ways.

Notes

WeakHashMap is not a solution for implementing cache, SoftReference's could be better utilized for implementing cache.

Applications of a WeakHashMap

WeakHashMap stores its keys using WeakReference, and can be used to map transient objects with their metadata. Let's suppose we have a socket application which creates sockets on client's request and socket lives there for sometime. Now if we want to associate some metadata with this socket such as identity of the user, then WeakHashMap is a ideal container for storing such associative information. Since we are not managing the lifecycle of the socket in this case, WeakHashMap will automatically remove all the metadata as soon as the socket dies.

Applications of a SoftReference

Soft references can be used to build memory sensitive cache which automatically collects items as soon as the cache is under high memory load, which otherwise has to be achieved by the programmer.

Q 33. What is transaction Isolation level?

Let's first get familiar with the common problems occurring in concurrent database applications, for example

Dirty Read
Occurs when uncommitted results of one transaction are made visible to another transaction.

Unrepeatable Reads
Occurs when the subsequent reads of same data by a transaction results in seeing different values.

Phantom Reads
One transaction performs a query returning multiple rows, and later executing the same query again sees some additional rows that were not present the first time.

We also call above three as Isolation Hazards, and the Transaction Isolation levels are related to these three problems.

Isolation Level	Dirty read	Unrepeatable read	Phantom read
Read Uncommitted	Yes	Yes	Yes
Read Committed	No	Yes	Yes
Repeatable Read	No	No	Yes
Serializable	No	No	No

For most of databases, the default Transaction Isolation Level is Read Committed.
(Read Committed does not see any inconsistent state of other transaction, with a fair amount of concurrency)

READ_UNCOMMITTED isolation level states that a transaction may read data that is still uncommitted by other transactions. This constraint is very relaxed in what matters to transactional concurrency but it may lead to some issues like dirty reads.

READ_COMMITTED isolation level states that a transaction can't read data that is not yet committed by other transactions. But the repeated read within the same transaction may get different results.

REPEATABLE_READ isolation level states that if a transaction reads one record from the database multiple times the result of all those reading operations must always be the same. This eliminates both the dirty read and the non-repeatable read issues.

SERIALIZABLE isolation level is the most restrictive of all isolation levels. Transactions are executed with locking at all levels (read, range and write locking) so they appear as if they were executed in a serialized way. This leads to a scenario where none of the issues mentioned above may occur, but in the other way we don't allow transaction concurrency and consequently introduce a performance penalty.

Please be noted that the above four isolation levels are in decreasing order of their concurrency. So for scalability reasons, Serializable is rarely a good choice of design, as it offers only a single thread to work at a given time.

General practice to choose Isolation level
Choose the lowest isolation level that can keep our data safe.

How to Set Isolation Level?

Database Level
Isolation Level can be set at the DB level as per DB specifications.

Hibernate Level
We can set the default Isolation for all hibernate transaction using the below mechanism

```
<property name="hibernate.connection.isolation">x</property>
```

Where x is the transaction Isolation Number choosen from below option
public static final int TRANSACTION_NONE = 0;
public static final int TRANSACTION_READ_UNCOMMITTED = 1;
public static final int TRANSACTION_READ_COMMITTED = 2;
public static final int TRANSACTION_REPEATABLE_READ = 4;
public static final int TRANSACTION_SERIALIZABLE = 8;

Using Spring Annotation at Method/Service Level
The below annotation can be configured at method or service level to set the appropriate Isolation Level

```
@Transactional(isolation=Isolation.READ_COMMITTED)
```

Plain Java JDBC Level
If we are using Plain Java JDBC with java.sql.Connection, then the below code snippet can be used to set appropriate Transaction Isolation Level.

```
Connection conn = getConnection();
System.out.println("Transaction Isolation Level: " + conn.getTransactionIsolation());

// Set transaction isolation to SERIALIZABLE
conn.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);
```

Q 34. What is difference between Primary key and Unique Key?

1. The main purpose of a Primary key is to uniquely identifies a given record in a Table, while the same is not true for Unique Key constraint.
2. A table can have one and only one Primary Key, while there could be multiple unique keys inside a single table.
3. Primary Key can not have Null values while Unique key column can contain a Null value
4. Primary key creates the Clustered index, but unique key creates the Non clustered index.

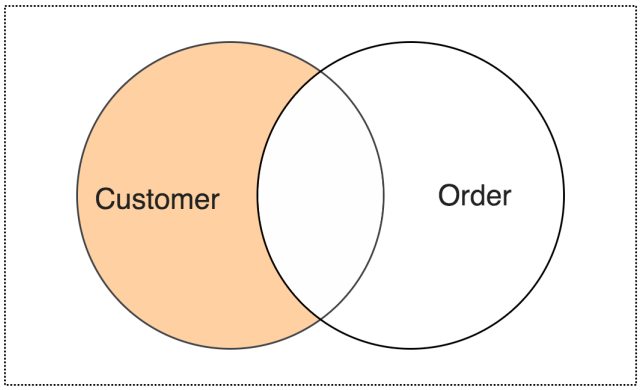
Q 35. Why do we need indexing on database table columns?

Indexing is a way of sorting a number of records on multiple fields. Creating an index on a field in a table creates another data structure which holds the field value, and pointer to the physical record it relates to. This index structure is then sorted, allowing Binary Searches [Time Complexity O(log(n))] to be performed on it .

Creating Index on database table column improves table query performance. It is a usual practice to create Index on Table Columns that are frequently used in where clause of query. There is a slight drawback of creating too many indexes on table columns that it has a performance hit on Table row insertion/updation since that will require index updation as well, also they require additional space on Disc.

Q 36. How will you list all customers from customer table who have no Order(s) yet?

This is a typical problem that can be solved using Left Outer Join. As illustrated in the following figure, we want to fetch the customers (colored area) who have not placed any orders yet.




```
SELECT
   c.name, c.contact, c.email
FROM
   Customer c
      LEFT JOIN
   Orders o ON c.id = o.c_id
WHERE
   o.c_id IS NULL;
```

LEFT OUTER JOIN

Here we are using Left Outer Join (also called Left Join) on Orders with Customers to select all the Customer(s) and then putting a where clause to eliminate the records without any Orders.
Another approach (but inefficient) to do the same could be to use sub-queries, as shown below:

```
select * from Customer c where c.id not in (select o.c_id from Order o)
```

Q 37. How would you fetch Employee with nth highest Age from Employee Table using SQL?

Each row of Employee needs to be compared to very other row to fetch the above mentioned details, thus the Time Complexity of this operation would be quite high (O (n^2))

```
SELECT *
FROM Employee E1
WHERE (N-1) = (SELECT COUNT(DISTINCT(E2.Age))
            FROM Employee E2
            WHERE E2.Age > E1.Age)

To find 2nd highest Age, the query would become
SELECT *
FROM Employee E1
WHERE (2-1) = (SELECT COUNT(DISTINCT(E2.Age))
            FROM Employee E2
            WHERE E2.Age > E1.Age)
```

Q 38. What is difference between Drop, Truncate and Delete commands in SQL?

Delete is used to delete rows from a table with optional where clause, we need to commit or rollback after calling this operation. This operation will cause all DELETE triggers to be fired on the table.

```
DELETE FROM Employee WHERE age < 14;
```

Truncate removes all rows from table, this operation can not be rolled back and no triggers are fired, thus it is faster in performance as well.

```
Truncate Table Employee;
```

Drop command will remove a table from the schema, all data rows, indexes, privileges will be removed, no triggers will be fired and no rollback.

```
Drop Table Employee;
```

Q 39. What is Inner Join, Left Outer Join and Right Outer Join?

INNER JOIN

This is the most common and the default join operation. This join creates a resultset by combining the column values of two tables (L and R) based upon the predicate. Each row of L (left table) is compared with each row of R (right table) to find all pairs of rows that satisfy the join predicates. When the join-predicate is satisfied, column values for each matched pair of rows of L and R are combined into a result row.
Example query is shown below.

Explicit Join Notation

```
SELECT *
FROM employee INNER JOIN department
  ON employee.DepartmentID = department.DepartmentID;
```

Implicit Join Notation

```
SELECT *
FROM employee, department
WHERE employee.DepartmentID = department.DepartmentID;
```

OUTER JOIN

An outer join does not require each record in the two joined tables to have a matching record. The joined table retains each record—even if no other matching record exists.

LEFT OUTER JOIN

The result of a left outer join (or simply left join) for table L and R always contains all records of the "left" table (L), even if the join-condition does not find any matching record in the "right" table (R). This means that if the ON clause matches 0 (zero) records in R (for a given record in L), the join will still return a row in the result (for that record)—but with NULL in each column from R. A left outer join returns all the values from an inner join plus all values in the left table that do not match to the right table.

```
SELECT *
FROM employee LEFT OUTER JOIN department
  ON employee.DepartmentID = department.DepartmentID;
```

RIGHT OUTER JOIN

A right outer join (or right join) closely resembles a left outer join, except with the treatment of the tables reversed. Every row from the "right" table (R) will appear in the joined table at least once. If no matching row from the "left" table (L) exists, NULL will appear in columns from A for those records that have no match in R.

```
SELECT *
FROM employee RIGHT OUTER JOIN department
  ON employee.DepartmentID = department.DepartmentID;
```

References

http://en.wikipedia.org/wiki/Join_(SQL)

Q 40. What are clustered and non-clustered indexes in database?

Clustered Index

Clustered index physically rearrange the data that users inserts in your tables. It is nothing but a dictionary type data where actual data remains. And thus the physical order of the rows is the same as the logical order (indexed order). This type of index is often build over the Primary Key of a table

Non-Clustered Index

Non-Clustered Index contains pointers to the data that is stored in the data page. It is a kind of index backside of the book where you see only the reference of a kind of data.

Clustered index usually provides faster data retrieval than the non-clustered index. Moreover clustered indexes provides faster access to the contiguous rows because those rows are present physically adjacent in the actual table.

References

http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.ase_15.0.sqlug/html/sqlug/sqlug537.htm
http://en.wikipedia.org/wiki/Database_index#Types_of_indexes

Q 41. How would you handle lazily loaded entities in web application development using hibernate?

There are at least two approaches to handle problem of initializing lazily loaded objects in web layer.

1. Hibernate.initialize(<entity>) - this static factory method will Force initialization of a proxy or persistent collection. This method should only be called inside the transaction otherwise it will throw exception. If we are using Spring then we can write something like this

```
@Override
@Transactional(readOnly = false)
public TaskData findById(long id) {
   TaskData taskData = taskDao.findById(id);
   if (taskData != null) {
      Hibernate.initialize(taskData.getTodoResources()); //TodoResources is lazy loaded object in TaskData entity
   }
   return taskData;
}
```

2. Incase of web applications, you can declare a special filter in web.xml, it will open session per request

```
<filter>
   <filter-name>openSessionInViewFilter</filter-name>
   <filter-class>org.springframework.orm.hibernate3.support.OpenSessionInViewFilter</filter-class>
</filter>
<filter-mapping>
   <filter-name>openSessionInViewFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

Depending upon the requirements, you can choose the best suited approach for your project.

Q 42. What are OneToOne, OneToMany and ManyToMany relationship mappings in database design?

When mapping entities with each other, we describe the relation among entities using OneToOne, OneToMany, ManyToOne or ManyToMany mappings.

OneToOne

A Person has a PAN (Card) is a perfect example of One To One association.
Unidirectional - Person can refer to PAN entity
Bidirectional - PAN entity can refer back to Person

```
@Entity
public class Person {
@Id private int id;
@OneToOne
@JoinColumn(name="PAN_ID")
private PAN pan;
// ...
}

@Entity
public class PAN {
@Id private int id;
@OneToOne(mappedBy="pan")
private Person person;
// ...
}
```

OneToMany

A Person has many Skill(s), But a skill can not be shared among Person(s). A Skill can belong to utmost One Person. One more example could be relationship between Employee and Department where an Department is associated with Collection of Employee(s)
Unidirectional - A Department can directly reference Employee(s) by collection
Bidirectional - Each Employee has a reference back to Department

```
@Entity
public class Employee {
@Id private int id;
@ManyToOne
@JoinColumn(name="DEPT_ID")
private Department department;
// ...
}

@Entity
public class Department {
@Id private int id;
@OneToMany(mappedBy="department")
private Collection<Employee> employees;
// ...
}
```

Employee Table would keep DEPT_ID foreign key in its table, thus making it possible to refer back to Dept.

ManyToMany

One Person Has Many Skills, a Skill is reused between Person(s). One more example of this could be

relationship between Employee and Project. Each employee can work on multiple Project(s) and each Project can be worked upon by multiple Employee(s). One more example could be relationship between Customer(s) and Product(s) where One or More Customer(s) purchase many different Product(s) and Product(s) can be purchased by different Customer(s)
Unidirectional - A Project can directly reference its Employee(s) by collection
Bidirectional - An Employee has Collection of Projects that it relates to.

```
@Entity
public class Employee {
@Id private int id;
@ManyToMany
private Collection<Project> projects;
// ...
}

@Entity
public class Project {
@Id private int id;
@ManyToMany(mappedBy="projects")
private Collection<Employee> employees;
// ...
}
```

Association or junction table is must to implement a ManyToMany relationship, this separate table connects one line from Employee to one line from Poject using foreign keys. And each primary key of Employee and Project can be copied over multiple times to this table.

Q 43. How would you implement ManyToMany mappings with the self entity in JPA?

We need to maintain two different mappings in the same entity for ManyToMany relationship as shown below -

```
@ManyToMany
@JoinTable(name="table_friends",  joinColumns=@JoinColumn(name="personId"),
inverseJoinColumns=@JoinColumn(name="friendId"))
private Set<User> friends;

@ManyToMany
@JoinTable(name="table_friends",  joinColumns=@JoinColumn(name="friendId"),
inverseJoinColumns=@JoinColumn(name="personId"))
private Set<User> friendOf;
```

In the above Bidirectional Mapping, One side of relationship will maintain the User's list of friends (friends), and the inverse side of relationship will maintain how many people have this User in their friend list (friendOf).

The Inverse side of the relationship can also be described as -

```
@ManyToMany(mappedBy="friends")
private Set<User> friendOf = new HashSet<User>();
```

Q 44. What are Inheritance strategies in JPA?

JPA defines three inheritance strategies namely, SINGLE_TABLE, TABLE_PER_CLASS and JOINED.

Single table inheritance is default, and table per class is optional so all JPA vendors may not support it.
JPA also defines mapped super class concept defined through the @MappedSuperClass annotation. A Mapped Super Class is not a persistent class, but allows a common persistable mapping to be defined for its subclasses.

Single Table Inheritance
In this inheritance, a single table is used to store all the instances of the entire inheritance hierarchy. The Table will have a column for every attribute of every class in the hierarchy. Discriminator columns identifies which class a particular row belongs.

Table Per Class Inheritance
A table is defined for each **concrete** class in the inheritance hierarchy to store all the attribute of that class and all its super classes.

Joined Table
This inheritance replicates the object model into data model. A table is created for each class in the hierarchy to store only the local attributes of that class.

Question: We want to extract common behavior in a super class in JPA entities but we do not want to have table for that super class. How would you achieve this?
Answer - If we create a normal class as the super class, then as per JPA specifications, the fields for that class are not persisted in the database tables. We need to create a super class extracting the common fields and then annotate that class with @MappedSuperClass in order to persist the fields of that super class in subclass tables. A mapped super class has no separate table defined for it.

References
http://en.wikibooks.org/wiki/Java_Persistence/Inheritance

Q 45. How will you handle Concurrent updates to an database entity in JPA i.e. when two users try to update the same database entity in parallel?

There are two main approaches to handle transaction concurrency using JPA 2.0[1]

1. Optimistic Concurrency (Scalable Option) - This approach is as simple as adding a version column to the database entity, as shown in the below code. When version column is present, JPA will increment the version field for us upon every update to the row. Thus when two detached entities with the same version try to update the database, one will fail (throws OptimisticLockException) because of mismatch in version column value. This approach offer higher concurrency throughput compared to Pessimistic Locking, because it does not serializes the thread access. This approach will work even for the detached entities where a single database row was read in parallel by two threads, and later point in time these two threads try to update the contents of detached database entities. This approach gives best performance for applications with very less contention among the concurrent transactions.

```
public class Employee {
@ID int id;
@Version int version;

JPA will issue DML something similar to this command
"UPDATE Employee SET ..., version = version + 1 WHERE id = ? AND version = readVersion"
```

1 https://blogs.oracle.com/carolmcdonald/entry/jpa_2_0_concurrency_and

2. Pessimistic Concurrency (badly-scalable) - In this approach, JPA will lock the database row (not object in memory) when the data is read, and releases the lock upon completion of transaction. This way only one database transaction can update the same entity at same time. In Oracle database, it's similar to the following SQL statement -
   (SELECT . . . FOR UPDATE [NOWAIT])

In Spring Framework, you can set this transaction Level by adding below annotation to the service method

```
@Transactional(readOnly = false, isolation = Isolation.SERIALIZABLE)
public void cleanTaskHistory() { ... }
```

Pessimistic approach works best for applications where contention ratio is high among the concurrent transactions, otherwise it is a badly scalable option for handling concurrency.

Q 46. How to efficiently generate ID's for an Entity in Hibernate/JPA?

We can use table HiLo generator in hibernate/JPA that can cache sertain predefined number of IDs for allocation, thereby reducing the number of database trips. Here is how we implement it.

Annotate the JPA Entity with the below annotations

```
@Entity
@Table(name="MyPatient", uniqueConstraints=
@UniqueConstraint(columnNames = {"firstName", "birthDate", "primaryIdentifier", "gender"}))
public class Patient {
@GeneratedValue(strategy=GenerationType.TABLE, generator="tbl-gen")
@TableGenerator(name="tbl-gen",
      pkColumnName="ENTITY_TBL_NAME", allocationSize=500,
      table="hibernate_hilo")
@Id
private Long id;
```

If it is Hibernate, then we can use Hibernate specific code

```
@Entity
@Table(name="MyPatient", uniqueConstraints=
@UniqueConstraint(columnNames = {"firstName", "birthDate", "primaryIdentifier", "gender"}))
public class Patient {
@GeneratedValue(strategy=GenerationType.TABLE, generator="tbl-gen")
@GenericGenerator(name="table-hilo-generator", strategy="org.hibernate.id.TableHiLoGenerator",
      parameters={
            @org.hibernate.annotations.Parameter(value="hibernate_id_generation", name="table"),
            @org.hibernate.annotations.Parameter(value="next_hi", name="column")
      })
@GeneratedValue(generator="table-hilo-generator")
@Id
private Long id;
```

Q 47. How does a typical Hibernate Transaction look like?

A typical database transaction should use the following idiom -

```
Session sess = factory.openSession();
Transaction tx;
try {
    tx = sess.beginTransaction();
    doWork();
    tx.commit();
} catch (Exception e) {
    if (tx != null) tx.rollback();
    throw e;
} finally {
    sess.close();
}
```

Q 48. How will you handle batch insert in hibernate for optimal usage of memory, network and CPU?

A naive approach to insert 1M rows in the database using Hibernate might look like this -

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();
for ( int i=0; i<1000000; i++ ) {
    Customer customer = new Customer(.....);
    session.save(customer);
}
tx.commit();
session.close();
```

But when we try to run this code we may run into OutOfMemory exception and the performance of the method will also be low.

Optimization Steps

- You will need to enable the use of JDBC batching in hibernate config file, for example batch size can be set to 50
  hibernate.jdbc.batch_size 50
- You can disable hibernate second level caching in hibernate config
  hibernate.cache.use_second_level_cache false
  However, this is not absolutely necessary, as we can explicitly set the CacheMode to disable interaction with the second-level cache.
- Use batch insert code that clear the cache after flushing the records to database, as shown below -

```
Session session = sessionFactory.openSession();
session.setCacheMode(CacheMode.IGNORE);
Transaction tx = session.beginTransaction();
for ( int i=0; i<1000000; i++ ) {
    Customer customer = new Customer(.....);
    session.save(customer);
    if ( i % 50 == 0 ) { //50, same as the JDBC batch size
        //flush a batch of inserts and release memory:
        session.flush();
        session.clear();
    }
}
tx.commit();
session.close();
```

Another good way of doing the same thing is to use Hibernate's Stateless Session

```
StatelessSession session = sessionFactory.openStatelessSession();
Transaction tx = session.beginTransaction();
ScrollableResults customers = session.getNamedQuery("GetCustomers")
      .scroll(ScrollMode.FORWARD_ONLY);
while (customers.next()) {
   Customer customer = (Customer) customers.get(0);
   customer.updateStuff(...);
   session.update(customer);
}
tx.commit();
session.close();
```

Or alternatively using HIbernate's session.doWork() method as shown below -

```
sessionFactory.getCurrentSession().doWork(new Work() {
   @Override
   public void execute(Connection connection) throws SQLException {
      StatelessSession statelessSession = sessionFactory.openStatelessSession(connection);
      try {
         DemoEntity demoEntity = (DemoEntity) statelessSession.createQuery("from DemoEntity where id = 1").
uniqueResult();
         demoEntity.setProperty("test");
         statelessSession.update(demoEntity);
      } finally {
         statelessSession.close();
      }
   }
});
```

Q 49. How will you operate on records of a large database table with million of entries in it using Hibernate?

We can use StatelessSession for this purpose which gives very good performance using optimal system resources.
Let's say we want to stream all records from a huge database table and scan for duplicate files -

```
public void analyzeAllDuplicates() {
      StatelessSession session = getSessionFactory().openStatelessSession();
      Transaction tx = session.beginTransaction();
      ScrollableResults fileDocuments = session
            .getNamedQuery(FileDocument.GET_ALL_DOCUMENTS)
            .scroll(ScrollMode.FORWARD_ONLY);
      while (fileDocuments.next()) {
         FileDocument document = (FileDocument) fileDocuments.get(0);
         checkAlreadyExists(document);
//          document.updateStuff(...);
//          session.update(document);
      }
      tx.commit();
      session.close();
   }
```

Q 50. Do you think Hibernate's SessionFactory and Session objects are thread safe?

SessionFactory is thread-safe and in normal application we should just create one SessionFactory object per classloader per JVM. SessionFactory class stores all the second level cache and query cache and thus is very heavy weight object.

Session on the other hand should be created per transaction, as it is not thread-safe.
A typical transaction should use the following idiom:

```
Session sess = factory.openSession();
<begin transaction>
<do your work>
<end transaction>
sess.close();
```

Q 51. What is difference between Hibernate's first and second level cache?

Hibernate's first level cache resides at Session Level. Session cache caches object within the current session but this is not enough for long level i.e. session factory scope.
Hibernate's second level cache resides at Session-factory level. This cache exists as long as the session factory is alive.

Important point about Hibernate's first and second level cache is that it only works when we fetch records by ID, for non-id fields cache does not come into picture at all.

Q 52. What is difference between session.get() and session.load() in hibernate?

Both these methods are used to retrieve an object from underlying datastore. The only difference is in the mechanism of fetching data.

session.get()
This method will actually hit the database and return the real object along with its properties. It is a EAGER loaded operation that requires round trip to database. If no record is found for the given ID, then null will be returned.

session.load()
This method returns a reference (proxy) to the entity with the given identifier. This method does not hit the database until a property on the return proxy is accessed for the first time. If no row is found, it will throw an ObjectNotFoundException.

Which one to choose?
The only difference is in the number of database roundtrips b/w these methods. If you need not to get the properties of a given object, then you shall prefer session.load(). The valid scenarios for such situation could be ManyToOne relationship or OneToOne relationship, where you just need to get reference to a related object and assign it to object under construction.

Q 53. What is usecase for GET, PUT, POST and DELETE method in REST API?

GET
To retrieve a resource, you shall use HTTP GET. We should not modify the state of resources using HTTP GET method ever.

POST
To create a resource on the server, use HTTP POST. An example could be new user registration REST endpoint, it will create a new user in the system.

PUT
To change the state of a resource or to update it on the server, use PUT

DELETE
To remove or delete a resource on server, use DELETE

Following this conventions has many advantages, including:
1. Its easy on the server side to implement Spring Security. For example, you can add an ant matcher to protect all POST endpoints with a given ROLE
2. CSRF works by default for POST method only, so if you are really modifying the state of an entity using GET operation, you may be compromising on security.
3. Implementing CORS is easy & safe if there are no violations to the above rules.
4. It allows user-agents (browsers) to implement caching, bookmarking etc. for GET resources.

Q 54. What are different types of Http Status Codes?

There is a complete list of Http Status Codes available at www.wikipedia.org
http://en.wikipedia.org/wiki/List_of_HTTP_status_codes

Generally used HTTP Status Codes are:
1xx - Informational Status Code. It means request has been received and process is continuing. (Its rarely used in production)
2xx - Success. Action was successfully received, understood and accepted.
    200 Ok
    201 Created - new resource is created
    202 Accepted - The request has been accepted for processing, but the processing has not been completed
3xx - Redirection. Further action must be taken in order to complete the request.
    302 - Temporarily Redirect (Found)
    304 - Not Modified
4xx - Client Error
    400 - Bad Request (malformed request syntax, invalid request message, etc)
    401 - Unauthorized (may ask for credentials). authenticating may make difference
    403 - Request was valid but server is refusing to respond to it, authenticating will make no difference
    404 - Not Found
5xx - Server Error
    500 - Internal Server Error, server encountered unexpected condition
    501 - Not Implemented
    503 - Service Unavailable (may be down for maintenance)

Q 55. What is difference between HTTP Redirect and Forward?

Forward

1. Control is forwarded to the resource available within the server from where the call is made, the transfer of control is made internally by the container, where client is completely unaware that a forward is happening.
2. When forward is done, the original request and response objects are transferred along with the additional parameters if needed.
3. Forward can't transfer control to some other domain.
4. Original URL at the client side remains intact, refreshing the page will cause the whole step to repeat again.
5. Session object is not lost in forward or redirect.

Redirect

1. A redirect is a two step process where web application instructs the browser client to fetch the fetch the second URL which differs from the original.
2. Server sends Http Status Code of 301 to the client, and then client follows the instructions.
3. If someone reloads the page on browser, then original request will not be repeated. Just the second url will be fetched again.
4. Redirect is marginally slower than forward, since it requires two requests.
5. Objects placed in request scope are not available to second request.
6. There are several ways to perform a redirect for example,

```
Http/1.1 301 moved permanently
Location : http://www.foobar.org/
<meta http-equiv="refresh" Content="0; URL=http://www.example.org/"/>
<script type="text/javascript"> location.href ="http://www.foobar.org" </script>
```

Redirect Should be used when

1. If you need to transfer of control to a different domain.
2. To achieve separation of tasks.

For example, database update and data display can be separated by redirect. In this case if user presses F5 button the browser then only display part will execute again and not the database update part.
Do the PaymentProcess and then redirect to the display payment info, if the client refreshes the browser, only the displayInfo will be done again and not the payment process.

Use Forward when database SELECT operations are used

Q 56. What are the best practices for handling TimeZone in database transactions?

There are multiple ways to handle Time Zone in your Java Application which deals with database transactions -

1. While using PreparedStatement, we should always prefer setDate(int parameterIndex, Date date, Calendar cal) method to specify the Calendar in desired time zone.
2. For Spring JDBCTemplate, we should pass Calendar (with desired TimeZone) instance instead if plain Date object.
3. We can also set application wide TimeZone using TimeZone.setDefault(TimeZone.getTimeZone(String ID))
4. JVM wide time zone can be set by passing JVM argument -Duser.timezone=GMT

Do's -

1. Prefer JodaTime API for handling TimeZone specific calculations in your application. JodaTime provides simple and better api's for playing with Date & TimeZone.
2. While persisting time in your application, always prefer to use GMT or any other TimeZone which is not affected by the Day Light Savings. And always include the original timezone name while storing the date so

that you can easily re-construct the date to the same value.
3. Business rules should always work on GMT time.
4. Only convert to local time at the last possible moment.
5. TimeZones and Offsets are not fixed and may change in future, always design your application keeping this thing in mind.

Don't -

1. Do not use javascript based Date and Time Calculations in your web applications unless absolute necessary as time and date on client machine may be different or incorrect.
2. Never trust Client DateTime on your server application.
3. Do not compare client datetime with server datetime.

Q 57. How will you check the owner information of a given domain name in web?

There is a Domain information lookup utility named whois (provided by sysinternals). it will list all the information related to a given domain name.
whois www.google.com

Connecting to IN.whois-servers.net...

Domain ID:D8357-AFIN

Domain Name:GOOGLE.CO.IN

Created On:23-Jun-2003 14:02:33 UTC

Last Updated On:22-May-2014 09:17:32 UTC

Expiration Date:23-Jun-2015 14:02:33 UTC

Sponsoring Registrar:Mark Monitor (R84-AFIN)

Status:CLIENT DELETE PROHIBITED

Status:CLIENT TRANSFER PROHIBITED

Registrant ID:mmr-108695

Registrant Name:Christina Chiou

Q 58. What happens when you type www.google.com in your browser's address bar from an Indian Location?

Skills - HTTP protocol, DNS Lookup, Networking, TCP connection, etc

There are series of events that happen when we type in www.google.com into browser's address bar from a given location, we will cover few main steps here -

1. User enters www.google.com into the address bar
2. Browser checks if the typed address is www url or the search term, if it is search term then it will use pre configured web search server (may be google or bing, etc) to search the typed term from web.
3. If the requested Object is in browser's cache and cache is valid, content is rendered from cache, otherwise
4. DNS Lookup takes place - Browser resolves the IP address for the mentioned server (www.google.com)
    i. It checks in the browser cache
    ii. checks the OS Cache
    iii. checks the router cache
    iv. checks the ISP cache
    v. DNS recursive search until it reaches authoritative DNS Server. If multiple ip addresses are found for a given server address, then DNS Round Robin algorithm picks up any one for the further communication. If it does not recognize the domain then it gives error message.
    we can use <nslookup www.google.com> command on windows to check what all IP addresses are mapped to this www.google.com domain, incase they are multiple DNS server will use round robin algorithm to pick up any one from the list.
5. Browser initiates TCP connection with the IP address and issues HTTP GET request to the server, it passes along an HttpRequest that includes metadata about browser, user preferences (language, locale etc.) and cookies for that domain.
6. Google.com server receives the request, uses the passed information (cookies) to find who the user is, locale, language and region and sends http redirects (HTTP GET 302) to browser to use local regional google server, i.e. www.google.co.in in our case (temporarily redirect)
7. Browser receives the response (302 in this case) and sends a fresh request to the newly mentioned location in the previous response, passing the user information again (cookies for that domain, metadata, etc)
8. Google.co.in receives the request the decodes the user and send the appropriate HTML response including headers (status code 200 OK, content type, etc)
9. The Browser receives the response and begins to parse it for display. if it is compressed, browser will decompress it, The HTML body will include links to css, images, js. All these links will trigger additional calls back to server to retrieve those files. CDN (Content Delivery Networks) may serve these static resource requests to speedup the process.
10. Browser layout engine will start to assemble the final page for display. css, js information may alter the layout of the page.
11. The final page is assembled and rendered to the user.
12. After this the browser may send further AJAX request to communicate with the web server even after the page is rendered.

What is General Behavior for a Generic Http Request?

1. Type something in browser bar
    i. browser parses the url (protocol, etc)
    ii. Is it a url or search term - when no protocol or valid domain is given, browser feeds the text given in address bar to browser's default web search engine.
2. DNS Lookup takes place
    i. browser checks cache; if requested object is in cache and is fresh
    ii. browser asks OS for server's IP address
    iii. OS makes a DNS lookup and replies the IP address to the browser

3. Opening of a socket (port 80 for HTTP and 443 for HTTPS)
    i. browser opens a TCP connection to server (this step is much more complex with HTTPS)
    ii. browser sends the HTTP request through TCP connection
4. Browser receives HTTP response and may close the TCP connection, or reuse it for another request
5. Browser checks if the response is a redirect (3xx result status codes), authorization request (401), error (4xx and 5xx), etc.; these are handled differently from normal responses (2xx), in case of redirect it sends a another Get request as per location specified in previous response.
6. if cacheable, response is stored in cache
7. Browser decodes response (e.g. if it's gzipped)
8. Browser determines what to do with response (e.g. is it a HTML page, is it an image, is it a sound clip?) the Content-type of header instructs the browser to render the response content as HTML, instead of say downloading it as a file.
9. Browser renders response, or offers a download dialog for unrecognized types
10. The browser begins rendering the HTML and sends the request for object embedded in HTML as many sites deliver their CSS,Images/Sprite files and scripts file from a content delivery network (CDN). the browser will again send the GET request for each of the embedded URL which again goes by the same procedure of look up and other above mention steps.
11. After this the browser may send further AJAX request to communicate with the web server even after the page is rendered.

Also, there are many other things happening in parallel to this (processing typed-in address, adding page to browser history, displaying progress to user, notifying plugins and extensions, rendering the page while it's downloading, pipelining, connection tracking for keep-alive, etc.).

What is nslookup command?

nslookup stands for "name server lookup" and it gives us all IP addresses mapped by a given domain name. Most DNS will provide Round Robin mechinaism to choose single IP from the list of IP's, but our local machine also caches the DNS resolution and will usually use the same IP adderss over and over until it expires (Time To Live, TTL).

Example usage -

nslookup www.google.com

Non-authoritative answer:

Name: www.google.com

Addresses: 2001:4860:400b:c01::6a

74.125.20.104
74.125.20.105
74.125.20.103
74.125.20.99
74.125.20.106
74.125.20.147

ping www.google.com

Pinging www.google.com [74.125.20.105] with 32 bytes of data:

Reply from 74.125.20.105: bytes=32 time=244ms TTL=41

we can see here that ping uses just one IP address while nslookup shows multiple ip addresses mapped by a given domain name.

Useful Links -

http://edusagar.com/articles/view/70/What-happens-when-you-type-a-URL-in-browser

https://technet.microsoft.com/en-in/library/cc772774(v=ws.10).aspx

https://github.com/alex/what-happens-when#check-hsts-list

Q 59. Why do we need Spring Framework?

Spring Framework is very lightweight and provides many key benefits, including
1. Spring has modular and layered architecture i.e. use what you need and leave what you don't need right now. For example, if we don't need Spring transaction management features, we don't need to add that dependency for Transaction Management module in our project.
2. Dependency Injection (or Inversion of Control) to write components that are loosely coupled from each other. Spring container takes care of wiring different components together in a seamless fashion. It takes care of most of Boiler Plate Code thus we can spend our useful time in writing just the actual business logic in our software.
3. AOP (Aspect Oriented Programming) - A technology for separating crosscutting concerns, something usually hard to do in object-oriented programming.
4. Spring MVC framework capable of creating web applications and restful web services that can return response in JSON or XML format
5. Consistent Transaction Management that can be configured using simple annotations at method and class level. It also provides a generic abstraction layer for transaction management making it easy to demarcate transactions without dealing with low-level issues.
7. Data Access Framework - Spring Data JPA
8. Batch Framework for processing batch requests that can be executed by multiple worker nodes
9. Spring Security - A pluggable library for Authentication and Authorization
10. Spring IoC container manages bean lifecycle
11. Writing unit test cases are easy in Spring framework because our business logic doesn't have direct dependencies with actual resource implementation classes. We can easily write a test configuration and inject our mock beans for testing purposes.

Q 60. What is Inversion of Control (or Dependency Injection)?

The basic concept of IOC (Dependency of Injection) is that you do not create your objects but describe how they should be created.
You don't directly connect your component and services together in code but describe which services are needed by which component in configuration file.
You just need to describe the dependency, the Spring container is then responsible for wiring it all up. It promotes Loose coupling with minimal effort and least intrusive mechanism. IoC container have configurable option for eager or lazy initialization of beans services.

Q 61. What is Bean Factory in Spring?

A Bean Factory is like a factory class that contains collections of beans. The Bean Factory holds bean definition of multiple beans within itself and then instantiates the bean when asked by client.

Bean Factory is actual representation of the Spring IOC container that is responsible for containing and managing the configured beans. XmlBeanFactory is the commonly used BeanFactory implementation.

Q 62. What is Application Context?

A bean factory is fine to simple applications, but to take advantage of the full power of the Spring framework, you may want to move up to Springs more advanced container, the application context. On the surface, an application context is same as a bean factory. Both load bean definitions, wire beans together, and dispense beans upon request. But it also provides:

- A means for resolving text messages, including support for internationalization.
- A generic way to load file resources.
- Events to beans that are registered as listeners.

Q 63. What are different types of Dependency Injection that spring support? or in other words what are the ways to initialize beans in Spring?

- Constructor Injection : Dependencies are provided as constructor parameters.

```
<bean id="authenticator" class="com.mycompany.service.AuthenticatorImpl"/>

<bean id="accountService" class="com.mycompany.service.AccountService">
  <constructor-arg ref="authenticator"/>
</bean>
```

- Setter Injection : Dependencies are assigned through JavaBeans properties.

```
<bean id="authenticator" class="com.mycompany.service.AuthenticatorImpl"/>

<bean id="accountService" class="com.mycompany.service.AccountService">
  <property name="authenticator" ref="authenticator"/>
</bean>
```

We should constructor injection for all mandatory collaborators and setter injection for all other properties.

Q 64. What are different Bean Scope in Spring?

singleton: Return a single bean instance per Spring IOC container.
prototype: Return a new bean instance each time when requested.
request: Return a single bean instance per HTTP request.
session: Return a single bean instance per HTTP session.
global session: Return a single bean instance per global HTTP session and only valid when used in portlet context.

We should use the prototype scope for all beans that are stateful and the singleton scope should be used for stateless beans.

Q 65. What are some important Spring Modules?

- Spring Context – for dependency injection.
- Spring AOP – for aspect oriented programming.
- Spring DAO – for database operations using DAO pattern
- Spring JDBC – for JDBC and DataSource support.
- Spring ORM – for ORM tools support such as Hibernate
- Spring Web Module – for creating web applications.
- Spring MVC – Model-View-Controller implementation for creating web applications, web services etc.
- Spring Transaction - for transaction management

Q 66. How will you load hierarchy of property files in Spring Context?

```
<bean class="org.springframework.context.support.PropertySourcesPlaceholderConfigurer">
      <property name="locations">
        <list>
          <value>classpath*:**/default.properties</value>
```

```
            <value>classpath*:**/database-${env:qa}.properties</value>
            <value>classpath*:**/${env:qa}.properties</value> <!--more specific properties goes here-->
        </list>
    </property>
    <property name="ignoreUnresolvablePlaceholders" value="true"/>
    <property name="ignoreResourceNotFound" value="true"/>
</bean>
```

Please note that property files supplied later have higher precedence and will override the existing properties with same name in earlier files.

Q 67. How to handle Bean Post Initialization and Pre Destroy Tasks in Spring Framework? For example resource loading after bean construction and resource cleanup before shutdown of spring context?

Spring framework provides two convenient annotations to handle Bean initialization and cleanup -

- @PostConstruct - annotate this on a method that will initialize a bean post its construction.
- @PreDestroy - annotate this on a method that will do cleanup before Application context closes.

For example, below is the file demonstrating the same

```
@Service
public class AppContextService {
    private AppContext appContext;

    public AppContext getAppContext() {
        return appContext;
    }

    @PostConstruct
    public void loadSettings() throws InstantiationException, IllegalAccessException {
        System.out.println("Loading Settings File");
        appContext = Utils.load(AppContext.class, "file-searcher.xml");
    }

    @PreDestroy
    public void saveSettings(){
        System.out.println("Saving Settings File");
        Utils.save(AppContext.class, appContext, "file-searcher.xml");
    }
}
```

Whenever Spring container starts, method with @PostConstruct annotation will be invoked. And whenever we call context.close(), method with @PreDestroy annotation will be invoked. Below is the typical standalone method to start and stop the Spring Application Context.

```
public class Main {
    public static void main(String[] args) throws IOException {
        System.setProperty("java.util.concurrent.ForkJoinPool.common.parallelism", "8");
        ConfigurableApplicationContext context = new ClassPathXmlApplicationContext("spring-config.xml");
        FileSearcherApp fileSearcherApp = context.getBean(FileSearcherApp.class);
        System.out.println(fileSearcherApp.sayHello());
        System.out.println("Press Enter to Exit");
```

```
            System.in.read();
            context.close();
        }
    }
```

Q 68. What is syntax of Cron Expression?

Quartz cron expression is used to schedule jobs in a job scheduler, syntax of cron expression is

```
0 0/30 * * * ?
```

The field order of cronExpression is -
1. Seconds
2. Minutes
3. Hours
4. Day of Month
5. Month
6. Day of Week
7. Year - Optional Field

There should be at least 6 fields provided in the cron expression, otherwise quartz scheduler will give error.
Example Cron Expressions -

- 0 0 * * * * - Top of every hour of every day
- */10 * * * * * - every ten seconds
- 0 0 9-17 * * MON-FRI - on the hour 9AM to 5PM weekdays
- 0 0/30 8-10 * * * - 8:00, 8:30, 9:00, 9:30 and 10:00 o'clock every day

Q 69. How will you embed a PNG/GIF image inside a CSS file?

Base64 encoded images can be placed inline into a CSS file, most browsers support this feature. We can use Java 8's Base64 encode() and decode() method to convert an image into Base64 and vice versa.

Encoding an PNG Image into Base64 format

```
//Encode base 64
String encodedBytes = Base64.getEncoder()
        .encodeToString(Files.readAllBytes(Paths.get("test.png")));
System.out.println("encodedBytes " + encodedBytes);

//Decode base 64
byte[] decodedBytes = Base64.getDecoder().decode(encodedBytes);
System.out.println("decodedBytes " + new String(decodedBytes));
```

Then the encoded PNG image can be placed inline into the required CSS file as shown below -

```
<style type="text/css">

input:required:valid { background-image: url(<base64 contents goes here>);
background-position: right top; background-repeat: no-repeat; }

</style>
```

Q 70. Explain Java 8 Stream API?

What is a Stream?

A stream is a sequence of data elements supporting sequential and parallel aggregate operations. Computing the sum of all elements in a stream of integers, mapping all names in list to their lengths, etc. are examples of aggregate operations on streams.

How is it different than Collection?

Collections focus on the storage of data elements for efficient access whereas streams focus on aggregate computations on data elements from a data source that is typically, but not necessarily, collections.
Few other differences are -

- Streams have no storage.
- Streams can represent a sequence of infinite elements.
- The design of streams is based on internal iteration.
- Streams are designed to be processed in parallel with no additional work from the developers.
- Streams are designed to support functional programming.
- Streams support lazy operations.
- Streams can be ordered or unordered.
- Streams cannot be reused.

What is ParallelStream?

Java provides two main functionalities out of the box -
a) Partitioning of the stream of data into smaller streams to make parallel processing possible.
b) Parallel processing of the data

Creating a Stream

Streams can be created in multiple ways -

- Stream from Values

```
Stream<String> stream = Stream.of("Ken", "Jeff", "Chris", "Ellen");
Stream<Integer> integerStream = Stream.of(1, 2, 3, 4, 5);
IntStream oneToFive = IntStream.range(1, 6);
```

- Stream from Functions

```
Stream<Long> oddNaturalNumbers = Stream.iterate(1L, n -> n + 2);
Stream.iterate(1L, n -> n + 2).limit(5).forEach(System.out::println);
Stream.generate(Math::random).limit(5).forEach(System.out::println);
```

- Stream from Arrays

```
IntStream numbers = Arrays.stream(new int[]{1, 2, 3});
```

- Stream from Collections

```
// Create and populate a set of strings
Set<String> names = new HashSet<>();
names.add("Ken");
names.add("jeff");

// Create a sequential stream from the set
Stream<String> sequentialStream = names.stream();

// Create a parallel stream from the set
```

```
Stream<String> parallelStream = names.parallelStream();
```

- Stream from Files, etc

```
Stream<String> lines = Files.lines(path)
```

Operations on Streams (Bulk Operations, Map and Reduce)

- Take Odd numbers from the stream square them and them add them all

```
int sum = IntStream.of(1, 2, 3, 4, 5)
        .peek(e -> System.out.println("Taking integer: " + e))
        .filter(n -> n % 2 == 1)
        .peek(e -> System.out.println("Filtered integer: " + e))
        .map(n -> n * n)
        .peek(e -> System.out.println("Mapped integer: " + e))
        .reduce(0, Integer::sum);
System.out.println("sum = " + sum);
```

Result : sum = 35

- Increase Income of All Females by 10%

```
persons.stream()
        .filter(Person::isFemale)
        .forEach(p -> p.setIncome(p.getIncome() * 1.10));
```

- Square all the numbers in the array

```
IntStream.rangeClosed(1, 5)
        .map(n -> n * n)
        .forEach(System.out::println);
```

- Get All males having income greater than 5000

```
Person.persons()
        .stream()
        .filter(p -> p.isMale() && p.getIncome() > 5000.0)
        .map(Person::getName)
        .forEach(System.out::println);
```

- Get Combined Income of All Employees

```
double sum = Person.persons()
        .stream()
        .map(Person::getIncome)
        .reduce(0.0, Double::sum);
System.out.println(sum);
```

- Find first line from log file that contains word 'password

```
Stream<String> lines = Files.lines(path);
Optional<String> passwordEntry = lines.filter(s -> s.contains("password")).
findFirst();
```

- Remove All Nulls from a Stream of Objects

```
stream.filter(Object::nonNull)
```

Q 71. Most useful Code Snippets in Java 8?

- Traversing File Tree using Stream API

```
public static void listFileTree1() {
    Path dir = Paths.get("");
    System.out.printf("%nThe file tree for %s%n", dir.toAbsolutePath());
    try (Stream<Path> fileTree = Files.walk(dir)) {
        fileTree.forEach(System.out::println);
    } catch (IOException e) {e.printStackTrace();}
}
```

- Calculate Prime Number using Stream API

```
public class PrimeUtils {
    public static void main(String[] args) throws ExecutionException, InterruptedException {
        ForkJoinPool forkJoinPool = new ForkJoinPool(1);
        List<Integer> primeList = forkJoinPool.submit(() -> new PrimeUtils().collectPrimes(10000000)).get();
        System.out.println("primeList = " + primeList);
        forkJoinPool.shutdown();
        long t2 = System.currentTimeMillis();
    }
    private List<Integer> collectPrimes(int max) {
        return IntStream.range(1, max).parallel().filter(this::isPrime).boxed().collect(Collectors.toList());
    }
    private long countPrimes(int max) {
        return IntStream.range(1, max).parallel().filter(this::isPrime).count();
    }
    private boolean isPrime(long n) {
        return n > 1 && IntStream.rangeClosed(2, (int) Math.sqrt(n)).noneMatch(divisor -> n % divisor == 0);
    }
}
```

There is a magic in above program, we are executing parallel stream operation inside a forkJoinPool, and number of threads consumed by the parallel stream depends upon the constructor of ForkJoinPool. That means if ForkJoinPool is created with 6 threads then the concurrency level of parallel stream will be 6.

- Copy File using FileChannel Java NIO

```
public class FileCopy {
     public static void fileCopy(File in, File out) throws IOException {
      try (FileChannel inChannel = new FileInputStream(in).getChannel();
           FileChannel outChannel = new FileOutputStream(out).getChannel();) {
         // inChannel.transferTo(0, inChannel.size(), outChannel);     // original -- apparently has trouble copying large
files on Windows
         // magic number for Windows, 64Mb - 32Kb)
         int maxCount = (64 * 1024 * 1024) - (32 * 1024);
         long size = inChannel.size();
         long position = 0;
         while (position < size) {
            position += inChannel.transferTo(position, maxCount, outChannel);
         }
      }
   }
   private static void copyFileUsingApacheCommonsIO(File source, File dest)
         throws IOException {
      FileUtils.copyFile(source, dest);
   }
   private static void copyFileUsingFileChannels(File source, File dest)
         throws IOException {
      try (FileChannel inputChannel = new FileInputStream(source).getChannel();
```

```
            FileChannel outputChannel = new FileOutputStream(dest).getChannel();) {
            outputChannel.transferFrom(inputChannel, 0, inputChannel.size());
        }
    }
    public static void main(String[] args) throws IOException, ExecutionException, InterruptedException {
        copyFileUsingFileChannels(new File("source"), new File("target"));
    }
}
```

- Find and Replace using Regex Pattern matching

```
public class MatchAndReplace {
    public static void main(String[] args) {
        // Prepare the regular expression
        String regex = "\\b(\\d{3})(\\d{3})(\\d{4})\\b";
        String replacementText = "(+91) $1 $2-$3";
        String source = "8010106410, 2339829, and 8427220717";
        // Compile the regular expression
        Pattern p = Pattern.compile(regex);
        // Get Matcher object
        Matcher m = p.matcher(source);
        // Replace the phone numbers by formatted phone numbers
        String formattedSource = m.replaceAll(replacementText);
        System.out.println("Text: " + source);
        System.out.println("Formatted Text: " + formattedSource);
    }
}
```

- Print All Lines of File using Stream

```
// Print the lines in a file, then "done"
try (Stream lines = Files.lines(path, UTF_8)) {
    lines.onClose(() -> System.out.println("done"))
         .forEach(System.out::println);
}
```

- Sorting on multiple columns ignoring the case using Java 8 Lambda expression

```
people.sort(
    Comparator.comparing(Person::getLastName)
        .thenComparing(Person::getFirstName)
        .thenComparing(
            Person::getEmailAddress,
            Comparator.nullsLast(CASE_INSENSITIVE_ORDER)));
```

- List Files of a directory sorted by Last Modified Date

```
public void method1() throws IOException {
        Files.list(Paths.get("I:\\"))
                .map(Path::toFile)
                .sorted(Comparator.comparing(File::lastModified))
                .map(File::getName)
                .forEachOrdered(System.out::println);
    }

public void sortByLastModified() throws IOException {
    Files.list(Paths.get("D:\\git\\dli-downloader"))
            .sorted(comparingLong(s -> s.toFile().lastModified()))
            .map(Path::getFileName)
            .forEachOrdered(System.out::println);
}
public void sortByLatestModified() throws IOException {
    Files.list(Paths.get("D:\\git\\dli-downloader"))
            .sorted((comparingLong((Path s) -> s.toFile().lastModified())).reversed())
```

```
            .map(Path::getFileName)
            .forEachOrdered(System.out::println);
    }
```

- Joining Collections to make a Single String -

```
String ids = String.join(", ", ZoneId.getAvailableZoneIds());
List<String> nameList = asList("first", "second","third");
String names = String.join(", ", nameList);
```

If the input collection is not of type string, then a map function can be provided to map a given object to String using some method e.g. toString() in this case.

```
String listString = list.stream().map(Object::toString).collect(Collectors.joining(", "));
```

- Lazy Message Logging using Java 8

```
logger.finest(() -> "x: " + x + ", y:" + y);
```

- Printing all items of a Map

```
map.forEach(threshold,  (k, v) -> System.out.println(k + " -> " + v));
```

- Parallel Sort contents [words] of a Big file

```
String contents = new String(Files.readAllBytes(Paths.get("alice.txt")), StandardCharsets.UTF_8);
String[] words = contents.split("[\\P{L}]+"); // Split along nonletters
Arrays.parallelSort(words);
```

- Stream and ParallelStream usage

```
import static java.util.stream.Collectors.toList;
//Using Serial processing - single cpu
List<Apple> heavyApples =  inventory.stream().filter((Apple a) -> a.getWeight() > 150).collect(toList());
//Using Parallel processing - multiple cpu
List<Apple> heavyApples = inventory.parallelStream().filter((Apple a) -> a.getWeight() > 150).collect(toList());
```

- Find an Artist with maximum Name length

```
private static Comparator<Artist> byNameLength = comparing(artist -> artist.getName().length());
public static Artist byCollecting(List<Artist> artists) {
      return artists.stream()
                  .collect(Collectors.maxBy(byNameLength))
                  .orElseThrow(RuntimeException::new);
 }
```

- Find Sum of Squares for a given integer range

```
public static int sumOfSquares(IntStream range) {
   return range.parallel()
         .map(x -> x * x)
         .sum();
}
```

- Apply 12% VAT on the purchase items and calculate the Bill

```
List<Integer> costBeforeTax = Arrays.asList(100, 200, 300, 400, 500);
double bill = costBeforeTax.stream().map((cost) -> cost + 0.12*cost).reduce((sum, cost) -> sum + cost).get();
System.out.println("Total : " + bill);
```

Q 72. How will you replace tokens in a given text with properties loaded from a property file using Java Regular Expressions?

We can use java regex to find and replace special tokens from a given text.
We can switch on case insensitiveness of pattern by providing Pattern.CASE_INSENSITIVE in the Pattern.compile() method.

```
import java.util.HashMap;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegexTokenReplacer {
   private Pattern pattern = Pattern.compile("\\{(.+?)\\}", Pattern.CASE_INSENSITIVE);
   public String replaceTokens(String text, Map<String, String> replacements) {
      Matcher matcher = pattern.matcher(text);
      StringBuffer buffer = new StringBuffer();
      while (matcher.find()) {
         String replacement = replacements.get(matcher.group(1).toLowerCase());
         if (replacement != null) {
            matcher.appendReplacement(buffer, "");
            buffer.append(replacement);
         }
      }
      matcher.appendTail(buffer);
      return buffer.toString();
   }
   public static void main(String[] args) {
      RegexTokenReplacer tokenReplacer = new RegexTokenReplacer();
      Map<String, String> props = new HashMap<>();
      props.put("name", "shunya");
      props.put("contact", "+91808066XX");
      String replaceTokens = tokenReplacer.replaceTokens("my name is {Name}, {name}'s contact is {contact}",
props);
      System.out.println(replaceTokens);
   }
}
```

This program should print - my name is shunya, shunya's contact is +91808066XX

If we just want to replace all 'cat' words with 'dog' in a given paragrapgh (i.e. no special token recognizer), then we can use the below Java Program

```
public void replaceAllCatWithDog() {
   Pattern p = Pattern.compile("cat");
   Matcher m = p.matcher("one cat two cats in the yard");
   StringBuffer sb = new StringBuffer();
   while (m.find()) {
      m.appendReplacement(sb, "dog");
   }
   m.appendTail(sb);
   System.out.println(sb.toString());
}
```

Output should be - one dog two dogs in the yard

Q 73. How will you configure custom sized ThreadPool for Java 8 Stream API parallel operations?

Java does not provide any direct mechanism to control the number of threads and ThreadPool used by parallel() method in stream API, but there are two indirect way to configure the same.

1. Configure Default Common Pool

Its documented that parallel() method utilizes the common pool available per classloader per jvm, and we have a mechanism to control the configuration of that default common pool using below 3 System properties

- **java.util.concurrent.ForkJoinPool.common.parallelism** - the parallelism level, a non-negative integer
- **java.util.concurrent.ForkJoinPool.common.threadFactory** - the class name of a ForkJoinPool. ForkJoinWorkerThreadFactory
- **java.util.concurrent.ForkJoinPool.common.exceptionHandler** - the class name of a Thread. UncaughtExceptionHandler

For example, set the System property before calling the parallel stream

```
long start = System.currentTimeMillis();
IntStream s = IntStream.range(0, 20);
System.setProperty("java.util.concurrent.ForkJoinPool.common.parallelism", "20");
s.parallel().forEach(i -> {
   try {
      Thread.sleep(100);
   } catch (Exception ignore) {
   }
   System.out.print((System.currentTimeMillis() - start) + " ");
});
```

2. Run the parallel() operation inside a custom ForkJoinPool

There actually is a trick how to execute a parallel operation in a specific fork-join pool. If you execute it as a task in a fork-join pool, it stays there and does not use the common one. The trick is based on ForkJoinTask. fork documentation which specifies:

"*Arranges to asynchronously execute this task in the pool the current task is running in, if applicable, or using the ForkJoinPool.commonPool() if not in ForkJoinPool()*"

```
class StreamExampleJava8 {
   public static void main(String[] args) throws IOException, ExecutionException, InterruptedException {
      ForkJoinPool forkJoinPool = new ForkJoinPool(4); // Configure the number of threads
      forkJoinPool.submit(() -> IntStream.range(1, 1_000_000).parallel().filter(StreamExampleJava8::isPrime).boxed().
collect(toList())).get();
      forkJoinPool.shutdown();
   }

   private static boolean isPrime(long n) {
      return n > 1 && IntStream.rangeClosed(2, (int) sqrt(n)).noneMatch(divisor -> n % divisor == 0);
   }
}
```

Chapter 2

Core Java Interview Questions

Q 74. What are new features added in Java 8?

- Lambda Expressions enable us to treat functionality as the method argument, or say code as data. These expressions make single-method interface more compact. for example
  Single-method interface usage has become compact as seen in below code snippet
  Before Java 8,

```
Thread tOld = new Thread(new Runnable() {
        @Override
        public void run() {
            System.out.println("Munish Chandel");
        }
    });
```

Now, in Java 8

```
Thread t = new Thread(() -> System.out.println("Munish Chandel"));
```

Code Compactness in Collections Operations

```
people.stream()
    .filter(
        p -> p.getGender() == Person.Sex.MALE
            && p.getAge() >= 18
            && p.getAge() <= 25)
    .map(p -> p.getEmailAddress())
    .forEach(email -> System.out.println(email));
```

The above code snippet filters collection of people who are Male and in age range of 18-25, get their e-mail address and print them on System.out, few other examples of lambda expression to create a map of Person based on their Gender, using sequential and parallel approach

```
Map<Person.Sex, List<Person>> byGender =
roster.stream().collect(Collectors.groupingBy(Person::getGender));

ConcurrentMap<Person.Sex, List<Person>> byGender =
roster.parallelStream().collect(
Collectors.groupingByConcurrent(Person::getGender));
```

- Default methods enable new functionality to be added to the interfaces of libraries and ensure binary compatibility with code written for older versions of those interfaces.
- Optimistic Locking in Code using Stamped Locks provides very lightweight synchronization
- Concurrent Adders & Accumulators - DoubleAdder, DoubleAccumulator, LongAdder, LongAccumulator
- Array Parallel Sorting API
- Complete New Date API
- Functional Interfaces
- Stream API in Collections enables bulk operations, such as sequential and parallel map-reduce functions
- Improvements in ConcurrentHashMap - added methods for atomic & bulk operations

- New classes added to java.util.concurrent to support scalable updatable variables
- PermGen Space removed, Metaspace added

Q 75. What is difference between method overloading, method overriding, method and variable hiding?

Rules for Overriding[1]

In the overriding method,

- The argument list must exactly match that of the overridden method. If they don't then overloading will be the result instead of overriding
- The return type must be of covariant type (same class or sub-class) i.e. we can narrow down the return type
- Only throw the same or narrowed checked exception i.e. we can narrow down exception
- Access level can be less restrictive i.e. we can broaden the visibility of methods
- Free to throw any kind of Runtime exception
- Private and final methods are not inherited and hence can't be overridden
- Static methods can't be inherited and can't be overridden
- No inheritance no overriding
- The type of the actual object on the heap decides which method is selected at runtime

Rules for Overloading

- Method name must be the same
- Argument List must change in overloaded method
- Overloaded method can change the return type
- Access modifier of overloaded method can change
- New or Broader exceptions can be thrown by overloaded method
- Inherited method from the super class (non private) can be overloaded in the subclass
- Reference type determines which overloaded version is selected based on argument types at compile time

Method Signature

Method name, plus the number and type of its parameters constitute the method signature. Return type is not part of method signatures.

Best Practice to avoid any confusion for a overridden method

When overriding a method, you might want to use the @Override annotation that instructs the compiler that you intend to override a method in the superclass. If, for some reason, the compiler detects that the method does not exist in one of the superclasses, it will generate an error.[2]

Hiding Variables

Overriding works for Instance methods, In case of Class methods If a subclass defines a class method with the same signature as a class method in super class, the method in subclass hides the one is super class.

Similarly variable names are never overridden by the sub class, but they can be hide. If the super class contains a variable named x and subclass also contains x (irrespective of the type of variable) then the subclass variable hides the super class variable. Remember that all non-private super class variable can always be referenced by the subclass using super.variable.

In a subclass, you can overload the methods inherited from the superclass. Such overloaded methods neither hide nor override the superclass methods—they are new methods, unique to the subclass.

1 SCJP Sun® Certified Programmer for Java™ 6 Study Guide Exam (310-065) Page 106
2 http://docs.oracle.com/javase/tutorial/java/IandI/override.html

Question : What are different Access Levels in Java?
Answer : Access Levels available in Java are mentioned in the below table

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Question : If a method throws NullPointerException in super class, can we override it with a method which throws RuntimeException, please note that NPE is sub-class of RuntimeException?
Answer : Yes, it can throw - there is no policy for RuntimeException (unchecked exceptions) in method overriding.

Question : If a method throws FileNotFoundException in a super Class, can we override this method in sub-class by throwing IOException?
Answer : FileNotFoundException extends IOException, and both of these are Checked Exceptions. So As per overriding rules, sub-class method can only narrow down the scope of Exception i.e. overriding method can only throw the same Exception or sub-classes of that Exception. So overriding method can not throw IOException in this case.

Q 76. What is order of calling constructors in case of Inheritance?

Constructors are called from the top to down hierarchy. For example as shown in the below code snippet, A is super class of B. Creating a instance of new B() will invoke B's constructor which will in-turn call super() and this constructor of A will get invoked. Call to super() is inserted by the compiler on our behalf and is not visible to us. So instance of A will be created first, and if the constructor of A contains any method which is overridden in sub class, the subclass version will be invoked except for the static method calls.

Compiler introduces the call to super() only if the default no-arg constructor is present in the super class, otherwise its responsibility of the programmer to introduce such call with proper constructor arguments.

Java Source

```
class A {
   A() {
      greeting();
      prints();
   }
   void greeting() {
      System.out.println("instance method from A");
   }
   static void prints() {
      System.out.println("Static method from A");
   }

   public static void main(String[] args) {
      new B();
   }
}
class B extends A {
```

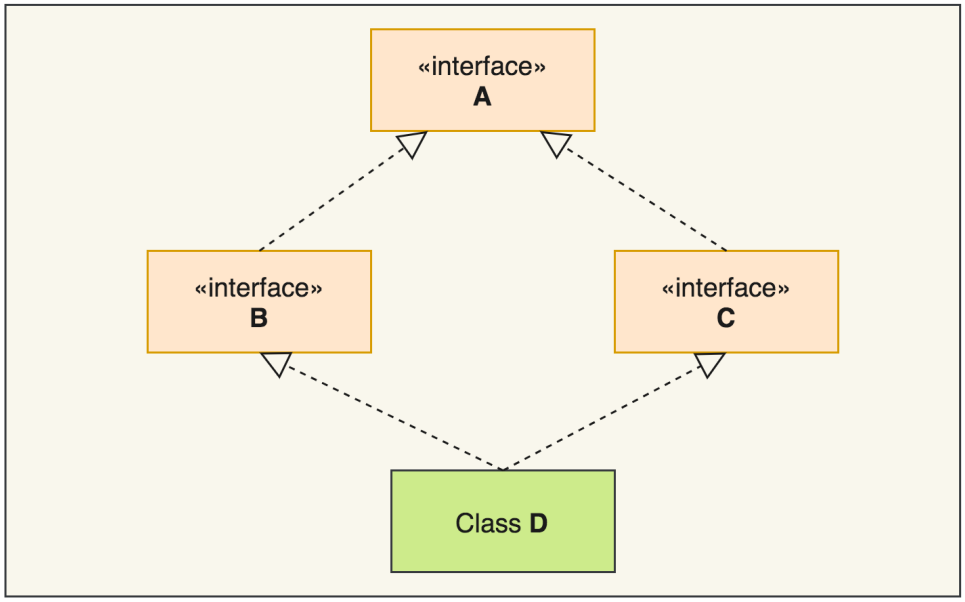
```
    B() {
/*Compiler will automatically insert calls to no-args A's constructor using super();*/
        greeting();
        prints();
    }
    void greeting() {
        System.out.println("instance method from B");
    }

    static void prints() {
        System.out.println("Static method from B");
    }
}
```

Program Output

instance method from B
Static method from A
instance method from B
Static method from B

Q 77. What is diamond problem of multiple inheritance?

Diamond Problem of inheritance is an ambiguity that can arise as a consequence of allowing multiple inheritance in language like C++. The below diagram illustrates the diamond problem of multiple inheritance.



Consider the below classes in C++

```
class A {
        void foo() { ... }
    }
    class B extends A {}
```

```
class C extends A {}

class D extends B, C {}
```

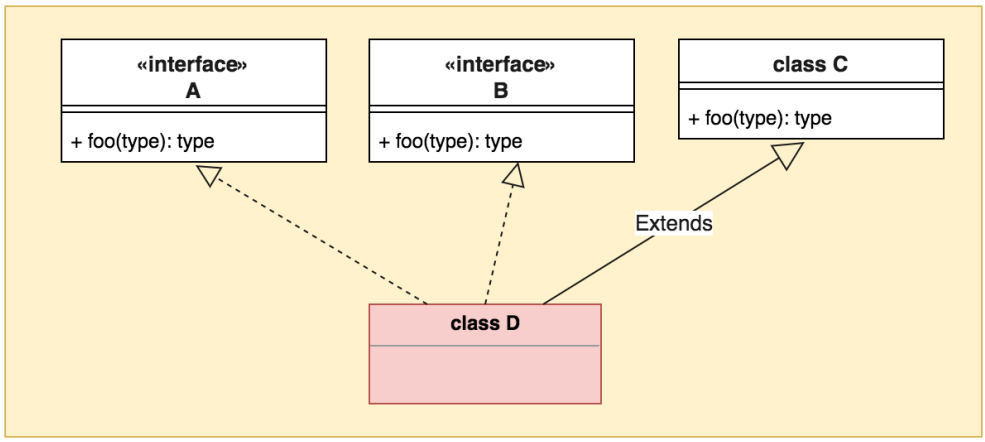
In the code above, the implementation of method `foo()` inherited by class D is unambiguously that defined by class A. But as soon as class B and class C starts to provide its own implementation for method `foo()`, the ambiguity will arrive in method resolution by Class D. This trap is known as diamond problem of multiple inheritance.
Since Java does not allow multiple inheritance for classes (only multiple interfaces are allowed), so diamond problem can not exist in Java. At any given point in time, a given Java class can extend from only one super class.

Q 78. How does Java 8 tackles diamond problem of multiple inheritance?

Java 8 brought a major change where interfaces can provide default implementation for its methods. Java designers kept in mind the diamond problem of inheritance while making this big change. There are clearly defined conflict resolution rules while inheriting default methods from interfaces using Java 8.

Rule 1

Any method inherited from a class or a superclass is given higher priority over any default method inherited from an interface.

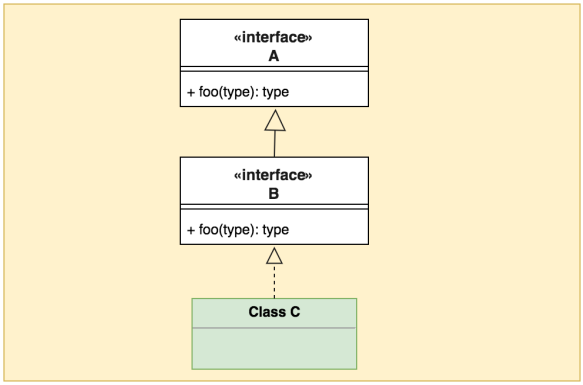


In the diagram above, `foo()` method of class D will inherit its implementation from class C.

Rule 2

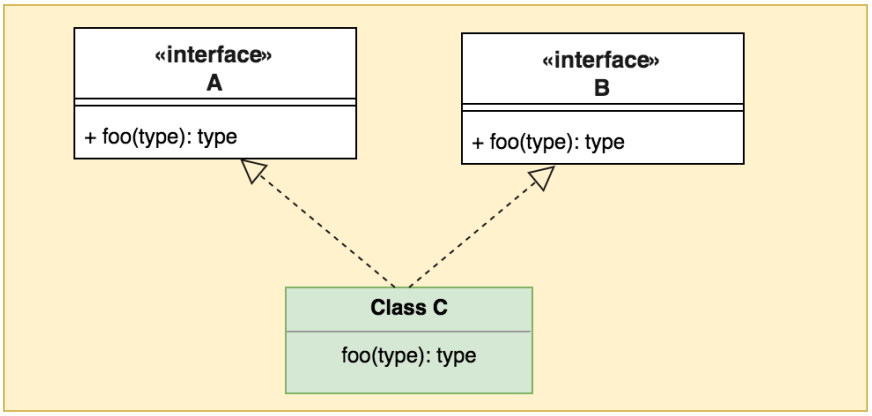
Derived interfaces or sub-interfaces take higher precedence than the interfaces higher-up in the inheritance hierarchy.

In the shown class diagram, `foo()` of class C will inherit its implementation from default method of interface B.



Rule 3

In case Rule 1 and Rule 2 are not able to resolve the conflict then the implementing class has to specifically override and provide a method with the same method definition.



In above class diagram, since interface A & B are at same level, to resolve conflict, class C must provide its own implementation by overriding method `foo()`.

Q 79. When should we choose Array, ArrayList, LinkedList over one another for a given Scenario and Why?

LinkedList (Doubly-linked list) and ArrayList (Resizable-array) both are two different implementations of List Interface.

LinkedList

LinkedList provides constant-time (Big O(1)) methods for insertion and removal using Iterators. But the methods to find the elements have Big O(n) time complexity (Linear Time, proportional to the size of list) and thus are poor performing. LinkedList has more memory overhead because it needs two nodes for each element which point to previous and next element in the LinkedList. If you are looking for random access of elements then ArrayList is the way to go for.

ArrayList

ArrayList on the other hand allows Big O(1) time complexity (constant time) for read/update methods. If position of the element is known then it can be grabbed in constant time using get(index) operation. But adding or removing elements from ArrayList (other than at end) requires shifting elements, either to make a new space for the element or for filling up the gap. Thus if frequent insertions and removals are required by your application logic then ArrayList will perform poorly (roughly Linear Time Big O(n)). The size, isEmpty, get, set, iterator, and listIterator operations run in constant time. Also if more elements are needed than the capacity of the ArrayList then a new underlying array with twice the capacity if created and the old array is copied to the new one which is time consuming operation (roughly Big O(n)). To avoid higher cost of resizing operation, we should always assign a appropriate initial capacity to the ArrayList at the time of construction.

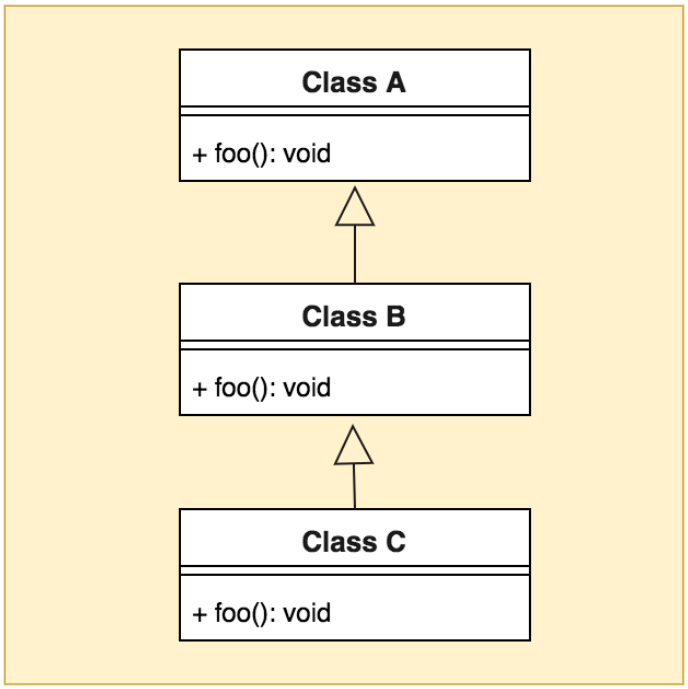
Array

Array is a fixed size primitive collection which can hold primitive or Objects. Array itself is a object and memory for array object is allocated on the Heap. Array does not provide useful collections methods like add(), addAll(), remove, iterator etc.
We should choose array only when the size of input is fixed and known in advance and underlying elements are of primitive type.

Q 80. We have 3 Classes A, B an C. Class C extends Class B and Class B extends Class A. Each class has an method foo(), is there a way to call A's foo() method from Class C?

Let's try to create a Class diagram for this scenario.

```
public class A {
   void foo() {System.out.println("Foo A");}
}

class B extends A {
   void foo() {System.out.println("Foo B");}
}

class C extends B {
   void foo() {
      System.out.println("Foo C");
   }

   public static void main(String[] args) {
      C c = new C();
      c.foo();
   }
}
```



Inheritance relationship among classes is shown in the diagram shown here, where every class has foo() method with the same signature thus Class B is overriding A's foo() method and Class C is overriding Class B's foo() method.

Now its possible for B and C to call their super class's foo() method by using super.foo() call.

But the interviewer is asking us to invoke A's foo() method from Class C, which is not possible because it violates the OOPs concept in Java. Java does not support multiple inheritance, that means C can see only a single super class which will have just one foo() method implementation. C can never see A's foo() method because otherwise how would it know which foo() method to invoke - B's or A's

The only way to do this

The only way it is possible to invoke A's foo() method from Class C is if Class B calls super.foo() method in its foo() implementation as shown below.

```
class B extends A {
    void foo() {super.foo();}
}
```

Q 81. Why wait is always used inside while loop as shown in the below snippet? Discuss all the probable reasons.

```
public synchronized void put(T element) throws InterruptedException {
      while(queue.size() == capacity) {
         wait();
      }
      queue.add(element);
      notify();
}
```

There are two main reasons that force use to use wait() method inside a while loop[1].

Spurious WakeUp[2]

In certain rare scenarios, a thread can wakeup without any reason even when no other thread signaled the condition. To gracefully handle those scenarios, we must recheck for the required condition before proceeding to execute the rest of the condition dependent code.

Multiple Threads Waiting for the Single Signals

If a thread calls notifyAll() upon meeting certain condition, then all the consumer threads will wakeup, even though only one thread will be expected to proceed in that scenario.
Let's analyze problem with the below mentioned Queue's take() method. Suppose there are 2 consumer threads awaiting for any new item on this shared queue. As soon as the Producer thread will put a single element into this queue, it will invoke notifyAll() and thus resuming all the 2 Consumer threads. Both the Consumer threads will come out of waiting state and will fight to acquire lock executing the rest of the code (line 5-7) one at a time. This will cause the second thread to throw exception because there was just one element in the queue.

```
1.  public synchronized T take() throws InterruptedException {
2.        if(queue.isEmpty()) {                    <=== Problematic If condition
3.           wait();
4.        }
5.        T item = queue.remove();
6.        notify();
7.        return item;
8.     }
```

Replacing **if** condition with a **while** loop can solve this problem without much effort. While loop will force each resuming thread to *test the condition on wakeup*, and putting the thread to waiting state again if required condition is not met.
So always remember to use wait() method from inside the while loop testing the condition that caused the thread to awaken, as shown below.

```
synchronized (obj) {
    while (<condition does not hold>)
       obj.wait(timeout);
    ... // Perform action appropriate to condition
}
```

1 see Section 3.2.3 in Doug Lea's "Concurrent Programming in Java (Second Edition)" , or Item 50 in Joshua Bloch's "Effective Java Programming Language Guide" (Addison-Wesley, 2001)
2 http://en.wikipedia.org/wiki/Spurious_wakeup

Q 82. We have a method which iterates over a Collection. We want to remove certain elements from that collection inside the loop in certain criteria is matched, How should we code this scenario?

Intent here is to check if you are aware of technique of modifying the collection structure while iterating over it. If we call collection.remove() from within the for loop then ConcurrentModificationException will be thrown by the JVM at runtime.

So lets take code snippet the given method

```
/***Failing Program, Never call Collection.remove(Object) while iterating***/
package org.shunya.interview;
import java.util.ArrayList;
import java.util.List;
import static java.util.Arrays.asList;

public class Test {
   public void removeFromCollection(List<Integer> marks) {
      for (Integer mark : marks) {
         if (mark < 40)
            marks.remove(mark);        //==> Will throw java.util.ConcurrentModificationException
      }
   }

   public static void main(String[] args) {
      Test test = new Test();
      test.removeFromCollection(new ArrayList<Integer>(asList(10, 20, 50, 60)));
   }
}
```

Actually, the right way to handle such scenario is to use Iterator to remove the element from the underlying Collection while iterating over it. ConcurrentModificationException is thrown because the for loop internally creates a fail-fast iterator which throws exception whenever it finds any structural modification in the underlying data structure (ArrayList in this case).

The correct Java 7 implementation for removal method would look something like,

```
public void removeFromCollection(List<Integer> marks) {
   for (Iterator<Integer> iterator = marks.iterator(); iterator.hasNext(); ) {
      Integer mark = iterator.next();
      if (mark < 40)
         iterator.remove();  //==> Safe to call remove() on Iterator
   }
}
```

Java 8 provides us with removeIf(Predicate) feature to remove objects from a collection, for example to remove all integers that are larger than 15, we can use the following code

```
public void removeFromCollection() {
   List<Integer> collect1 = Stream.of(10, 20, 30, 40).collect(toList());
   collect1.removeIf(integer -> integer > 15);
   collect1.forEach(System.out::println);
}
```

Q 83. We are writing an API which will accept a Collection<Integer> as an argument and duplicate an element in the Original Collection if certain criteria in met. How would you code such an API method?

This question is based on the fundamentals explored in the last question. If we try to modify a Collection inside a for loop without using an explicit Iterator then ConcurrentModicifactionException is thrown. So will not repeat the same wrong code in this solution.

Unfortunately Iterator does not provide any add() method in its interface, so it would be hard to use Iterator in this API to structurally modify the data structure. We are left with two options here -

1.) Use ListIterator's add() method which works only for LinkedList as the underlying data structure rather than any Collection.

```
public void addIntoCollection(LinkedList<Integer> marks) {
    for (ListIterator<Integer> iterator = marks.listIterator(); iterator.hasNext(); ) {
        Integer mark = iterator.next();
        if (mark < 40)
            iterator.add(mark);
    }
    System.out.println("marks = " + marks);
}
```

2.) Create another List and add stuff to that while we iterate over the input collection, and in the end append all elements of this newly created List to the original Collection.

```
public void addIntoCollection2(LinkedList<Integer> marks) {
    List<Integer> tempFooList = new ArrayList<Integer>();
    for (Integer mark : marks) {
        if (mark < 40)
            tempFooList.add(mark);
    }
    marks.addAll(tempFooList);
    System.out.println("marks = " + marks);
}
```

Q 84. If hashcode() method of an object always returns 0 then what will be the impact on the functionality of software?

Hashcode is used to fairly distribute elements inside a map into individual buckets. If the hashcode returned is zero for each element then the distribution will no more be fair and all the elements will end up into a single bucket. Each bucket in a HashMap contains list of HashEntry objects, so in a way HashMap will act as a map with single bucket holding all of its elements in a list. That will drastically reduce HashMap's performance to that of a LinkedList for get and put operations.
So time complexity of get and put method will become : Big O(n) instead of Big O(1)
Although, functionally it will still behave correctly.

Q 85. Iterator interface provides remove() method but no add() method. What could be the reason for such behavior?

Iterator interface contains three methods namely remove(), hasNext() and next().

It intentionally does not provide any add() method because it should not !
Iterator does not know much about the underlying collection. Underlying collection could be of any type (Set, ArrayList, LinkedList, etc) and might be offering the guaranteed ordering of its elements based on some algorithm. For example TreeSet maintains the order of its element using Red Black Tree datastructure. Now if iterator tries to add an element at a given location, then it might corrupt the state of the underlying data structure. And that is not the case while removing elements.
Thus Iterator does not provide any add() method.

List Iterator does provide the add() method because it know the location where it needs to add the newly created element as List preserves the order of its elements.

Q 86. What does Collections.unmodifiableCollection() do? Is it a good idea to use it safely in multi-threading scenario without synchronization, Is it immutable?

Collections.unmodifiableCollection() returns a unmodifiable dynamic view of underlying data structure. Any attempt direct or via iterator to modify this view throws UnsupportedOperationException, but any changes made in the underlying data structure will be reflected in the view.
This method is no substitute for the other thread safety techniques because iterating over a collection using this view may throw ConcurrentModificationException if original collection is structurally modified during the iteration.
So **external synchronization is must** if we are going to modify the underlying collection.
For example, the following code will throw ConcurrentModificationException in the for loop.

```
public class UnModifiableCollection {
   private List<String> names = new ArrayList<>();
   public void testConcurrency() {
      names.addAll(asList("1", "2", "3", "4"));
      Collection<String> dynamicView = Collections.unmodifiableCollection(names);
      for (String s : dynamicView) {//throws ConcurrentModification in 2nd iteration
         System.out.println("s = " + s);
         names.remove(0);          //The culprit line modifying the underlying collection
      }
   }
   public static void main(String[] args) {
      UnModifiableCollection test = new UnModifiableCollection();
      test.testConcurrency();
   }
}
```

It provides an immutable view to mutable collection, its not possible to modify the underlying collection through methods of this object.

Q 87. If we don't override hashcode() while using a object in hashing collection, what will be the impact?

Then the Object's default hashcode() method will be used to calculate the hashcode, which in turn will return the memory address of the object in hexadecimal format. So in a way the hashmap will behave like a identity

hashmap which will consider two elements equal if and only if two objects are same as per their memory address (and not logically). For example two String Objects with same contents might be treated different by this hashmap if they are different on heap.

Q 88. How would you detect a DeadLock in a running program?

Deadlock occurs in software program due to circular dependencies on shared resources by multiple threads. This causes the partial or complete hanging of the software program and the program itself can not recover from the dead lock situation. But still its possible using multiple ways to detect the Dead Lock in a running JVM.

1. Using Jconsole - JDK installation ships with jconsole tool which can connect to a running java process using JMX protocol. Jconsole can tell us whether there is a dead lock in the program or not.
2. Using JMX Management package as shown below

```
JDK 1.5 Has some API from java.lang.management package

ThreadMXBean tmx = ManagementFactory.getThreadMXBean();
long[] ids = tmx.findDeadlockedThreads();
```

This will list down the thread ids for the troubleshooting purpose.

Q 89. How would you avoid deadlock in a Java Program?

There are many tactics to write a deadlock free program in Java -

1. Avoid acquiring multiple locks at once, if it is absolutely required then always acquire the lock in the same order and release them in opposite order across the multiple methods/threads.
2. Avoid calling un-trusted foreign code while holding a lock. Time consuming calls should be avoided from within the locks.
3. Use timed & interruptible locks i.e. put a timeout on the lock attempt, if a thread is unable to acquire the lock within the given timeout value then it should free up all the acquired locks and retry after sometime. Java provides Lock Interface for this specific purpose.
4. Avoid locks using lock-free data-structures like ConcurrentLinkedQueue instead of a synchronized ArrayList, ConcurrentHashMap instead of synchronized Hashtable. Java provides many lock-free APIs in its atomic package like AtomicInteger, etc. Use compare and set (CAS) wherever possible.
5. Try to keep your locks local to your object and do not expose them for global access e.g. see ConcurrentHashMap implementation
6. Prefer Immutable types where we simply create an object copy upon modification, instead of sharing object data among threads.

Q 90. What is difference between Vector and ArrayList, why should one prefer ArrayList over Vector?

Main differences between the two dynamic List data structures are -

1. Vector methods is synchronized, ArrayList is not
2. Data Growth - Vector doubles its size when its full, ArrayList increases its size by 50% when its full.
3. Vector class retrofitted to implement the List interface, making it a member of the Java Collections Framework in JDK 1.2
4. Stack is sub-class of Vector class

One should normally use ArrayList - it offers better performance.
Vector is synchronized for concurrent modification. But Vector synchronizes on each individual operation. That's almost never what one want to do.

Generally you want to synchronize a whole sequence of operations. Synchronizing individual operations is useless for practical purpose (if you iterate over a Vector, for instance, you still need to take out a lock to avoid anyone else changing the collection at the same time, which otherwise would cause a ConcurrentModificationException in the iterating thread) and also slower due to repeated locking - once at individual method level inside Vector another at block level in your code.

In almost all scenario's you can utilize ArrayList in your application code, if you are looking for a synchronized version of List you can decorate a collection using the calls such as Collections.synchronizedList()

As for a Stack equivalent - you can have a look at Deque/ArrayDeque.

Q 91. How would you simulate DeadLock condition in Java?

DeadLock happens in multi-threaded scenario when two more threads have mutual dependencies on two or more shared resources. Let's understand with the following code,

```
import java.util.concurrent.TimeUnit;

public class DeadLock {
   static class Resource {
      final String name;
      Resource(String name) {this.name = name;}
      synchronized void print() {
         System.out.println("this is resource " + name);
      }

      synchronized void print(Resource another) {
         try {TimeUnit.MILLISECONDS.sleep(100);} catch (InterruptedException e) {}
         System.out.println("Thread "+Thread.currentThread().getName()+" acquired resource " + name);
         another.print();                    // --> The line that could cause a deadlock
    }
   }

   public static void main(String[] args) {
      final Resource r1 = new Resource("r1");
      final Resource r2 = new Resource("r2");
      new Thread(() -> r1.print(r2)).start();
      new Thread(() -> r2.print(r1)).start();
   }
}
```

In the above code, two threads operate over two shared Resources r1 and r2. Resource class has two synchronized methods (which will require the threads to obtain lock over the instance) and unfortunately r1 has a inter-dependency on r2. There is a great probability that the above code will block for ever causing a deadlock.
Using jconsole we can detect the deadlock, below is the message shown in jconsole for this java process

Name: Thread-1
State: BLOCKED on org.shunya.power.interview.DeadLock$Resource@354949 owned by: Thread-0
Total blocked: 2 Total waited: 1
Name: Thread-0
State: BLOCKED on org.shunya.power.interview.DeadLock$Resource@661a11 owned by: Thread-1
Total blocked: 1 Total waited: 1

Q 92. Which data type would you choose for storing currency values like Trading Price? What's your opinion about Float, Double and BigDecimal?

Float & Double are Bad for financial world, never use them for monetary calculations.

There are two main reasons supporting this statement -

- All floating point values that can represent a currency amount (in dollars and cents) can not be stored exactly as it is in the memory. So if we want to store 0.1 dollar (10 cents), float/double can not store it as it is. Let's try to understand this fact by taking this simple example

```
public class DoubleForCurrency {
   public static void main(String[] args) {
      double total = 0.2;
      for (int i = 0; i < 100; i++) {
         total += 0.2;
      }
      System.out.println("total = " + total);
   }
}
```

OUTPUT : total = 20.19999999999996
The output should have been 20.20 (20 dollars and 20 cents), but floating point calculation made it 20.1999999999..

- There is not much flexibility provided by Math.round() method for rounding the given calculation result compared to functionality offered by MathContext. RoundingMode provides options such as ROUND_UP, ROUND_DOWN, ROUND_CEILING, ROUND_FLOOR, ROUND_UNNECESSARY, etc

BigDecimal For the Rescue
BigDecimal represents a signed decimal number of arbitrary precision with an associated scale. BigDecimal provides full control over the precision and rounding of the number value. Virtually its possible to calculate value of pi to 2 billion decimal places using BigDecimal.

That's the reason we should always prefer BigDecimal or BigInteger for financial calculations.

Notes

Primitive type - int and long are also useful for monetary calculations if decimal precision is not required

We should really avoid using BigDecimal(double value) constructor instead prefer BigDecimal(String) because BigDecimal (0.1) results in 0.100000...5..3 being stored in BigDecimal instance. In contrast BigDecimal ("0.1") stores exactly 0.1

Question : What is Precision and Scale?
Precision is the total number of digits (or significant digits) of a real number
Scale specifies number of digits after decimal place

For example, 12.345 has precision of 5 and scale of 3

How to format BigDecimal Value without getting exponentiation in the result & Strip the trailing zeros?

We might get exponentiations in the calculation result if we do not follow some best practices while using Bigdecimal. Below is the code snippet which shows a good usage example of handling the calculation result using Bigdecimal.

```
import java.math.BigDecimal;
public class BigDecimalForCurrency {
   public static void main(String[] args) {
      int scale = 4;
      double value = 0.11111;
      BigDecimal tempBig = new BigDecimal(Double.toString(value));
      tempBig = tempBig.setScale(scale, BigDecimal.ROUND_HALF_EVEN);
      String strValue = tempBig.stripTrailingZeros().toPlainString();
      System.out.println("tempBig = " + strValue);
   }
}
```

How would you print a given currency value for Indian Locale (INR Currency)?

NumberFormat class is designed specifically for this purpose. Currency symbol & Rounding Mode is automatically set based on the locale using NumberFormat. Lets see this example

```
public static String formatRupees(double value) {
   NumberFormat format = NumberFormat.getCurrencyInstance(new Locale("en", "in"));
   format.setMinimumFractionDigits(2);
   format.setMaximumFractionDigits(5);
   return format.format(value);
}

public static void main(String[] args) {
   BigDecimal tempBig = new BigDecimal(22.1214);
   System.out.println("tempBig = " + formatRupees(tempBig.doubleValue()));
}
```

Output tempBig = Rs.22.12

Thus everything is taken care by NumberFormat, nothing lese to worry about.

Some precautions

- BigDecimal(String) constructor should always be preferred over BigDecimal(Double)
- Convert Double value to string using Double.toString(double) method
- Rounding mode should be provided while setting the scale
- StripTrailingZeros chops off all the trailing zeros
- toString() may use scientific notation but, toPlainString() will never return exponentiation in its result

For further reading -

https://blogs.oracle.com/CoreJavaTechTips/entry/the_need_for_bigdecimal

Q 93. How would you round a double value to certain decimal Precision and Scale?

Firstly let us understand the difference between Precision and Scale.

If the number is 9232.129394, then
Precision represents the number of number of significant digits to which a number is calculated i.e. 4 digits (9232)
Scale represents the number of digits to the right of the decimal point i.e. 6 in above case (129394)

Some other examples are,

Precision 4, scale 2: 99.99
Precision 10, scale 0: 9999999999
Precision 8, scale 3: 99999.999
Precision 5, scale -3: 99999000

No one wants to loose the precision of the number as it will change the value by large amount. If you still want to loose the precision simply divide the number by 10 to the power precision.

There are multiple ways in Java to round the double value to certain scale, as mentioned in the below example

```
import java.math.BigDecimal;
import java.math.RoundingMode;
import java.text.DecimalFormat;

public class RoundDouble {
   public double round1(double input, int scale) {
      BigDecimal bigDecimal = new BigDecimal(input).setScale(scale, RoundingMode.HALF_EVEN);
      return bigDecimal.doubleValue();
   }

   public double round2(double input) {
      return Math.round(input * 100) / 100.0d;
   }

   public double round3(double input) {
      DecimalFormat df = new DecimalFormat("#.00");
      return Double.parseDouble(df.format(input));
   }

   public static void main(String[] args) {
      RoundDouble rd = new RoundDouble();
      System.out.println(rd.round1(9232.129394d, 2));
      System.out.println(rd.round2(9232.129394d));
      System.out.println(rd.round3(9232.129394d));
   }
}
```

The first method of rounding using BigDecimal should be preferred in most scenarios.

Q 94. How great is the Idea of synchronizing the getter methods of a shared mutable state? What if we don't?

From **Effective Java 2nd Edition - Item 66**
When multiple threads share mutable data, each thread that reads or writes the data must perform synchronization. In fact, synchronization has no effect unless both read and write operations are synchronized.

Synchronization serves two major purposes in a multi-threaded scenario, one is atomicity of the operation and second is the memory visibility of the changes made by one thread to all other threads (Brian Goetz article on read and write barriers)[1]. In case of getters the changes made to the shared variable will get reflected to the new thread if the code block is synchronized, otherwise dirty reads may happen and the thread may see the stale state of the shared object.

So all the methods returning the mutable protected state of the shared object must be synchronized unless the field returned is immutable, final or volatile.
Let's take example of a simple Counter Class.

```
public class Counter {
   private int c = 0;
   public synchronized void increment() {
      c++;
   }

     /**Getter Must be synchronized to see the guaranteed correct value*/
   public synchronized int getValue() {
      return c;
   }
}
```

That's the reason that get() method of vector class is synchronized & must be synchronized.

Q 95. Can the keys in Hashing data structure be made Mutable?

Interviewer's intent is to know how good you know about Hashing Data Structure

The answer is **NO**. If we make the keys mutable then the hashcode() of the key will no more be consistent over time which will cause lookup failure for that object from the data structure. Let's analyze this example.

```
public void testMutableKey() {
   Map<MutableKey, Object> testMap = new HashMap<>();
   MutableKey mutableKey = new MutableKey();
   mutableKey.setName("TestName");
   testMap.put(mutableKey, new Object());
   Object o = testMap.get(mutableKey);
   System.out.println("before changing key = " + o);
   mutableKey.setName("abc");<====Problematic Instruction
   o = testMap.get(mutableKey);
   System.out.println("after changing key = " + o);
}

public static void main(String[] args) {
   MutableHashKey test = new MutableHashKey();
   test.testMutableKey();
}
```

1 http://www.ibm.com/developerworks/library/j-jtp08223/

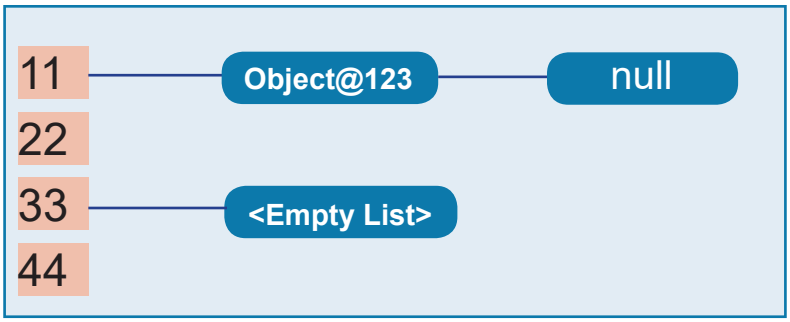
Program Output :
before changing key = java.lang.Object@489bb457
after changing key = null

From the above example we see that as soon as we change the key, we are not able to get the associated object from the Map.

Let's see what's happening inside

When we put the mutableKey to HashMap then hashcode() is calculated for the key, suppose it comes out to be 11. So the Object123 is successfully inserted into the HashMap at bucket Location 11.

Then we modify the key and try to get the object. HashMap's get() method again calculates the hashcode of the Key, since the Key is changed in between, so suppose hashcode() comes out to be 33 this time. Now the get() method goes to the bucket at address 33 and tries to retrieve the object, but it find nothing over there and returns the null.



HashMap Internal Structure

Never make changes to the hashmap's key, otherwise the associated object can not be fetched using get() method. Though it will be accessible using other methods which iterate over the entire collection.

Q 96. Is it safe to iterate over collection returned by Collections.synchronizedCollection() method, or should we synchronize the Iterating code?

We should synchronize the code block doing any kind of iteration as stated by the Java Docs

```
public static <T> Collection<T> synchronizedCollection(Collection<T> c)
```

Returns a synchronized (thread-safe) collection backed by the specified collection. In order to guarantee serial access, it is critical that all access to the backing collection is accomplished through the returned collection.
It is imperative that the user manually synchronize on the returned collection when iterating over it:

```
Collection c = Collections.synchronizedCollection(myCollection);
   ...
synchronized (c) {
    Iterator i = c.iterator(); // Must be in the synchronized block
    while (i.hasNext())
       foo(i.next());
}
```

Failure to follow this advice may result in non-deterministic behavior.

Q 97. What are different type of Inner classes in Java? How to choose a type with example?

An inner class is a class defined within another class, or even within an expression. They are mostly used to simply our code by putting closely related classes together in one source file, instead creating class burst. Event handlers are best examples of Inner Classes.

Types of Inner Class -
- Regular Inner Classes (classes defined within the curly braces of a regular class)
- Static Inner Classes (that can be accessed without having an instance of outer class)
- Method-local Inner Classes (Inner class defined within a method body)
- Anonymous Inner Classes (Without any class name)

Notes

Question : Why do we need to declare a local variable final if inner class declare within a method needs to use it?
Local variables always live on the stack, the moment method is over all local variables are gone. Inner class objects might be on heap even after the method is over, so in that case it would not be able to access the local variable, since they are gone. There is also a possibility that the variable could change before the inner class accesses it. Making the local variable final prevents these scenarios.

Q 98. When should we need a static inner class rather than creating a top level class in Java program?

A static Class interacts with the instance members of its outer class and other classes just like any top level class. In fact, a static nested class is behaviorally a top level class that has been nested in another top level class for packaging convenience.

If we take a example of LinkedList.Entry class, there is no need of it being a top level class as it is only used by LinkedList. Otherwise it will cause class burst inside a package, moreover there are other static inner classes by the same name as well like Map.Entry
And since these does need access to LinkedList/Map's internal so it makes sense to make them static inner classes.

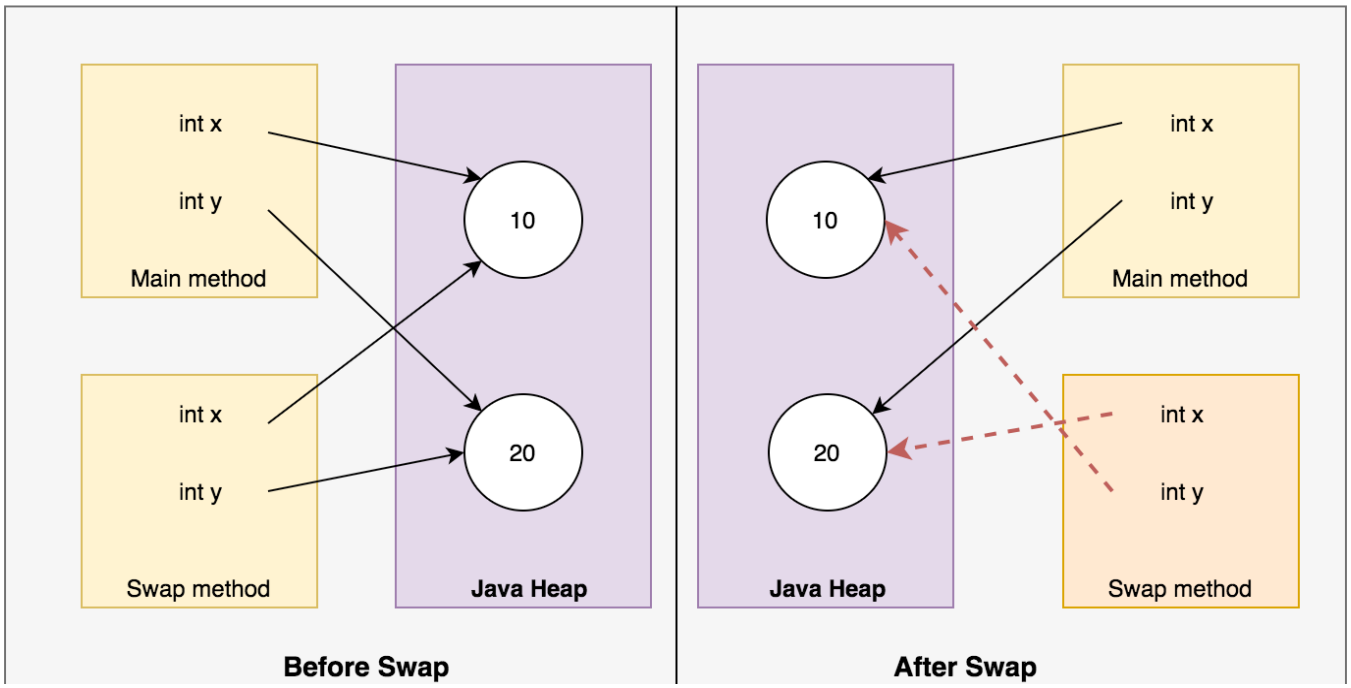
Why to use it?
1. It is a way of logically grouping the classes that are only used in one place. If a class is useful to only one other class, then it is logical to embed it in that class and keep the two together.
2. It increases encapsulation.
3. Nested classes can lead to more readable and maintainable code. Nesting small classes within top-level classes places the code closer to where it is used.

Examples
Iterator in most of the collection types are implemented as a inner class and Entry is implemented as static inner class.

Q 99. How does method parameter passing works in Java? Does it pass-by-reference or pass-by-value?

Method parameters are always pass-by-value [of-the-reference] in Java language irrespective of the type of variable (primitive or objects). Lets consider the below scenario -

```
public class Swap {
   public static void main(String[] args) {
      Integer x = 10;
      Integer y = 20;
      swap(x, y);
      System.out.println("x = " + x + ", y = " + y);
   }

   static void swap(Integer a, Integer b) {
      Integer tmp = a;
      a = b;
      b = tmp;
   }
}
```



Here, when we call method swap(x, y) from main() method, then we pass the value of reference x and y (i.e. Value stored at memory address S1 and S2) rather than entire object from heap. In swap method, two new stack variables are created with name a and b, pointing to address H1 and H2 on heap. Now when you reassign these method variables while doing swap, you are actually changing the memory location that a and b are pointing to. Since these method variables are local to method swap, so variable x and y in main method will not see any effect of swap. In nutshell, swap method can not change values pointed by variable x and y in main method.

For a primitive variable, a copy of the bits representing the value is passed to the method. For example, if you

pass an int variable with the value of x, you're passing a copy of the bits representing x. The called method then gets its own copy of the value, to do with it what it likes.

For Objects, a copy of the bits representing the reference to an object is passed to the method. The called method then gets its own copy of the reference variable, to do with it what it likes. **But because two identical reference variables refer to the exact same object, if the called method modifies the object (by invoking setter methods, for example), the caller will see that the object the caller's original variable refers to has also been changed.**

The bottom line on pass-by-value: the called method can't change the caller's variable, although for object reference variables, the called method can change the object the variable referred to. What's the difference between changing the variable and changing the object?

That's the reason we can never write a method in Java which can swap two Integers or even mutable objects.

Q 100. Is it possible to write a method in Java which swaps two int/Integer?

The answer is **No**.

One must understand two basic concepts -

1. Everything (primitive and objects) is passed by value in Java method calls
2. All Wrapper classes (Integer, Double, etc.) are Immutable by design, modification of same object is not possible. If you add two Integers then third Integer will be created on fly to hold the results.

Incase of Integer Wrapper Class

For objects, the reference to the Object are copied by value to the calling method. If we reassign these reference copies then the changes will not be reflected in the method calling this swap(x,y).

```
// This code will never work as intended
public void swap(Integer x, Integer y){
      Integer tmp =x;
       x=y;
       y=tmp;
   }
```

Above implementation will change the values referenced by variable x and y, but it will not reflect in the method that calls swap(Integer, Integer). Thus swap will not work as intended.

The only way to have this possible was using some kind of setter on Integer class which could have modified the underlying value. But Java declares all Wrapper classes as Immutable for thread-safety perspective, thus there is no way to swap Integers in Java.

So how will you make it work?

Create a mutable wrapper class that holds these Integer values, then pass this mutable Wrapper to the swap method. Swap method shall set (not reassign) value on these wrapper objects while doing swap. Correct working Java Code is shown below -

```
public class SwapValues {
   static class Wrapper<T> {
      T value;

      public Wrapper(T value) {
         this.value = value;
      }
   }

   public static void main(String[] args) {

      Wrapper<Integer> first = new Wrapper<>(10);
      Wrapper<Integer> second = new Wrapper<>(20);

      swap(first, second);

      System.out.println(first.value);
      System.out.println(second.value);
   }

   static void swap(Wrapper<Integer> first, Wrapper<Integer> second) {
      Integer tmp = first.value;

      first.value = second.value;
      second.value = tmp;
   }
}
```

Q 101. What all collections require hashcode() method?

Only hashing data structures uses hashcode() method along with equals() method, though the equals() is used by almost every class.
hashcode is useful for creating hashing based datastructures like Hashtable, HashMap, LinkedHashMap, ConcurrentHashMap, HashSet. (Basically any Java collection that has Hash inside the name of it)

Hashcode is used to provide best case O(1) time complexity for searching the stored element.

TreeMap, TreeSet uses Comparator/Comparable for comparing the elements against each other, so these data structures do not require hashcode() method. The best case time complexity offered by these datastructures for lookup operation is logarithmic O (log n) compared to O (1) offered by hashing datastructures.

Q 102. What is problem with mutable static variables in Java class?

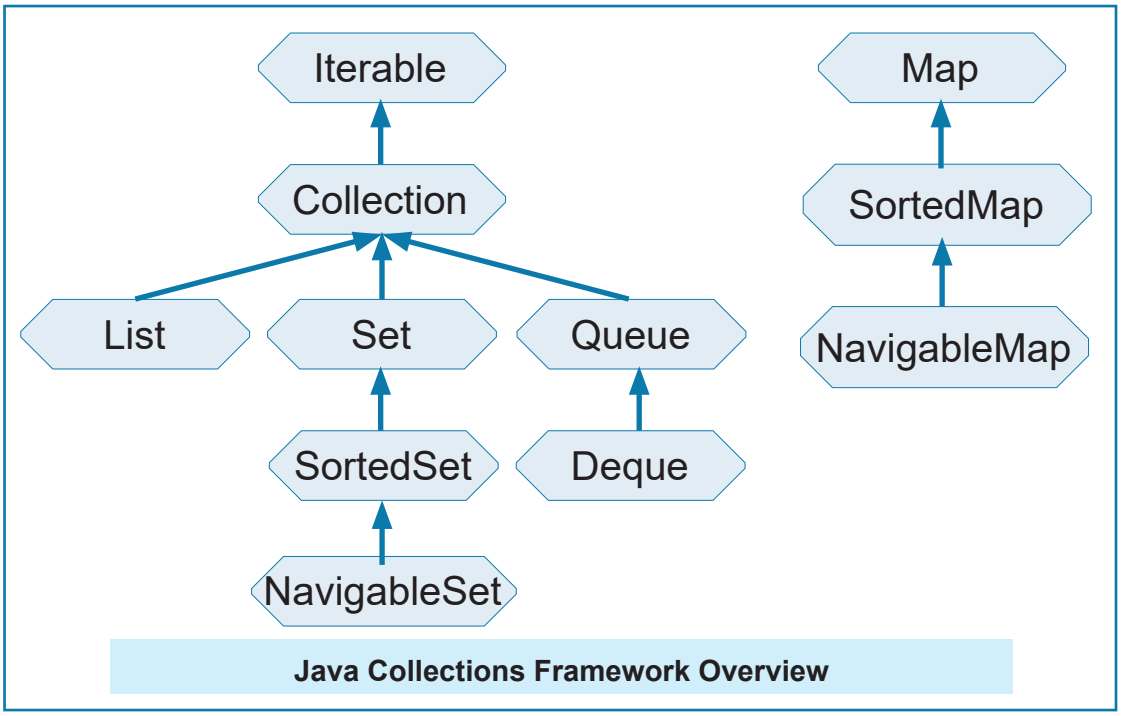
Using mutable static variables make your java code vulnerable to bugs

- Problem sharing a mutable static variable in multi-threaded environment. It's very tough to write & maintain a thread safe code with Mutable non-private static fields.
- Problem in Single Threaded design because we have to be very careful while updating static variable, since the next bit of code might expect some other state for the same.
- Code that relies on static objects can't be easily unit tested, and statics can't be easily mocked and hence does not promote TDD.

- If you are using static keyword without final for declaring a fields then you should reconsider your design, since the mutable static fields can be just dangerous !!

Q 103. Provide a diagram for collections framework.

Collections framework contains two main Interfaces Collection and Map. Collection is further extended by List, Set and Queue Interface.



There is a separate utility class named *Collections* which provides various static factory methods for playing with collections (Algorithm Part)

There are multitude Implementations provided for the above mentioned interfaces, few implementations implements more than one such Interface.

Set - A collection that does not allow duplicate elements (models mathematical set abstraction) and represents entities such as courses making up of student's schedule, ISBN number of books, Social security Number, PAN number, processes running on a machine, etc

List - A collection that maintains order of its elements. Lists can contain duplicate elements. ListIterator provides precise control over where to add the new item to the collection.

Queue - Queue is a First In First Out data structure which maintains order of its original elements. Most List implementations like LinkedList implements Queue interface as well.

Q 104. What is Immutable Class. Why would you choose it? How would you make a class immutable?

What is Immutable Object
When the state of object can not be changed after its construction then the object is called Immutable.

Why do we need it
Immutable objects are inherently thread-safe, thus help writing multi-threading code without much worries. Immutable questions are meant for multi-threading program. *If someone is talking bout immutability then indirectly he is talking about multi-threaded context*. Immutable classes are easy to understand, as they possess a single state, which is controlled by their constructor. Immutable objects are good candidate for hash keys because their hashcode can be cached and reused for better performance.

Which Objects should be Immutable
Immutable classes are ideal for representing ADT's (Abstract Data Type) value.
Joshua Bloch suggests that
"*All classes should be designed to be immutable unless there is a specific reason not to do so*"

Guidelines for Making a class Immutable
1. All fields should be declared final
2. Class itself is declared final so that the derived classes do not make it Mutable.
3. ***this*** reference should not be allowed to escape during object construction such as in anonymous inner classes (for example adding action listener)
4. Any field that contains reference to mutable objects (such as arrays, collections, StringBuffer, etc)
   i. Are private
   ii. Are never returned or exposed to the caller
   iii. Are the only reference to the Objects that they refer
   iv. Do not change the state of the referenced object after the construction.
   v. If mutable fields must be returned to the caller, then a defensive copy should be returned so that the changes do not reflect in the inner data structure.

```
public List<String> getList() {
    return Collections.unmodifiableList(list);   <=== defensive copy of the mutable field before returning it to caller
}
```

   vi. If a mutable Object is passed in the constructor (like an array), then Immutable class should first make a defensive copy of the mutable object before storing its reference.

How does this reference escape during object construction?
When we construct a non static anonymous inner class, the class always has a pointer to the outer class, thus making all the fields available to that inner class. The compiler substitutes the reference of the outer class on behalf of us.

An Object is said to be Immutable when,
Its state can't be changed after construction, all its fields are final and this reference does not escape during construction.

Example of Immutable Classes in JDK
All primitive Wrapper classes (Number, Integer, Long, Float, Double, etc), String Class, Color Class, BigInteger & BigDecimal class, CopyOnWriteArrayList & CopyOnWriteArraySet

Q 105. Discuss Exception class hierarchy in Java. When should we extend our custom exception from RuntimeException or Exception?

Throwable sits at the top of Java Exception Class hierarchy. Exception and Error are the two direct subclasses that extends Throwable.

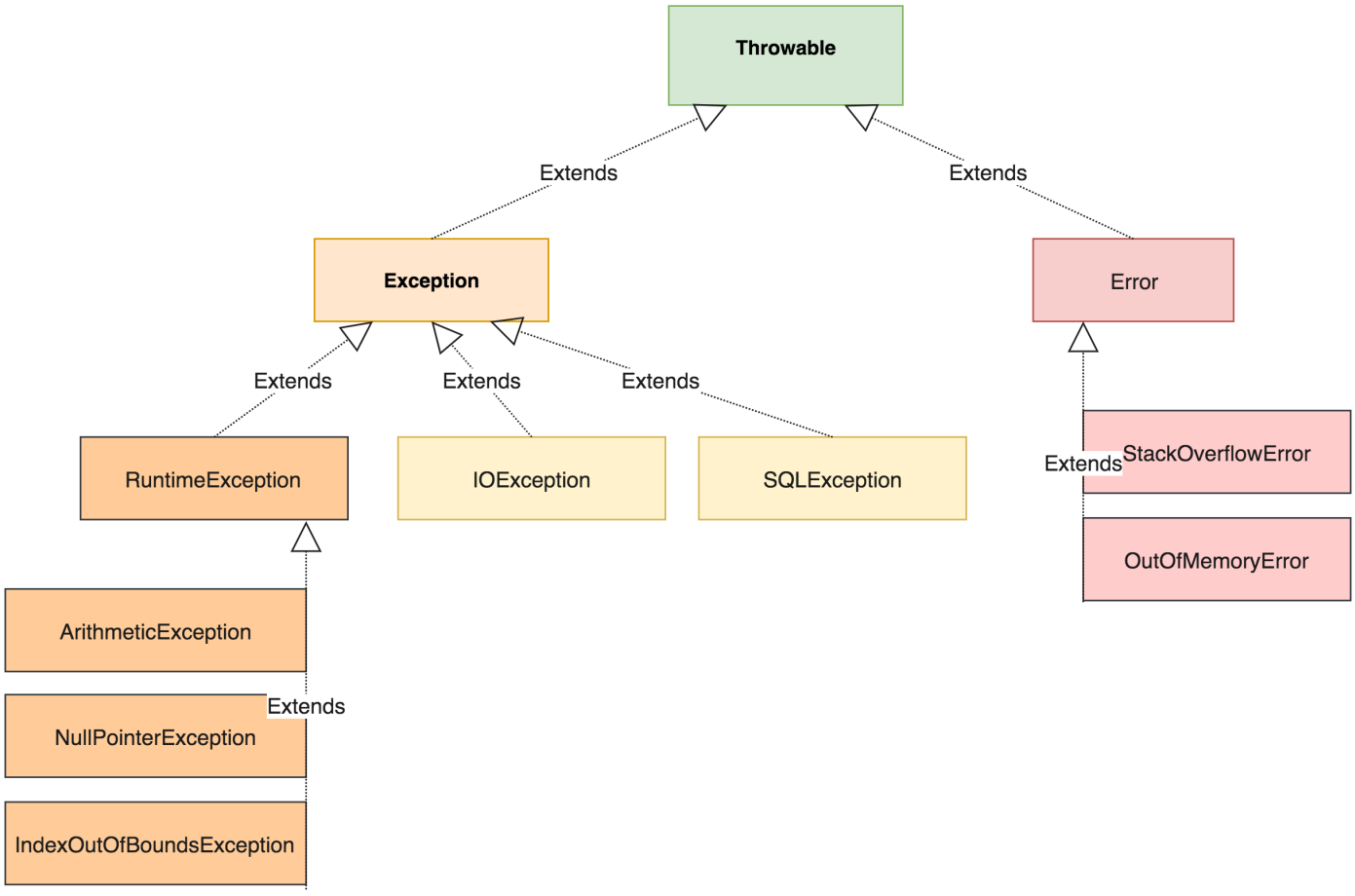
Logically, there are three main types of Exceptions in Java:

Checked Exceptions Represents exceptional scenario which if occurred, must dealt with in some way. example is IOException, FileNotFoundException. We need to declare these exceptions along with the code dealing with such scenarios. Custom checked exceptions can be created by extending your class from java. lang.Exception Class.

Unchecked/Runtime Exceptions Represents an error in our program's logic which can not be reasonably recovered from at run time, for example NullPointerException, ArrayIndexOutOfBoundsException. We do not need to declare/catch such exception in the method signature because these are not expected by any programmer. Custom unchecked exceptions can be created by extending from RuntimeException

Error is a subclass of Throwable that indicates serious problems that a reasonable application should not try to catch. A custom error can be created by extending our class from Throwable.

Below is the rough diagram for Java Exception Class Hierarchy:



Q 106. How does an ArrayList expands itself when its maximum capacity is reached?

When the internal array of an ArrayList becomes full, then new array with double the capacity is created efficiently by the ArrayList using the following method.

```
elementData = Arrays.copyOf(elementData, newCapacity);
```

If it needs to shift the elements in order to add something over the existing index, then it displaces the elements using following System method -

```
System.arraycopy(elementData, index, elementData, index + 1, size - index);
```

If we know in advance the capacity requirements for the ArrayList object, then we should always create the ArrayList with that capacity to reduce the amount of incremental reallocation effort.

Also, as of Java 8, when an Array reaches its capacity (capacity is not size but length of internal array, for arraylist), its size is increased 1.5 times compared to earlier capacity. This is excerpt from Java Source code -

```
int newCapacity = oldCapacity + (oldCapacity >> 1);
```

which is 1 + 1/2 = 1.5 times the older capacity. Please note that right shifting an integer by 1 is equivalent to division by 2.

See more on bitwise operator related questions in this book.

Q 107. How does HashMap expands itself when threshold is reached?

Incase of hashmap, there is a concept of load factor. When size of the hashmap reaches (capacity * load factor), then hashmap grows itself to double its previous size.
Please note that the size of hashmap is always kept in power of two in order to reduce the runtime complexity for finding the hash for given key. Hashing function utilizes bitwise operation (XOR and right Shift) instead of modulus operator to speed up the lookup process. Below is the excerpt from Java 8 HashMap Source

```
static final int hash(Object key) {
   int h;
   return (key == null) ? 0 : (h = key.hashCode()) ^ (h >>> 16);
}
```

Q 108. What is StringPool In Java?

JVM has a string pool where it keeps at most one object of any String. String literals objects are always created in the string pool for reusability purpose. String objects created with the new operator do not refer to objects in the string pool.

Pool generally resides in PermGen space as of JDK 1.6 (not much documentation is found on this)
The pooling concept has been made possible because the String objects are Immutable in Java.

Q 109. What is instance level locking and class level locking?

In Java, every object has a built in lock (or monitor) that gets activated when Object has synchronized method code. It is basically of two types

Instance Lock

When we enter a non-static synchronized code then JVM acquires instance level lock on the Object whose method we are executing. Instance lock can also we acquired using the following syntax with block level synchronization.

Notes about instance locking

- Lock is mutually exclusive, which means that only one thread can acquire it and other threads have to wait for their turn until first thread releases it.
- Each Java object has just one lock (or monitor)
- Non-synchronized methods (& a single static synchronized method) can be executed in parallel with a single synchronized method.
- If a thread goes to sleep then it holds any lock it has.
- A single thread can acquire multiple lock on multiple objects.
- Lock is Reentrant, meaning that same thread can acquire the same lock multiple times (2^{31} times)

```
private static final Object lock = new Object();
synchronized(lock){
…
}
```

Class Lock (Locking on static methods or on Class object)

It is a lock acquired over java.lang.Class of an Object and is used to protect static methods of that class from thread safety point of view. There are two ways to acquire such lock - First one is using a static synchronized method and second one using block level synchronization over Object.getClass() as shown below.

```
synchronized(MyClass.class){
…
}
```

Example

Lets take this sample example to understand class and instance level Lock.

```
public class MyClass{
public static synchronized classMethod(){ ... }

public synchronized void instanceMethod(){ ... }
}
private MyClass reference = new MyClass();
```

1. One Thread can call reference.classMethod() and other thread can call reference.instanceMethod() in parallel because class level and instance level locks do not interfare.
2. But both the threads can't call the same instanceMethod() or classMethod() in parallel, because of the Mutual Exclusiveness of the Instance Lock and Class Lock.

Q 110. Explain threading jargons?

Race Condition
A race condition occurs when the correctness of a computation depends on the relative timing of multiple threads by the runtime. In this scenario Getting the right result relies on the lucky timings.
Dead Lock
Dead lock occurs when two or more threads are blocked forever, waiting for each other to release up the shared resource. For two threads, it happens when two threads have a circular dependency on a pair of synchronized shared resources.
Starvation
Describes a situation where a thread is unable to gain regular access to shared resource and is unable to make any progress
This happens when shared resources are made unavailable for long periods by "greedy" threads. For example, suppose an object provides a synchronized method that often takes a long time to return. If one thread invokes this method frequently, other threads that also need frequent synchronized access to the same object will often be blocked.
Mutex
Mutex stands for mutually exclusive, only one kind of operation (READ or WRITE) is allowed at a given time frame.
Live Lock
A thread often acts in response to the action of other threads, if the other thread's action is also in response to another thread, then live lock may result. No progress but threads are not blocked.
Synchronizer
A synchronizer is any object that coordinates the control of flow of threads based on its state. For example, semaphore, CountDownLatch, FutureTask, Exchanger, CyclicBarrier, etc.
Latch
A synchronizer that can delay the progress of threads until it reaches the terminal state.
Semaphore
Counting semaphore are used to control the number of activities that can access a certain shared resource or perform a given action at the same time. Semaphores are normally used implement resource pool or to impose a bound on a collection.
Exchanger
A two party barrier in which the parties exchange data at the barrier point.

Question : How will you Produce a Race Condition in your Java Program?
Answer
The following Class's object, if called from two different threads, can produce a Race Condition. When two thread will try to add value to the single shared object, chances are there that race condition will occur and as a result, the outcome of add method will be unpredictable.

```
public class RaceCondition {
   protected long money = 0;

   public void add(long value){
      this.money = this.money + value;
   }
}
```

Q 111. What is float-int implicit conversion while doing calculation on mixed data type in Java?

As per Java Language Specifications

If at least one of the operands to a binary operator is of floating-point type, then the operation is a floating-point operation, even if the other is integral.

For example let's consider the following example

```
int i1 = 5;
float f = 0.5f;
int i2 = 2;
System.out.println(i1 * f);              // Result will be a float
System.out.println(i1 / i2);             // Result will be an integer
System.out.println(((float) i1) / i2);   // Result will be a float
```

Result

```
2.5
2
2.5
```

Primitive Casting
Casting lets you convert primitive values from one type to another. Casts can be explicit (narrowing conversions) or implicit (widening the conversions).
Compiler does implicit conversion when you try to put smaller item into bigger bucket but not the other way.
By default all literal integers are implicitly interpreted as int by the compiler. for example,
int x = 27; //Literal assignment

Q 112. Discuss Comparable and Comparator? Which one should be used in a given scenario?

Comparable and Comparator both are used for allowing sorting a collection of objects.

Comparable should be used to define the natural ordering behavior of an Object. Normally a class implements this interface to define the natural ordering behavior of its objects. For example, java.lang.String implements Comparable interface to provide natural order to Strings (Compares two strings lexicographically)

Comparator should be used to provide an external controllable ordering behavior which can override the default ordering behavior (natural ordering) and when we might require different type of ordering behavior for the same Object. Comparator is implemented like an Adaptor Design Pattern where a separate class is dedicated for providing the comparison behavior.

Using Comparator in Java 8 is very easy with Lambda Expression

```
Arrays.sort(strings, (a, b) -> a.compareTo(b));
Arrays.sort(strings, String::compareToIgnoreCase);  // this case using method reference
Arrays.sort(people, Comparator.comparing(Person::getLastName));  // sorts collection of people with lastname
```

Q 113. How would you sort a collection of data based on two properties of an entity in Java, analogical to SQL's Order by firstField, SecondField desc?

Sorting based on multiple Object properties is easily achievable in Java Collections Framework. We just need to redesign our Comparator to accommodate for multiple fields. Let's see how can we achieve that .

Lets assume we want to sort Person objects based on Age, and then Name (when two person has same age).

```
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
public class Sorting {
    static class Person {
        String name;
        int age;
        Person(String name, int age) {
            this.name = name;
            this.age = age;
        }
        @Override
        public String toString() {return "name=" + name + ",age=" + age + '}';}
    }
    public static void main(String[] args) {
        List<Person> persons = new ArrayList<>();
        persons.add(new Person("Second", 26));
        java.util.Collections.sort(persons, new Comparator<Person>() {
            @Override
            public int compare(Person o1, Person o2) {
                if (o1.age == o2.age) {
                    return o1.name.compareToIgnoreCase(o2.name);
                } else if (o1.age < o2.age) {
                    return -1;
                }
                return 1;
            }
        });
        System.out.println(java.util.Arrays.toString(persons.toArray()));
    }
}
```

In the above code snippet, we can see the implementation for Person Comparator (code highlighted in red), whenever age of two persons is equal, then return the result based on Name comparison. That's quite easy?

Achieving the same result in Java 8 is compact and easy, as seen below

```
persons.sort(
    Comparator.comparing(Person::getLastName)
        .thenComparing(Person::getFirstName)
        .thenComparing(
            Person::getEmailAddress,
            Comparator.nullsLast(CASE_INSENSITIVE_ORDER)));
```

That's the magic of Java 8, its compact with Lambda expressions and functional interfaces.

Q 114. How would you convert time from One Time Zone to another in Java?

java.util.Date class is not TimeZone aware, as it does not store any time zone specific information in its object. This is clearly mentioned in the Java Docs for Date Class -

> In Date, A milliseconds value represents the number of milliseconds that have passed since January 1, 1970 00:00:00.000 GMT.

The internal representation of the time inside Date object remains same for a given time, when we print the date object using System.out.println(date) method then date.toString() method is invoked which prints the date in local TimeZone of the JVM.

Custom TimeZone formatting can be achieved using SimpleDateFormat class.

```
Calendar instance = Calendar.getInstance();
Date date = instance.getTime();
DateFormat formatter = new SimpleDateFormat("MM/dd/yyyy hh:mm:ss Z");
formatter.setTimeZone(TimeZone.getTimeZone("Europe/London"));
System.out.println(formatter.format(date));
formatter.setTimeZone(TimeZone.getTimeZone("Asia/Calcutta"));
System.out.println(formatter.format(date));
```

Thus a given time in milliseconds can be represented in different TimeZone using different TimeZone specific Date formatters.

Notes

Always prefer to user Calendar API over Date due to various benefits of Calendar Class - Calendar handles TimeZone information and it correctly measures the duration of a year in milliseconds keeping into account the leap years.

Question: What will be output of the following Java Program?

```
Calendar instance = Calendar.getInstance(TimeZone.getTimeZone("Asia/Calcutta"));
Date date = instance.getTime();
System.out.println("date = " + date);

instance.setTimeZone(TimeZone.getTimeZone("GMT"));
Date date2 = instance.getTime();
System.out.println("date2 = " + date2);
```

Answer

Both the System.out will print the same date value because Date class object is always printed in local TimeZone and changing the TimeZone on Calendar class does not alter the underlying milliseconds value from the epoch time (Since January 1, 1970 00:00:00.000 GMT).

Question: How will you write a method to add weekdays to a given date?

Answer - The following method can add given weekdays to a given Date.

```
public Date addBusinessDays(Date date, int numberOfDays) {
   int count = 0;
   Calendar calendar = Calendar.getInstance();
   calendar.setTime(date);
   while (count < numberOfDays) {
      calendar.add(Calendar.DAY_OF_YEAR, 1);
      if (calendar.get(Calendar.DAY_OF_WEEK) == Calendar.SUNDAY || calendar.get(Calendar.DAY_OF_WEEK) ==
Calendar.SATURDAY)
```

```
            count++;
        }
        return calendar.getTime();
    }
```

Q 115. Will WeakHashMap's entry be collected if the value contains the only strong reference to the key?

```
public class Key {};
public class Value{
      final public Key key;
      public Value (Key key){
      this.key = key;
      }
}
```

It will not be collected.

Let's understand why?

The value object in the WeakHashMap are held by ordinary strong references. Thus care must be taken to ensure that Value objects do not strongly refer to their own keys, either directly or indirectly, since that will prevent the keys from being garbage collected. If it is strongly required to store the key's reference in Value object then we can assign a Weak reference of Key in value object, as shown follow.

```
final public WeakReference<Key> key;
public Value(Key key){
      this.key = new WeakReference<Key>(key);
}
```

Q 116. Why HashMap's initial capacity must be power of two?

HashMap's initial capacity must be power of 2, and its default value is set to 16.
The reason is that, hashmap utilizes binary shift operation for calculating the modulus (%) of x/n for optimization reasons. But the bitwise hack expects the initial capacity to be power of two.

Even if we specify a custom initial capacity, it will round it to the nearest power of two using the following code.

```
int capacity = 1;
while(capacity < intialCapacity){
      capacity = capacity << 1;
}
```

Q 117. Can we traverse the list and remove its elements in the same iteration loop?

Yes, that is feasible, provided

1. No other thread is modifying the collection at that traversal time (it should be single threaded model)
2. Iterator is used to traverse and to remove the elements from within that loop

Here is the perfect working example,

```
public class RemoveViaIterator {
    private List<String> names = new ArrayList<>(asList("1st", "2nd", "3rd", "4th"));
```

```
    public void remove(){
        Iterator<String> iterator = names.iterator();
        while (iterator.hasNext()) {
            Object next = iterator.next();
            System.out.println("next = " + next);
            iterator.remove();
        }
        System.out.println(names.size());
    }

    public static void main(String[] args) {
        RemoveViaIterator test = new RemoveViaIterator();
        test.remove();
    }
}
```

Q 118. Do I need to override object's equals() and hashcode() method for its use in a TreeMap?

TreeMap does not use hashcode() method

Only hashing data structures require hashcode() method for retrieving data in Big O(1) time complexity. TreeMap, on the other hand uses Comparator/Comparable for sorting/ranking its elements, equals() is only used to search an element inside the collection using contains(object o). Its always a good practice to keep equals() method in sync with the Comparator to have consistency in your code.
HashMap, hashtable, ConcurrentHashMap, LinkedHashMap are few of the hashing data structures that are dependent on both hashcode() and equals() method.

Q 119. Implement a thread-safe BlockingQueue using intrinsic locking mechanism.

Here is the rough implementation of Blocking Queue using Java Intrisic Locking aka synchronized keyword.

```
import java.util.LinkedList;
import java.util.Queue;
public class BlockingQueue<T> {
    private Queue<T> queue = new LinkedList<T>();
    private int capacity;

    public BlockingQueue(int capacity) {
        this.capacity = capacity;
    }

    public synchronized void put(T element) throws InterruptedException {
        while (queue.size() == capacity) {
            wait();
        }
        queue.add(element);
        notify();
    }

    public synchronized T take() throws InterruptedException {
        while (queue.isEmpty()) {
            wait();
        }
```

```
            T item = queue.remove();
            notify();
            return item;
        }
    }
```

Q 120. Is there a way to acquire a single lock over ConcurrentHashMap instance?

Acquiring a single lock over entire ConcurrentHashMap will defeat the original purpose of using it, because CHM intentionally uses multiple internal locks to make it thread-safe and achieve better throughput in multi-threaded environment. CHM's dynamic re-sizing recursively acquires lock over all of its buckets in order to make them bigger, but that's of no concern to the API user.

If we really want a single lock for the entire collection, then we should really prefer synchronized hashmap instead of ConcurrentHashMap.

Q 121. How will you implement a Blocking Queue using Lock and Condition Interface provided in JDK?

Lock is analogical to synchronized keyword and Condition is similar to wait/notify. Here is the implementation for BlockingQueue that uses Lock and Condition.

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class BlockingQueue<T> {
    private Queue<T> queue = new LinkedList<T>();
    private int capacity;
    private Lock lock = new ReentrantLock();
    private Condition notFull = lock.newCondition();
    private Condition notEmpty = lock.newCondition();

    public BlockingQueue(int capacity) {
        this.capacity = capacity;
    }

    public void put(T element) throws InterruptedException {
        lock.lock();
        try {
            while (queue.size() == capacity) {
                notFull.await();
            }
            queue.add(element);
            notEmpty.signal();
        } finally {
            lock.unlock();
        }
    }

    public T take() throws InterruptedException {
        lock.lock();
```

```
        try {
            while (queue.isEmpty()) {
                notEmpty.await();
            }
            T item = queue.remove();
            notFull.signal();
            return item;
        } finally {
            lock.unlock();
        }
    }
}
```

Please note that the thread contention for this class will be slightly less compared to similar implementation using synchronized keyword, because here we maintain two different Condition Queues instead of just one.

Q 122. How would you cancel a method execution after time-out expires using Java Future?

This can be easily achieved using Future utility class provided by concurrent package in JDK 1.5

Future represents the life cycle of a task and calling cancel() on future attempts to cancel the task execution, if not already in completed state. Thread can even be interrupted if we pass true as a parameter to cancel() method. And this interrupt can be checked using Thread.interrupted() method inside the running task.

JAVA Source

```
import java.util.concurrent.*;

public class TimedExecution {
    private ExecutorService executorService = Executors.newCachedThreadPool();

    public void timedRun(Runnable runnable, long timeout, TimeUnit unit) throws InterruptedException,
ExecutionException {
        Future<?> task = executorService.submit(runnable);
        try {
            task.get(timeout, unit);
        } catch (TimeoutException e) {
            System.err.println("Timeout occurred.");
        } finally {
            task.cancel(true);
        }
    }

    public void stop() {
        executorService.shutdown();
    }

    public static void main(String[] args) throws ExecutionException, InterruptedException {
        TimedExecution timedExecution = new TimedExecution();

        timedExecution.timedRun(new Runnable() {
            @Override
            public void run() {
                while (!Thread.interrupted()) {
                    System.out.println("Test me..");
```

```
                }
            }
        }, 100, TimeUnit.MICROSECONDS);
        timedExecution.stop();
    }
}
```

This task will be cancelled after 100 microseconds. Do not forget to call task.cancel(true), otherwise the thread will continue executing the task in background.

Q 123. Java already had Future interface, then why did they provide Completable Future class in Java 8?

There is an inherent problem with the existing Future class in Java 1.5, lets see what's wrong with it !
The java.util.concurrent library provides a Future<T> interface to denote a value of type T that will be available at some point in the future.

Let's say our requirement is to chain below two asynchronous operations in single method
1. Read the web page
2. Get links from the web page

We will analyze if it is feasible to chain these two operations using existing Future available in Java.

Let's assume there are two methods corresponding to above two operations

```
public void Future<String> readPage(URL url) {...definition...}
public static List<URL> getLinks(String page) {...definition...}
```

How can we chain the above two method to get the links from web page? Unfortunately, there is only one way. First, call the get method on the future to get its value when it becomes available. Then, process the result:

```
public void getLinks(String url) {
    Future<String> contents = readPage(url);
    String page = contents.get();
    List<URL> links = Parser.getLinks(page);
}
```

But the call to content.get() is a blocking call. We are really no better off than with a method public String readPage(URL url) that blocks until the result is available. There was no easy way of saying: “When the result becomes available, here is how to process it.” This is the crucial feature that the new CompletableFuture<T> class provides.

```
CompletableFuture<Void> links  = CompletableFuture.supplyAsync(() -> blockingReadPage(url))
        .thenApply(Parser::getLinks)
        .thenAccept(System.out::println);
```

Unlike a plain Future, a CompleteableFuture has a method thenApply to which you can pass the post-processing function.

Q 124. What is difference between intrinsic synchronization and explicit locking using Lock?

JVM provides intrinsic synchronization through monitor locks. Each object in Java owns a monitor on which the threads can be synchronized. JDK 1.5 introduced concept of explicit synchronization using Lock[1] and Condition classes which offers advanced features over intrinsic synchronization.

```
public interface Lock {
    void lock();
    void lockInterruptibly() throws InterruptedException;
    boolean tryLock();
    boolean tryLock(long time, TimeUnit unit) throws InterruptedException;
    void unlock();
    Condition newCondition();
}
```

Usage semantics for Lock

```
Lock lock = ...;
    if (lock.tryLock()) {
        try {
            // manipulate protected state
        } finally {
            lock.unlock();
        }
    } else {
        // perform alternative actions
    }
```

Intrinsic synchronization	Explicit Locking using Lock and Condition
• Its easy to use technique with code more readable and compact • It is not possible to interrupt a thread waiting to acquire a lock, or attempt to acquire a lock without being willing to wait for it forever. • Responsibility of releasing a lock is handled by JVM even in case a exception occurs • JVM can do some performance optimization if synchronized keyword is used like lock elision & lock coarsening • As per Java Documentation "*The use of synchronized methods or statements provides access to the implicit monitor lock associated with every object, but forces all lock acquisition and release to occur in a block-structured way: when multiple locks are acquired they must be released in the opposite order, and all locks must be released in the same lexical scope in which they were acquired.*"	• Provides same mutual exclusion & memory visibility guarantee as the synchronized block • Provides option for timed, polled or Interruptible locks helping avoid probabilistic deadlock. lock.tryLock() used for polling tryLock(long time, TimeUnit unit) for timed locking lockInterruptibly() for interruptible locking Interruptible lock acquisition allows locking to be used within cancelable activities. • Offers choice of fairness to the lock acquisition when multiple threads try to acquire the shared lock setting fairness flag to true as shown below. Lock lock = new ReentrantLock(true); This has significant performance cost when sued. • Ability to implement non-block-structured locking. Lock doesn't have to be released in the same block of code, unlike synchronized locks. • Lock has to be released manually in a finally block once we have modified the protected state

1 http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/Lock.html

Q 125. What are Stamped Locks? How they are useful in optimistic scenario where thread contention is rare?

StampedLock Class has been introduced in Java 8 for Optimistic Locking scenario, it provides three modes for controlling read/write access. The state of a StampedLock consists of a version and mode.

1. writeLock() possibly blocks waiting for exclusive access, returning a stamp that can be used in method unlockWrite(long) to release the lock.
2. readLock() possibly blocks waiting for non-exclusive access, returning a stamp that can be used in method unlockRead(long) to release the lock.
3. tryOptimisticRead() returns a non-zero stamp only if the lock is not currently held in write mode. Method validate(long) returns true if the lock has not been acquired in write mode since obtaining a given stamp. This mode can be thought of as an extremely weak version of a read-lock, that can be broken by a writer at any time. The use of optimistic mode for short read-only code segments often reduces contention and improves throughput.

Example Usage

We should use StampedLock for writing thread-safe classes that have minimum thread contention, as StampedLock highly relies on the Optimistic Locking and thus is very fragile to concurrent updates.

- Idiom for using StampedLock in our Classes

```
public void testStampedLock() {
   long stamp = lock.tryOptimisticRead(); // non blocking path
   doWork(); // we're hoping no writing will go on in the meanwhile
   if (lock.validate(stamp)) {
      //success! no contention with a writer thread
   } else {
      //another thread must have acquired a write lock in the meanwhile, changing the stamp.
      //let's downgrade to a heavier read lock / write lock
      stamp = lock.readLock(); //this is a traditional blocking read lock
      try {
         doWork();   //no writing can happen now from any other thread
      } finally {
         lock.unlock(stamp); // release using the correlating stamp
      }
   }
}
```

- We can improve the Java Vector Class's get method by using lighter weight optimistic lock

```
public class Vector {
   private int size;
   private Object[] elements;
   private StampedLock lock = new StampedLock();

   public Object get(int n) {
      long stamp = lock.tryOptimisticRead();
      Object[] currentElements = elements;
      int currentSize = size;
      if (!lock.validate(stamp)) { // Someone else had a write lock
         stamp = lock.readLock(); // Get a pessimistic lock
```

```
            currentElements = elements;
            currentSize = size;
            lock.unlockRead(stamp);
        }
        return n < currentSize ? currentElements[n] : null;
    }
}
```

- Implementing Bank Account with StampedLock (bankAccounts have minimum thread contention, as normally single person uses the account at a given time)

```
import java.util.concurrent.locks.StampedLock;

public class BankAccountStampedLock {
    private final StampedLock sl = new StampedLock();
    private long balance;

    public BankAccountStampedLock(long balance) {
        this.balance = balance;
    }

    public void deposit(long amount) {
        long stamp = sl.writeLock();
        try {
            balance += amount;
        } finally {
            sl.unlockWrite(stamp);
        }
    }
    public void withdraw(long amount) {
        long stamp = sl.writeLock();
        try {
            balance -= amount;
        } finally {
            sl.unlockWrite(stamp);
        }
    }
    public long getBalance() {
        long stamp = sl.readLock();
        try {
            return balance;
        } finally {
            sl.unlockRead(stamp);
        }
    }

    public long getBalanceOptimisticRead() {
        long stamp = sl.tryOptimisticRead();
        long balance = this.balance;
        if (!sl.validate(stamp)) {
            stamp = sl.readLock();
            try {
                balance = this.balance;
            } finally {
                sl.unlockRead(stamp);
            }
```

```
        }
        return balance;
    }
}
```

Q 126. How will you find out first non-repeating character from a string? For example, String input = "aaabbbeggh", answer should be 'e'

We can find the first non-repeating char from a string using the following algorithm in Big O(n) Time

First Pass

We can maintain a counting array for all possible alphabet values (ASCII code 0-128) and keep on counting the position of array based on Ascii value of character. This will be Big O(n) Time Complexity Task where n is number of letters in the string.

Second Pass

Iterate through the counting array and find the first index position where value is exactly 1, and then break. That will give us the ascii code of character that is non repeating.

Java Code

```
String str = "zzzzzbbbccccddehhhhiii";
int[] countingArray = new int[128];
str.chars().forEach(value -> countingArray[value]++);
int nonRepeatingCharAsInt = 0;
for (int i = 0; i < countingArray.length; i++) {
    if (countingArray[i] == 1) {
        nonRepeatingCharAsInt = i;
        break;
    }
}
System.out.println("character = " + Character.valueOf((char) nonRepeatingCharAsInt));
```

There is a second method using hashmap to store the count, using Java 8

```
import java.util.LinkedHashMap;
import java.util.function.Consumer;
import java.util.stream.Collectors;
import static java.util.function.Function.identity;

public class NonRepeatingLetter {
    public static void main(String[] args) {
        findFirstNonRepeatingLetter(args[0], System.out::println);
    }
    private static void findFirstNonRepeatingLetter(String s, Consumer<Character> callback) {
        s.chars()
                .mapToObj(i -> Character.valueOf((char) i))
                .collect(Collectors.groupingBy(identity(), LinkedHashMap::new, Collectors.counting()))
                .entrySet().stream()
                .filter(entry -> entry.getValue() == 1L)
                .map(entry -> entry.getKey())
                .findFirst().map(c -> {
            callback.accept(c);
            return c;
        });
    }
}
```

Q 127. What is difference between Callable and Runnable Interface?

As per Java documentation :

"*Callable interface is similar to Runnable, in that both are designed for classes whose instances are potentially executed by another thread. A Runnable, however, does not return a result and cannot throw a checked exception.*"

```
public interface Callable<V> {
        V call() throws Exception;
}
```

In order to convert Runnable to Callable use the following utility method provided by Executors class

```
Callable callable = Executors.callable(Runnable task);
```

Callable, however must be executed using a ExecutorService instead of Thread as shown below.

```
result = exec.submit(aCallable).get();
```

Submitting a callable to ExecutorService returns Future Object which represents the lifecycle of a task and provides methods to check if the task has been completed or cancelled, retrieve the results and cancel the task.

Future Interface looks like this -

```
    public interface Future<V> {
      boolean cancel(boolean mayInterruptIfRunning);
      boolean isCancelled();
      boolean isDone();
      V get() throws InterruptedException, ExecutionException;
      V get(long timeout, TimeUnit unit)
          throws InterruptedException, ExecutionException, TimeoutException;
    }
```

So eventually, we can cancel a future, get its status or synchronously wait for its completion by invoking blocking get() method.

Q 128. What will happen when an exception occurs from within a synchronized code block? Will lock be retained or released?

When an exception occurs from within a synchronized code block, then JVM smartly releases all the locks acquired by the current thread and will start unwinding the execution stack, till the exception is handled using catch block, otherwise killing the thread.

But the same does not happen when we write explicit locking code using Lock interface. In that case we need to release the lock manually in the finally block.

Q 129. What is difference between sleep(), yield() and wait() method?

Object.wait() and Thread.sleep() are entirely two different method used in two different contexts. Here are few differences between these two methods.

1. wait() method releases the acquired lock when the thread is waiting till someone calls notify() while Thread.sleep() method keeps the lock even if thread is waiting.

```
synchronized(monitor) {
   Thread.sleep(1000);          // LOCK is held by the current thread
}
synchronized(monitor) {
   monitor.wait();              // LOCK is released by current thread
}
```

2. wait() can only be called from synchronized context otherwise it will throw IllegalMonitorStateException, while sleep can be called from any code block.
3. wait() is called on an Object while sleep is called on a Thread
4. waiting thread can be awaken by calling notify()/notifyAll() methods while sleeping thread can't be awaken[1] (though can be interrupted)
5. Incase of sleep() Thread immediately goes to Runnable state after waking up while in case of wait(), waiting thread first fights back for the lock and then go to Runnable state.
6. Major difference between yield and sleep in Java is that yield() method pauses the currently executing thread temporarily for giving a chance to the remaining waiting threads of the same priority to execute. If there is no waiting thread or all the waiting threads have a lower priority then the same thread will continue its execution.

In Layman's Terms

sleep(n) - Thread is done with its time slot, and please don't give it another one for at least n milliseconds. The OS doesn't even try to schedule the sleeping thread until requested time has passed.

yield() - Thread is done with its time slot, but it still has work to do. The OS is free to immediately give the thread another time slot, or to give some other thread or process the CPU the yielding thread just gave up.

wait() - Thread is done with its time slot, Don't give it another time slot until someone calls notify(). As with sleep(), the OS won't even try to schedule your task unless someone calls notify() or one of a few other wakeup scenarios occurs (spurious wakeup).

Q 130. What is difference between StringBuilder and StringBuffer?

StringBuffer is heavy weight thread-safe Class while StringBuilder is light weight non-thread-safe class, otherwise their functionality is similar. One should choose StringBuilder if thread-safety is not a concern for that particular piece of code.

1 Thread.interrupt() would cause InterruptedException on both sleep() and wait() methods.

Q 131. How does Generics work in Java?

Why to choose Generics?

- Generics add stronger compile time type safety to our code, thus reducing the number of production bugs.
- It tries to eliminate explicit type casting to a greater extent, making our code more readable.
- Generics enable types (classes and interfaces) to be parameters when defining classes, interfaces and methods, allowing us to implement generic algorithms which works for different types.

Example of Generic class

class name<T1, T2, ..., Tn> { /* ... */ }

Naming Convention

E - element
K - key
N - number
T - type
V - value
S,U,V etc - 2nd, 3rd & 4th type

Can we add a Double value to List<Number>?

Because an Integer is a kind of Number, so this is perfectly allowed due to inheritance.

```
Box<Number> box = new Box<Number>();
box.add(new Integer(10));   // OK
box.add(new Double(10.1));  // OK
```

Why List<String> can not be assigned to List<Object>?

Let's consider the following example,

```
List<String> stringList =null ;
List<Object> objectList = stringList;   //ERROR
```

Here List<String> is not a sub type of List<Object>, hence as per inheritance rule, this is not allowed.

Inheritance in Generics - Can we assign List<String> to Collection<String>?

If the type parameter is same, then we can very well extend the classes as per Java Inheritance rules. So it is perfectly legal to assign List<String> to Collection<String> reference.

Can we do T a = new T[100]? Why not?

No, this is not possible, because Arrays in Java contains Type information, at runtime, about its component type. So we must know the component type when we create an array. Because Type information is not retained at runtime using generics, thus we can't create an array like this. Please refer to Type Erasure in Generics Java for more information.

What is Type Erasure?

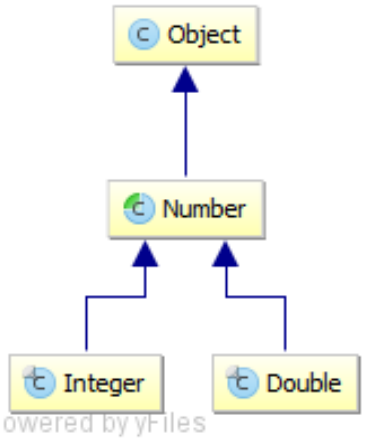
Generics provide compile time safety to our Java code. Type erasure happens at compile time, to remove those generic type information from source and adds casts needed and deliver the byte code. Thus the java byte code will be no different than that the non-generic Java code.

Q 132. What are Upper and Lower bounds in Generics? Where to choose one?

Upper and Lower bounded wildcard are used in Generics to relax the restriction on a variable.

Upper Bounded Wildcards[1]

Upper bounded wildcard restricts the unknown type to be a specific type or subtype of that type. For example, If we want to write a method that accepts List<Number> and its subtypes i.e. List<Double> and List<Integer>, etc then we can use Upper bounded wildcard. Below is the sample signature of upper bounded wildcard method.




```
public static void process(List<? extends Number> list) { /* ... */ }
```

Lower Bounded Wildcards

Lower bounded wildcard restricts the unknown type to be a specific type or super type of that type. Say you want to write a method that puts Integer objects into a list. To maximize flexibility, you may like the method to work on List<Integer>, List<Number>, but not List<Double> - anything that can hold Integer values. The Syntax in that case would be -

```
public static void addNumbers(List<? super Integer> list) {/*.....*/}
```

When to choose what?

Lets first define In and Out terminology
In variable
An "in" variable serves up data to the code. Imagine a copy method with two arguments: copy(src, dest). The src argument provides the data to be copied, so it is the "in" parameter.
Out Variable
An "out" variable holds data for use elsewhere. In the copy example, copy(src, dest), the dest argument accepts data, so it is the "out" parameter.

Wildcard Guidelines

•An "in" variable is defined with an upper bounded wildcard, using the extends keyword.
•An "out" variable is defined with a lower bounded wildcard, using the super keyword.
•In the case where the "in" variable can be accessed using methods defined in the Object class, use an unbounded wildcard.
•In the case where the code needs to access the variable as both an "in" and an "out" variable, do not use a wildcard.

Multiple Bounds

A type parameter can have multiple bounds as described below.
<T extends B1 & B2 & B3>

1 http://docs.oracle.com/javase/tutorial/java/generics/wildcardGuidelines.html

Q 133. Discuss memory visibility of final fields in multi-threading scenario.

As per Java Lang Specifications for final fields[1]

"Fields declared final are initialized once, but never changed under normal circumstances. The detailed semantics of final fields are somewhat different from those of normal fields. In particular, compilers have a great deal of freedom to move reads of final fields across synchronization barriers and calls to arbitrary or unknown methods. Correspondingly, compilers are allowed to keep the value of a final field cached in a register and not reload it from memory in situations where a non-final field would have to be reloaded.

final fields also allow programmers to implement thread-safe immutable objects without synchronization. A thread-safe immutable object is seen as immutable by all threads, even if a data race is used to pass references to the immutable object between threads. This can provide safety guarantees against misuse of an immutable class by incorrect or malicious code. final fields must be used correctly to provide a guarantee of immutability.

An object is considered to be completely initialized when its constructor finishes. A thread that can only see a reference to an object after that object has been completely initialized is guaranteed to see the correctly initialized values for that object's final fields.

The usage model for final fields is a simple one: Set the final fields for an object in that object's constructor; and do not write a reference to the object being constructed in a place where another thread can see it before the object's constructor is finished. If this is followed, then when the object is seen by another thread, that thread will always see the correctly constructed version of that object's final fields. It will also see versions of any object or array referenced by those final fields that are at least as up-to-date as the final fields are."

Briefly speaking

Memory visibility of final fields are guaranteed by the Java Memory Model and hence are thread safe in multi-threaded scenario. Thus we prefer to use Immutable objects in the concurrent application to avoid any memory visibility related problems.

Notes

Question

We have a legacy application in which a thread reads a config file and starts creating beans. It does so by first creating the object and then setting the required properties on them. All this happens without any synchronization because everything is happening serially. Once the objects are created, other threads pick those objects and start processing. But somehow, we got the problem that one of the client thread is not seeing the Correct Value for a bean.

Analysis

This could be a typical memory visibility issue in a multi-threaded environment. A properly constructed bean would have to make its fields final in absence of synchronization, otherwise other threads may see the default values for its fields. Thus moving the bean initialization code (setters) into constructor and making the fields final should solve this problem. Otherwise we might need to synchronize the access to this particular bean - read articles about proper publishing of an object in multi-threaded environment.[2]

A Properly Constructed Object in Multi-threaded scenario[3]

1 http://docs.oracle.com/javase/specs/jls/se7/html/jls-17.html#jls-17.7
2 http://www.ibm.com/developerworks/java/library/j-jtp0618/index.html
3 http://www.cs.umd.edu/~pugh/java/memoryModel/jsr-133-faq.html

continued on 126

The values of final fields are populated inside an object's constructor. And if the object is constructed properly (this reference does not escape during construction, don't start thread from within constructor, fields are final), then the values assigned to the final fields in the constructor are guaranteed to be visible to all other threads without the need of any synchronization. In addition, the visible values for any other object or array referenced by those final fields will be at least as up-to-date as the final fields. Let's understand the following example,

```
import java.util.ArrayList;

class FinalFieldExample {
   final int x;
   final ArrayList<String> names;
   int y;
   static FinalFieldExample f;

   public FinalFieldExample() {
      x = 3;
      y = 4;
      names = new ArrayList<>();
      names.add("First");
      names.add("Second");
   }

   static void writer() {
      f = new FinalFieldExample();
      f.names.add("third");
   }

   static void reader() {
      if (f != null) {
         int i = f.x;
         int j = f.y;
         ArrayList<String> myNames = f.names;
      }
   }
}
```

In this example, suppose Thread A calls FinalFieldExample.write() method and thus creates the object. Thread B on other hand calls FinalFieldExample.reader() method and thus access's the object. Further suppose that Thread B calls reader() once the writer() is finished creating the object. As per new Java Memory Model (JSR 133),

- Thread B executing the reader is guaranteed to see the value of 3 for field f.x - its a final primitive variable
- Thread B is guaranteed to see the value of "First" and "Second" for field f.names because names array is referenced by a final field, and the value assigned to such object inside the constructor boundary will be visible to other threads.
- There is not guarantee that Thread B will see "third" inside array f.names
- There is not guarantee that Thread B will see value of 4 for field f.y

Q 134. Where would you use LinkedHashSet provided by Java Collections?

LinkedHashSet maintains the order of its original elements over and above the the uniqueness of its elements. Its more efficient when we iterate over its elements because it maintains the elements in a linked list. This is not possible with the HashSet because the in order to iterate over its elements, every bucket in a hash table must be visited whether it is occupied or not.

So we would choose it over HashSet when
1. We require the collection to preserve the original order of its elements.
2. We require better performance for iteration, next operation is performed in constant time O(1).

Q 135. What do you think is the reason for String Class to be Immutable?

String class in Java is implemented as Immutable and there are various reasons for doing that,

- Immutability brings inherent thread-safety to the usage of String class in a concurrent program. So multiple threads can work on the same String object without any data corruption. There is absolutely no need to synchronize the code because of String objects.
- StringPooling is possible only because of immutability because the underlying contents of the String will never change. This helps reducing the overall memory footprint of the application using lots of String objects.
- Hash code for Immutable objects are calculated lazily on first usage and then cached for future reference. This gives the benefit of performance when we use Immutable Key's in any hashing data structure.

Q 136. How is String concatenation implemented in Java using + operator? for example, String name = "hello" + "world"

String concatenation is implemented internally through the StringBuilder(as of JDK 1.5) class and its append method. As of Java 5, Java will automatically convert the following string concatenation

```
String h = "hello" + "world";
```

to either

```
String h ="helloworld";
```

or into

```
String i = new StringBuilder().append("hello").append("world").toString();
```

Thus no temporary objects will be created for this type of concatenation. This JVM optimization may improved further in upcoming releases.

Does that mean, we should never prefer StringBuilder over String?

No, we never meant that. StringBuilder has its own importance, but in slightly different scenario. Like the code we discussed above, if the same thing is happening inside a loop then O(n) StringBuilder objects will be created (one per iteration), causing overall time complexity Big O(n^2) where n is the number of strings that we are concatenating. So in that case, for performance reasons, we should create a single StringBuilder Object outside the loop and then append the data to that StringBuilder Object. That's the reason, experts advice to use StringBuilder inside a loop. Using StringBuilder, the code should look like this :

```
StringBuilder result = new StringBuilder(10000);
```

continued on 128

```
for(int i=0; i<=1000;i++){
    result.append("Hello"+i);
}
return result.toString();
```

Time Complexity : Big O (n) where n is the number of strings, 1000 in this case.

Q 137. Which data type would you choose for storing sensitive information, like passwords, and Why?

Normally, a character array should used for storing passwords. Here is the reason for choosing char array over String -

- There is no way to clear a String Object from the memory, its up to GC to collect it.
- String objects are immutable and stored in a String Pool (may reside inside a PermGen space) which may not at all be gc'd.
- Any person taking the heap dump can easily see the String literals.
- In case of an char array, we can always nullify it once we are done with the information, so not much dependency on the GC, thus we are narrowing the time window for the life of sensitive data.

Q 138. What is difference between using Serializable & Externalizable Interfaces in Java?

- Externalizable Interface extends the java.io.Serializable adding two methods - writeExternal & readExternal.
- In case of Serializable, default serialization is used, while in case of Externalizable, the complete serialization control goes to the application. Stating that means, we can not benefit from the default serialization process when we choose Externalizable interface.
- We generally choose Externalizable when we want to save the output in our custom format which is other than Java default serialization format like, csv, database, flat file, xml, etc
- readExternel() and writeExternal() methods are used to handle the serialization in case of Externilizabel interface.
- In case of externalizable interface, we need to handle super type state, default values in transient variable and static variables.
- Incase of Serialization, object is reconstructed using data read from ObjectInputStream but incase of Externalizable, public no-arg constructor is used to reconstruct the object.

Q 139. How would you design Money Class in Java?

Money is composed of two fundamental entities - Amount and Currency.
BigDecimal is ideal data type provided in Java for representing monetary values, and Java also provides Currency Class implementation.

Below is the sample code for writing Money Class

```
public class Money implements Comparable<Money>{
   private static final Currency INR = Currency.getInstance(new Locale("en", "in"));
   private static final NumberFormat format = NumberFormat.getCurrencyInstance(new Locale("en", "in"));
   private final BigDecimal amount;
   private final Currency currency;
```

```
    public static Money rupees(BigDecimal amount) {
        return new Money(amount, INR);
    }
    Money(BigDecimal amount, Currency currency) {
        this.amount = amount;
        this.currency = currency;
    }
    public BigDecimal getAmount() {
        return amount;
    }

    public Currency getCurrency() {
        return currency;
    }
    public boolean isSameCurrencyAs(Money aThat){
        boolean result = false;
        if ( aThat != null ) {
            result = this.currency.equals(aThat.currency);
        }
        return result;
    }
    private void checkCurrenciesMatch(Money aThat){
        if (! isSameCurrencyAs(aThat)) {
            throw new RuntimeException(aThat.getCurrency() + " doesn't match the expected currency : " + currency);
        }
    }
    public Money minus(Money aThat){
        checkCurrenciesMatch(aThat);
        return new Money(amount.subtract(aThat.amount), currency);
    }
    @Override
    public String toString() {
        return format.format(amount);
    }
    public String toString(Locale locale) {
        return getCurrency().getSymbol(locale) + " " + getAmount();
    }
    @Override
    public int compareTo(Money o) {
        return o.getAmount().compareTo(amount);
    }

    public static void main(String[] args) {
        Money rupees = Money.rupees(new BigDecimal("100"));
        System.out.println("rupees = " + rupees);
    }
}
```

Notes

The code shown above is not thread safe, ideally NumberFormat should be created local to a thread using ThreadLocal class, instead making it a static field of class.

continued on 130

Q 140. What is difference between HashMap, TreeMap and LinkedHashMap?

All three classes (HashMap, TreeMap and LinkedHashMap) implements Map interface, and therefore represents mapping from unique key to values. But there are different usage scenarios for each of them -

1. HashMap is a hashing data structure which works on hashcode of keys. Keys must provide consistent implementation of equals() and hashCode() method in order to work with hashmap. Time complexity for get() and put() operations is Big O(1).
2. LinkedHashMap is also a hashing data structure similar to HashMap, but it retains the original order of insertion for its elements using a LinkedList. Thus iteration order of its elements is same as the insertion order for LinkedHashMap which is not the case for other two Map classes. Iterating over its elements is lightly faster than the HashMap because it does not need to traverse over buckets which are empty. LinkedHashMap also provides a great starting point for creating a LRU Cache object by overriding the removeEldestEntry() method, as shown in the following code snippet.
3. TreeMap is a SortedMap, based on Red-Black Binary Search Tree which maintains order of its elements based on given comparator or comparable. Time complexity for put() and get() operation is O (log n).

Property	HashMap	LinkedHashMap	TreeMap
Time Complexity (Big O notation) Get, Put, Remove	O(1)	O(1)	O(log n)
Iteration Order	Random	Sorted according to either Insertion Order or Access Order (as specified during construction)	Sorted according to either natural order of keys or comparator (as specified during construction)
Null Keys	allowed	allowed	Not allowed if Key uses Natural Ordering or Comparator does not support comparison on null Keys
Interface	Map	Map	Map, SortedMap and NavigableMap
Synchronization	none, use Collections.synchronizedMap()	None, use Collections.synchronizedMap()	none, use Collections.synchronizedMap()
Data Structure	List of Buckets, if more than 8 entries in bucket then Java 8 will switch to balanced tree from linked list	Doubly Linked List of Buckets	Red-Black Tree (a kind of self-balancing binary search tree) implementation of Binary Tree. It offers O (log n) for Insert, Delete and Search operations and O (n) Space Complexity
Applications	General Purpose, fast retrieval, non-synchronized. ConcurrentHashMap can be used where concurrency is involved.	Can be used for LRU cache, other places where insertion or access order matters	Algorithms where Sorted or Navigable features are required. For example, find among the list of employees whose salary is next to given employee, Range Search, etc.
Requirements for Keys	Equals() and hashCode() needs to be overwritten	Equals() and hashCode() needs to be overwritten	Comparator needs to be supplied for Key implementation, otherwise natural order will be used to sort the keys

Q 141. How would you write high performing IO code in Java? Can you write a sample code for calculating checksum of a file in time efficient manner?

Intent of the interviewer is to know if you are familiar with Java's High Performance IO Channels.

Few of the times we wish the speed of C for doing some IO intensive task in our Java program. Calculation of CRC is one of the task which requires an efficient IO implementation in order to give good performance which is very close to the one we see in a similar C program (though not equivalent)

An InputStream in Java can be easily converted into an FileChannel using its getChannel() method. Let's understand how to use channels using the following Checksum Calculation Program.

```
public static long calculateCRC(File filename) {
    final int SIZE = 16 * 1024;
    try (FileInputStream in = new FileInputStream(filename);) {
        CRC32 crc = new CRC32();
        FileChannel channel = in.getChannel();
        int length = (int) channel.size();
        MappedByteBuffer mb = channel.map(FileChannel.MapMode.READ_ONLY, 0, length);
        byte[] bytes = new byte[SIZE];
        int nGet;
        while (mb.hasRemaining()) {
            nGet = Math.min(mb.remaining(), SIZE);
            mb.get(bytes, 0, nGet);
            crc.update(bytes, 0, nGet);
        }
        return crc.getValue();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    throw new RuntimeException("unknown IO error occurred ");
}
```

If the input file is very large > 1 GB, then its better to calculate the CRC in iterations -

```
public static Long calculateCRC(File file) {
    long t1 = System.currentTimeMillis();
    long crcValue;
    final int SIZE = 16 * 1024;
    final int SIZE2 = 16 * 1024 * 1024 * 100;
    try (FileInputStream in = new FileInputStream(file)) {
        CRC32 crc = new CRC32();
        FileChannel channel = in.getChannel();
        long length = channel.size();
        long iterations = length / SIZE2;
        long reverseLength = 0l;
        for (int i = 0; i <= iterations; i++) {
            MappedByteBuffer mb = channel.map(FileChannel.MapMode.READ_ONLY, reverseLength, i == iterations?
length % SIZE2 : SIZE2);
            reverseLength += SIZE2;
            byte[] bytes = new byte[SIZE];
            int nGet = 0;
            while (mb.hasRemaining()) {
                nGet = Math.min(mb.remaining(), SIZE);
```

```
                mb.get(bytes, 0, nGet);
                crc.update(bytes, 0, nGet);
            }
        }
        crcValue = crc.getValue();
        long t2 = System.currentTimeMillis();
        System.out.println("Time for CRC = "+ (t2-t1));
        return crcValue;
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    throw new RuntimeException("unknown IO error occurred ");
}
```

Java's FileChannel provide much better performance compared to the InputStream, BufferedInputStream and RandomAccessFile methods, because it utilizes the operating system specific optimization techniques under the hood. On the similar basis, SHA256 or MD5 can be calculated efficiently using FileChannels

```
public static String calculateSHA256(File file) throws IOException, NoSuchAlgorithmException {
    long t1 = System.currentTimeMillis();
    final int SIZE = 32 * 1024;
    final int SIZE2 = SIZE * 1024 * 100;
    try (FileInputStream in = new FileInputStream(file)) {
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        FileChannel channel = in.getChannel();
        long length = channel.size();
        long iterations = length / SIZE2;
        long reverseLength = 0l;
        for (int i = 0; i <= iterations; i++) {
            MappedByteBuffer mb = channel.map(FileChannel.MapMode.READ_ONLY, reverseLength, i == iterations?
length % SIZE2 : SIZE2);
            reverseLength += SIZE2;
            byte[] bytes = new byte[SIZE];
            int nGet;
            while (mb.hasRemaining()) {
                nGet = Math.min(mb.remaining(), SIZE);
                mb.get(bytes, 0, nGet);
                md.update(bytes, 0, nGet);
            }
        }
        byte[] mdbytes = md.digest();
        //convert the byte to hex format
        StringBuffer sb = new StringBuffer("");
        for (int i = 0; i < mdbytes.length; i++) {
            sb.append(Integer.toString((mdbytes[i] & 0xff) + 0x100, 16).substring(1));
        }
        long t2 = System.currentTimeMillis();
        System.out.println(sb.toString());
        System.out.println("Time for SHA2 = "+ (t2-t1));
        return sb.toString();
    }
}
```

Java Channels can be used in wide variety of IO tasks. For example, an efficient implementation of Http File Download can be written using a FileChannel, as shown below.

```
public boolean download(String rootUrl, String fileName) throws IOException {
    Path path = Paths.get(fileName);
    long totalBytesRead = 0L;

    HttpURLConnection con = (HttpURLConnection) new URL(rootUrl + fileName).openConnection();

    con.setReadTimeout(10000);
    con.setConnectTimeout(10000);

    try (ReadableByteChannel rbc = Channels.newChannel(con.getInputStream());
         FileChannel fileChannel = FileChannel.open(path, EnumSet.of(CREATE, WRITE));)
    {
        totalBytesRead = fileChannel.transferFrom(rbc, 0, 1 << 22); // download file with max size 4MB
        return true;
    } catch (IOException e) {
        e.printStackTrace();
        throw e;
    }
}
```

Above code snippet will transfer bytes from HttpURLConnection's Stream into FileChannel using transferFrom() method of FileChannel.

Here is snippet of Java Doc for this method

```
public abstract long transferFrom(java.nio.channels.ReadableByteChannel src,
                                  long position,
                                  long count)
                           throws java.io.IOException
```

" This method is potentially much more efficient than a simple loop that reads from the source channel and writes to this channel. ***Many operating systems can transfer bytes directly from the source channel into the filesystem cache without actually copying them."***

Other features of FileChannel

Bytes may be read or written at an absolute position in a file in a way that does not affect the channel's current position.

A region of a file may be mapped directly into memory; for large files this is often much more efficient than invoking the usual read or write methods.

Updates made to a file may be forced out to the underlying storage device, ensuring that data are not lost in the event of a system crash.

Bytes can be transferred from a file to some other channel, and vice versa, in a way that can be optimized by many operating systems into a very fast transfer directly to or from the filesystem cache.

A region of a file may be locked against access by other programs.

File channels are safe for use by multiple concurrent threads.

Q 142. We have an Application and we want that only Single Instance should run for that Application. If Application is already running then second instance should never be started. How would you handle this in Java?

There are two main ways to handle such scenario in Java -
1.) Use a Socket networking in your application and start a server socket on a predefined port. When second instance try to start up then check if the port os already occupied or not and accordingly take the decision.

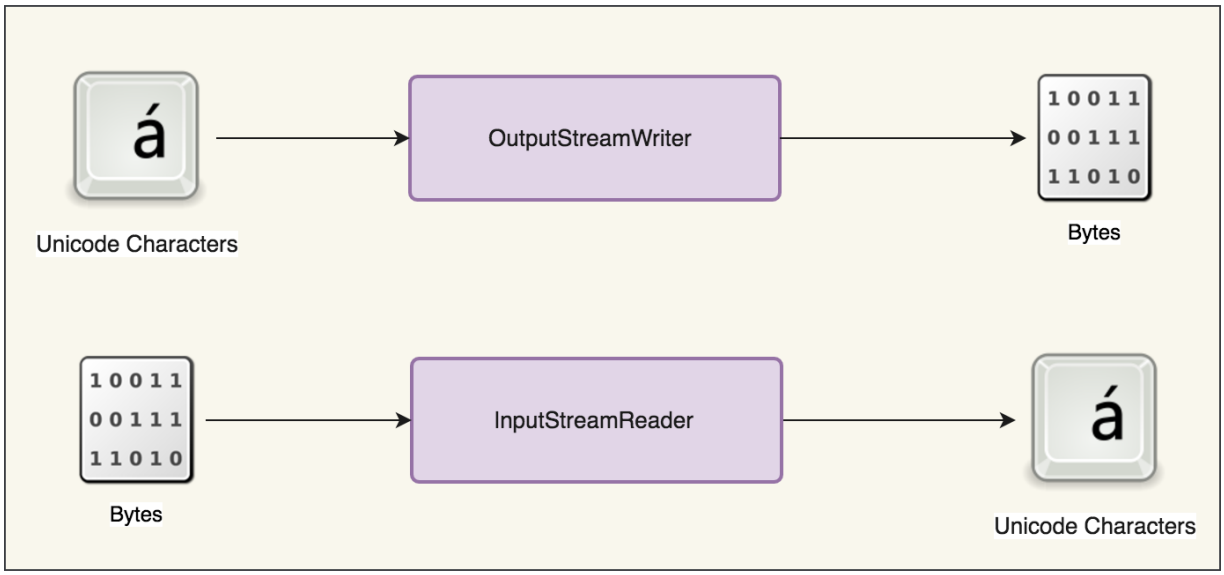
2.) Use a shared file lock using FileChannel and check if that temp file is already locked by some running process or not. If yes then terminate the startup process for second instance. Let's see how we can achieve this in the following code -

```
import java.io.File;
import java.io.RandomAccessFile;
import java.nio.channels.FileChannel;
import java.nio.channels.FileLock;
import java.nio.channels.OverlappingFileLockException;
public class SingleInstanceLock {
   private String appName;
   private File lockFile;
   private FileLock fileLock;
   private FileChannel fileChannel;
   public SingleInstanceLock(String appName) {
      this.appName = appName;
   }
   public boolean isAppActive() {
      try {
         lockFile = new File(System.getProperty("user.home"), appName + ".tmp");
         fileChannel = new RandomAccessFile(lockFile, "rw").getChannel();
         try {
            fileLock = fileChannel.tryLock();
         } catch (OverlappingFileLockException e) {
            System.out.println("Already Locked");
            closeLock();
            return true;
         }
         if (fileLock == null) {
            System.out.println("Could not obtain lock");
            closeLock();
            return true;
         }
         lockFile.deleteOnExit();
         return false;
      } catch (Exception e) {
         closeLock();
         return true;
      }
   }
  private void closeLock() {
      try { fileLock.release();} catch (Exception e) {}
      try { fileChannel.close();} catch (Exception e) {}
  }
}
```

Q 143. Why do we need Reader Classes when we already have Streams Classes? What are the benefit of using a Reader over a stream, in what scenario one should be preferred.

InputStream and OutputStream operates at byte level (also called byte streams) while Reader and Writer classes operates at the character level (char streams). Reader class is essentially a wrapper over InputStream where it delegates the I/O related work to the byte stream and performs the translation of byte to character using the given character encoding and character set. So Reader class provides a easy mechanism to the developer to deal with the Character stream with an option to deal with different CharacterSets.

It is possible to convert byte stream to a character stream using InputStreamReader and OutputStreamWriter.



Convert non-Unicode bytes to Unicode Characters using the below code

```
FileInputStream fis = new FileInputStream("test.txt");
InputStreamReader isr = new InputStreamReader(fis, "UTF8");
```

Convert Unicode Characters (from String object) to non-Unicode bytes using below code

```
static void writeOutput(String str) throws IOException {
   FileOutputStream fos = new FileOutputStream("test.txt");
   Writer out = new OutputStreamWriter(fos, "UTF8");
   out.write(str);
   out.close();
}
```

Chapter 3

Concurrency in Java

Q 144. What is Concurrency? How will you implement Concurrency in your Java Programs?

Concurrency is the property of an software program to run several computations in parallel. Java provides us with the multiple mechanisms to create Threads so as to utilize the multiple processor cores of a given hardware in order to achieve high throughput.

Java provides various ready to use utilities for writing concurrent programs, which otherwise is difficult to implement.

JDK 1.6 provides many useful utility classes in java.util.concurrent package -
Executors, Queues, TimeUnit, Synchronizers classes (Semaphore, CountDownLatch, CyclicBarrier, Exchanger), Concurrent collections (ConcurrentHashMap, CopyOnWriteArrayList, ConcurrentSkipListMap), java.util.concurrent.atomic package for non-blocking algorithms & collections, Lock Interface (ReentrantLock), and well defined Java Memory Model for memory consistency in concurrent environment.

an example usage of TimeUnit and Lock interface could be to try obtaining lock for 50ms as shown in below code snippet

```
Lock lock = ...;
  if (lock.tryLock(50L, TimeUnit.MILLISECONDS))
```

Such a method call can never cause a deadlock scenario because it will try acquiring lock only for 50 ms only.

What is Thread-Safety and how to achieve it?

A Class is thread safe when it behaves correctly when accessed & modified from multiple threads in parallel without any changes in the code calling it.

There are certain ways to make our class thread-safe, as follow

1. Use synchronization mechanism (intrinsic or explicit) on the accessor methods, voletile fields, atomic updates etc.
2. Make the class Immutable and this making it inherently thread-safe.
3. Don't expose the shared state across threads (for e.g. Keep objects local to thread using ThreadLocal)

What is Synchronization?

Synchronization avoids thread interference and memory consistency errors by providing serialized access to the shared state.

Synchronization has two major aspects

1. It makes sure that the compound actions executes atomically by providing mutually exclusive access to the shared state across the threads.
2. It ensures the memory consistency by making the changes visible to all the threads upon method exit.

Lets examine the following program for its correctness in concurrent environment, It maintains the integer

counter.

Counter.java

```
@NotThreadSafe
class Counter {
	private int c = 0;
	public void increment() {
		c++;
	}
	public int value() {
	return c;
	}
}
```

This program will work absolutely fine in single threaded environment but will not behave correctly in multi-threaded environment, because

1. increment() method will not be executed atomically so data race may corrupt the counter value.
2. value() method may not return the latest value of counter because of caching in processor's registers.

So lets make this program thread-safe.

Counter.java

```
@ThreadSafe
class Counter {
	private int c = 0;
	public synchronized void increment() {	//this will make the operation execution atomic across the threads
		c++;
	}
	public synchronized int value() {		//this will make sure the changes are visible to the calling thread
	return c;
	}
}
```

Please Make Sure

That you make the getter method as synchronized until the getter returns an immutable object. Otherwise the memory effects may not be consistent and the calling thread may see a stale value of the variable.

Weaker form of synchronization using volatile

Volatile variable can not make the method execution atomic, but it can make sure that the updates to the variable are propagated predictably to the other threads. volatile variables basically, are not cached into the processor registers, rather they are always fetched and written to the main memory on heap.

Concurrency vs Parallelism

Concurrency is when two tasks can start, run, and complete in overlapping time periods. It doesn't necessarily mean they'll ever both be running at the same instant. Eg. multitasking on a single core machine.
Parallelism is when tasks literally run at the same time, eg. on a multicore processor.

Q 145. There are two Threads A and B operating on a shared resource R, A needs to inform B that some important changes has happened in R. What technique would you use in Java to achieve this?

Object R's method wait(), notify() & notifyAll(), can be used for inter-thread

continued on 138

communication. This will allow all threads which hold lock over R, to communicate among them selves. You can explore a typical Producer-Consumer problem to see how it works.

Q 146. What are different states of a Thread? What does those states tells us?

A thread in JVM can have 6 different states as defined in Thread.State enum. At any given time, thread must be in any of these states.

NEW
This state is for a thread which has not yet started.

RUNNABLE
This state is for the currently running thread which is executing in java virtual machine, but it may be waiting for the other resources from operating system such as processor.

BLOCKED
Thread state for a thread blocked waiting for a monitor lock. A thread in this state can be waiting for a monitor lock to enter a synchronized block/method or reenter a synchronized method after calling Object.wait.

WAITING
A thread is waiting due to calling on one of the method -
Object.wait with no timeout
Thread.join with no timeout
LockSupport.park

A Thread in this state is waiting for another thread to perform a particular action. For example, a thread that has called Object.wait() on an object is waiting for another thread to call Object.notify() or Object.notifyAll() on that object. A thread that has called Thread.join() is waiting for a specified thread to terminate.

TIMED_WAITING
Thread state for a waiting thread with a specified waiting time. A thread is in the timed waiting state due to calling one of the following methods with a specified positive waiting time -

Thread.sleep
Object.wait with timeout
Thread.join with timeout
LockSupport.parkNanos
LockSupport.parkUntil

TERMINATED
Thread state for a terminated thread. The thread has completed execution.

References
This content has been taken directly from the Java 7 Docs - Thread.State enum.

Q 147. Question: What do you understand by Java Memory Model? What is double-checked locking? What is different about final variables in new JMM?

Interviewer's Intent - *Interviewer wants to understand your capabilities to write robust concurrent code.*

Java Memory Model[1] defines the legal interaction of threads with the memory in a real computer system. In a way, it describes what behaviors are legal in multi-threaded code. It determines when a Thread can reliably **see** writes to variables made by other threads. It defines semantics for volatile, final & synchronized, that makes guarantee of visibility of memory operations across the Threads.

Let's first discuss about Memory Barrier which are the base for our further discussions. There are two type of memory barrier instructions in JMM - read barriers & write barrier.

A read barrier invalidates the local memory (cache, registers, etc) and then reads the contents from the main memory, so that changes made by other threads becomes visible to the current Thread.
A write barrier flushes out the contents of the processor's local memory to the main memory, so that changes made by the current Thread becomes visible to the other threads.

JMM semantics for synchronized

When a thread acquires monitor of an object, by entering into a synchronized block of code, it performs a read barrier (invalidates the local memory and reads from the heap instead). Similarly exiting from a synchronized block as part of releasing the associated monitor, it performs a write barrier (flushes changes to the main memory)
Thus modifications to a shared state using synchronized block by one Thread, is guaranteed to be visible to subsequent synchronized reads by other threads. This guarantee is provided by JMM in presence of synchronized code block.

JMM semantics for Volatile fields

Read & write to volatile variables have same memory semantics as that of acquiring and releasing a monitor using synchronized code block. So the visibility of volatile field is guaranteed by the JMM. Moreover afterwards Java 1.5, volatile reads and writes are not reorderable with any other memory operations (volatile and non-volatile both). Thus when Thread A writes to a volatile variable V, and afterwards Thread B reads from variable V, any variable values that were visible to A at the time V was written are guaranteed now to be visible to B.

Let's try to understand the same using the following code

```
Data data = null;
volatile boolean flag = false;

Thread A
-------------
data = new Data();
flag = true;            <-- writing to volatile will flush data as well as flag to main memory

Thread B
-------------
if(flag==true){         <-- reading from volatile will perform read barrier for flag as well data.
use data;               <--- data is guaranteed to visible even though it is not declared volatile because of the JMM
                             semantics of volatile flag.
}
```

JMM semantics for final fields & Initialization safety

1 http://www.ibm.com/developerworks/library/j-jtp03304/

continued on 140

JSR 133 (new JMM with JDK 1.5 onwards) provides a new guarantee of initialization safety - that as long as an object is properly constructed (this reference does not escape during the construction), then all threads will see the correct value for its final fields that were set in its constructor, regardless of whether or not synchronized is used to publish the object from one thread to another. Further, any variable that can be reached through a final field of a properly constructed object, such as fields of an object referenced by a final field, are also guaranteed to be visible to the other threads. For example, if a final field contains reference to a ArrayList, in addition to the correct value of the reference being visible to other thread, also the contents of that ArrayList at construction time, would be visible to other threads without synchronization.
For all the final fields, when a constructor completes, all of the writes to final fields and to the variables reachable through those final fields becomes frozen, and any thread that obtains a reference to that object after the freeze is guaranteed to see the frozen values for all frozen fields. So it is a kind of *happen-before* relationship between the write of a final field in the boundary of constructor and the initial load of a shared reference to that object in another Thread.

Double-Checked Locking Problem

In earlier times (prior to JDK 1.6) a simple uncontended synchronization block was expensive and that lead many people to write double-checked locking to write lazy initialization code. The double-checked locking idiom tries to improve performance by avoiding synchronization over the common code path after the helper is allocated. But the DCL never worked because of the limitations of pervious JMM. This is now fixed by new JMM (JDK 1.5 onwards) using volatile keyword.

NonThreadSafe Singleton (This will not work under current JMM), so **never use it**

```
public class Singleton
{
  private Singleton() {}
  private static Singleton instance_ = null;      ==> A global static variable that will hold the state

 public static Singleton instance()
 {
  if(instance_==null)            ==> un-synchronized access to this fields may see partially constructed objects because
  {                                      of instruction reordering by the compiler or the cache
    synchronized(Singleton.class)
    {
      if(instance_==null)
        instance_= new Singleton();
    }
  }
  return instance_;
}
}
```

JMM will not guarantee the expected execution of this static singleton.

Why above code idiom is broken in current JMM?

DCL relies on the un synchronized use of _instance field. This appears harmless, but it is not. Suppose Thread A is inside sycnhronized block and it is creating new Singleton instance and assigning to _instance variable, while thread B is just entering the getInstance() method. Consider the effect on memory of this initialization. Memory for the new Singleton object will be allocated; the constructor for Singleton will be called, initializing the member fields of the new object; and the field resource of SomeClass will be assigned a reference to the newly created object. There could be two scenarios now

- Suppose Thread A has completed initialization of _instance and exits synchronized block as thread B enters getInstance(). By this time, the _instance is fully initalized and Thread A has flushed its local memory to main memory (write barriers). Singleton's member fields may refer other objects stored in memory which will also be flushed out.. While Thread B may see a valid reference to the newly created _instance, but

continued on 141

because it didn't perform a read barrier, it could still see stale values of _instance's member fields.

- Since thread B is not executing inside a synchronized block, it may see these memory operations in a different order than the one thread A executes. It could be the case that B sees these events in the following order (and the compiler is also free to reorder the instructions like this): allocate memory, assign reference to resource, call constructor. Suppose thread B comes along after the memory has been allocated and the resource field is set, but before the constructor is called. It sees that resource is not null, skips the synchronized block, and returns a reference to a partially constructed Resource! Needless to say, the result is neither expected nor desired.

Fixed double-checked Locking using volatile in new JMM (multi-threaded singleton pattern JDK 1.5)

The following code makes the helper volatile so as to stop the instruction reordering. This code will work with JDK 1.5 onwards only.

```
class Foo {
        private volatile Helper helper = null;
        public Helper getHelper() {
            if (helper == null) {
                synchronized(this) {
                    if (helper == null)
                        helper = new Helper();
                }
            }
            return helper;
        }
    }
```

If Helper is an **immutable** object, such that all of the fields of Helper are final, then double-checked locking will work without having to use volatile fields. The idea is that a reference to an immutable object (such as a String or an Integer) should behave in much the same way as an int or float; reading and writing references to immutable objects are atomic.

Alternatives to DCL[2]

Now a days JVM is much smarter and the relative expense of synchronized block over volatile is very less, so it does not really make sense to use DCL for performance reasons.

The easiest way to avoid DCL is to avoid it. We can make the whole method synchronized instead of making the code block synchronized.

Another option is to use *eager initialization* instead of lazy initialization by assigning at the creation time

Here is the example demonstrating eager initialization

```
class MySingleton {
  public static Resource resource = new Resource();
}
```

Using Initialization On Demand Holder idiom[3]

Inner classes are not loaded until they are referenced. This fact can be used to utilize inner classes for lazy initialization as shown below

```
public class Something {
    private Something() {
    }
    private static class LazyHolder {
        private static final Something INSTANCE = new Something();
```

2 http://en.wikipedia.org/wiki/Double-checked_locking#Usage_in_Java

3 http://en.wikipedia.org/wiki/Initialization_on_demand_holder_idiom#Example_Java_Implementation

```
    }
    public static Something getInstance() {
        return LazyHolder.INSTANCE;
    }
}
```

This code is guaranteed to be correct because of the initialization guarantees for static fields; if a field is set in a static initializer, it is guaranteed to be made visible, correctly, to any thread that accesses that class.

Using final wrapper to hold the Instance
Semantics of final field in Java 5 can be employed to safely publish the helper object without using volatile.

```
public class FinalWrapper<T> {
    public final T value;
    public FinalWrapper(T value) {
        this.value = value;
    }
}

public class Foo {
    private FinalWrapper<Helper> helperWrapper = null;
    public Helper getHelper() {
        FinalWrapper<Helper> wrapper = helperWrapper;
        if (wrapper == null) {
            synchronized(this) {
                if (helperWrapper == null) {
                    helperWrapper = new FinalWrapper<Helper>(new Helper());
                }
                wrapper = helperWrapper;
            }
        }
        return wrapper.value;
    }
}
```

The local variable wrapper is required for correctness.

And finally using Enum for Thread-Safe Singleton

```
public enum Singleton{
    INSTANCE;
}
```

For further readings -

http://www.ibm.com/developerworks/library/j-jtp03304/
http://en.wikipedia.org/wiki/Double-checked_locking#Usage_in_Java
http://en.wikipedia.org/wiki/Initialization_on_demand_holder_idiom#Example_Java_Implementation
http://en.wikipedia.org/wiki/Double-checked_locking#Usage_in_Java
http://www.cs.umd.edu/~pugh/java/memoryModel/DoubleCheckedLocking.html
http://www.javaworld.com/jw-02-2001/jw-0209-double.html
http://www.ibm.com/developerworks/java/library/j-jtp0618/index.html
http://www.ibm.com/developerworks/library/j-jtp02244/index.html
http://www.ibm.com/developerworks/java/library/j-jtp06197/index.html
http://www.cs.umd.edu/~pugh/java/memoryModel/jsr-133-faq.html
http://www.cs.umd.edu/~pugh/java/memoryModel/jsr133.pdf

Q 148. Is i++ thread-safe (increment operation on primitive types)?

No, because the ++ operator is non atomic.

Let's understand the following code -

```
public class Counter{
    private int i;

    public int increment(){
        i++;
    }
}
```

The method increment() in the above code is not thread safe, because i++ require multiple cpu instruction cycles to compute the summation. Data race condition may happen if the shared object is incremented from multiple threads, simultaneously.

How To Make It thread-safe?

There are mainly two ways to make it thread-safe in Java -
1. By making the increment method synchronized. (Preferred when thread contention is moderate to high)
2. By using AtomicInteger to maintain the increment, utilizing CAS under the hood. (Preferred for single CPU, and for low to moderate thread contention)

If you want, you can write your custom **AbstractQueueSynchronizer** to achieve the same, but that discussion is out of scope for this writing.

Q 149. What happens when wait() & notify() method are called?

When wait() method is invoked from a synchronized context, the following things happen
- The calling thread gives up the lock.
- The calling thread gives up the CPU.
- The calling thread goes to the monitor's waiting pool.

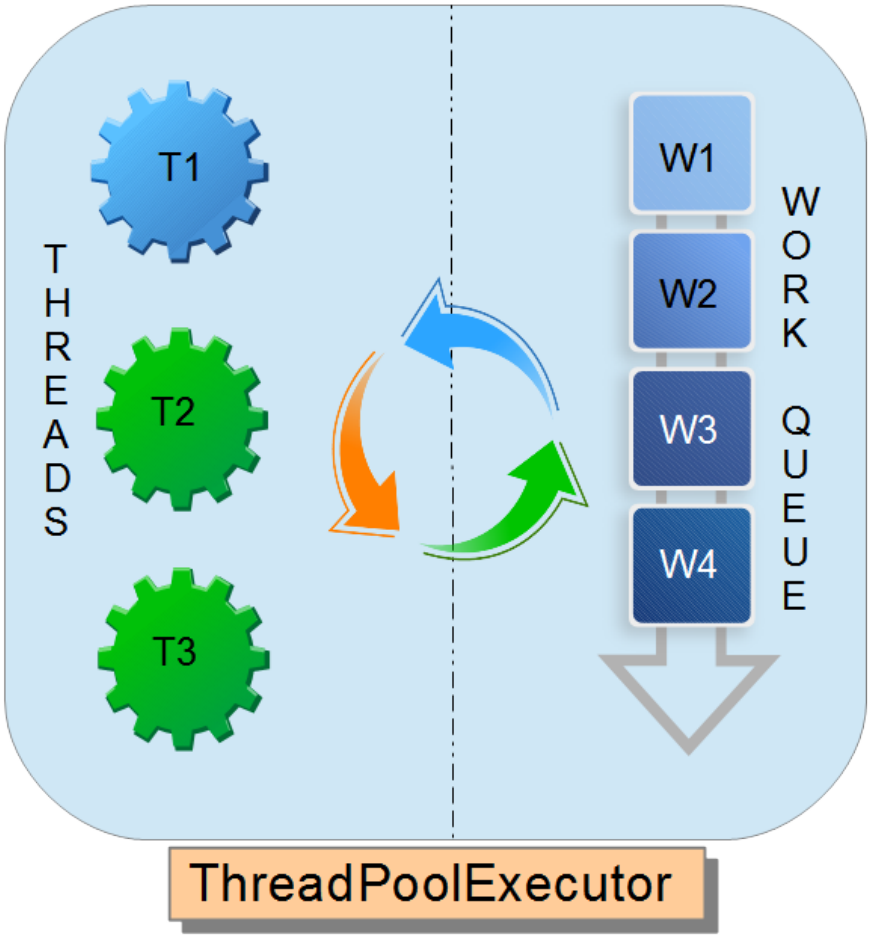
And in case of notify() method, following things happen
- One of the waiting thread (may be a random thread) moves out of the monitor's waiting pool.
- Thread comes into ready state (RUNNABLE).
- Tries its best to require the monitor lock before it can proceed to the method execution.

Q 150. Discuss ThreadPoolExecutor? What different Task Queuing Strategies are possible? How will you gracefully handle rejection for Tasks?

ThreadPoolExecutor is an ExecutorService that uses a pool of thread (comprising of one or more threads) to execute each submitted tasks. It provides many benefits over naive approach of creating a new thread for each submitted task -
1. Usually improved performance when executing a large number of asynchronous tasks, due to sharing of thread pool for consecutive tasks.
2. It provides a means to bound the resources as per configuration thus keeping your application predictable at extreme input load. Both threads and Work Items can be bounded to maximum limit while creating a new

ThreadPoolExecutor instance.

3. Configurable rejection handler gets invoked when a particular task can not be executed due to some known reason (Thread Pool shutting down, queue can not hold more work items, etc.)



Different Task Queuing Strategies Available

There are 3 general strategies available for task queuing.

1. **Direct handoff (default choice)** - this is default choice for work queue. This uses a SynchronousQueue to hand off tasks to threads without otherwise holding them. An attempt to queue a task will fail if no immediate thread is available to run the task, so a new thread might be constructed to run the submitted task. Direct handoffs generally require unbounded maximumPoolSizes to avoid rejection of new submitted tasks. This in turn admits the possibility of unbounded thread growth when commands continue to arrive on average faster than they can be processed.
2. **Unbounded queues** - Using an unbounded queue (for example a LinkedBlockingQueue without a predefined capacity) will cause new tasks to wait in the queue when all corePoolSize threads are busy. Thus, no more than corePoolSize threads will ever be created. There is a possibility of unbounded work queue growth when commands continue to arrive on average faster than they can be processed
3. **Bounded queues** - A bounded queue (for example, an ArrayBlockingQueue) helps prevent resource exhaustion when used with finite maximumPoolSizes, but can be more difficult to tune and control. Queue sizes and maximum pool sizes may be traded off for each other: Using large queues and small pools minimizes CPU usage, OS resources, and context-switching overhead, but can lead to artificially low throughput.

Task Rejection Handler

New tasks submitted in method execute(Runnable) will be rejected when the Executor has been shut down, and also when the Executor uses finite bounds for both maximum threads and work queue capacity, and is saturated. In either case, the execute method invokes the RejectedExecutionHandler. rejectedExecution(Runnable, ThreadPoolExecutor) method of its RejectedExecutionHandler. Four predefined handler policies are provided:

1. In the default ThreadPoolExecutor.AbortPolicy, the handler throws a runtime RejectedExecutionException upon rejection.
2. In ThreadPoolExecutor.CallerRunsPolicy, the thread that invokes execute itself runs the task. This provides a simple feedback control mechanism that will slow down the rate that new tasks are submitted.
3. In ThreadPoolExecutor.DiscardPolicy, a task that cannot be executed is simply dropped.
4. In ThreadPoolExecutor.DiscardOldestPolicy, if the executor is not shut down, the task at the head of the work queue is dropped, and then execution is retried (which can fail again, causing this to be repeated.)

Here is the Java Code example that creates a new ThreadPoolExecutor with AbortPolicy()

```
int poolSize=2;
int maxPoolSize=5;
int queueSize=3;
long aliveTive=1000;
ArrayBlockingQueue<Runnable> queue= new ArrayBlockingQueue<Runnable>(queueSize);
ThreadPoolExecutor threadPoolExecutor= new ThreadPoolExecutor(poolSize,maxPoolSize,aliveTive,
      TimeUnit.MILLISECONDS,queue,new ThreadPoolExecutor.AbortPolicy());
```

Q 151. How will you write a custom ThreadPoolExecutor that can be paused and resumed on demand? You can extend the existing ThreadPoolExecutor to add this new behavior.

We can easily extend the Java provided ThreadPoolExecutor to add pause/resume behavior, as illustrated in the below program.

```
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.ReentrantLock;

class PausableThreadPoolExecutor extends ThreadPoolExecutor {
   private boolean isPaused;
   private ReentrantLock pauseLock = new ReentrantLock();
   private Condition unpaused = pauseLock.newCondition();

   public PausableThreadPoolExecutor(int corePoolSize,
                                     int maximumPoolSize,
                                     long keepAliveTime,
                                     TimeUnit unit,
                                     BlockingQueue<Runnable> workQueue) {
      super(corePoolSize, maximumPoolSize, keepAliveTime, unit, workQueue);
   }

   protected void beforeExecute(Thread t, Runnable r) {
```

```
            super.beforeExecute(t, r);
            pauseLock.lock();
            try {
                while (isPaused) unpaused.await();
            } catch (InterruptedException ie) {
                t.interrupt();
            } finally {
                pauseLock.unlock();
            }
        }

        public void pause() {
            pauseLock.lock();
            try {
                isPaused = true;
            } finally {
                pauseLock.unlock();
            }
        }

        public void resume() {
            pauseLock.lock();
            try {
                isPaused = false;
                unpaused.signalAll();
            } finally {
                pauseLock.unlock();
            }
        }
    }
```

Q 152. How will you write your own custom thread pool executor from scratch?

Thread pool executor requires a Queue for holding tasks, and a collection of Worker Threads that will pick up tasks from the work queue start running them. Let us try to write our own simple Thread Pool Executor implementation.

```
package org.shunya.interview.example2;

import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;

public class CustomThreadPoolExecutor {
    private final BlockingQueue<Runnable> workerQueue;
    private final Thread[] workerThreads;

    public CustomThreadPoolExecutor(int numThreads) {
        workerQueue = new LinkedBlockingQueue<>();
        workerThreads = new Thread[numThreads];

        int i = 0;
        for (Thread t : workerThreads) {
            t = new Worker("Custom Pool Thread " + ++i);
            t.start();
```

```
        }
    }

    public void addTask(Runnable r) {
        try {
            workerQueue.put(r);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    class Worker extends Thread {
        public Worker(String name) {
            super(name);
        }

        public void run() {
            while (true) {
                try {
                    workerQueue.take().run();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                } catch (RuntimeException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
public static void main(String[] args) {
    CustomThreadPoolExecutor threadPoolExecutor = new CustomThreadPoolExecutor(10);
    threadPoolExecutor.addTask(() -> System.out.println("First print task"));
    threadPoolExecutor.addTask(() -> System.out.println("Second print task"));
    threadPoolExecutor.addTask(() -> System.out.println("Third print task"));
}
```

This is a very basic functionality for a Thread Pool Executor, more behavior can be added to it as and when required.

Q 153. What is difference between ExecutorService's submit() and execute() method?

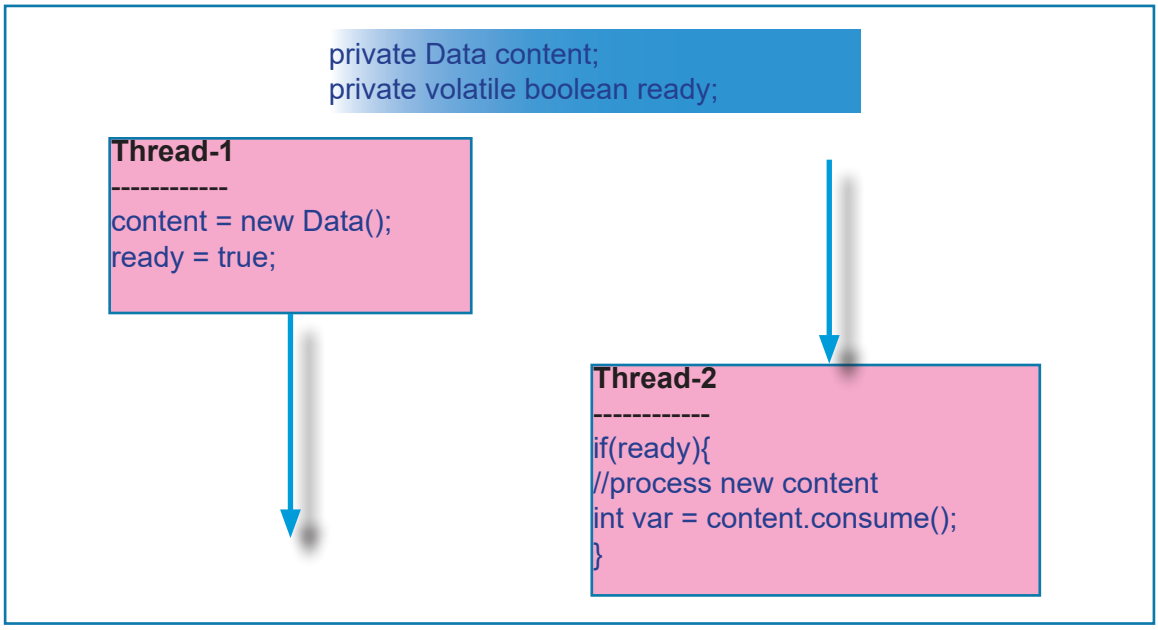
1. The execute() method is declared in Executor interface while submit() method is declared in ExecutorService interface.
2. The submit() method can accept both Runnable as well as Callable tasks, but execute() method can only accept a Runnable Task.
3. execute() method returns void while submit() method returns Future object. Using future you can cancel the execution or retrieve the results of computation done by runnable/callable task.
4. There is a difference when looking at exception handling. If your tasks throws an exception and if it was submitted with execute this exception will go to the uncaught exception handler (when you don't have provided one explicitly, the default one will just print the stack trace to System.err). If you submitted the task with submit any thrown exception, checked exception or not, is then part of the task's return status. For a task that was submitted with submit and that terminates with an exception, the Future.get will re-throw this exception, wrapped in an ExecutionException.

Q 154. Discuss about volatile keyword and Java Memory Model?

Volatile is a mechanism for lighter weight synchronization where memory visibility of the protected state is guaranteed to all consecutive threads.

A write to volatile variable not only flush changes of the volatile variable but all the non volatile variables changed before write to volatile. Thus a simple flag of volatile type can serve the memory visibility guarantee of all the other variables changed before. The following figure explain it in entirety.

Figure : *Effects of volatile variables on time scale*



Non-atomic treatment of long and double

For the purposes of the Java programming language memory model, a single write to a non-volatile long or double value is treated as two separate writes: one to each 32-bit half. This can result in a situation where a thread sees the first 32 bits of a 64-bit value from one write, and the second 32 bits from another write.

Writes and reads of volatile long and double values are always atomic.

Writes to and reads of references are always atomic, regardless of whether they are implemented as 32-bit or 64-bit values.

It is safe to perform read-modify-write operations on a shared volatile variable as long as you ensure that the volatile variable is only written from single thread.
volatile variables are liable to get into race conditions because atomicity is required to solve race condition

Q 155. What is a CAS? How does it help writing non-blocking scalable applications? Tell something about Atomic Package provided by Java 1.6

Problem with Traditional Locking (using synchronized keyword or Lock objects)[1]

If a thread tries to acquire a lock that is already held by some other thread, then the thread has to block until the lock becomes available. This could lead to scalability hazards if the new thread was performing high priority tasks. Deadlock is second problem when dealing with locking due to inconsistency in acquiring multiple locks. Thirdly for managing very lightweight operations (in terms of CPU cycles) like increment counter & concurrent updates to a variable, acquiring lock could cause more overhead than the real computation logic.

Compare and Swap (CAS) for the rescue

It provides us with the finer-grained mechanism for managing lock-free thread safe concurrent updates to individual variables, after seeking some hardware level support to achieve the same. A CAS operation includes three commands - a memory location, the expected old value and a new value. The underlying processor will atomically update the given memory location to new value if there matches the expected old value, otherwise it will do nothing.

Atomic Package (java.util.concurrent.atomic)

Atomic package is a small toolkit of classes that exposes CAS operations and thus helping us write lock-free thread safe programming on single variables. Most atomic classes provide the following method for CAS

```
boolean compareAndSet(expectedValue, updateValue);
```

This method (which varies in argument types across different classes) atomically sets a variable to the updateValue if it currently holds the expectedValue, reporting true on success.

Memory effects of Atomic Classes

Memory effects for read and update of a atomic variable generally follow the rules for the volatile -

- get() has memory effects of reading a volatile variable.
- set() has memory effects of writing a volatile variable.
- compareAndSet, getAndIncrement has memory effects of read and write to volatile variable.

Lock-free and wait-free Algorithms using Compare and Swap (CAS)

In a lock-free algorithm, at least some thread always make progress

In a wait-free algorithm, every thread will continue to make some progress in face of arbitrary delay of other threads.

Below is a small example utilizing CAS for Implementing a non-blocking sequence generator.

```
class Sequencer {
  private final AtomicLong sequenceNumber = new AtomicLong(0);
  public long next() {
    return sequenceNumber.getAndIncrement();
  }
}
```

We should keep it in mind that CAS operations should be preferred to locking code only when :

1. The operation is very lightweight and confined to a single variable update
2. Thread contention is low to moderate, under heavy contention, performance will suffer dramatically, as the JVM spends more time dealing with scheduling threads and managing contention and queues of waiting threads and less time doing real work, like incremental counters.

1 http://www.ibm.com/developerworks/java/library/j-jtp11234/

Q 156. There is a object state which is represented by two variables. How would you write a high throughput non-blocking algorithm to update the state from multiple threads?

Non-blocking algorithms provide better throughput at low thread contention compared to the locking counterpart. This can only be achieved in Java using CAS[1] (compare and swap) utilities provided in atomic package. AtomicReference along with Immutable object can be used to write a non-blocking algorithm maintaining a current state of Trade Volume.
There are key points to be noted while writing non-blocking algorithm[2] are:

- Immutability of TradeVolume as in below example is must to ensure proper initialization at it's assignment time. Immutability is achieved by making all fields final and providing constructor initialization.
- compareAndSet operation must be called repetitively in a while loop till the time it returns true.

```
public class NonBlockingTradeUpdate {
    /***
     * This TradeVolume Class must be Immutable otherwise it may create Java Memory Model Problems
     * while using with Atomic Reference. All the fields must be final to guarantee the initialization
     * and assignment at the same time from other thread.
     */
    @Immutable
    private static class TradeVolume {
        final long quantity;
        final long price;
        private TradeVolume(long quantity, long price) {
            this.quantity = quantity;
            this.price = price;
        }
    }
    private final AtomicReference<TradeVolume> tradeVol = new AtomicReference<>(new TradeVolume(100, 200));
    public long getQuantity() {
        return tradeVol.get().quantity;
    }
    public long getPrice() {
        return tradeVol.get().price;
    }
    /**
     * A non-blocking update method which updates the TradeVolume Object using AtomicReference.
     * This method is likely to perform better under multi-core environment with low thread contention.
     * @param quantity Quantity of the Trade
     * @param price    Price of the Trade
     */
    public void update(long quantity, long price) {
        while (true) {
            TradeVolume oldValue = tradeVol.get();
            TradeVolume newValue = new TradeVolume(quantity+ oldValue.quantity, price+ oldValue.price);
            if (tradeVol.compareAndSet(oldValue, newValue))
                return;
        }
    }
}
```

1 See Concurrency In Practice chapter 15.3 - Atomic Variable Classes
2 https://www.ibm.com/developerworks/java/library/j-jtp04186/

The update method can be written concisely using the updateAndGet() method introduced in Java 8

JAVA 8 version of the update method discussed earlier

```
public void update(long quantity, long price) {
    tradeVol.updateAndGet(x -> new TradeVolume(quantity + x.quantity, price + x.price));
}
```

Here is the documentation for updateAndGet method from Java Docs

"Atomically updates the current value with the results of applying the given function, returning the updated value. The function should be side-effect-free, since it may be re-applied when attempted updates fail due to contention among threads."

Q 157. How would you implement AtomicFloat /AtomicDouble using CAS?

Java does not provide Atomic implementation for Double/Float as they have stated in their documentation[3]

"Atomic classes are not general purpose replacements for java.lang.Integer and related classes. They do not define methods such as hashCode and compareTo. (Because atomic variables are expected to be mutated, they are poor choices for hash table keys.) Additionally, classes are provided only for those types that are commonly useful in intended applications. For example, there is no atomic class for representing byte. In those infrequent cases where you would like to do so, you can use an AtomicInteger to hold byte values, and cast appropriately. You can also hold floats using Float.floatToIntBits and Float.intBitsToFloat conversions, and doubles using Double.doubleToLongBits and Double.longBitsToDouble conversions."

There are two ways to implement AtomicDouble/AtomicFloat.

First method is to use AtomicReference to hold Double value as shown in below snippet

```
import java.util.concurrent.atomic.AtomicReference;

public class AtomicDoubleUpdater {
    private final AtomicReference<Double> curr = new AtomicReference<>();

    public void add(Double aDouble) {
        for (; ; ) {
            Double oldVal = curr.get();
            Double newVal = oldVal + aDouble;
            if (curr.compareAndSet(oldVal, newVal))
                return;
        }
    }
}
```

Second method is to use AtomicInteger for storing Float bit values as hinted by above Java docs

```
import java.util.concurrent.atomic.AtomicInteger;
import static java.lang.Float.*;
```

3 http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/atomic/package-summary.html

continued on 152

```
class AtomicFloat extends Number {
    private AtomicInteger bits;

    public AtomicFloat() {
        this(0f);
    }
    public AtomicFloat(float initialValue) {
        bits = new AtomicInteger(floatToIntBits(initialValue));
    }
    public final boolean compareAndSet(float expect, float update) {
        return bits.compareAndSet(floatToIntBits(expect),
                floatToIntBits(update));
    }
    public final void set(float newValue) {
        bits.set(floatToIntBits(newValue));
    }
    public final float get() {
        return intBitsToFloat(bits.get());
    }

    public float floatValue() {
        return get();
    }

    public final float getAndSet(float newValue) {
        return intBitsToFloat(bits.getAndSet(floatToIntBits(newValue)));
    }

    public final boolean weakCompareAndSet(float expect, float update) {
        return bits.weakCompareAndSet(floatToIntBits(expect),
                floatToIntBits(update));
    }

    public double doubleValue() { return (double) floatValue(); }
    public int intValue()       { return (int) get();           }
    public long longValue()     { return (long) get();          }
}
```

Notes

Java 8 provides LongAdder and DoubleAdder to address atomic update problems for Long and Double. Java Also provides LongAccumulator and DoubleAccumulator to accumulate values from different threads and then consolidate them in final stage.

Typical LongAdder Usage would look like -

```
final LongAdder adder = new LongAdder();
for (...)
  pool.submit(() -> {
      while (...) {
       ...
       if (...) adder.increment();
      }
    });
...
long total = adder.sum());
```

And LongAccumulator can be used like this -

```
LongAccumulator adder = new LongAccumulator(Long::sum, 0);
// from multiple some thread...
```

continued on 153

```
    adder.accumulate(value);
    //final thread
    adder.doubleValue();
```

Q 158. How LongAdder and LongAccumulator are different from AtomicLong & AtomicInteger?

When you have a very large number of threads accessing the same atomic values, performance suffers because the optimistic updates require too many retries. Java 8 provides classes LongAdder and LongAccumulator to solve this problem. A LongAdder is composed of multiple variables whose collective sum is the current value. Multiple threads can update different summands, and new summands are automatically provided when the number of threads increases. This is efficient in the common situation where the value of the sum is not needed until after all work has been done. The performance improvement can be substantial.

Thus, If you anticipate high contention, you should simply use a LongAdder instead of an AtomicLong.

LongAdder Example

```
public void demo1() throws InterruptedException {
    ExecutorService executorService = Executors.newFixedThreadPool(8);
    LongAdder adder = new LongAdder();
    int numberOfThreads = 8;
    int numberOfIncrements = 100;
    Runnable incrementAction = () -> IntStream
            .range(0, numberOfIncrements)
            .forEach(i -> adder.increment());

    for (int i = 0; i < numberOfThreads; i++) {
        executorService.execute(incrementAction);
    }

    executorService.shutdown();
    executorService.awaitTermination(10, TimeUnit.SECONDS);
    final long sum = adder.sum();
    assert sum == numberOfThreads * numberOfIncrements;
}
```

In the above example, multiple threads (8 in number) are concurrently incrementing a LongAdder object. LongAdder uses a clever tactics to reduce the thread contention during operations by multiple threads. It does so by keeping an array of counters that can grow on demand. So there is at an expense of higher memory consumption.

Q 159. Can we implement check & update method (similar to compare and swap) using volatile alone?

No, this is not possible using volatile keyword alone. Volatile keyword can not guarantee atomicity of operation. It's a lighter weight synchronization which can guarantee memory visibility only.
The only way to implement CAS is either using synchronized block (Lock Interface as well) or using java provided hardware level CAS in it's atomic package i.e. using AtomicReference, AtomicInteger, etc

Q 160. What is difference between Fork/Join framework and ExecutorService?

The main difference between Fork/Join and Executor framework is the **Work Stealing Algorithm.**
Unlike the Executor Framework, in Fork/Join framework, when a task is waiting for the completion of the sub-tasks it has created using the join operation, the worker thread that is executing that task looks for another tasks that have not been executed yet and steal them to start their execution. This makes Fork Join framework much more efficient in terms of cpu usage, thereby improving the performance of the application.

Existing Java APIs using Fork Join framework

There are some generally usable features in Java SE 8 that are already implemented using the fork/join framework.

- parallelSort() method introduced in java.util.Arrays leverage concurrency via the fork join framework. This makes parallel sorting operation faster on multi-core machines compared to sequential sorting.
- Parallelism implemented in Stream.parallel() uses fork join framework under the hood.

Q 161. How does ForkJoinPool helps in writing concurrent applications? Please provide few examples for RecursiveTask and RecursiveAction.

Fork Join framework reduces the contention for the work queue by using work stealing technique. Each worker thread has its own work queue, which is implemented using a double-ended queue (`Deque, ArrayDeque, LinkedBlockingDeque`). When a task forks a new thread, it pushes it onto the head of its own Deque. When a task executes a join operation with another task that has yet to complete, rather than sleeping until the target task is complete (as Thread.join() does), it pops another task from the head of its Deque and executes that. In case the thread's task Deque is empty, it tries to steal task from the tail of another thread's Deque.

There are at least two advantages of using the Deque instead of normal Queue in this case -

- Reduced Contention - Only worker thread ever accesses the head of its own Deque, there is never contention for the head of the Deque. Similarly tail of the Deque is only ever accessed when a thread runs out of work, there is rarely contention for the tail of any thread's Deque either. This reduction in contention dramatically reduces the synchronization costs compared to a traditional thread-pool based frameworks.
- Reduced Stealing - LIFO ordering of the tasks means that the largest tasks sit at the tail of the Deque and thus when another thread has to steal a task, it steals a large one that can be decomposed into smaller ones, reducing the need to steal again in the near future.

Work Stealing thus produces reasonable load balancing with no central coordination and minimal synchronization costs.

Few important points to consider before we start implementing Fork Join Pool for a given requirement:

- Choosing the optimum threshold value for sequential computation is very important, otherwise we may not get the better results on multi-core hardware
- If the threshold is too small, then the overhead of task creation and management could become significant. In this case machine memory may become bottleneck and counter any gain in the throughput.
- If the threshold is too large, then the program might not create enough tasks to fully take advantage of the available cores in processor
- In case of doubt, we should perform a benchmark to see if there is actually gain in throughput due to Fork Join Pool
- JDK documentation suggests, a task should perform more than 100 and less than 10000 basic computational steps, and should avoid indefinite looping.

Example Program that sums all elements of an Array

```
import java.util.concurrent.*;

class SummationTask extends RecursiveTask<Long> {
    private final int THRESHOLD = 2;
    final long[] array;
    final int lo, hi;

    SummationTask(long[] array, int lo, int hi) {
        this.array = array;
        this.lo = lo;
        this.hi = hi;
    }
    protected Long compute() {
        if (hi - lo < THRESHOLD) {
            long result = 0;
            for (int i = lo; i < hi; ++i)
                result += array[i];
            return result;
        } else {
            int mid = (lo + hi) >>> 1;
            SummationTask left = new SummationTask(array, lo, mid);
            left.fork();
            SummationTask right = new SummationTask(array, mid, hi);
            long result = right.compute() + left.join();
            return result;
        }
    }
    public static void main(String[] args) throws ExecutionException, InterruptedException {
        long[] array = {1, 2, 3, 4, 5, 6, 7, 8, 9};
        SummationTask incrementTask = new SummationTask(array, 0, 9);
        ForkJoinPool forkJoinPool = new ForkJoinPool();
        forkJoinPool.invoke(incrementTask);
        System.out.println(incrementTask.get());
    }
}
```

Diagram on the next page explains the working of RecursiveTask execution in Fork Join Pool (using Divide and Conquer approach).

Q 162. How will you track the largest value monitored by different threads in an non-blocking fashion (using atomic operations)?

Lets first understand the problem:

There is a shared long value which is updated simultaneously by multiple threads. Interviewer wants you to track the largest value taking into consideration the thread-safety aspects.

We can use AtomicLong (from java.util.concurrent package) to achieve this behavior in a non-blocking fashion.

A naive approach could be to use AtomicLong.set(<long>), like shown in the following code:

A naive-approach (it is not thread-safe)

```
   public class LargestLongTracker {
       private final AtomicLong largest = new AtomicLong();

       public void updateAndTrack1(long observedValue) {
           largest.set(Math.max(largest.get(), observedValue));
       }

       public long getLargest() {
           return largest.get();
       }
   }
```

The above code is prone to error due to race condition. Calculating the max value and setting the calculated value into largest is not a atomic operation.

Correct approach using Java 7 would be to compute the new value and use compareAndSet in a loop:

Java 7 approach

```
public class LargestLongTracker {
    private final AtomicLong largest = new AtomicLong();

    public void updateAndTrack2(long observedValue) {
        long oldValue, newValue;
        do {
            oldValue = largest.get();
            newValue = Math.max(oldValue, observedValue);
        } while (!largest.compareAndSet(oldValue, newValue));
    }
}
```

The important point to note here is that this operation may fail and we are retrying failed attempts again and again till we succeed.

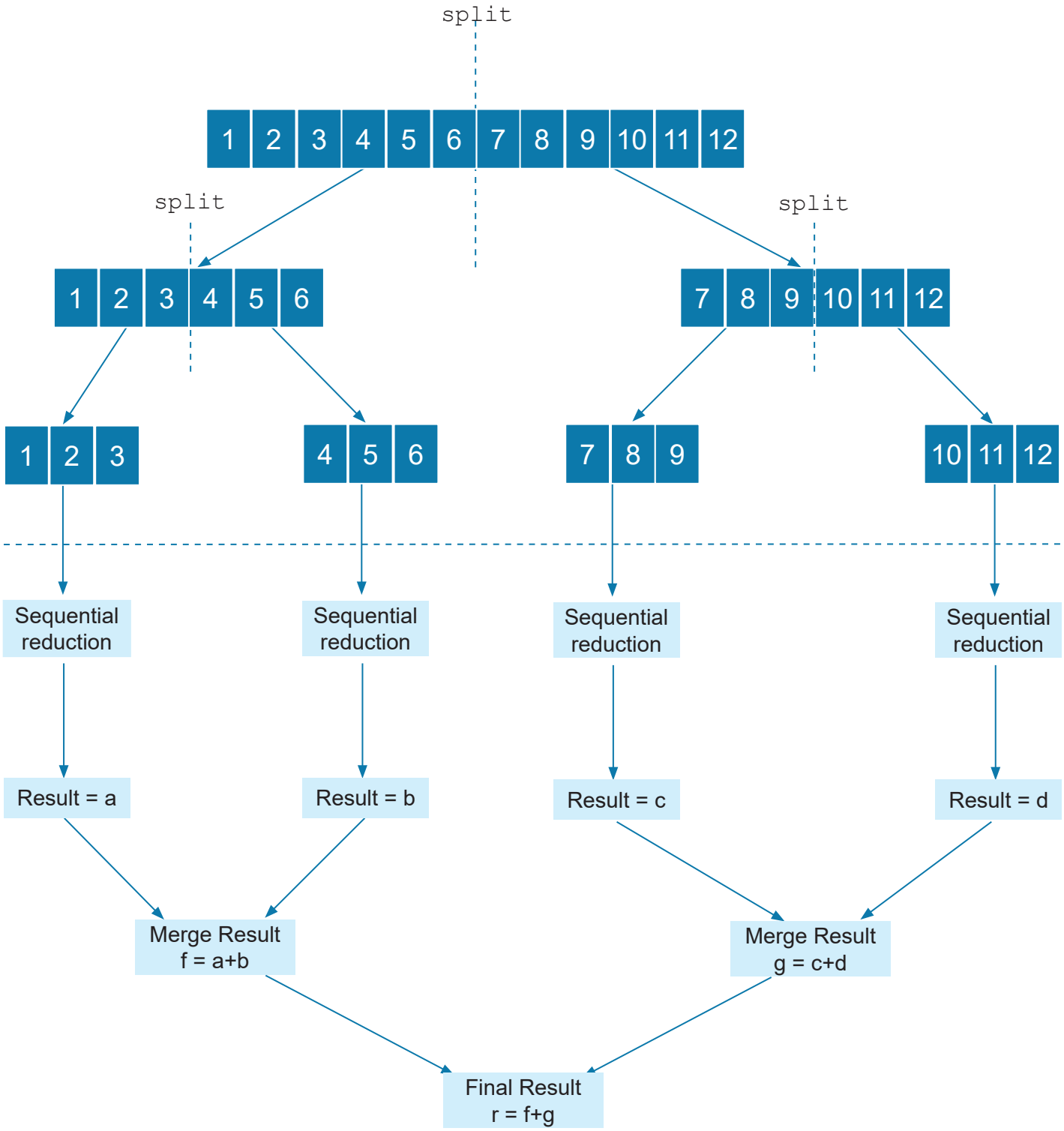
Best approach

Java 8 introduced accumulateAndGet & updateAndGet that accepts a accumulator function (lambda) to perform the action in non-blocking fashion.

Java 8 approach (Atomic update)

```
public class LargestLongTracker {
    private final AtomicLong largest = new AtomicLong();

    public void updateAndTrack3(long observedValue) {
        largest.accumulateAndGet(observedValue, Math::max);
        //OR
        largest.updateAndGet(x -> Math.max(x, observedValue));
    }

}
```

In new Java 8 code, we shall always prefer this approach over other options.



Diagrammatic representation of Recursive Task that sums all elements of array using Fork Join Pool

Q 163. How will you calculate Fibonacci Sequence on a multi-core processor?

Java provides us with Fork Join framework where we can use Divide and Conquer approach to divide a given task onto multiple cores of machine. Below is the example Fibonacci program that can utilize the multiple cores of processor -

Regular Fibonacci Program prior to Java 7

```
public int fib(int n) {
   if (n <= 1)       //Base Condition
      return n;
   else {          //Recursive case
      return fib(n - 1) + fib(n - 2);
   }
}
```

Fibonacci Program using Fork Join Pool

```
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveTask;

class Fibonacci extends RecursiveTask<Integer> {
   final int n;

   Fibonacci(int n) {
      this.n = n;
   }

   protected Integer compute() {
      if (n <= 1)
         return n;
      Fibonacci f1 = new Fibonacci(n - 1);
      f1.fork();
      Fibonacci f2 = new Fibonacci(n - 2);
      return f2.compute() + f1.join();
   }

   public static void main(String[] args) {
      Fibonacci fibonacci = new Fibonacci(100);
      ForkJoinPool.commonPool().invoke(fibonacci);
   }
}
```

When we invoke fork() on a RecursiveTask, a new sub-task is pushed to head of its own Deque. If the current thread is working on its full capacity, another thread may steal this sub task and start executing it. Invoking join() on the task, causes current thread to halt the current execution and pops the another task from the head of its Deque, thus making Fork Join approach much more efficient.

Q 164. How will you increment each element of an Integer array, utilizing all the cores of processor?

We can use Fork and Join Task to divide this problem into smaller subsets that can be executed in multiple cpu cores, as shown below -

```
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveAction;

class IncrementTask extends RecursiveAction {
   private final int THRESHOLD = 100;
   final long[] array;
   final int lo, hi;

   IncrementTask(long[] array, int lo, int hi) {
      this.array = array;
      this.lo = lo;
      this.hi = hi;
   }

   protected void compute() {
      if (hi - lo < THRESHOLD) {
         for (int i = lo; i < hi; ++i)
            array[i]++;
      } else {
         int mid = (lo + hi) >>> 1;
         invokeAll(new IncrementTask(array, lo, mid), new IncrementTask(array, mid, hi));
      }
   }

   public static void main(String[] args) {
      long[] array = {1, 2, 3, 4, 5, 6, 7, 8, 9};
      IncrementTask incrementTask = new IncrementTask(array, 0, 9);
      ForkJoinPool forkJoinPool = new ForkJoinPool();
      forkJoinPool.invoke(incrementTask);
   }
}
```

Q 165. You are writing a multi-threaded software piece for NSE for maintaining the volume of Trades made by its individual brokers (icici direct, reliance). It's highly concurrent scenario and we can not use lock based thread safety due to high demand of throughput. How would handle such scenario?

```
private ConcurrentHashMap<String, BigDecimal> sumByAccount;

this hashmap could contain entries like :
'ICICI Direct' -> 10000.00
'Reliance Money' -> 20000.00
```

Since the multiple threads could be simultaneously adding the value to their respective broker, the designed code should be thread safe. Un-Thread safe version looks like :

```
public void addToSum(String account, BigDecimal amount){
    BigDecimal newSum = sumByAccount.get(account).add(amount);
    sumByAccount.put(account, newSum);
}
```

Solution

CAS can be utilized for achieving the high throughput requirement of the underlying system in this case. AtomicReference<BigDecimal> could be used to store the BigDecimal value atomically.

```
ConcurrentHashMap<String, AtomicReference<BigDecimal>> map;
public void addToSum(String account, BigDecimal amount) {
    AtomicReference<BigDecimal> newSum = map.get(account);
    for (;;) {
        BigDecimal oldVal = newSum.get();
        if (newSum.compareAndSet(oldVal, oldVal.add(amount)))
            return;
    }
}
```

AtomicReference uses CAS to atomically compare and assign a single reference. In the above code the compareAndSet(oldVal, oldVal.add(amount)) method checks if the AtomicReference == oldVal (by their memory location instead of actual value), if true then it replaces the value of field stored in AtomicReference with the oldVal.add(amount). All this comparison and swapping happens atomically by the JVM. Afterwards invoking the newSum.get() will return the added amount.
For loop is required here because it is possible that multiple threads are trying to add to the same AtomicReference and doing so just one thread succeeds and other fails. The failed threads must try again the operation to make the addition to BigDecimal.
Please be noted that CAS is recommended for moderate Thread contention scenarios. Synchronized should always be preferred for high contention code blocks. Or prefer to use LongAccumulator/ DoubleAccumulator if you are using Java 8

Java 8 Can use DoubleAccumulator for the same purpose with much compact code (Preferred Approach), addToSum(account, amount) method will look like below in Java 8

```
public void addToSum(String account, double amount) {
   sumByAccount.computeIfAbsent(account -> new LongAccumulator()).accumulate(amount);
}
```

Q 166. Calculate the time spread for 10 threads - Suppose T1 started earliest and T5 finished last, then the difference between T5 and T1 will give time spread.

This is a typical thread synchronization problem which can be solved using various available techniques in Java. We will discuss three main approaches to solve this problem - first one using a synchronized object, second one using non-blocking CAS, third using existing synchronizer CountDownLatch. Algorithm for the both is same - Two times will be recorded, first time for the thread which started earliest, and second time for the thread which finished last. The difference of the two times will give us time window.

Writing custom synchronizer to address this problem
We will write a custom synchronized class which records the first start time and last stop time.

```
public class TimeSpread2 {
   int threads;
   long startTime;
   long stopTime;
   boolean started = false;
   public TimeSpread2(int threads) {this.threads = threads;}
   public synchronized void start(){
      if(!started){
         started = true;
         startTime = System.currentTimeMillis();
      }
   }
   public synchronized void stop(){
      if(--threads<=0){
         stopTime = System.currentTimeMillis();
         notifyAll();
      }
   }
   public synchronized long timeSpread() throws InterruptedException {
      while(threads >0){      wait();   }
      return stopTime-startTime;
   }
   public static void main(String[] args) throws InterruptedException {
      int threads1 = 100;
      final TimeSpread2 timeSpread = new TimeSpread2(threads1);
      Runnable t = new Runnable(){
         public void run() {
            timeSpread.start();
            try {TimeUnit.SECONDS.sleep(5);} catch (InterruptedException e) {}
            timeSpread.stop();
         }
      };

      for(int i=0;i< threads1;i++){
         Thread thread = new Thread(t);thread.start();
      }
      System.out.println("time spread = " + timeSpread.timeSpread());
   }
}
```

continued on 162

Writing Non-Blocking version for same using Atomic package

This version does not require the calling thread to obtain lock on the Object, thus it could be slightly faster.

```
public class TimeSpread {
   final AtomicBoolean started = new AtomicBoolean(false);
   final AtomicInteger stopCounter;
   long startTime;
   long stopTime;
   public TimeSpread(int threads) {
      stopCounter = new AtomicInteger(threads);
   }
   public void start() {
      if (!started.get()) {
         if (started.compareAndSet(false, true)) {
            startTime = System.currentTimeMillis();
         }
      }
   }
   public void stop() {
      if(stopCounter.getAndDecrement()==1){
         stopTime = System.currentTimeMillis();
      }
   }
   public long timeConsumed(){
      return stopTime-startTime;
   }
   public static void main(String[] args) throws InterruptedException {
      int threads = 300;
      final TimeSpread timeSpread = new TimeSpread(threads);
      Runnable t = new Runnable(){
         public void run() {
            timeSpread.start();
            try {TimeUnit.SECONDS.sleep(5);} catch (InterruptedException e) {}
            timeSpread.stop();
         }
      };
      List<Thread> list= new ArrayList<>(threads);
      for(int i=0;i< threads;i++){
         Thread thread = new Thread(t);
         thread.start();
         list.add(thread);
      }
      for (Thread thread : list) {
         thread.join();
      }
      System.out.println("time spread = " + timeSpread.timeConsumed());
   }
```

Using existing Synchronizer - CountDownLatch for Calculating the time window
We can use two CountDownLatch (with permit 1 and 10) and countdown the first latch at the first line of run method, second latch just before the exit of run method.

Here is the sample Java code illustrating the same,[1]

1 Effective Java 2nd Edition : Item 69

continued on 163

```
public static long time(Executor executor, int concurrency, final Runnable action) throws InterruptedException {

    final CountDownLatch ready = new CountDownLatch(concurrency);
    final CountDownLatch start = new CountDownLatch(1);
    final CountDownLatch done = new CountDownLatch(concurrency);

    for (int i = 0; i < concurrency; i++) {
        executor.execute(new Runnable() {
            public void run() {
                ready.countDown(); // Tell timer we're ready
                try {
                    start.await(); // Wait till peers are ready
                    action.run();
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                } finally {
                    done.countDown();  // Tell timer we're done
                }
            }
        });
    }

    ready.await();    // Wait for all workers to be ready
    long startNanos = System.nanoTime();
    start.countDown(); // And they're off!
    done.await();     // Wait for all workers to finish
    return System.nanoTime() - startNanos;
}
```

Notes

We can evaluate any of these three approaches for our requirement and pick one. But definitely, using CountDownLatch seems a cleaner approach where the latch hides the boiler-plate code of the synchronization.

Q 167. What are fail-fast Iterator? what is fail safe?

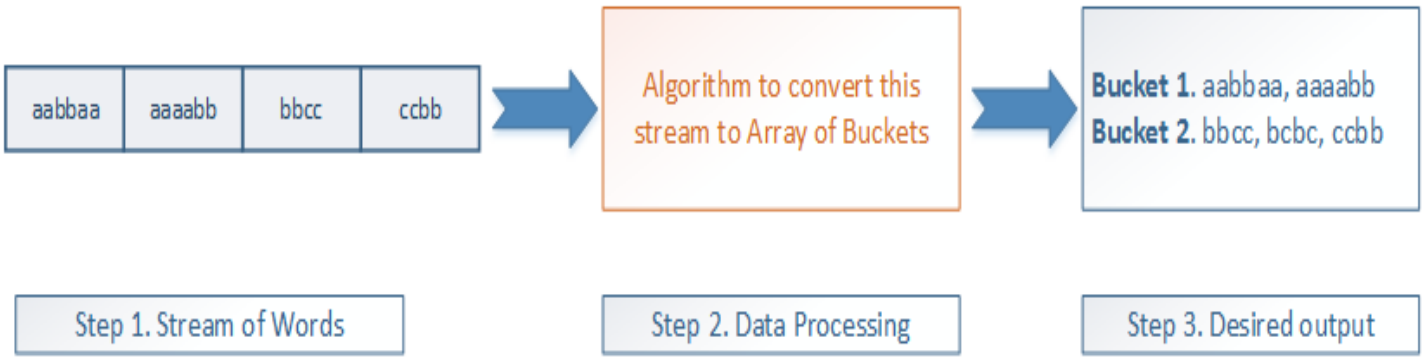
Fail-Fast Iterator (java.util package - HashMap, HashSet, TreeSet, etc)

Iterator fails as soon as it realizes that the structure of the underlying data structure has been modified since the iteration has begun. Structural changes means adding, removing any element from the collection, merely updating some value in the data structure does not count for the structural modifications. It is implemented by keeping a modification count and if iterating thread realizes the changes in modification count, it throws Concur rentModificationException. Most collections in package java.util are fail-fast by Design.

Fail-Safe Iterator (java.util.concurrent - ConcurrentSkipListSet, CopyOnWriteArrayList, ConcurrentMap)

Fail-safe Iterator is "Weakly Consistent" and does not throw any exception if collection is modified structurally during the iteration. Such Iterator may work on clone of collection instead of original collection - such as in CopyOnWriteArrayList. While ConcurrentHashMap's iterator returns the state of the hashtable at some point at or since the creation of iterator. Most collections under java.util.concurrent offer fail-safe Iterators to its users and that's by Design. Fail safe collections should be preferred while writing multi-threaded applications to avoid conurrency related issues. Fail Safe Iterator is guaranteed to list elements as they existed upon construction of Iterator, and may reflect any modifications subsequent to construction (without guarantee).

Q 168. There is a stream of words which contains Anagrams. How would you print anagrams in a single bucket from that stream?



What is Anagram?

A word or phrase spelled by rearranging the letters of another word or phrase, using all the original letters exactly once. For example, aabbaa is anagram of aaaabb, ababab and bbaaaa

How do we find an Anagram?

If we two words are equal after sorting, then they can be considered anagrams of each other. Sorting can take anywhere from O (n log n) to O (n^2) time complexity.

```
public boolean isAnagram(String s1, String s2){
    char[] a1 = s1.toCharArray();
    char[] a2 = s2.toCharArray();

    Arrays.sort(a1);
    Arrays.sort(a2);

    if (Arrays.toString(a1).equals(Arrays.toString(a2))){
        return true;
    }
    return false;
}
```

There is O(n) Time Complexity algorithm for finding the anagram as well, you just need to iterate through the first word and count the number of instances of each letter. Then iterate over the second word and do the same. Finally make sure that the letter frequency matches for both the words.

```
public class AnagramEfficient {

    public boolean isAnagram(String first, String second) {
        int letterCount[] = new int[126];

        for (char ch : first.toCharArray()) {
            letterCount[ch]++;
        }

        for (char ch : second.toCharArray()) {
            letterCount[ch]--;
        }
```

```
        for (int count : letterCount) {
            if (count != 0)
                return false;
        }
        return true;
    }
}
```

Please note that the above implementation will not work for Unicode Characters, you may need to increase the size of letterCount array in order to support characters outside ASCII range.

Unfortunately, this efficient algorithm can not be used with this requirement due to bucketing logic.

Steps to solve the given problem

1. Create a hashmap with string as key and list<string> to hold all anagrams of a given key string.
2. For each word in the input stream, create a key by sorting the word and append this word to that list whose key is the sorted word. For example [aakk -> akka, akak] If it does not exist then create a new list with the sorted word as key in map.
3. Iterate through hashmap keys and print corresponding values.

Source Code

```
import java.util.*;

public class Anagrams {
    private static Map<String, List<String>> anagramsMap = new HashMap<>(100);

    public static void main(String[] args) {
        String [] input = {"akka","akak","baab","baba","bbaa"};
        for (String s : input) {
            char[] word = s.toCharArray();
            Arrays.sort(word);
            String key = String.valueOf(word);
            if(!anagramsMap.containsKey(key)){
                anagramsMap.put(key, new ArrayList<String>());
            }
            anagramsMap.get(key).add(s);
        }
        System.out.println("anagramsMap = " + anagramsMap);
    }
}
```

Time Complexity

If we ignore the time consumed by sorting an individual string then we can say that the above approach takes Big O(n) time complexity. Otherwise the actual runtime complexity would be

O (n log n) for sorting + n (for hashmap comparison assuming zero collision)

Which is equivalent to O (n log n)

Q 169. Describe CopyOnWriteArrayList? Where is it used in Java Applications?

Functionality wise this collection is very much similar to ArrayList except the fact that CopyOnWriteArrayList is thread-safe. It maintains thread-safety using Immutability approach, where any modification operations results in creating a fresh copy of the underlying array.

This is ordinarily too costly, but may be more efficient than alternatives when traversal operations vastly outnumber mutations, and is useful when you cannot or don't want to synchronize traversals, yet need to preclude interference among concurrent threads.

The "snapshot" style iterator method uses a reference to the state of the array at the point that the iterator was created. This array never changes during the lifetime of the iterator, so interference is impossible and the iterator is guaranteed not to throw ConcurrentModificationException. The iterator will not reflect additions, removals, or changes to the list since the iterator was created. Element-changing operations on iterators themselves (remove, set, and add) are not supported. These methods throw UnsupportedOperationException.

This collection is used to write Scalable Concurrent Applications.

Q 170. There are M number of Threads who work on N number of shared synchronized resources. How would you make sure that deadlock does not happen?

If a single thread uses more than one protected shared resource, then we should make sure that we acquire shared resources in particular order and release them in **reverse** order, otherwise we might end up into a deadlock scenario.

Q 171. Are there concurrent version for TreeMap and TreeSet in Java Collections Framework?

Java Collection Framework have ConcurrentSkipListMap and ConcurrentSkipListSet which are concurrent replacement for TreeMap and TreeSet. These classes implement SortedMap and SortedSet interface respectively. So if our application demands fair concurrency then instead of wrapping TreeMap and TreeSet inside synchronized collection, we can prefer these concurrent utilities. These also implement NavigableMap and NavigableSet interface with methods like lowerKey, floorKey, ceilingKey, higherKey, headMap and tailMap.

Concurrent vs Synchronized version

Key Point to note here is that there is difference between synchronized version of TreeMap and Concurrent version of TreeMap (ConcurrentSkipListMap). Synchronized allows just one thread at a time i.e. access to the sharedobject is serialized, thus throughput will be low. But Insertion, removal, update, and access operations can be safely execute concurrently by multiple threads in case of ConcurrentSkipListMap. There is underlying difference in the implementation of ConcurrentSkipListMap to support this. Same is the case with synchronized and concurrent version of TreeSet. Thus care must be taken while designing scalable concurrent applications and preference should be given to concurrent versions.

Time Complexity

Average time complexity is log(n) for the containsKey, get, put, remove ad the variant operations of the ConcurrentSkipListMap

Q 172. Is it safe to iterate over an ArrayList and remove its elements at the same time? When do we get ConcurrentModificationException & hidden Iterator?

Iterator returned by the ArrayList (and many other collection types) is fail-fast. If the list is structurally modified at anytime after the iterator is created, in any way except through the Iterator's own remove() method, the iterator will throw ConcurrentModificationException and thus fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

Structural Modification

"A structural modification is any operation that adds or deletes one or more elements, or explicitly resizes the backing array; merely changing the values associated with a key that an instance already contains is not a structural modification." - Java Docs

Further, the structural modification could happen either from single thread or from multiple threads. The behavior of ArrayList would be different in both the cases as mentioned below.

Single Threaded Scenario

Never call list.remove(element) to remove a item from list while traversing it. Rather use Iterator.remove() mehtod.

```
private List<String> list = new ArrayList<>(asList("first", "second", "third", "fourth"));
public void unsafeMethod() {
      for (String item : list) {              // Will throw ConcurrentModificationException
         list.remove(item);
      }
}
public void safeMethod() {
      Iterator<String> iterator = list.iterator();
      while (iterator.hasNext()) {        // safe to call iterator.remove()
         String item = iterator.next();
         iterator.remove();
       }
}
```

Multi-Threading Scenario

ArrayList implementation is not thread-safe because it provides no synchronization mechanism for protecting the shared state of its fields. If multiple threads access an ArrayList instance concurrently, and at least one of the threads modifies the list structurally, it **must** be synchronized externally. This is typically accomplished by synchronizing on some object that naturally encapsulates the list.

If no such object exists, the list should be "wrapped" using the Collections.synchronizedList() method. This is best done at creation time, to prevent accidental unsynchronized access to the list :

```
List list = Collections.synchronizedList(new ArrayList(...));
public void safeMethod() {
synchronized(list) {
...rest of the code as shown in previous method
}
```

Hidden Iterators

There are certain ArrayList methods which uses Iterators in a hidden form the API user. size() and toString() are few of them. So care must be taken to call these methods from synchronized block in case of multi-threaded scenario.

Java 8 provides method to conditionally remove items from stream using a filter

```
List<String> names = new ArrayList(asList("munish","ravneesh", "rajneesh"));
 names.removeIf(name -> "munish".equalsIgnoreCase(name));
 System.out.println("names = " + names);
```

Though lambda makes removal quite compact, but the operation is not thread-safe and must not be used in multi-threaded environment without explicit synchronization in place.

Q 173. What is ThreadLocal class, how does it help writing multi-threading code? any usage with example?

ThreadLocal class provides a simple mechanism for thread safety by creating only one object instance per thread. These variables differ from their normal counterparts in that each thread that accesses one (via its get or set method) has its own, independently initialized copy of the variable. ThreadLocal instances are typically private static fields in classes that wish to associate state with a thread (e.g., a user ID or Transaction ID).

Each thread holds an implicit reference to its copy of a thread-local variable as long as the thread is alive and the ThreadLocal instance is accessible; after a thread goes away, all of its copies of thread-local instances are subject to garbage collection (unless other references to these copies exist)

ThreadLocal is used to achieve thread-safety in our multi-threaded code by creating a copy local to a thread and thus no more sharing of state.

When get() method is invoked the first time, ThreadLocal calls initialValue() and returns the newly created Object, the same Object is returned on subsequent invocations by the same thread untill we clear the Object.

ThreadLocal Usage Scenarios

- EntityManager in JPA - EntityManager instance is not thread safe but creating too many EntityManagers could be expensive, thus in a servlet environment ThreadLocal copy could be created using a servlet Filter and re-used the same EntityManager for the whole request life cycle, committing the changes in the end. In this case, when the thread calls get() for the first time, a new EntityManager instance is created and the same is re used on subsequent calls to get() method by the same thread.

```
public interface SessionCache {
   public EntityManager getUnderlyingEntityManager();
}

public class ThreadLocalSession implements SessionCache{
   @Override
   public EntityManager getUnderlyingEntityManager() {
      return threadLocalJPASession.get();
   }
   private static class ThreadLocalJPASession extends ThreadLocal<EntityManager>{
      @Override
      protected EntityManager initialValue() {
         return JPASessionFactory.getInstance().getSession();
      }
   }
   public static final ThreadLocalJPASession threadLocalJPASession = new ThreadLocalJPASession();
   public void set(EntityManager em){
      threadLocalJPASession.set(em);
   }
   public void clear() {
```

continued on 169

```
            threadLocalJPASession.get().close();
            threadLocalJPASession.remove();
        }
    }
```

- Using Calendar class in multi-threading environment : Calendar.getInstance() is not safe from multi-threading perspective and a copy of it could be created per thread and stored in ThreadLocal.
- Random Number Generator, ByteBuffers, XML parsers can utilize ThreadLocal for optimization purpose.

Three main Criteria for choosing ThreadLocal's applicability

- Object is non-trivial to construct
- Instance of object is frequently needed
- Application is multi-threaded

Notes

ThreadLocal instances are typically private static fields in classes that wish to associate state with a thread (e.g., a user ID or Transaction ID).

Each thread holds an implicit reference to its copy of a thread-local variable as long as the thread is alive and the ThreadLocal instance is accessible; after a thread goes away, all of its copies of thread-local instances are subject to garbage collection (unless other references to these copies exist).

Q 174. How would you implement your own Transaction Handler in Core Java, using the EntityManager created in last question?

Sometimes we do not want to use Spring Transaction API's and want to write our own (though very should never do that unless we are very good at it). In the last question we discussed how we can write a ThreadLocal EntityManager class. Now we will leverage the same class for writing our basic reusable transaction handler.

```
public interface Transatomatic{
    public<T> T run(UnitOfWork<T> unitOfWork);
    public static interface UnitOfWork <T>{
        public T run();
    }
}

public class JPATransatomatic implements Transatomatic {
    private final ThreadLocalSession threadLocalSession;

    public JPATransatomatic(ThreadLocalSession threadLocalSession){
        this.threadLocalSession = threadLocalSession;
    }

    @Override
    public<T> T run(UnitOfWork<T> unitOfWork) {
        final EntityManager em = threadLocalSession.getUnderlyingEntityManager();
        EntityTransaction tx = null;
        try {
            tx = em.getTransaction();
            tx.begin();
            T result = unitOfWork.run();
```

```
            tx.commit();
            return result;
        } finally {
            if (tx != null && tx.isActive()) {
                tx.rollback();
            }
            threadLocalSession.clear();
        }
    }
}
```

Now any client code who wants to run a database specific code inside a transaction can create instance of JPATransatomatic class, set appropriate ThreadLocalEntityManager and use it, as shown below

```
ThreadLocalSession threadLocalSession = new ThreadLocalSession();
JPATransatomatic  transatomatic = new JPATransatomatic(threadLocalSession);

public <T> T find(final Class<T> clazz, final long id) {
    return transatomatic.run(new Transatomatic.UnitOfWork<T>() {
        @Override
        public T run() {
            return catalogue.find(clazz, id);
        }
    });
}
```

Q 175. What is AtomicInteger class and how is it different than using volatile or synchronized in a concurrent environment?

Read & write to volatile variables have same memory semantics as that of acquiring and releasing a monitor using synchronized code block. So the visibility of volatile field is guaranteed by the JMM.

AtomicInteger class stores its value field in a volatile variable, thus it is a decorator over the traditional volatile variable, but it provides unique non-blocking mechanism for updating the value after requiring the hardware level support for CAS (compare and set).
Under low to moderate thread contention, atomic updates provides higher throughput compared to synchronized blocking increment operation.

Here is the implementation for getAndIncrement() method of AtomicInteger Class.

```
public final int getAndIncrement() {
    for (;;) {
        int current = get();
        int next = current + 1;
        if (compareAndSet(current, next))
            return current;
    }
}
```

You can see that no lock is acquired to increment the value, rather CAS is used inside infinite loop to update the new value.

Q 176. You are writing a server application which converts microsoft word documents into pdf format. Under the hood you are launching a binary executable which does the actual conversion of document. How would you restrict the parallel launch of such binaries to 5 in Java, so as to limit the total load on the server.

This is a typical problem of controlling the parallel access to the shared scarce resource so as to avoid the thread starvation.

JDK 1.5 provides a class specifically designed to address this kind of problem - Semaphore

Semaphore

Counting semaphores are used to control the number of activities that can access a certain resource or perform a given action at the same time, and it could be used for

- Resource pooling for e.g. Database connection pooling.
- To turn any collection into a blocking bounded collection.

Java Source For PDFConverter

PDFConverter that allows a given concurrencyLevel in multi-threaded environment i.e. at max <concurrencyLevel> number of documents can be converted in parallel.

```
import java.io.File;
import java.util.concurrent.Semaphore;

public class PDFConverter {
   private final Semaphore semaphore;

   public PDFConverter(int concurrencyLevel) {
      this.semaphore = new Semaphore(concurrencyLevel);
   }

   public void convertToPdf(File input, File output) throws InterruptedException {
      try {
         semaphore.acquire();    // Excess threads have to wait here till a permit becomes available
         // Convert the input file to PDF and then write it to the output file.
      } finally {
         semaphore.release();
      }
   }
}
```

Convert Multiple Documents to PDF using Java 8 parallelStream() API

```
PDFConverter pdfConverter = new PDFConverter(5);
      List<File> documents = new ArrayList<>();
     //   add documents to the list using  documents.add(doc1);
      documents.parallelStream().forEach(document -> {
         try {
            pdfConverter.convertToPdf(document, new File(document.getName() + ".pdf"));
         } catch (InterruptedException e) {
            e.printStackTrace();
```

Continued on 172

```
        }
    });
```

Using Executor Framework to achieve the same Task

```
ExecutorService executorService = Executors.newCachedThreadPool();
for (File document : documents) {
    executorService.submit(() -> {
        try {
            pdfConverter.convertToPdf(document, new File(document.getName() + ".pdf"));
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    });
}
executorService.shutdown();
executorService.awaitTermination(...);
```

Notes

Before Java 1.5, we had to write the semaphore functionality from scratch using synchronization (along with wait and notify for inter thread communication)
Semaphores can be used to convert a standard Java Collection into Bounded Collection after which the collection would hold only certain number of elements. Once the allowed elements are In, the thread has to wait till some other thread removes from that collection.

BoundedHashSet Example

```
import java.util.Collections;
import java.util.HashSet;
import java.util.Set;
import java.util.concurrent.Semaphore;

public class BoundedHashSet<T> {
    private final Set<T> set;
    private final Semaphore sem;

    public BoundedHashSet(int bound) {
        this.set = Collections.synchronizedSet(new HashSet<T>());
        sem = new Semaphore(bound);
    }

    public boolean add(T o) throws InterruptedException {
        sem.acquire();
        boolean wasAdded = false;
        try {
            wasAdded = set.add(o);
            return wasAdded;
        } finally {
            if (!wasAdded)
                sem.release();
        }
    }

    public boolean remove(Object o) {
        boolean wasRemoved = set.remove(o);
        if (wasRemoved)
```

```
            sem.release();
        return wasRemoved;
    }
}
```

Q 177. What are common threading issues faced by Java Developers?

Below is the list of issues that can occur in a multi-threading application.

1. Non-synchronized access to the fields of a shared mutable class.
2. Reusable objects used as a lock.
3. Check then act problems like double checked locking.
4. Invoking a blocking method with lock hold.
5. Visibility issues with the data because getters were not synchronized/published correctly.
6. Starting a thread from within the constructor causing partially initialized objects.
7. CPU starvation problems - A given task submits a new task to the common single threaded executor causing the deadlock in the program.
8. ConcurrentModificationException due to illegal structural modification to the collections.
9. Failed to handle exception from background thread.
10. Atomic and volatile variables.
11. Synchronized on immutable objects like String is useless and error prone.
12. Deadlock due to wrong resource locking order - resources must be locked in same order from all the threads and locks should be release in opposite order.

Chapter 4

Algorithms & Data Structures

Q 178. Given a collection of 1 million integers ranging from 1 to 9, how would you sort them in Big O(n) time?

This is a typical Integer Sorting problem with a constraint that the number range to sort is very limited in spite 1 million total entries. Integer Sorting with limited range is achieved efficiently with Bucket Sorting.

Algorithm

Create a array of size 9 and at each index store the occurrence count of the respective integers. Doing this will achieve this sorting with time complexity of Big O(n) and even the memory requirements are minimized. In Order to print the output just traverse the above created array.

Source Class

```
public class BucketSort {
    public int[] sort(int[] array, int min, int max) {
        int range = max - min +1;
        int[] result = new int[range];
        for (int i : array) {
            result[i]++;
        }
        return result;
    }
}
```

Test Class

```
public class BucketSortTest {
    @Test
    public void testBucketSortFor1To9() {
        int[] array = {2, 1, 5, 1, 2, 3, 4, 3, 5, 6, 7, 8, 5, 6, 7, 0};
        int[] sort = new BucketSort().sort(array, 0, 8);
        for (int i = 0; i < sort.length; i++) {
            for(int j=0;j<sort[i];j++){
                System.out.println(i);
            }
        } } }
```

Program output : 0,1,1,2,2,3,3,4,5,5,5,6,6,7,7,8

Notes

Bloom Filter[1] could help us achieve something similar.

Bucket sort, counting sort, radix sort, and van Emde Boas tree sorting all work best when the key size is small; for large enough keys, they become slower than comparison sorting algorithms…
Integer Sorting Techniques : http://en.wikipedia.org/wiki/Integer_sorting#Algorithms_for_few_items
Sorting Algorithms : http://en.wikipedia.org/wiki/Sorting_algorithm

1 http://en.wikipedia.org/wiki/Bloom%5Ffilter

Q 179. Given 1 million trades objects, you need to write a method that searches if the specified trade is contained in the collection or not. Which collection would you choose for storing these 1 million trades and why?

HashSet is a good choice for storing this collection because it will offer Big O(1) time complexity. In order to use HashSet we must override equals() and hashcode() method for the Trade Object. If that's not possible then we should created a Trade Wrapper class which overrides these methods.

```
public class Trade{
...
@Override
public boolean equals(Object o) {...}
@Override
public int hashCode() {...}
}
```

Q 180. I have an Integer array where every number appears even number of time except one. Find that number.

Approach

This problem can be solved by utilizing bitwise operators in O(1) space and O(n) time complexity.
XOR all the number together and the final result would the odd number.

How does XOR works?

Here is the complete solution using XORing

```
public class OddNumberProblem {
    private int[] array = {1,1,2,3,4,5,2,3,4};
    public int findSingleOdd(){
        int result =0;
        for (int i : array) {
            result=result^i;
        }
        return result;
    }

    public static void main(String[] args) {
        OddNumberProblem test = new OddNumberProblem();
        int singleOdd = test.findSingleOdd();
        System.out.println("singleOdd = " + singleOdd);
    }
}
```

Output:

```
singleOdd = 5
```

Q 181. how would you check if a number is even or odd using bit wise operator in Java?

Least significant bit (rightmost) can be used to check if the number is even or odd.
For all Odd numbers, rightmost bit is always 1 in binary representation.

```
public static boolean checkOdd(long number){
    return ((number & 0x1) == 1);
}
```

Notes

We prefer bitwise operator for checking even odd because the traditional way of checking even by n % 2 ==0 is compassionately expensive compared to bitwise & operator (Big O(1) time complexity)

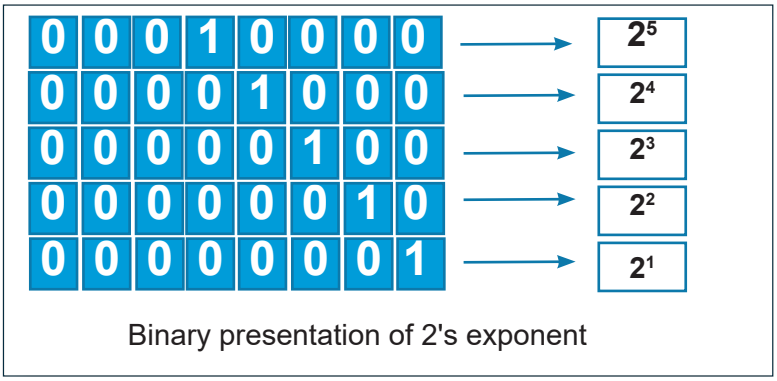
Q 182. How would you check if the given number is power of 2?

This can be easily checked using bitwise operators in Java. Firstly let's see how a number looks in binary when it is power of two.

From the figure, we can see that only 1 bit is set for all numbers which are exponent of 2.

Let's now write a method to check if only nth bit of a number is set to true.

```
int isPowerOfTwo (int x)
{
  return ((x != 0) && !(x & (x - 1)));
}
```



Binary presentation of 2's exponent

The first condition (x!=0) checks if the number is positive, because the second conition works only for positive numbers.

The second condition (x & (x-1)) will return zero for a number which is exponent of two (provided the number is positive). For example, in case of number 32,

```
        00010000 (2^5)
&       00001111  (2^5-1)
--------------------------------
        00000000 (0)
```

Thus through the above code, we are checking if the number is positive and is power of two.

Q 183. What is a PriorityQueue? How is it implemented in Java? What are its uses?

What is a PriorityQueue?

It is an abstract data type similar to queue but the elements are stored in a sorted order according to some priority associated with them, and the element with the higher priority is served before the element with lower priority. Priority is decided using the Comparator provided at the time of its construction. If no comparator is provided, then the natural ordering of elements is used to prioritize them.

For example, if all elements are of type Integer and no comparator is provided, then the natural order is used resulting in highest priority to the smallest Integer value.

Implementation - Binary Heap

Binary Heap[1] data structure is used as the underlying for implementing a PriorityQueue in Java. A Binary Min-Heap is a complete binary tree such that -

- Each node is less than or equal to each of its children according to the comparison predicate (Comparator) provided at the time of construction.
- The tree is a complete binary tree - all levels of the tree are full except, at the level farthest from root. And if the last level of the tree is not complete, then the nodes of that level are filled starting from left.

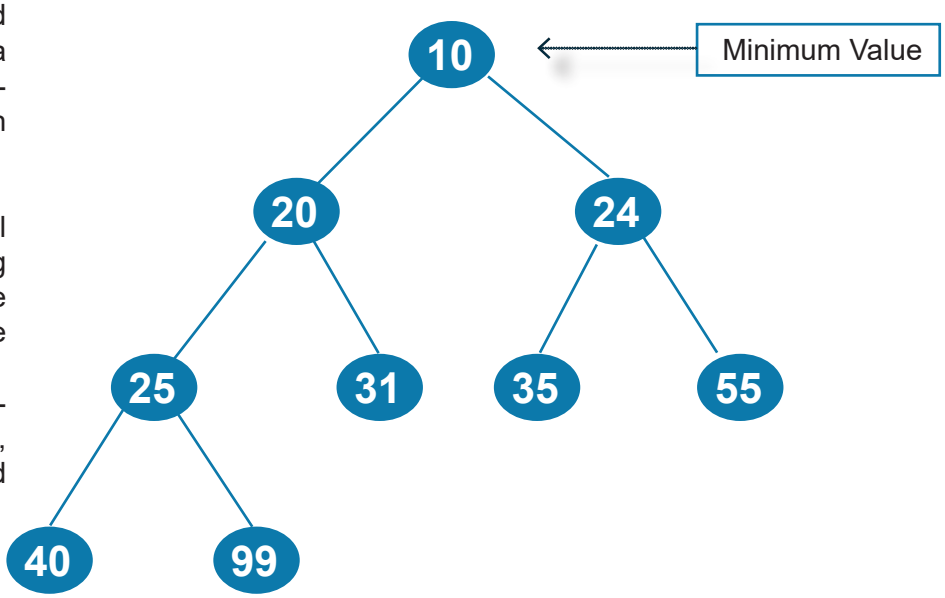


Figure : **Binary Min-Heap (minimum on top)**

A complete binary trees can be implemented with an random access array, which makes the implementation fast for retrieval.

Given the index of an element, element's children can be accessed in constant time using an random access array. Children of the element at index i are at indexes (i << 1)+1 and (i << 1)+2. And the parent of an element at index j is at (j-1) >>1

How is it different from Binary Search Tree?

Please note that a Binary heap is not a binary search tree.

- The ordering in binary heap is top to bottom compared to left to right in case of binary search tree.
- Duplicate elements are allowed in Binary Heap which is not the case with Binary Search Tree (a duplicate key are overwritten by the new key).
- A binary heap is a complete binary tree which may not be true for a Binary Search Tree.

Time Complexity for PriorityQueue

- Big O(1) time for retrieval methods - peek(), element() and size().
- Big O(log n) time for enqueing and dequeing method - offer, poll, remove() and add.
- Big O(n) time for remove(Object) and contains(Object) methods.

1 http://en.wikipedia.org/wiki/Binary_heap

PriorityQueue Usages

- A network printer where multiple people submit print jobs at the same time, While one big print job is executing, PriorityQueue could re-arrange other jobs so that the small print jobs (with very less number of pages) execute on priority compared to big ones.
- Emergency department in any hospital handles patients based on their severity, thus priority queue could used to implement such logic.

Notes

Binary Heap can be used for solving algorithmic problems, like the following -

- Finding top 10 most frequently used words in a very large file in O(n)
- Finding top 1 million numbers from a given large file containing 5 billion numbers in O(n)
- You have a file with 1 trillion numbers, but memory can hold only 1 million, How would you find the lowest 1 million out of all these numbers?
  Hint - Create a binary-max-heap with 1 million capacity, so the largest number will be at the top. Now go over each number in the file, compare it with the peek(), if it is smaller then poll() the element and add the new one. The total complexity should be less than O (n log n). Selection Rank algorithm could also be used to solve this problem, provided there exists no duplicate number.

Q 184. What is difference between Collections.sort() and Arrays.sort()? Which one is better in terms of time efficiency?

Collections.sort() internally calls Arrays.sort() and thus the underlying algorithm for both of these methods is same. The only difference is the type of input these methods accept.
Merge Sort algorithm is used by Arrays.sort() method as of JDK 6.

Q 185. There are 1 billion cell-phone numbers each having 10 digits, all of them stored randomly in a file. How would you check if there exists any duplicate? Only 10 MB RAM is available to the system.

Approach 1

Hash all these numbers into 1000 files using hash(num)%1000, then the duplicates should fall into the same file. Each file will contain 1 million numbers roughly, then for each file use HashSet to check for the duplicates.

(If sufficient memory is available)

Approach 2

Use BitSet to represent those 1 billion numbers and then traverse the file and set the appropriate Bit in the BitSet. But check the Bit value before setting it, thus listing the duplicate.

Approach 3

Use bucket Sort to partition the numbers based on some common prefix. Then the duplicate numbers should fall under the same bucket.

Approach 4

Build a TRIE from this huge file (This will load every thing into memory) and then search the number before putting it into TRIE.

Some similar questions -

http://stackoverflow.com/questions/7703049/check-1-billion-cell-phone-numbers-for-duplicates
http://stackoverflow.com/questions/7153659/find-an-integer-not-among-four-billion-given-ones

Q 186. What is a Binary Search Tree? Does Java provide implementation for BST? How do you do in-order, pre-order and post-order Traversal of its elements?

A Binary Search Tree[1]

(also known as sorted binary tree) is a node based binary tree data structure which has the following properties,

- All elements in the left subtree are less than the root element.
- All elements in the right subtree are greater than the root element.
- Both, left and right subtree must also be binary search trees.
- There can not be any duplicate element in the entire tree.

TreeMap in Java 6

Java provides its Binary Search Tree implementation in TreeMap class. TreeMap is special kind of BST which is height balanced and known as red-black-tree.

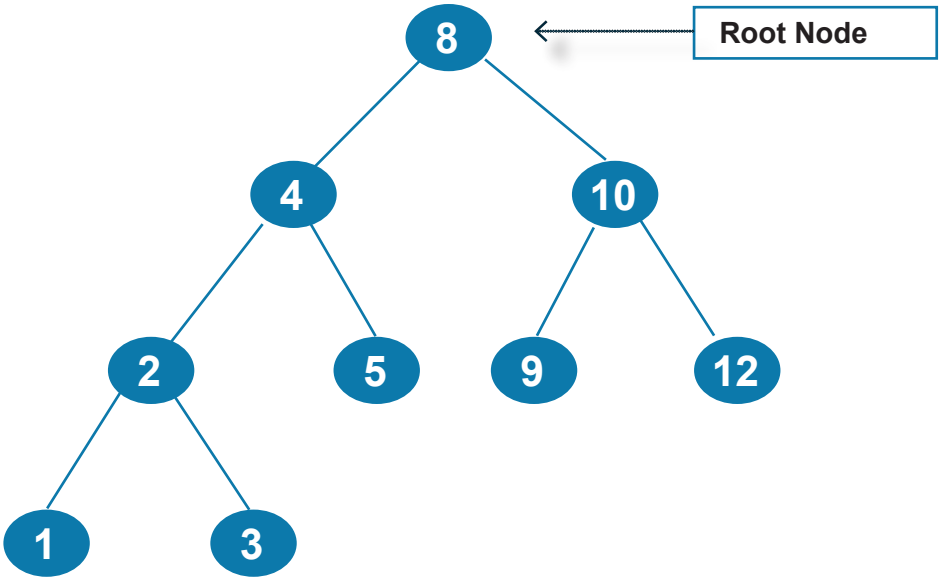


Figure : **Binary Search Tree Example**

Tree Traversal

There are three types of depth-first traversal, namely pre-order, in-order and post-order.

Pre-order Traversal

- Visit the root node
- Traverse the left subtree
- Traverse the right subtree

In-order Traversal

- Traverse the left subtree
- Visit the root node
- Traverse the right subtree

Post-order Traversal

- Traverse the left subtree
- Traverse the right subtree
- Visit the node

Pseudo Code for Traversal

Given a Node with definition, Node{data, Node left, Node right}

Pre-order

```
preOrder(node){
    if(node == null)
        return
                visit(node)
    preOrder(node.left)
    preOrder(node.right)
}
```

In-order

1 http://en.wikipedia.org/wiki/Binary_search_tree

```
inOrder(node){
    if(node==null)
        return;
    inOrder(node.left)
    visit(node)
    inOrder(node.right)
}
```

Post-order

```
postOrder(node){
    if(node==null)
        return;
    postOrder(node.left)
    postOrder(node.right)
    visit(node)
}
```

Q 187. What is technique to sort data that is too large to bring into memory?

To sort data that is residing on secondary storage (disc, tape, etc) rather than in main memory (primary storage), we use a sorting technique that is called *external sort.* There could be two different scenario where data can not fit into main memory -

- Items to be sorted are themselves too large to fit into main memory (files, images, audio, video, etc), but there are not many items. In this case we can only sort the keys and a value indicating the location of data on disc. After the key-value pairs are sorted as per required criteria, the data is rearranged on disc into correct order.
- Items to be sorted are too many to fit into main memory at one time, but the items themselves are small enough to fit into memory (age, employee data, dates, numbers, strings, etc). In this case the data can be divided in to partitions that fit into main memory, and then resulting files can be merged into single file. Merge Sort or Radix sort can be used as external sorting technique in this case.

Q 188. Check if a binary tree is a Binary Search Tree or not?

Source Code

```
public boolean isValid(Node root) {
   return isValidBST(root, Integer.MIN_VALUE,
         Integer.MAX_VALUE);
}
private boolean isValidBST(Node node, int MIN, int MAX) {
   if(node == null)
      return true;
   if(node.value > MIN
         && node.value < MAX
         && isValidBST(node.left, MIN, node.value)
         && isValidBST(node.right, node.value, MAX))
      return true;
   else
      return false;
}
```

The recursive call makes sure that subtree nodes are within the range of its ancestors. The time complexity will be O(n) since every node will be examined once.

Further reading - http://cslibrary.stanford.edu/110/BinaryTrees.html#java

Q 189. How would you convert a sorted integer array to height balanced Binary Search Tree?

Input: Array {1, 2, 3}
Output: A Balanced BST

```
    2
  /  \
 1    3
```

Algorithm

1. Get the middle element of the sorted array (start+(end-start)/2) and put this element at the root
2. Recursively repeat the same for the for the left and right child
   i. Get the middle of the left half and make it left child of the root created in step 1.
   ii. Get the middle of right half and make it right child of the root created in step 1.

Time Complexity : Big O(n)

Java Source

```
public class SortedArrayToBST {
   static class Node {
      Node left;
      Node right;
      int data;
   }

   Node sortedArrayToBST(int arr[], int n) {
      return sortedArrayToBST(arr, 0, n - 1);
   }

   Node sortedArrayToBST(int arr[], int start, int end) {
      if (start > end) return null;
      int mid = start + (end - start) / 2; // same as (start+end)/2, but it avoids overflow.
      Node node = new Node();
      node.data = arr[mid];
      node.left = sortedArrayToBST(arr, start, mid - 1);
      node.right = sortedArrayToBST(arr, mid + 1, end);
      return node;
   }

   public static void main(String[] args) {
      int[] sortedArray = {10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20};
      SortedArrayToBST test = new SortedArrayToBST();
      Node node = test.sortedArrayToBST(sortedArray, 11);
      //printTree(node);
   }
}
```

Similar questions on the web -

http://cslibrary.stanford.edu/110/BinaryTrees.html#java
http://www.geeksforgeeks.org/sorted-array-to-balanced-bst/
http://leetcode.com/2010/11/convert-sorted-array-into-balanced.html

Q 190. How would you calculate depth of a binary tree?

Height of a binary tree can be easily calculated using the following recursive algorithm.

Algorithm

1. Base condition - if nod is null then return 0.
2. Calculate height of left sub tree, height of right sub tree, and then return the height of the tree as the max of two heights+1.

```
static class Node {
   Node left;
   Node right;
   int data;
}

public static int height(Node node){
   if(node == null){
      return 0;
   }
   int hLeft = height(node.left);
   int hRight = height(node.right);
   int height = 1+Math.max(hLeft, hRight);
   return height;
}
```

Q 191. Calculate factorial using recursive method.

General formula for calculating factorial of a number is:

factorial(n) = n x factorial(n-1)

Its easy to convert the same to a recursive formula.

Recursive Algorithm

1. Base Condition - factorial of 1 is equal to 1.
2. Recursive Condition - factorial of n = n*factorial(n-1).

```
public int factorial(int n) {
   if (n == 1) {
      return n;
   } else {
      return n * factorial(n - 1);
   }
}
```

Be noted that integer variable can hold maximum value of 2^{31}, which will overflow once number is large. To fix that issue, you can use BigInteger class that can hold relatively large numbers.

Q 192. You have a mixed pile of N nuts and N bolts and need to quickly find the corresponding pairs of nuts and bolts. Each nut matches exactly one bolt, and each bolt matches exactly one nut. By fitting a nut and bolt together, you can see which is bigger. But it is not possible to directly compare two nuts or two bolts. Given an efficient method for solving the problem.

This can be solved quickly using a customized Quick Sort algorithm.

A simple modification of Quicksort shows that there are randomized algorithms whose expected number of comparisons (and running time) are O(n log n).

Approach

Pick a random bolt and compare it to all the nuts, find its matching nut and compare it to all the bolts (they match, the nut is larger, the nut is small). This will divide the problem into two problems, one consisting of nuts and bolts smaller than the matched pair and the other consisting of larger pairs. Repeat and divide this until all the nuts and bolts are matched. This is very similar to quick sort in action and achieving the result in O(n log n) time.

References

http://www.wisdom.weizmann.ac.il/~naor/PUZZLES/nuts_solution.html
http://algs4.cs.princeton.edu/23quicksort/

Q 193. Your are give a file with 1 million numbers in it. How would you find the 20 biggest numbers out of this file?

There are multiple ways to solve this problem.

Sorting Approach (Slow on time)

Sort all the numbers and pick the first 20. If we use Merge Sort Algorithm to sort elements (as used by Java 6 Collections.sort(List<T> list)) then this will take at least O(n log n) time complexity and O(n) memory, where N is 1 million in this case.

Max Heap (PriorityQueue in Java) Approach (Preferred)

Create a Max heap with size of 20. Now traverse all the elements from the source file using a stream and push it to heap if the minimum element in the heap is less than the current element.
This will take require a constant size of O(k) where k = 20
Complexity of inserting an item to a PriorityQueue (Heap based) is O(log N) where N =20 in this case. Hence the overall Time Complexity for this approach would be O(n log k) where N = 1 million and K=20.

Selection Rank Algorithm

It is an algorithm to find the ***k***th smallest (or largest) element from a given array in worst case linear time.
This algorithm can be used to find the 20th largest element and then traverse the entire file and compare if the given number is greater than the 20th largest number, if yes then print it.

For more details about Selection Algorithm, please refer to below link
http://en.wikipedia.org/wiki/Selection_algorithm

Q 194. Reverse the bits of a number and check if the number is palindrome or not?

Answer : int numBitsReversed = Integer.reverse(num);

then XOR the number with the reversed number if zero then palindrome

see also Integer.reverseByte(num),

Reversing bits of number in java

```
while(x!=0){
b<<1;
b|=(x&1);
x>>1;
}
```

Q 195. How would you mirror a Binary Tree?

Mirroring will change any given tree into its mirror image. For example,
This tree...

```
      7
     / \
    4   9
   / \
  1  3
```

Will be changed to...

```
      7
     / \
    9   4
       /  \
      3   1
```

Java Source[1]

```
    private void mirror(Node node) {
       if (node != null) {
          // do the sub-trees
          mirror(node.left);
          mirror(node.right);

          // swap the left/right pointers
          Node temp = node.left;
          node.left = node.right;
          node.right = temp;
       }
    }
```

1 http://cslibrary.stanford.edu/110/BinaryTrees.html#java

Q 196. How to calculate exponentiation of a number using squaring for performance reason?

Exponentiation is a mathematical operation, written as b^n, involving two numbers, the base b and the exponent n. When n is a positive integer, exponentiation corresponds to repeated multiplication of the base: that is, b^n is the product of multiplying n bases i.e.

$b^n = b \times b \times ... \times b$ {b multiplied n times}

The following mathematical expression can help us writing a method similar to Math.pow(x, n) using squaring as a technique for achieving exponentiation. This technique greatly reduces the time complexity to O (log n) compared to normal way of multiplying the number n times.

x^n	1 if n = 0;
	$1/x^{-n}$ if n < 0;
	$x.(x^{n-1/2})^2$, if n is odd
	$(x^{n/2})^2$, if n is even

Simple recursive Algorithm can be written based on above expressions.

```
public class Maths {
    static long exponentiationBySquaring(int base, int exponent) {
        if (exponent == 0) //when exponent is 0, result is 1 i.e. 2^0 = 1
            return 1;
        if (exponent == 1) { //when exponent is 1, result is base, i.e. 2^1 = 2
            return base;
        } else if (exponent % 2 == 0) { // when exponent is even - square the number and reduce exponent by half i.e.
2^4 = (2*2)^2
            return exponentiationBySquaring(base * base, exponent / 2);
        } else { // when exponent is odd - square the number, reduce power by 2 and multiply once i.e. 2^5 = 2*((2*2)^2)
            return base * exponentiationBySquaring(base * base, (exponent - 1) / 2);
        }
    }
    public static void main(String[] args) {
        int base = 2;
        System.out.println("Calculating exponent for base - " + base);
        for (int i = 0; i < 10; i++) {
            long result = exponentiationBySquaring(base, i);
            System.out.println("2^" + i + " = " + result);
        }
    }
}
```

Program Output

```
Calculating exponent for base - 2
2^0 = 1
2^1 = 2
2^2 = 4
2^3 = 8
```

```
2^4 = 16
2^5 = 32
2^6 = 64
2^7 = 128
2^8 = 256
2^9 = 512
```

Notes

Time Complexity of this algorithm is O (log n) where n is the exponentiation. The number of multiplications reduces to half on each iteration.

Another method for calculating the pow

Other way to write the same recursive algorithm,

```
int power(int x, int y) {
   if (y == 0)
      return 1;
   else if (y % 2 == 0)
      return power(x, y / 2) * power(x, y / 2);
   else
      return x * power(x, y / 2) * power(x, y / 2);
}
void test() {
   int x = 2;
   int y = 3;
   System.out.println(power(10, 3));
}
```

But the time complexity in this case would be O (n) and space complexity O (1), above method can be further optimized to O (log n) by calculating the power(x , y/2) only once and storing it, also taking care of negative y values as well -

```
int power(int x, int y) {
   if (y == 0)
      return 1;
   else {
      int temp = power(x, y / 2);
      if (y % 2 == 0)
         return temp * temp;
      else {
         return y > 0 ? x * temp * temp : (temp * temp) / x;
      }
   }
}
```

Q 197. How will you implement a Queue from scratch in Java?

Given a class Node, we can easily construct a type safe (Generic) queue which provides two methods - enqueue and dequeue. Internally this queue keeps two references - first one for the Head of the queue and second one for the Tail of the queue. When we add new item to the queue, its added to the head, and when we remove an element then its removed from the Tail (**F**irst **I**n **F**irst **O**ut).

Below is the implementation of the same.

```
public class QueueUsingNode<T> {
   static class Node<T> {
```

```
        final T data;
        Node<T> next;

        Node(T data) {this.data = data;}
    }

    Node<T> first, last;

    void enqueue(T item) {
        if (first == null) {
            last = new Node(item);
            first = last;
        } else {
            last.next = new Node<T>(item);
            last = last.next;
        }
    }

    T dequeue(){
        if(first!=null){
            T item = first.data;
            first = first.next;
            return item;
        }
        return null;
    }
    public static void main(String[] args) {
        QueueUsingNode<Integer> test = new QueueUsingNode<>();
        test.enqueue(100);
        System.out.println("test = " + test.dequeue());
    }
}
```

Q 198. How will you Implement a Stack using the Queue?

Here is the Generic implementation of Stack using linked Nodes. Nodes are linked like in a LinkedList and the push and pop operations happen on the head of the Linked Nodes.

```
public class StackUsingNode<T> {
    static class Node<T> {
        final T data;
        Node<T> next;

        Node(T data) {this.data = data;}
    }

    Node<T> top;

    T pop() {
        if (top != null) {
            T item = top.data;
            top = top.next;
            return item;
        }
        return null;
```

```
    }

    void push(T item) {
        Node<T> t = new Node<T>(item);
        t.next = top;
        top = t;
    }

    T peek() {
        return top.data;
    }

    public static void main(String[] args) {
        StackUsingNode<Integer> stack = new StackUsingNode<>();
        stack.push(100);
        stack.push(200);
        System.out.println("stack = " + stack.pop());
    }
}
```

Q 199. How will you test if a given sentence is a Pangram or not?

Pangram is a sentence containing every letter of the alphabet (a-z).

The below program can be used to test if given input is pangram or not. this method utilizes bitwise OR and left shift operator to set nth bit in an integer. Since integer has 32 bits, so its sufficient to store and check if all alphabets (26 in number) are present in a given sentence.

```
public class Pangram {

    public boolean checkPangramEfficient(String input) {
        int i = 0;
        for (char c : input.toCharArray()) {
            int x = Character.toUpperCase(c);
            if (x >= 'A' && x <= 'Z') {
                i |= 1 << (x - 'A');
            }
        }
        if (i == (1 << (1 + 'Z' - 'A')) - 1) {
            System.out.println("input is a pangram");
            return true;
        } else {
            System.out.println("input is not pangram");
            return false;
        }
    }
}

public class PangramTest {
    private Pangram pangramChecker = new Pangram();

      @Test
      public void testPangramEfficient() {
          String input = "The quick brown fox jumps over the lazy dog";
          assertTrue(pangramChecker.checkPangramEfficient(input));
      }
}
```

Q 200. How would you implement a simple Math.random() method for a given range say (1-16)?

Generating random numbers is not that easy because there are lots of expectations from a perfect random number generator (fair distribution, randomness, fast, etc). The scope of this question is just to write a simple function without worrying about the fairness, speed.
In order to generate a random number, we would require a seed which provides us with the randomness. System.currentTimeMillis() could be a good substitute for providing seed value in our case.

```
public class RandomGenerator {
    public long generate(){
        return System.currentTimeMillis() % 16;
    }

    public static void main(String[] args) {
        RandomGenerator test = new RandomGenerator();
        long generate = test.generate();
        System.out.println("generate = " + generate);
    }
}
```

The value returned by the System.currTimeMillis() is very large and we need to make it fit to our bounds using the modulus operator (x % n) which will bound the upper value to be less than n.

Q 201. How an elevator decides priority of a given request. Suppose you are in an elevator at 5th floor and one person presses 7th floor and then 2nd presses 8th floor. which data structure will be helpful to prioritize the requests?

Generally elevator's software maintains two different queues - one for the upward traversal and another for downward traversal along with the direction flag which holds the current direction of movement. At any given point in time, only single queue is active for the serving the requests, though both the queues can enqueue requests.

PriorityQueue is used for storing the user requests where priority is decided based on following algorithm.

For upward movement PriorityQueue
The floor number with lower value has the higher priority.

For downward movement PriorityQueue
The floor number with higher value has the higher priority.

Requests are removed from the PiorityQueue as soon as they are served. If current floor is 5th and user presses 4th floor with upward moving elevator, then the requests are queued to the downward movement priority queue (which is not yet active)

Q 202. How would you multiply a number with 7 using bitwise hacks?

This can be achieved by multiplying the number with 8 and then subtracting the number from the result.

Multiplying a number with 8 using bit shift operators

If we left shift bits of a number by 2^3 then it would be equivalent to multiplying the number by 8.

```
public class MultiplyBy7 {
    public static void main(String[] args) {
        System.out.println(multiplyBy7(8));
    }

    private static int multiplyBy7(int number) {
        // multiply by 8 using bitwise left shift operator (2^3 = 8)
        int result =number << 3;
        result =  result - number;
        return result;
    }
}
```

Q 203. What is the best way to search an element from a sorted Integer Array? What would be it's time complexity?

Binary search is best when we want to search from within a sorted collection.
It narrows down the search area to half in each iteration and thus very time efficient.

Binary Search Algorithm

```
int low = 0;
int high = list.size()-1;

while (low <= high) {
    int mid = (low + high) >>> 1;
    Comparable<? super T> midVal = list.get(mid);
    int cmp = midVal.compareTo(key);

    if (cmp < 0)
        low = mid + 1;
    else if (cmp > 0)
        high = mid - 1;
    else
        return mid; // key found
}
return -(low + 1);  // key not found
```

Worst case Time Complexity for Binary Search is Big O (log n)

Q 204. How would you reverse a Singly linked List?

Using Stack to reverse the Linked List[1]

Push all the items to the stack and then re-construct the Linked List by popping the elements from the stack.

Using Recursion to Reverse the Linked List[2]

Breaking down problem statement into recursive method call

1. Base case : if the list is empty or of size one then return list
2. Recursive condition : If the list has n elements, then divide list into two parts - 1st element and rest of the elements. Reverse the position of these two and return.

```
public static Node reverse(Node node) {
        Node firstNode = node;
        if (firstNode.next == null || firstNode == null) {  // Base condition
            return firstNode;
        } else {                                    // Recursive condition
            Node secondNode = firstNode.next;
            firstNode.next = null;
            Node reverseNode = reverse(secondNode);
            secondNode.next = firstNode;
            return reverseNode;
        }
    }
```

Using Iterative approach to reverse a Linked List

Set current Node to the first node of the List
Set its previous node to null
Set next node to the 2nd node
Repeat the process for rest of the List Items.

```
public static Node reverseIterative(Node node) {
        Node prevNode = null;
        Node currNode = node;
        Node nextNode;
        while (currNode != null) {
            nextNode = currNode.next;
            currNode.next = prevNode;
            prevNode = currNode;
            currNode = nextNode;
        }
        return prevNode;
    }
```

Similar questions on the web

http://goodinterviewquestions.blogspot.in/2012/03/reverse-singly-linked-list.html
http://www.technicalypto.com/2010/01/java-program-to-reverse-singly-linked.html
http://stackoverflow.com/questions/12666178/reverse-singly-linked-list-java-check-whether-circular
http://crackinterviewtoday.wordpress.com/2010/03/24/reverse-a-single-linked-list-iterative-procedure/

1 http://datastructuresblog.wordpress.com/2007/03/30/reversing-a-single-linked-list-using-stack/
2 http://crackinterviewtoday.wordpress.com/2010/03/24/reverse-a-single-linked-list-recursive-procedure/

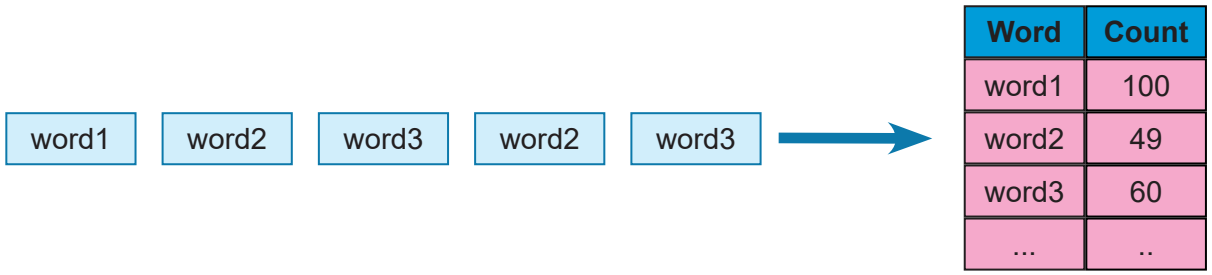
Q 205. How would you count word occurrence in a very large file? How to keep track of top 10 occurring words?

Questions worth asking - can file fit into main memory?, how many distinct words are there in the file?
Please note that there are limited number of natural language words available and all of them can be easily fit into today's computer RAM. For example oxford English dictionary contains total of around 0.6 million words.

We will discuss two approaches to solve this problem -
Approach 1 : Find the Top K Occurrence count using a hashmap and min-heap (PriorityQueue in Java)

Pseudocode for the algorithm

1. Finding the Word Occurrence Count - Stream the words into a HashMap (put operation is Big O(1)) keeping the value as word occurrence count. On every word occurrence, update the word count.
2. Track Top K occurring Words Using Binary Min Heap (PriorityQueue with Natural ordering) - This can be



achieved by maintaining a binary min heap of max size K, and then for each word count in hashmap -
   i. Check if the heap size if less than K - then add the new word count to min heap. Otherwise
   ii. Check if the peek element (that is minimum value in binary min heap) is less than the new word count, and if it is, then remove the existing number and insert the new word count into min heap.
   iii. When we are done traversing the entire word-counts then we will have heap containing the top K frequently occurring words.

Java 8 Source Code (Find Top 10 word occurrences)

```
import java.util.Arrays;
import java.util.Comparator;
import java.util.PriorityQueue;
import java.util.stream.Collectors;

public class SplitWordCount {
   public static void main(String[] args) {
      List<String> terms = Arrays.asList(
            "Coding is great",
            "Search Engine are great",
            "Google is a nice search engine",
            "Bing is also a nice engine");
      TopOccurrence topOccurrence = new TopOccurrence(2);
      terms.parallelStream()  //Utilizes multi-core hardware
            .flatMap(s -> Arrays.asList(s.split(" ")).stream())
            .collect(Collectors.toConcurrentMap(w -> w.toLowerCase(), w -> 1, Integer::sum)) // Big O(n)
            .forEach((s, integer) -> topOccurrence.add(new WordCount(s, integer)));
      System.out.println(topOccurrence);
   }

   static class TopOccurrence {
      private final PriorityQueue<WordCount> minHeap;
```

```
        private final int maxSize;
        public TopOccurrence(int maxSize) {
            this.maxSize = maxSize;
            this.minHeap = new PriorityQueue<WordCount>(Comparator.comparingInt(wc -> wc.count));
    //        This constructs a min heap (when order of elements is natural i.e. ascending order).
    //        We are using Natural order for integers (wc.count)
    //        In order to create a max-heap, we just need to provide reversed comparator i.e. that sorts in descending order,
    as shown below
    //        this.minHeap = new PriorityQueue<WordCount>(Comparator.comparingInt((WordCount wc) -> wc.count).
    reversed());
        }
        public void add(WordCount data) {
            if (minHeap.size() < maxSize) { // size() is Big O(1)
                minHeap.offer(data); // Big O(log(k)) where k is the number of top occurrences required
            } else if (minHeap.peek().count < data.count) { // peek() is Big O(1)
                minHeap.poll(); // Big O(log(k))
                minHeap.offer(data); // Big O(log(k))
            }
        }

        @Override
        public String toString() {
            return "TopOccurrence{" + "minHeap=" + minHeap +  ", maxSize=" + maxSize + '}';
        }
    }

    static class WordCount {
        protected final String word;
        protected final int count;
        WordCount(String word, int count) {
            this.word = word;
            this.count = count;
        }
        @Override
        public String toString() {
            return "{" +  "word='" + word + '\'' + ", count=" + count + '}'+"\r\n";
        }
    }
}
```

The overall time complexity of the above algorithm should be **O (n log k)** where n is the total number of elements and k is the number of top occurrence elements that we need. Space complexity would be Big O(k + d), where d is the total number of distinct words in the file.

Notes

We preferred to choose Binary Heap over TreeSet because TreeSet provide a get method with Big O(log n) time complexity over PriorityQueue's peek() method with Big O(1), so its a big time saver for the given requirement.

Binary Min Heap[1] is a *complete binary tree*[2] data structure in which each Node is less than or equal to each of its children. Heap is very efficient O(1) for finding minima and maxima from a given data set. PriorityQueue in JDK 1.6 is a implementation for *Binary Min Heap*.

1 http://en.wikipedia.org/wiki/Heap_%28data_structure%29
2 http://en.wikipedia.org/wiki/Complete_Binary_Tree

Is Heap Abstract Data Type?

A heap is a specific data structure and PriorityQueue (It's implementation) is the proper term for the abstract data type.
A heap data structure should not be confused with the heap which is a common name for dynamically allocated memory. The term was originally used only for the data structure.

Approach 2 : Using Counting Array Approach to find the Top x words

Steps for algorithm -

1. Iterate all words and maintain a count inside hash data structure. This is Big O(n) time complexity task, this will result in structure like this -
   hashMap = {"first":42, "second":100, "third":59};

2. Traverse the hash and find the with maximum frequency, "second":100 in this case, then create an array of string with size 100 - this is Big O(x) where x number of unique words in hashmap
3. Now Traverse the hash again and use the number of occurrences of words as array index and append the word at that position in the array, we will get something like this -
4. Then just traverse the array from the end and collect the k non null words. Time Complexity - Big O(k)

```
import java.util.*;
import java.util.concurrent.ConcurrentMap;
import java.util.stream.Collectors;
```


```
public class SplitWordCount2 {
   public static void main(String[] args) {
      List<String> terms = Arrays.asList(
            "Coding is great",
            "Search Engine are great",
            "Google is a nice search engine",
            "Bing is also a nice engine");
      ConcurrentMap<String, Integer> wordFreq = terms.parallelStream()  //Utilizes multi-core hardware
            .flatMap(s -> Arrays.asList(s.split(" ")).stream())
            .collect(Collectors.toConcurrentMap(w -> w.toLowerCase(), w -> 1, Integer::sum)); // Big O(n)
      OptionalInt max = wordFreq.values().parallelStream().mapToInt(Integer::valueOf).max();
      if (max.isPresent()) {
         String[] freq = new String[max.getAsInt()];
         wordFreq.forEach((s, integer) -> {
            if (freq[integer - 1] == null) {
               freq[integer - 1] = s;
            } else {
               freq[integer - 1] = freq[integer - 1] + "," + s;
            }
         });
         for (int i = max.getAsInt() - 1; i >= 0; i--) {
            System.out.println((i + 1) + " -- " + freq[i]);
         }
      }
   }
}
```

Approach 3 : Other Strategies for Scalable Design

If we have very limited memory in our device then we can utilize memory efficient data structures for storing words - TRIE and DAG.

TRIE - memory is shared between multiple words with common prefix and word count can be maintained along with the word termination mark, but it would be more time consuming than the HashMap

Approach 4 : Diving Data File into multiple if unique words are too big

If the number of unique words are too big to fit into main memory of computer, then we can use hashing technique to divide the words into different files and start processing those files one by one.
Pseudo algorithm

- Create N output files
- Fetch words from big data file
- For each word, calculate the hashcode
- output file sequence = hashcode%N (where N is appropriately set according to the main memory available)
- Now all duplicate words will go into same file, so we can count the word frequency from each output file separately and append them into a final merged file.

Approach 5 : Another Divide and Conquer approach

1. Read the file and write all words starting with A to file A.txt, words starting with B to file B.txt ... words starting with Z to file Z.txt.
2. If any of these files are still greater than your memory limit, divide the large files again using their second letters, i.e. words starting with AA goes into file AA.txt, words starting with AB goes into file AB.txt etc.
3. Since a word cannot appear in two different files now, you can easily count all words at each file and merge the results without further calculations.
4. You can divide each file in only one pass, so it would take linear time to divide the files. Then, you can count words in each file and merge the files in linear time.

Approach 6 : Utilizes Distributed Computing (Map-Reduce Approach)

1. Distribute your text files on different nodes as per their memory capacity to hold the entire data into memory.
2. Calculate each word frequency within the node. (Either using hashtable, parallel stream, unix command, space compressed prefix tree, etc)
3. Collect each result in a master node and combine them all. Here we are assuming that the unique words will fit into main computer's memory.
4. Either sort the results or build a min-heap with capacity of 10 and store top 10 occurrences in that min-heap. Java's PriorityQueue can be used to build a min-heap. You can also take a TreeMap in step 3 to store the results in sorted manner.

Approach 7 : Using unix shell to find the top occurrences of words

```
sort <input file with one word per line> | uniq -c | sort -nr | head -10
```

we can also add

```
awk '{print $2}'
```

to eliminate the count of words from the result display.

here is how you read it -

- sort groups (like words) together
- count each group and print one line with count and group
- sort -nr reverse sorts numerically by the count, so the highest count is on the top
- head -10 prints the top 10
- awk chops off the count

Alternatively run the below command in a text file containing multiple words in single line -

```
tr -s '[[:punct:][:space:]]' '\n' < words | sort | uniq -c | sort -nr | head -10 | awk '{print $2}'
```

The output will be something like shown below (if you remove awk command)-

```
2 you
1 when
1 visit
1 today
1 name
1 my
```

Useful discussion -
http://stackoverflow.com/questions/12190326/parsing-one-terabyte-of-text-and-efficiently-counting-the-number-of-occurrences

Q 206. What is difference between synchronized HashMap and a Hashtable?

Functionally both are same except the single difference that a hash table is one of the legacy collection class which was introduced well before the Java Collection Framework.

Both of these classes implement Map interface and are part of Collections framework as of JDK 2. Hashtable implements one extra interface - Dictionary which Hashmap does not.

HashMap should be preferred if thread-safety is not required, and ConcurrentHashMap should be preferred to Hashtable if highly concurrent implementation is required.

Q 207. What is difference between Iterator and ListIterator?

An Iterator class provides us with 3 methods - next(), hasNext() and remove(). Every Collection in Java is Iterable - posses an iterator to allow traversal and removal its underlying elements.

ListIterator - It is a specialized iterator for lists that allows to traverse the list bidirectionally, modify the list during iteration, and obtain the iterator's current position in the list. It allows complete modification - remove, add and update operations are provided.

Q 208. What do you understand by Token Bucket Algorithm. What is its use?

Token Bucket Algorithm

Token bucket algorithm is used to define the upper limits on bandwidth and burstiness on the data transmission in a software application. The token bucket algorithm is based on an analogy of a fixed capacity bucket into which tokens, normally representing a unit of bytes or a single packet of predetermined size, are added at a fixed rate.

Applications

1.) To provide download bandwidth limits in software applications like torrent & download managers.
2.) To control the download speed on 3G network by our cellular provider.

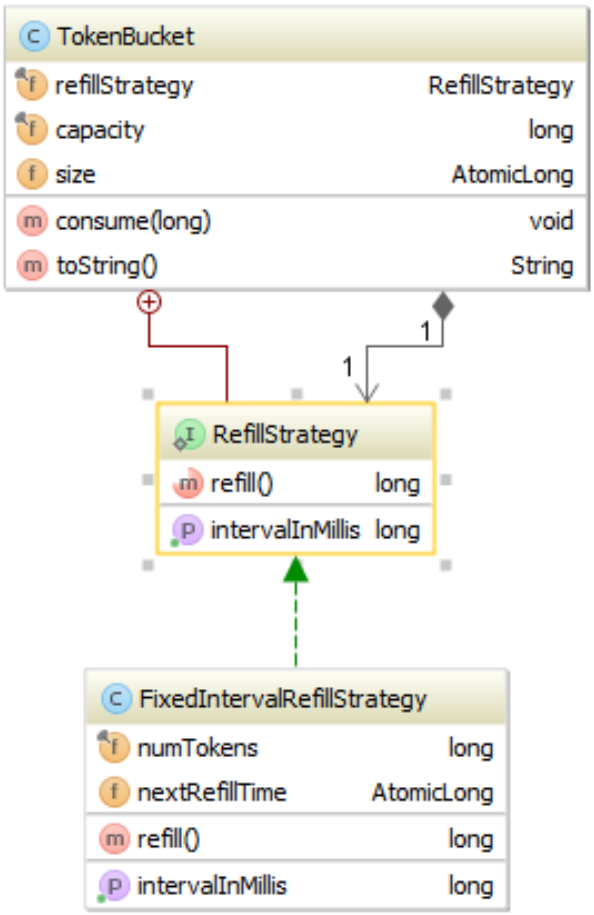
Implementation

Lets try to create an implementation for this algorithm. We will choose a Leaky Bucket Implementation, where a fixed amount of tokens are filled after a predefined interval into the bucket. If no one utilizes those token, then they do not get accumulated over time, they just over flow after the capacity of bucket is reached. Let's name this strategy as FixedIntervalRefillStrategy.

Our TokenBucket Class will have following properties

1.) Refill Strategy
2.) Maximum Capacity of Tokens - this is the maximum amount of tokens that a client can ask for, otherwise an exception is thrown.
3.) Size - it is the current size of the bucket which will keep on changing as it is refilled after specific interval and emptied by the clients.

TokenBucket's consume() method accepts the number of tokens to consume. This method will then remove those number of Tokens from the bucket, refilling the bucket if required. This method utilizes CAS (CompareAndSet) operation of AtomicLong to make the resize operation atomic so that no-locking is required. This will make the class thread-safe when multiple threads will simultaneously demand for the tokens.



Class Diagram For Token Bucket Algorithm

```
public class TokenBucket {
    private final RefillStrategy refillStrategy;
    private final long capacity;
    private AtomicLong size;

    public TokenBucket(long capacity, RefillStrategy refillStrategy) {
        this.refillStrategy = refillStrategy;
        this.capacity = capacity;
        this.size = new AtomicLong(0L);
    }

    public void consume(long numTokens) throws InterruptedException {
        if (numTokens < 0)
            throw new RuntimeException("Number of tokens to consume must be positive");
        if (numTokens >= capacity)
            throw new RuntimeException("Number of tokens to consume must be less than the capacity of the bucket");
        long newTokens = Math.max(0, refillStrategy.refill());
        while (!Thread.currentThread().isInterrupted()) {
            long existingSize = size.get();
            long newValue = Math.max(0, Math.min(existingSize + newTokens, capacity));
            if (numTokens <= newValue) {
                newValue -= numTokens;
                if (size.compareAndSet(existingSize, newValue))
                    break;
            } else {
                    if (existingSize + newTokens <= capacity)
                                    size.addAndGet(newTokens);
                    else
                                    size.addAndGet(capacity - existingSize);
                Thread.sleep(refillStrategy.getIntervalInMillis());
                newTokens = Math.max(0, refillStrategy.refill());
            }
        }
    }

    public static interface RefillStrategy {
        long refill();
        long getIntervalInMillis();
    }

    @Override
    public String toString() {  return "Capacity : " + capacity + ", Size : " + size;    }
}

public final class TokenBuckets
{
  private TokenBuckets() {}

  public static TokenBucket newFixedIntervalRefill(long capacityTokens, long refillTokens, long period, TimeUnit unit)
  {
    TokenBucket.RefillStrategy strategy = new FixedIntervalRefillStrategy(refillTokens, period, unit);
    return new TokenBucket(capacityTokens, strategy);
  }
}

public class FixedIntervalRefillStrategy implements TokenBucket.RefillStrategy {
```

```
    private final long numTokens;
    private final long intervalInMillis;
    private AtomicLong nextRefillTime;

    /**
     * Create a FixedIntervalRefillStrategy.
     *
     * @param numTokens The number of tokens to add to the bucket every interval.
     * @param interval  How often to refill the bucket.
     * @param unit      Unit for interval.
     */
    public FixedIntervalRefillStrategy(long numTokens, long interval, TimeUnit unit) {
        this.numTokens = numTokens;
        this.intervalInMillis = unit.toMillis(interval);
        this.nextRefillTime = new AtomicLong(-1L);
    }

    public long refill() {
        final long now = System.currentTimeMillis();
        final long refillTime = nextRefillTime.get();
        if (now < refillTime) {
            return 0;
        }
        return nextRefillTime.compareAndSet(refillTime, now + intervalInMillis) ? numTokens : 0;
    }

    @Override
    public long getIntervalInMillis() {
        return intervalInMillis;
    }
}
```

API Client User

```
TokenBucket bucket = TokenBuckets.newFixedIntervalRefill(1024 * 10, speedLimitKBps, 1, TimeUnit.SECONDS);
```

For further reading -
http://en.wikipedia.org/wiki/Token_bucket

Q 209. How will you implement fibonacci series using Iterative & Recursive approach in Java 8?

Iterative Approach to Fibonacci using Java

```
import java.io.DataInputStream; //imports all of the input stream class from the io classes
import java.io.IOException;

public class Fibonacci {
    public static void main(String args[]) throws IOException {
        DataInputStream input = new DataInputStream(System.in);
        long a = -1, b = 1, c;
        int n, i = 0;
```

```
        System.out.println("enter the no. of terms u want in d series...");
        n = Integer.parseInt(input.readLine());
        while (i < n) {
            c = a + b;
            System.out.println(c);
            a = b;
            b = c;
            i++;
        }
    }
}
```

Java 8 Recursive Approach with Caching for optimization

Here we will use ConcurrentHashMap as the cache to store the previously calculated fibonacci numbers, there by reducing the redundant computation. Instead of specifying the base condition in the fib() method, cache is pre-populated with the fibonacci sequence of 0 and 1. computeIfAbsent() method invokes the lambda expression only if the value does not exist in the cache. Math.addExact() throws exception in case overflow occurs for the long.

```
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ConcurrentMap;
import java.util.stream.IntStream;

public class Fibonacci2 {
    private ConcurrentMap<Integer, Long> cache = new ConcurrentHashMap<>(100);

    public Fibonacci2() {
        cache.put(0, 0L);
        cache.put(1, 1L);
    }

    public long fib(int n) {
        return cache.computeIfAbsent(n, x -> Math.addExact(fib(x - 1), fib(x - 2)));
                //Math.addExact throws ArithmeticException - if the result overflows a long
    }

    public static void main(String[] args) {
        Fibonacci2 fibonacci2 = new Fibonacci2();
        IntStream.range(0, 90).forEach(value -> System.out.println(fibonacci2.fib(value)));
    }
}
```

Java 7 Recursive Approach using Fork and Join Framework

We can utilize the multi-core hardware by using Fork & Join framework provided by Java 7, here we divide the computation into two steps i.e. fib(n-1) and fib(n-2) and first step is forked into new thread while second step is computed in the same thread.

```
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveTask;

class Fibonacci extends RecursiveTask<Integer> {
    final int n;

    Fibonacci(int n) {
```

```
        this.n = n;
    }

    protected Integer compute() {
        if (n <= 1)
            return n;
        Fibonacci f1 = new Fibonacci(n - 1);
        f1.fork();
        Fibonacci f2 = new Fibonacci(n - 2);
        return f2.compute() + f1.join();
    }

    public static void main(String[] args) {
        Fibonacci fibonacci = new Fibonacci(10);
        ForkJoinPool.commonPool().invoke(fibonacci);
    }
}
```

Please note that cache can be employed in this approach too in order to increase the speed of calculation.

Using Java 8 streams to generate Fibonacci in Iterative Approach

```
static void method1() {
    Stream.iterate(new int[]{0, 1},
            t -> new int[]{t[1], t[0] + t[1]})
            .limit(10)
            .map(t -> t[0])
            .forEach(System.out::println);
}

static void method2() {
    {
        IntSupplier fib = new IntSupplier() {
            private int previous = 0;
            private int current = 1;

            public int getAsInt() {
                int oldPrevious = this.previous;
                int nextValue = this.previous + this.current;
                this.previous = this.current;
                this.current = nextValue;
                return oldPrevious;
            }
        };
        IntStream.generate(fib).limit(10).forEach(System.out::println);
    }
}
```

Q 210. How will you write a multi-threaded HttpDownloader program using Java 8?

We can utilize parallel Stream API provided in Java 8 along with the ForkJoinPool to download the http urls using multiple threads, below is the non-production implementation of the same.

```
package org.shunya.crackingjavainterviews;

import java.io.IOException;
import java.net.HttpURLConnection;
import java.net.URL;
import java.nio.channels.Channels;
import java.nio.channels.FileChannel;
import java.nio.channels.ReadableByteChannel;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.EnumSet;
import java.util.List;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.ForkJoinTask;
import java.util.concurrent.atomic.AtomicInteger;

import static java.nio.file.StandardOpenOption.CREATE;
import static java.nio.file.StandardOpenOption.WRITE;

public class HttpDownloader {

    public static void main(String[] args) throws ExecutionException, InterruptedException {
        HttpDownloader httpDownloader = new HttpDownloader();
        List<String> urls = new ArrayList<>();
        urls.add("http://mysite/test1");
        urls.add("http://mysite/test2");
        urls.add("http://mysite/test3");
        urls.add("http://mysite/test4");
        httpDownloader.downloadAll(urls);
    }

    public void downloadAll(List<String> urls) throws ExecutionException, InterruptedException {
        AtomicInteger fileCounter = new AtomicInteger(0);
        ForkJoinPool pool = new ForkJoinPool(5);
        ForkJoinTask<?> task = pool.submit(() -> urls.parallelStream().forEach(url -> download(url, "test-download-" +
fileCounter.incrementAndGet())));
        task.get();
        pool.shutdown();
    }

    public void download(String rootUrl, String fileName) {
        try {
            Path path = Paths.get(fileName);
            long totalBytesRead = 0L;
            HttpURLConnection con = (HttpURLConnection) new URL(rootUrl).openConnection();
            con.setReadTimeout(10000);
            con.setConnectTimeout(10000);
            try (ReadableByteChannel rbc = Channels.newChannel(con.getInputStream());
```

```
                FileChannel fileChannel = FileChannel.open(path, EnumSet.of(CREATE, WRITE));) {
                totalBytesRead = fileChannel.transferFrom(rbc, 0, 1 << 22); // download file with max size 4MB
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Q 211. How will you find first non-repeatable character from a String using Java 8?

Below is the code to find the first non-repeating letter/character from a given string using Java.

```
package org.shunya.interview.example6;

import java.util.*;
import java.util.function.*;
import java.util.stream.Collectors;

import static java.util.function.Function.identity;

public class NonRepeatingLetter {
    public static void main(String[] args) {
        findFirstNonRepeatingLetter(args[0], System.out::println);
    }

    private static void findFirstNonRepeatingLetter(String s, Consumer<Character> callback) {
        s.chars()
                .mapToObj(i -> Character.valueOf((char) i))
                .collect(Collectors.groupingBy(identity(), LinkedHashMap::new, Collectors.counting()))
                .entrySet().stream()
                .filter(entry -> entry.getValue() == 1L)
                .map(entry -> entry.getKey())
                .findFirst().map(c -> {
            callback.accept(c);
            return c;
        });
    }
}
```

Q 212. How will you find Word Frequency in sorted order for a collection of words?

There are different mechanisms in Java 8 to get the word frequency, broadly they all can be categorized into 4 different categories -

(toMap , groupingBy) x (serial, concurrent)

We will list all of these combinations one by one.

```
    public static void wordsByFreqSorted() {
     List<String> keywords = Arrays.asList("Apple", "Ananas", "Mango", "Banana", "Beer", "Apple", "Mango",
"Mango");
     Map<String, List<String>> result = keywords.stream().sorted()
             .collect(Collectors.groupingBy(it -> it.toString()));
     System.out.println(result);
```

```
}
public static void allMethod(List<String> words) {
    Map<String, Long> frequency = words.stream().collect(groupingBy(Function.identity(), counting()));
    System.out.println("frequency = " + frequency);

    Map<String, Integer> frequency2 = words.stream().collect(groupingBy(Function.identity(), summingInt(e -> 1)));
    System.out.println("frequency2 = " + frequency2);

    Map<String, Long> frequency3 = words.stream().collect(groupingBy(e -> e, counting()));
    System.out.println("frequency3 = " + frequency3);

    Map<String, Integer> frequency4 = words.parallelStream().
            collect(Collectors.toConcurrentMap(w -> w, w -> 1, Integer::sum));
    System.out.println("frequency4 = " + frequency4);

    Map<String, Long> frequency5 = words.parallelStream().collect(
            Collectors.groupingByConcurrent(
                    Function.identity(), Collectors.<String>counting()));
    System.out.println("frequency5 = " + frequency5);

    Map<String, Long> frequency6 = words.stream().collect(
            Collectors.groupingBy(Function.identity(), Collectors.<String>counting()));
    System.out.println("frequency6 = " + frequency6);

}
```

Concurrent version should be ideally faster on multi-processor machines.

Q 213. How will you calculate MD5 hash of a given String in Java?

Java provides MessageDigest class for calculation of MD5, SHA, etc.

MD5 calculation in Java

```
public static String getMD5(String input) {
        try {
            MessageDigest md = MessageDigest.getInstance("MD5");
            byte[] messageDigest = md.digest(input.getBytes());
            BigInteger number = new BigInteger(1, messageDigest);
            String hashtext = number.toString(16);
            while (hashtext.length() < 32) {
                hashtext = "0" + hashtext;
            }
            return hashtext;
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e);
        }
    }
```

Q 214. There is a set of integer values representing the time consumed by a job execution in seconds, how will you find average execution time ignoring the extreme run times (0 seconds or a value much above the average execution time)?

There are many strategies to exclude the bad entries from the input data, in order to calculate the accurate average time for Job execution.

1. Compute the average using all the entries in first phase, then in second phase discard all those entries from input that deviate more than permissible limit, then in third phase recompute the average again.
2. First discard N highest and lowest values and compute arithmetic mean for the rest. Set N to suitable value so that, for example 1% or 10% of values are discarded.
3. Sort the input data in increasing order, then use the median, or middle value, or average of few middle values.
4. Use geometric mean that give less weight for the outliers.

Also see the below links -

http://en.wikipedia.org/wiki/Geometric_mean
http://en.wikipedia.org/wiki/Median
http://en.wikipedia.org/wiki/Mean

Q 215. There are millions of telephone numbers in a country, each state of the country has specific phone number range assigned to it. How will you determine which state a number belongs to?

For the sake of simplicity lets take 8 digit phone number and each assume that each state has the range shown in below format -

91XX - 91999999 belongs to State HP
92XX - 92999999 belongs to State PB
93XX - 93999999 belongs to State HR
And so on for the all the states

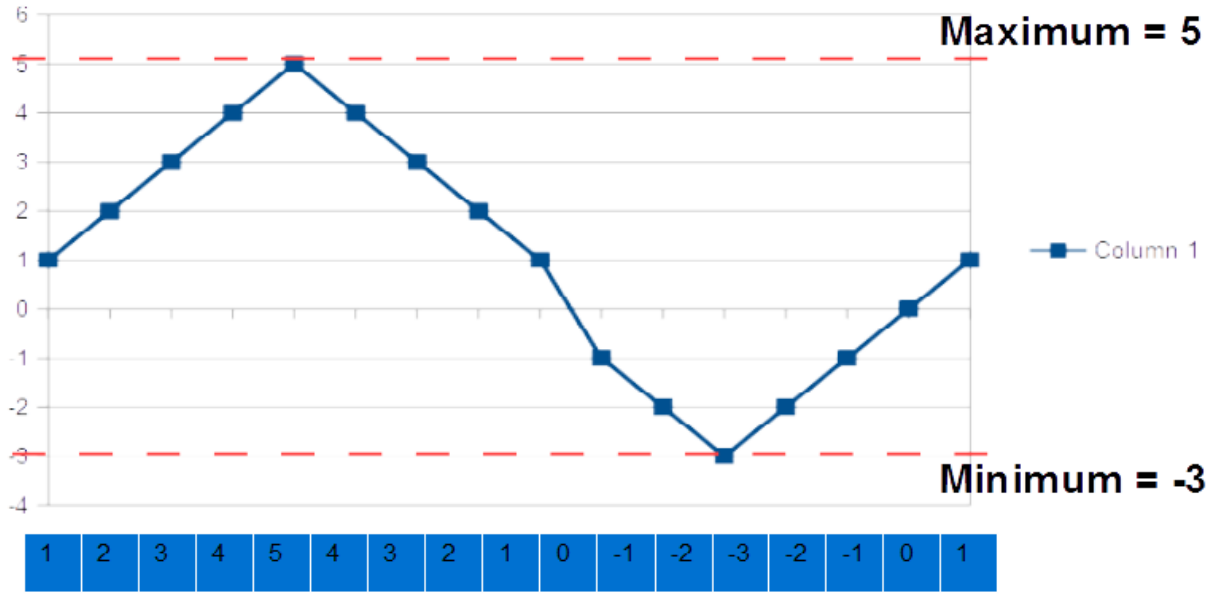
Now to solve such problem we can utilize the number range search algorithm. In Java we have ready made TreeMap implementation that can also effectively solve this problem. Please note that all the phone number ranges are mutually exclusive i.e. no phone number can belong to two states. We can use TreeMap's floorEntry(int) method to find the applicable range for a given input.

floorEntry() Returns a key-value mapping associated with the greatest key less than or equal to the given key, or null if there is no such key.

```
import java.util.TreeMap;
public class PhoneRangeSearch {
   private class PhoneRange {
      int start;
      int end;
      String state;

      public PhoneRange(int start, int end, String state) {
         this.start = start;
         this.end = end;
         this.state = state;
      }
   }

   TreeMap<Integer, PhoneRange> phoneRangeTreeMap = new TreeMap<>();

   public void addRange(int start, int end, String state){
      PhoneRange range = new PhoneRange(start, end, state);
      phoneRangeTreeMap.put(start, range);
   }

   public String getState(int phone){
      return phoneRangeTreeMap.floorEntry(phone).getValue().state;
   }
}


public class PhoneRangeSearchTest {

    @org.junit.Test
    public void testGetState() throws Exception {
        PhoneRangeSearch phoneRangeSearch = new PhoneRangeSearch();
        phoneRangeSearch.addRange(91000000, 91999999, "HP");
        phoneRangeSearch.addRange(92000000, 92999999, "JK");
        phoneRangeSearch.addRange(93000000, 93999999, "PB");
        phoneRangeSearch.addRange(94000000, 94999999, "HR");
        phoneRangeSearch.addRange(95000000, 95999999, "UP");

        assertEquals("HR", phoneRangeSearch.getState(94251212));
        assertEquals("PB", phoneRangeSearch.getState(93158200));
        assertEquals("HP", phoneRangeSearch.getState(91543543));
        assertEquals("JK", phoneRangeSearch.getState(92100004));
    }
```

Q 216. There is a number series in which subsequent number is either +1 or -1 of previous number. How will you determine if the range of supplied numbers are contained in this series in minimum time?

So basically we get a histogram of number series like this -



From the above series we can see that there is a maxima-minima values. If we calculate the maximum and minimum value from the input series in the first iteration (Big O N), then we can utilize those upper/lower bound values to check if the given input is contained in this bound or not.

Java Source

```
import java.util.IntSummaryStatistics;
import java.util.stream.IntStream;
/**
 * Created by Munish on 4/18/2015.
 */
public class MaximaMinimaSeries {
    public static void main(String[] args) {
        int [] input = {1, 2, 3, 4, 5, 4, 3, 2, 1, 0, -1, -2, -3, -2, -1, 0, 1};
        final IntSummaryStatistics summaryStatistics = IntStream.of(input).summaryStatistics();
        final int max = summaryStatistics.getMax();
        final int min = summaryStatistics.getMin();

        int [] numbersToChck = {3, 7, -8, -3, 5};
        for (int i : numbersToChck) {
            if(i <= max && i >= min){
                System.out.println("Number is contained in the series = " + i);
            }
        }
    }
}
```

Chapter 5

Object Oriented Design

Q 217. What are the key principles when designing a software for performance efficiency?

1. Stateless design using REST can help achieve scalability whereever possible. In such application, minimal session elements need to be replicated while distributing the application over multiple hosts. Users can save their favorite URLs and thus there should be no need for the page flow, if we use REST.
2. Logging can be done asynchronously to save precious time of a method call.
3. More processes vs more threads can be configured based on the demand of the target application. Generally it is advised to have a JVM with up to 2 GB memory because increasing memory beyond 2 GB incurs heavy GC pauses, and if we require more processing then we prefer to have a separate process for the JVM altogether. Multiple independent tasks should be run in parallel. Tasks can be partitioned to improve the performance.
4. If we improve upon the concurrency of the software piece, then we can increase its scalability. This can be achieved by reducing the dependency on the shared resources. We should try utilizing the latest hardware optimization through JAVA as much as possible. For example we can use Atomic utilities provided in java.util.concurrent.atomic package, or Fork & Join to achieve higher throughput in concurrent applications. We should try holding the shared locks for as little time as possible.
5. Resource pooling and caching can be used to improve the processing time. Executing jobs in batches can further improve the performance.
6. Picking up appropriate algorithm and data structure for a given scenario can help optimize the processing.
7. If we are using SQL in our application then we should tune the SQL, use batching whereever possible and create indexes on the essentials table columns for faster retrievals.
8. We should tune our JVM for optimum memory settings (Heap, PermGen, etc) and Garbage collection settings. For example if we do lot of text processing in our application with big temporary objects being created, then we should have larger Young Generation defined so that frequent gc run does not happen.
9. Keep up to date with new technologies for performance benefits.

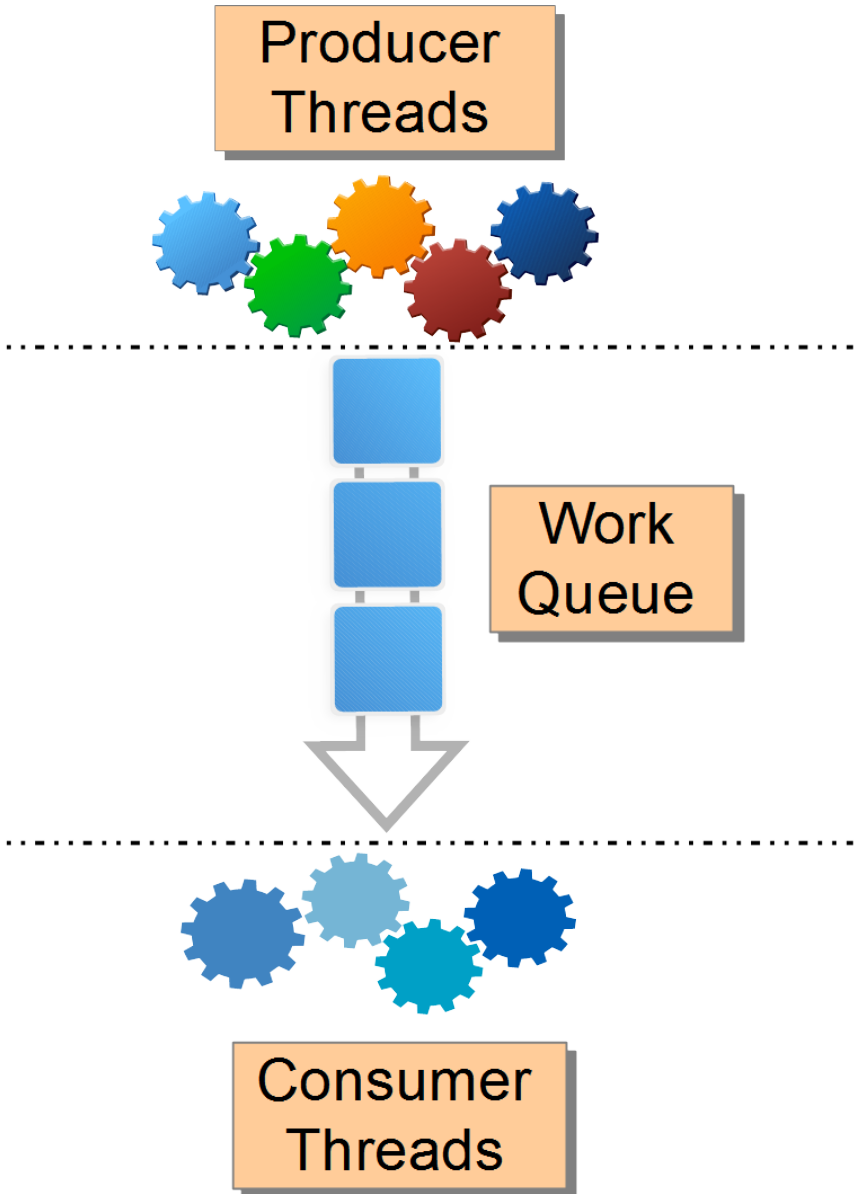
Q 218. How would you describe Producer Consumer problem in Java? What is its significance?

A producer consumer problem is a typical example of multi-thread synchronization problem. It describes a scenario where some shared resource (like a work queue, buffer, etc) is used by two types of threads - Producers & Consumers. Producer populates the task items into the shared queue, and the consumer polls those jobs and execute them. All this happens in parallel. There could be multiple Producer and multiple Consumer, and producer does not even need to know about the consumer. Synchronization is required among the threads so that Producer does not add more items than the size of the queue, and consumer does not take data when there is no item in the shared queue. All these problems are addressed in Producer Consumer Design.

To Summarize the benefits of Producer Consumer Queue -

- Producer does not need to know about the Consumer, thus there is abstraction of producers and consumers of work items i.e. separation of concerns.

- Producer can keep filling the Work Queue irrespective there is a Consumer available to immediately process it. Queue Capacity will determine the number of Work Items it can hold.
- Consumer can be added on demand i.e. if there is more work items in the Queue then we can add more consumers to the same shared Queue and thus share the work load.
- In a similar way N number of producer can be attached the Queue.
- Queue makes sure that no work item is picked up by more than one consumer thread. Java's thread synchronization mechanism can be utilized to achieve thread-safety.

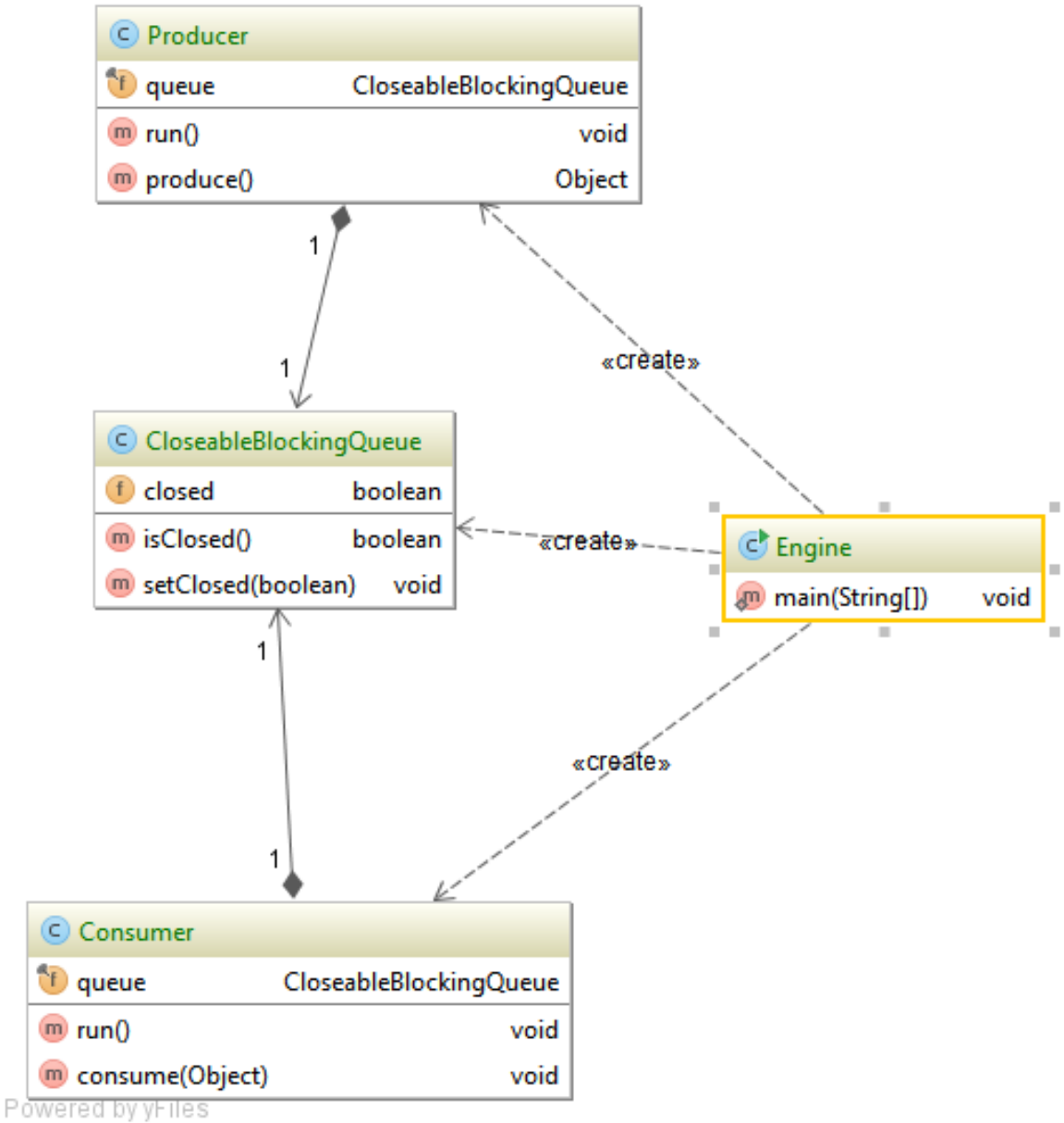


Using Java 1.5 onwards, this problem can be easily solved using Thread and shared BlockingQueue implementation, as demonstrated below.

```
import java.util.concurrent.LinkedBlockingQueue;

/**
 * Created by Munish on 4/14/2015.
 */
```

```
public class CloseableBlockingQueue<T> extends LinkedBlockingQueue<T> {
   private boolean closed = false;

   public synchronized boolean isClosed() {
      return closed;
   }

   public synchronized void setClosed(boolean closed) {
      this.closed = closed;
   }
}
public class Consumer implements Runnable {
   private final CloseableBlockingQueue queue;

   Consumer(CloseableBlockingQueue q) {
      queue = q;
   }

   public void run() {
      try {
         while (!queue.isClosed()) {
            consume(queue.take());
         }
      } catch (InterruptedException ex) {
         ex.printStackTrace();
      }
   }

   void consume(Object x) {
      System.out.println(x);
   }
}


public class Producer implements Runnable {
   private final CloseableBlockingQueue queue;

   Producer(CloseableBlockingQueue q) {
      queue = q;
   }

   public void run() {
      try {
         while (!queue.isClosed()) {
            queue.put(produce());
         }
      } catch (InterruptedException ex) {
         ex.printStackTrace();
      }
   }

   Object produce() {
      return new Object();
   }
}
```

```
class Engine {
   public static void main(String[] args) throws InterruptedException {
      CloseableBlockingQueue<Integer> q = new CloseableBlockingQueue<>();
      Producer p = new Producer(q);
      Consumer c1 = new Consumer(q);
      Consumer c2 = new Consumer(q);
      new Thread(p).start();
      new Thread(c1).start();
      new Thread(c2).start();

      Thread.sleep(1000);
      q.setClosed(true);
   }
}
```

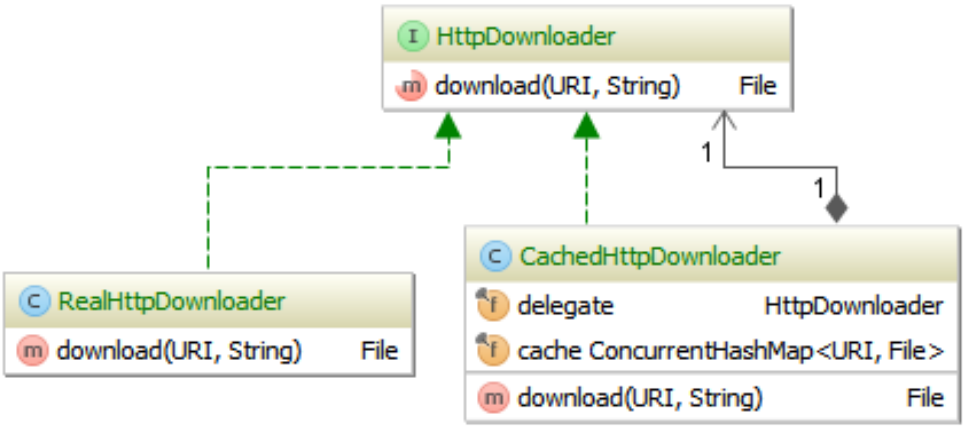


Class Diagram for Producer Consumer Problem

Q 219. How would you implement a Caching for HttpDownloader Task using Decorator Design Pattern?

Decorator Design pattern makes it very easy to enrich the behavior of an existing class by adding wrapper over it, thus maintaining the loose coupling at the same time. Let's first discuss the overall design for implementing caching to Real Http Downloader Task.

Class Diagram for CachedHttpDownloader using Decorator Design Pattern




```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.net.HttpURLConnection;
import java.net.URI;
import java.nio.channels.Channels;
import java.nio.channels.FileChannel;
import java.nio.channels.ReadableByteChannel;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.util.EnumSet;
import java.util.concurrent.ConcurrentHashMap;

public class RealHttpDownloader implements HttpDownloader2 {

    @Override
    public File download(URI uri, String fileName) throws IOException {
        Path path = Paths.get(fileName);
        long totalBytesRead = 0L;
        HttpURLConnection con = (HttpURLConnection) uri.resolve(fileName).toURL().openConnection();
        con.setReadTimeout(10000);
        con.setConnectTimeout(10000);

        try (ReadableByteChannel rbc = Channels.newChannel(con.getInputStream());
             FileChannel fileChannel = FileChannel.open(path, EnumSet.of(StandardOpenOption.CREATE,
```

```
    StandardOpenOption.WRITE));) {
            totalBytesRead = fileChannel.transferFrom(rbc, 0, 1 << 22); // download file with max size 4MB
            System.out.println("totalBytesRead = " + totalBytesRead);
            fileChannel.close();
            rbc.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        return path.toFile();
    }
}

interface HttpDownloader2 {
    public File download(URI uri, String fileName) throws IOException;
}

class CachedHttpDownloader implements HttpDownloader2 {
    private final org.shunya.interview.HttpDownloader delegate;
    private final ConcurrentHashMap<URI, File> cache = new ConcurrentHashMap<>(100);

    public CachedHttpDownloader(org.shunya.interview.HttpDownloader delegate) {
        this.delegate = delegate;
    }

    @Override
    public File download(URI uri, String fileName) throws IOException {
        if (cache.contains(uri))
            return cache.get(uri);
        return delegate.download(uri, fileName);
    }
}
```

Here we see that CachedHttpDownloader implements the same interface HttpDownloader and it has a delegator object which holds the instance of RealHttpDownloader which if required download from the Http.

Question : Why didn't we choose CachedHttpDownloader to extend from RealHttpDownloader?
We could have chosen another approach of creating the CachedHttpDownloader by extending it from RealHttpDownloader class, but that could have limited our implementation from the benefit of extending it from any other class. Java does not allow a class to extend from more than one class.

Q 220. Write Object Oriented design for library management system.

Terminology

Publication - A Publication is a written work meant for distribution to public (it has author, publisher)
Book - A Book is a specific type of publication extending the abstract Publication
Journal - A journal is also a specific type of publication which extends some properties of Publication.
Transaction - Return, Borrow and Renew are three main types of transactions happening inside a library.

Gathering Requirements

1. Authentication for the valid user
2. Managing (add, update, remove) the Inventory of all type of publications (Book, Journal, Magazine, etc)
3. Ability to search for a Publication by various parameters
4. Ability to borrow a Book from library

5. Ability to renew a book
6. Ability to return a book

Design

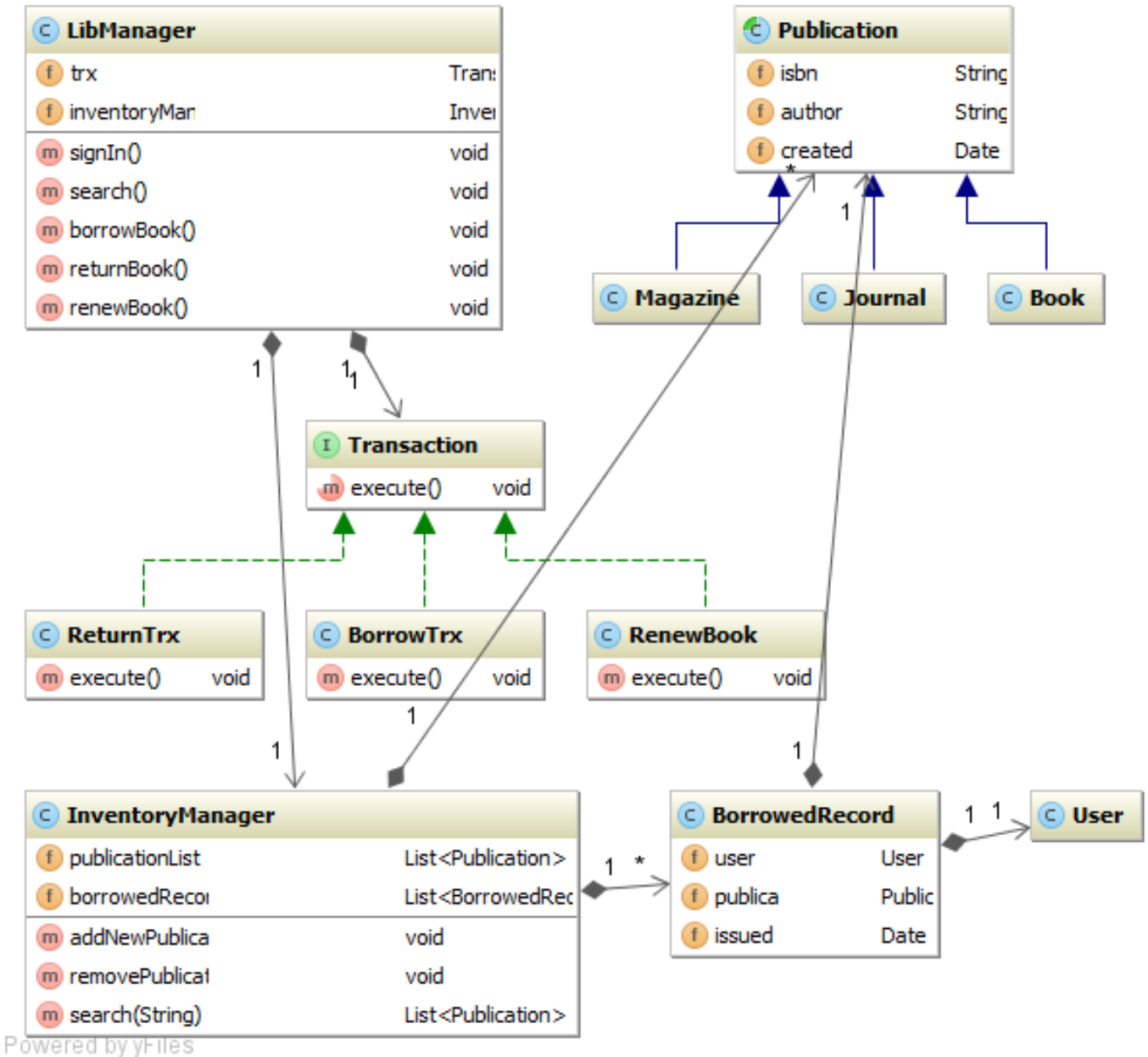
Journal and Book are specific type of Publication so we can map them into Object Oriented World by making Publication Class as Abstract and then Book, Magazine and Journal extending the Publication.

Similarly borrow, return and renew are the type of transaction that a library user will typically be performing. Transaction can be made an interface and Return, Borrow & Renew will implement this interface.

For further reading -

Chapter 6. Object oriented design with UML and Java

Here is the class diagram for Library Management System.



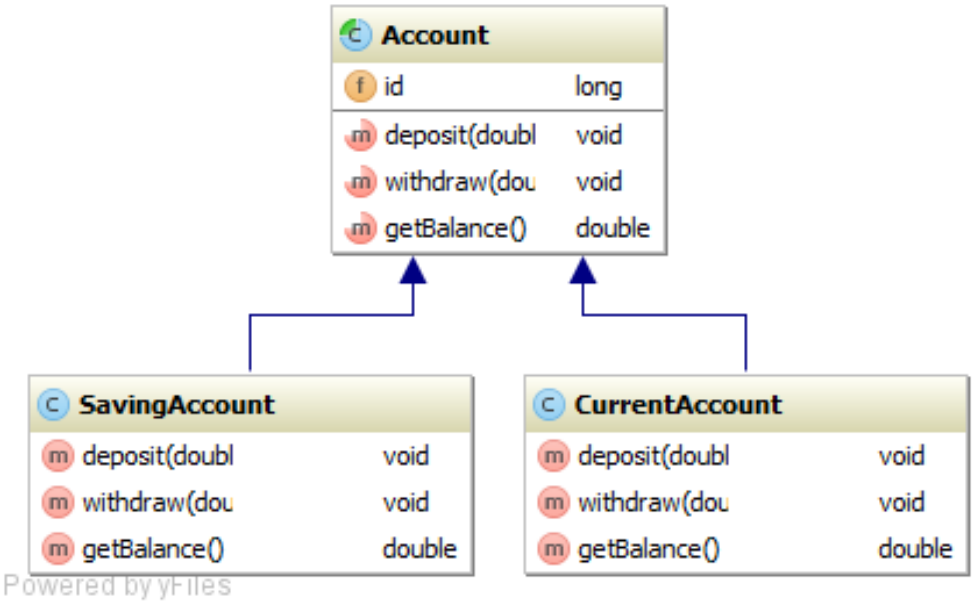
Q 221. Design ATM machine.

State Design Pattern along with generalization can be used to solve the problem.

Gathering Requirements
1. Authenticate user with PIN
2. Select account type - Current Account, Saving Account
3. Select Operation : Balance Inquiry, Withdrawal or Deposit
4. Execute any of the above operation after supplying necessary input.
5. Print the transaction if required.

Design
We can utilize state design pattern for 2nd step mentioned above to maintain the state of account selected by the user.

Account - > SavingAccount, CurrentAccount (using state pattern)

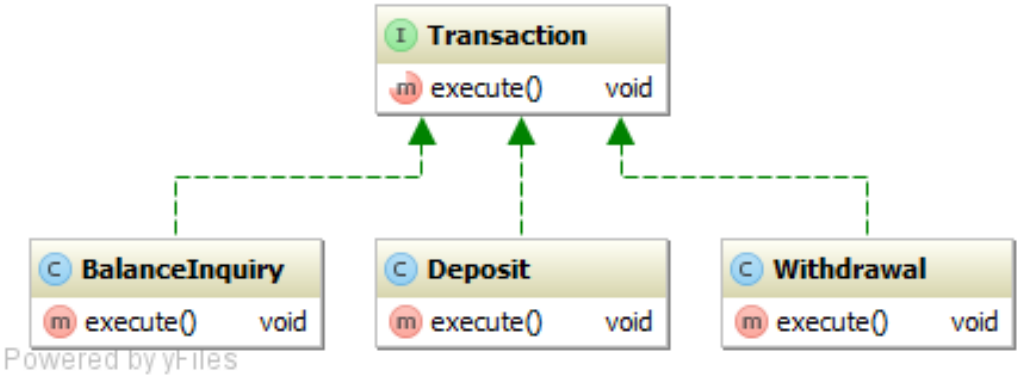


Account has two states - Saving Account & Current Account. User will select one of these account for executing a transaction, and the appropriate state will be set at that moment.

3rd step can be solved using polymorphism where Balance Inquiry, Withdrawal & Deposit represents a Transaction which can be executed.

Transaction -> Balance Inquiry, Withdrawal, Deposit (using generalization)

Prompt user if the user requires receipt or not and accordingly execute the action.



Q 222. Design a web crawler that will crawl for links(urls).

Designing a web crawler is very complex task and can not be explained in books in entirety. We will discuss few steps that can make a very simple Web Crawler running on your system.

Steps To Write a simple Web Crawler

Create a Tracker Object to kep track of visited URLs so that no cyclic reference occurs during crawling.
Start with the list of few seeding URL's and add them to the Local Queue and Tracker.
Iterate over the Queue and fetch links from each URL, and add each link into the same queue if the url has not been visited earlier. Tracker can easily tell if the new url was visited earlier or not.

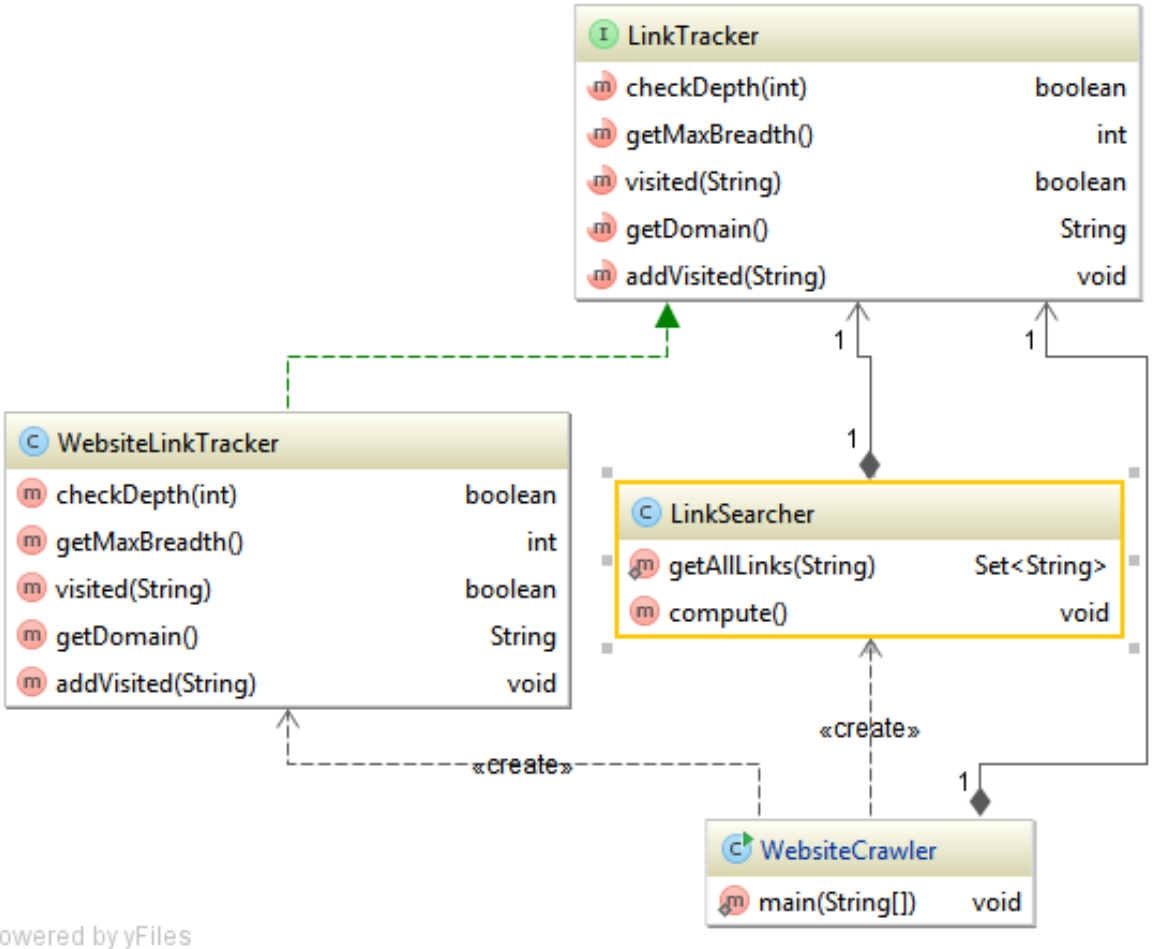
You can Create a ThreadPoolExecutor and a CallableTask which can fetch URL from the above created Queue and get the HTTP contents and index/crawl them. HttpClient can be used (JSoup, HtmlUnit, etc can also be used) to fetch the contents/links from a web page. IOChannels can be utilized to download the HTML contents from a URL in an efficient manner.

Java 7's Fork and Join Pool framework can give a better throughput for the same hardware. We will cover a simple implementation using ForkJoinPool framework instead of ExecutorService.

Java Source

```
import java.net.URI;
import java.net.URISyntaxException;
import java.util.concurrent.ForkJoinPool;

public class WebsiteCrawler {
   private String inputUrl;
   private ForkJoinPool pool;
   private LinkTracker linkTracker;

   public WebsiteCrawler(String startUrl, int maxThreadCount, int maxDepth, int maxBreadth) throws
URISyntaxException {
      this.inputUrl = startUrl;
      this.linkTracker = new WebsiteLinkTracker(maxDepth, maxBreadth, getDomainName(startUrl));
      this.pool = new ForkJoinPool(maxThreadCount);
   }
```

```
    private static String getDomainName(String url) throws URISyntaxException {
        URI uri = new URI(url);
        String domain = uri.getHost();
        return domain.startsWith("www.") ? domain.substring(4) : domain;
    }

    private void init() {
        pool.invoke(new LinkSearcher(0, this.inputUrl, linkTracker));
    }

    public static void main(String[] args) throws Exception {
//        new WebsiteCrawler("http://kalyan-gitapress.org/", 20, 3).init();
        new WebsiteCrawler("http://www.kalyana-kalpataru.org/", 20, 3, 100).init();
    }
}
```



Class Diagram for a Simplistic Web Crawler

```
package org.shunya.crawler;

public interface LinkTracker {
    boolean checkDepth(int depth);
    int getMaxBreadth();
    boolean visited(String link);
    String getDomain();
    void addVisited(String link);
}


import java.net.URI;
import java.net.URISyntaxException;
import java.util.concurrent.ForkJoinPool;

public class WebsiteCrawler {
    private String inputUrl;
    private ForkJoinPool pool;
    private LinkTracker linkTracker;

    public WebsiteCrawler(String startUrl, int maxThreadCount, int maxDepth, int maxBreadth) throws
URISyntaxException {
        this.inputUrl = startUrl;
        this.linkTracker = new WebsiteLinkTracker(maxDepth, maxBreadth, getDomainName(startUrl));
        this.pool = new ForkJoinPool(maxThreadCount);
    }

    private static String getDomainName(String url) throws URISyntaxException {
        URI uri = new URI(url);
        String domain = uri.getHost();
        return domain.startsWith("www.") ? domain.substring(4) : domain;
    }

    private void init() {
        pool.invoke(new LinkSearcher(0, this.inputUrl, linkTracker));
    }

    public static void main(String[] args) throws Exception {
        new WebsiteCrawler("http://www.kalyana-kalpataru.org/", 20, 3, 100).init();
    }
}



import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.select.Elements;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Set;
import java.util.concurrent.RecursiveAction;
```

```
import static java.util.stream.Collectors.toSet;

public class LinkSearcher extends RecursiveAction {
    private int currentDepth;
    private String url;
    private LinkTracker tracker;

    public LinkSearcher(int currentDepth, String url, LinkTracker tracker) {
        this.currentDepth = currentDepth;
        this.url = url;
        this.tracker = tracker;
    }

    public static Set<String> getAllLinks(String url) {
        try {
            Document document = Jsoup.connect(url).timeout(20 * 1000).userAgent("Mozilla").get();
            Elements links = document.select("a");
            return links.stream().map(element -> element.absUrl("href")).collect(toSet());
        } catch (IOException e) {
            e.printStackTrace();
        }
        return Collections.emptySet();
    }

    @Override
    public void compute() {
        if (!tracker.visited(url) && tracker.checkDepth(currentDepth)) {
            try {
                List actions = new ArrayList();
                Set<String> allLinks = getAllLinks(url);
                allLinks.stream().limit(tracker.getMaxBreadth()).forEach(link -> {
                    if (!link.isEmpty() && !link.endsWith(".pdf") && !link.endsWith(".jpg") && link.contains(tracker.
getDomain()) && !tracker.visited(link)) {
                        actions.add(new LinkSearcher(currentDepth + 1, link, tracker));
                    } else if (link.endsWith(".pdf")) {
                        System.out.println(currentDepth+" : found --> " + link);
                    }
                });
                tracker.addVisited(url);
                invokeAll(actions);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

For further reading

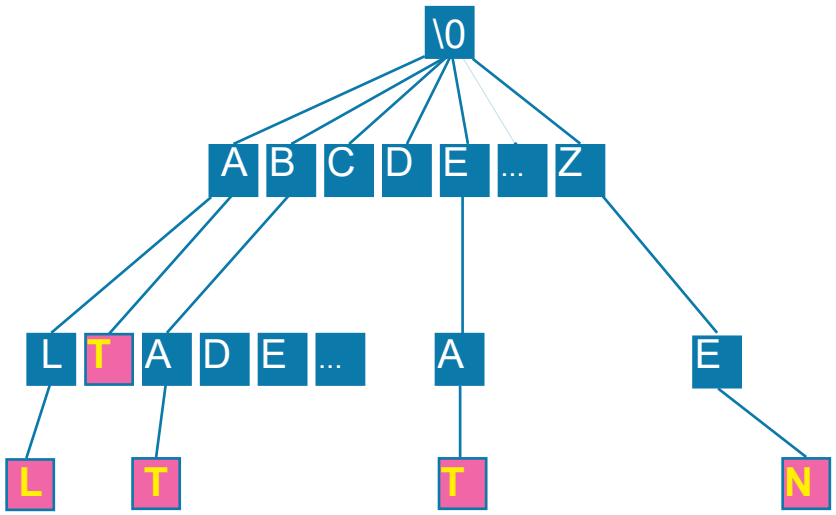
http://www.harding.edu/fmccown/classes/comp475-s09/WebCrawler.java.txt

Q 223. Design Phone Book for a mobile using TRIE (also known as prefix tree).

TRIE is an ordered tree data structure which store associative arrays and the keys are usually alphabets. The position in the tree defines the key with which it is associated. The root node is generally empty (\0) and it contains upto 26 children each representing a alphabet character. All descendants of a given node have a common prefix of string associated with that node. Each node contains a flag which tells if the current node is full word or not. In the diagram on right, pink colored boxes shows the full words.

The term trie comes from re**trie**val

TRIE is a generally good data structure for storing dictionary like data. Phone book can be perfectly implemented using TRIE which will save memory as well as time for prefix searching.



Prefix Tree (TRIE) Visualization

A typical Node of a Trie is defined as

```
static class TrieNode {
    char letter;
    TrieNode[] links;
    boolean fullWord;
    TrieNode(char letter) {
        this.letter = letter;
        links = new TrieNode[26];
        this.fullWord = false;
    }
}
```

Root node is always empty, as shown below

```
static TrieNode createTree() {
    return (new TrieNode('\0'));
}
```

To insert a word inside Trie, we would this method

```
static void insertWord(TrieNode root, String word) {
    int offset = 97;
    int l = word.length();
    char[] letters = word.toLowerCase().toCharArray();
    TrieNode curNode = root;
    for (int i = 0; i < l; i++) {
        if (curNode.links[letters[i] - offset] == null)
            curNode.links[letters[i] - offset] = new TrieNode(letters[i]);
        curNode = curNode.links[letters[i] - offset];
    }
    curNode.fullWord = true;
}
```

To find if given word exists in the Tree

```
static boolean find(TrieNode root, String word) {
    char[] letters = word.toCharArray();
    int l = letters.length;
    int offset = 97;
```

```
        TrieNode curNode = root;
        int i;
        for (i = 0; i < l; i++) {
            if (curNode == null)
                return false;
            curNode = curNode.links[letters[i] - offset];
        }
        if (i == l && curNode == null)
            return false;
        if (curNode != null && !curNode.fullWord)
            return false;
        return true;
    }
```

And, finally to print the whole Tree

```
    static void printTree(TrieNode root, int level, char[] branch) {
        if (root == null)
            return;
        for (int i = 0; i < root.links.length; i++) {
            branch[level] = root.letter;
            printTree(root.links[i], level + 1, branch);
        }
        if (root.fullWord) {
            System.out.println(String.valueOf(branch, 0, level + 1));
        }
    }
```

The main method looks like,

```
    public static void main(String[] args) {
        TrieNode tree = createTree();
        String[] words = {"an", "ant", "all", "allot", "alloy", "aloe", "are", "ate", "be", "beat", "beware", "beast", "bed", "bell"}
        for (int i = 0; i < words.length; i++)
            insertWord(tree, words[i]);
        find(tree, "ant");
    }
```

Applications of TRIE datastructure

1. Looking up data in a trie is faster in the worst case, O(m) time (where m is the length of a search string), compared to an imperfect hash table. An imperfect hash table can have key collisions. A key collision is the hash function mapping of different keys to the same position in a hash table. The worst-case lookup speed in an imperfect hash table is O(N) time, but far more typically is O(1), with O(m) time spent evaluating the hash. There are no collisions of different keys in a trie.
2. Autocomplete is a potential candidate for TRIE
3. Sorting can be accomplished using TRIE, using the below approach
   i. Insert all keys in a TRIE datastructure
   ii. Output all keys in the trie by means of pre-order traversal, which results in output that is in lexicographically increasing order. Pre-order traversal is a kind of depth-first traversal.
4. Full Text Search - A special kind of trie, called a suffix tree, can be used to index all suffixes in a text in order to carry out fast full text searches.

For further reading -

http://en.wikipedia.org/wiki/Trie
http://en.literateprograms.org/Suffix_tree_%28Java%29

Q 224. How would you resolve task's inter dependency, just as in maven/ant.

Let's consider the following task dependencies.

Task	Dependent On
3	1,5
2	5,3
4	3
5	1

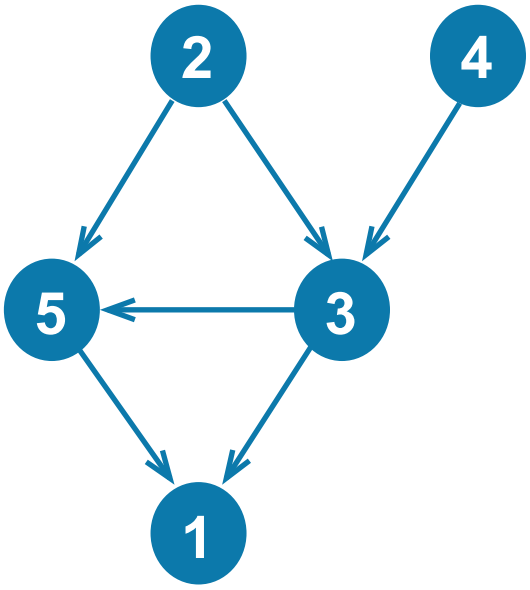
Here first row states that task 3 is dependent on task 1 and task 5, and so on. If the two consecutive tasks have no dependency, then they can be run in any order.

The result should look like - [1, 5, 3, 2 ,4] or [1, 5, 3, 4, 2]

Approach 1

It is a typical Graph traversal problem, that can be solved using Topological Sorting Algorithm[1] in linear time O(|V| + |E|),
Where
V = number of nodes
E = number of edges

Lets now discuss the algorithm as described by Kahn in 1962.



Direct Acyclic Graph for the Task Dependencies

First, find a list of "start nodes" which have no outgoing edges and insert them into a set S; at least one such node must exist in an acyclic graph.

1 http://en.wikipedia.org/wiki/Topological_sorting

Then we will follow this algorithm,

Pseudo Code For Algorithm

L ← Empty list that will contain the sorted elements
S ← Set of all nodes with no incoming edges

```
while S is non-empty do
   remove a node n from S
   insert n into L
   for each node m with an edge e from n to m do
      remove edge e from the graph
      if m has no other incoming edges then
         insert m into S

if graph has edges then
   return error (graph has at least one cycle)
else
   return L (a topologically sorted order)
```

Lets see the sample Java Implementation,

Java Source

```
public class Node {
   public int data;
   public final HashSet<Edge> inEdges;
   public final HashSet<Edge> outEdges;

   public Node(int data) {
      this.data = data;
      inEdges = new HashSet<Edge>();
      outEdges = new HashSet<Edge>();
   }
   public Node addEdge(Node node) {
      Edge e = new Edge(this, node);
      outEdges.add(e);
      node.inEdges.add(e);
      return this;
   }
   @Override
   public String toString() {
      return "" + data;
   }
   public static class Edge{
      public final Node from;
      public final Node to;
      public Edge(Node from, Node to) {
         this.from = from;
         this.to = to;
      }
      @Override
      public boolean equals(Object obj) {
         Edge e = (Edge)obj;
```

```
            return e.from == from && e.to == to;
        }
    }
}


public class TaskResolver {

    public static void main(String[] args) {
        Node one = new Node(1);
        Node two = new Node(2);
        Node three = new Node(3);
        Node four = new Node(4);
        Node five = new Node(5);

        //Adding dependencies to the nodes.
        three.addEdge(one).addEdge(five);  // task 3 depends on 1 & 5.
        two.addEdge(five).addEdge(three);       // Task 2 depends on 5 & 3.
        four.addEdge(three);
        five.addEdge(one);

        Node[] allNodes = {one, two, three, four, five,};
        //L <- Empty list that will contain the sorted elements

        ArrayList<Node> L = new ArrayList<Node>();
        //S <- Set of all nodes with no incoming edges
        HashSet<Node> S = new HashSet<Node>();
        for (Node n : allNodes) {
            if (n.inEdges.size() == 0) {
                S.add(n);
            }
        }

        //while S is non-empty do
        while(!S.isEmpty()){
            //remove a node n from S
            Node n = S.iterator().next();
            S.remove(n);

            //insert n into L
            L.add(n);

            //for each node m with an edge e from n to m do
            for(Iterator<Node.Edge> it = n.outEdges.iterator();it.hasNext();){
                //remove edge e from the graph
                Edge e = it.next();
                Node m = e.to;
                it.remove();//Remove edge from n
                m.inEdges.remove(e);//Remove edge from m

                //if m has no other incoming edges then insert m into S
                if(m.inEdges.isEmpty()){
                    S.add(m);
                }
            }
        }
```

```
        //Check to see if all edges are removed
        boolean cycle = false;
        for(Node n : allNodes){
            if(!n.inEdges.isEmpty()){
                cycle = true;
                break;
            }
        }

        if(cycle){
            System.out.println("Cycle present, topological sort not possible");
        }else{
            System.out.println("Topological Sort: "+ Arrays.toString(L.toArray()));
        }
    }
}
```

Approach 2

We can use HashMap to solve this problem.

1. Create a List of Tasks which will hold all completed tasks.
1. Find the tasks that are independent, like task 1 in above example and put it into a completed task list.
2. Build a map of task -> List of dependent tasks
3. Traverse the entire entrySet and remove all the tasks whose dependent tasks in completed task basket.
4. Add all tasks to the completed task basket whose dependent tasks are completed.
5. Repeat step 3-4 till the size of the map becomes zero.

For more details please refer to -

http://en.wikipedia.org/wiki/Topological_sorting

Q 225. How would you sort 900 MB of data using 100 MB of RAM?

External sort can be used for sorting data that can not fit into main memory.

Algorithm[1]

External Merge Sort is the answer to the above mentioned problem.

1. Read 100 MB of data in main memory and sort by some conventional method like quicksort.
2. Write the sorted data to the disk.
3. Repeat step 1 & 2 until all the data is in sorted 100 MB chunks (9 chunks) which now need to be merged into single output file.
4. Read first 10 MB of each sorted chunk into input buffer in main memory and allocate remaining 10 MB for the output buffer.
5. Perform 9 way merge and store the result in output buffer.

Lets try to understand this with a concrete example,

Imagine you have numbers 1-9,
{9 7 2 6 3 4 8 5 1}
And lets suppose that only 3 fit in the main memory.
So break the data into 3 chunks, sort each, store in separate files. The contents of 3 files will now become
2 7 9
3 4 6
1 5 8
Now you would open each of 3 files as streams and read the first value from each.
2 3 1
Output the lowest value 1, and get the next value from that stream, now you have
2 3 5
output the next lowest value , 2 and continue onwards until you have outputted the entire sorted list.

Similar Questions

Question[2]

There are 2 huge files A and B which contains numbers in sorted order. Make a combined file C which contains the total sorted order.

Solution

Merge Sort technique.

Question

There are k files each containing millions of numbers. How would you create a combined sort file out of these?

Solution

- Use a binary-min heap (increasing order, smallest at the top) of size k, where k is the number of files.
- Read first record from all the k files into the heap.
- Loop until all k files are empty.
- Poll() the minimum element from the binary heap, and append it to the file.
- Read the next element from the file from which the minimum element came.
- If some file has no more element, then remove it from the loop.
- In this way we will have one big file with all number sorted
- Time complexity will be O (n log k)

---

1 http://en.wikipedia.org/wiki/External_sorting#External_merge_sort
2 http://en.wikipedia.org/wiki/Merge_sort

Java Code Example For External Merge Sort

```
public class ExternalMergeSort {

    public static void main(String[] args) throws IOException {
        ExternalMergeSort mergeSort = new ExternalMergeSort();
        int recordCount = 1000000;
        Path path = mergeSort.createTestData(recordCount);
        int numPartitions = 10;
        List<Path> splitFiles = new ArrayList<>();
        for (int i = 1; i <= numPartitions; i++) {
            Path tempFile = Files.createTempFile("test", "random" + i);
            splitFiles.add(tempFile);
            mergeSort.splitFileAndSort((i - 1) * recordCount / numPartitions, recordCount / numPartitions, path, tempFile);
        }
        mergeSort.mergeResults(splitFiles, null, numPartitions, recordCount);
//        System.out.println("-----------------------");
//        Files.lines(path).mapToInt(Integer::valueOf).parallel().sorted().limit(100).forEachOrdered(System.out::println);
        splitFiles.stream().map(Path::toFile).forEach(File::delete);
        path.toFile().delete();
    }

    private void splitFileAndSort(int start, int maxSize, Path inputFile, Path outputFile) throws IOException {
        try (PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(outputFile.toFile())))) {
            Files.lines(inputFile).skip(start).limit(maxSize).parallel().mapToInt(Integer::valueOf).sorted().forEachOrdered(s
-> pw.println(s));
        }
    }

    private void mergeResults(List<Path> pathList, Path outFile, int numPartitions, int recordCount) throws
IOException {
        int[] topMost = new int[numPartitions];
        BufferedReader[] fileBuffers = new BufferedReader[numPartitions];
        for (int i = 0; i < numPartitions; i++) {
            fileBuffers[i] = new BufferedReader(new FileReader(pathList.get(i).toFile()), 1000);
            loadNext(topMost, fileBuffers, i);
        }

        for (int i = 0; i < recordCount; i++) {
            int min = topMost[0];
            int minIndex = 0;
            for (int j = 0; j < numPartitions; j++) {
                if (topMost[j] < min) {
                    min = topMost[j];
                    minIndex = j;
                }
            }
            loadNext(topMost, fileBuffers, minIndex);
            System.out.println(min);
        }
        closeAll(fileBuffers);
    }

    private void closeAll(BufferedReader[] fileBuffers) throws IOException {
        for (BufferedReader fileBuffer : fileBuffers) {
            fileBuffer.close();
        }
    }
```

```
    private void loadNext(int[] topMost, BufferedReader[] fileBuffer, int i) throws IOException {
        String line = fileBuffer[i].readLine();
        if (line != null) {
            topMost[i] = Integer.valueOf(line);
        } else {
            topMost[i] = Integer.MAX_VALUE;
        }
    }

    public Path createTestData(int recordCount) throws IOException {
        Path tempFile = Files.createTempFile("test", "random");
        System.out.println("tempFile = " + tempFile);
        PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(tempFile.toFile())));
        Random random = new Random(System.currentTimeMillis());
        random.ints(0, recordCount).limit(recordCount).forEach(value -> pw.println(value));
        pw.close();
        System.out.println("File created with random numbers = " + tempFile);
        return tempFile;
    }

}
```

In the above code,

CreateTestData() -> Creates a new file with specified number of random numbers in it.
splitAndSort() -> sorts a chunk of data and write it to a file at a time
main() -> driver program

Q 226. How would you design minimum number of platforms so that the buses can be accommodated as per their schedule?

BUS	Arrival Time (HRS)	Departure Time (HRS)
A	0900	0930
B	0915	1300
C	1030	1100
D	1045	1145
E	1100	1400

Bus Schedule for a given Platform

This problem is about finding the peak time when maximum number of buses are waiting to get into platform. Ideally we would not like to stop a bus outside the platform so every single bus would require one platform. So in this problem, the maximum number of buses arriving at the same time during the peak time of day will decide the number of platforms.

Algorithm Skills Required : Sorting, Finding Max from an Array.

Pseudocode

Calculate the peak time of the buses from the given bus schedule, the number of buses at the peak time will give us the number of platforms.

Step 1

Create a single array of bus timing after append A (for Arrival) and D (for departure) to each bus time.
So the above table should now become a one dimensional array like this :

0900A	0930D	0915A	1300D	1030A	1100D	1045A	1145D	1100A	1400D

Step 2

Sort the Bus Time (above array) in ascending order (Natural Order)
Create a Counting Array (mark +1 for Arrival, -1 for Departure)

0900A	0915A	0930D	1030A	1045A	1100A	1100D	1145D	1300D	1400D

Step 3

+1	+1	-1	+1	+1	+1	-1	-1	-1	-1

Create a array with Prefix Sum f(x) = x + (x-1), it gives total amount of buses at a given time. Prefix sum gives us the cumulative running total at any given array index i.e. the cumulative sum of **all buses present** at a station at a given time.
Now traverse the entire array and find the index with maximum value, this value is maximum number of Buses

1	2	1	2	3	4	3	2	1	0

at peak time.

Thus we can see that minimum number of platforms required is 4 at 1100 when there are 4 buses present at the platform.

Pseudo Code

```
var counter, max, peakTime;
for(each time entry){
        if(arrival)
          counter++;
        else
          counter--;
        if(counter > max)
        max= counter
        peakTime= time entry;
}
```

Here the max is the maximum number of platforms that should be built for accommodating all the buses at peak time as per given time table.

Java Source

```
public class PlatformDesigner {
  static class BusSchedule {
       final String name, arrivalTime, departureTime;
       BusSchedule(String name, String arrivalTime, String departureTime) {
          this.name = name;
          this.arrivalTime = arrivalTime;
          this.departureTime = departureTime;
       }
    }
  private List<BusSchedule> busScheduleList = new ArrayList<>();
  public void addBusSchedule(BusSchedule schedule) {
       busScheduleList.add(schedule);
    }
  public void calculateNoOfPlatforms(){
       List<String> scheduleTokens = new ArrayList<>();
       for (BusSchedule schedule : busScheduleList) {
          scheduleTokens.add(schedule.arrivalTime+"A");
          scheduleTokens.add(schedule.departureTime+"D");
       }
   Collections.sort(scheduleTokens);
   int max=0, counter =0;
   String peakTime ="";
       for (String token : scheduleTokens) {
          if(token.endsWith("A")){
             counter++;
          }else if(token.endsWith("D")){
             counter--;
          }
       if(counter > max){
             max=counter;
             peakTime = token;
          }
       }

       System.out.println("max number of platforms required at peak time ["+peakTime+"] = " + max);
    }
}
```

Q 227. There is a pricing service which connects to Reuters & Bloomberg and fetches the latest price for the given Instrument Tics. There could be multiple price events for the same Stock and we need to consider the latest one. Design a service to show prices for the Top 10 stocks of the Day?

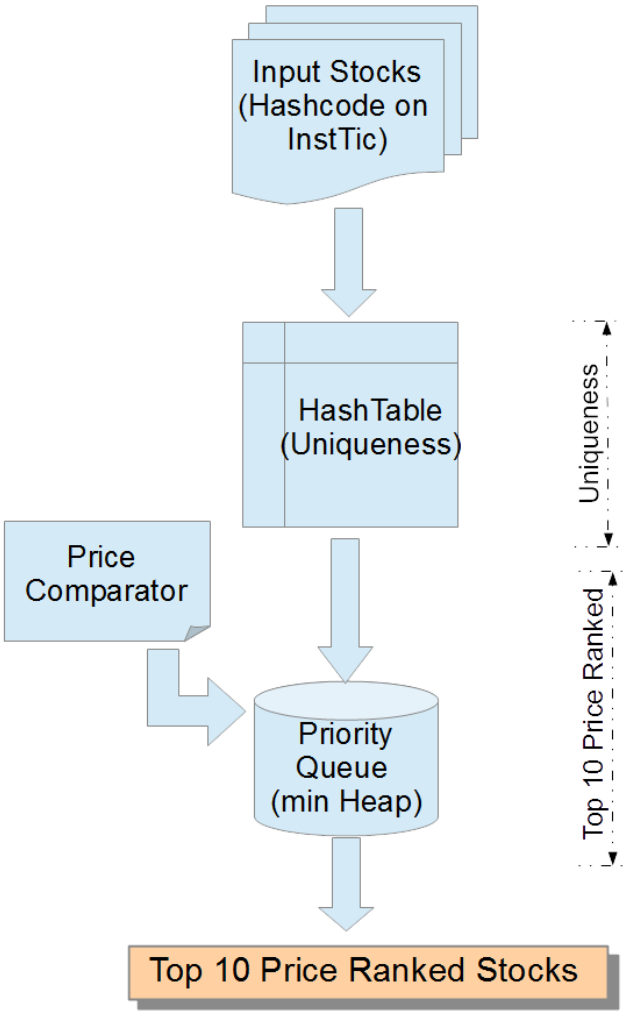
This problems requires us to collect price feeds from PricingService, remove duplicates based on InstrTic keeping the latest one, and then finally capture the top 10 feeds based on price. Keeping in mind that we need to remove duplicates based on Tic# and then sorting based on price i.e. 2 different fields for two different operations.

Maintaining Uniqueness based on Tic#

HashSet is a good option for removing duplicates from the collection, so iterate over the entire collection of feeds and then add them to HashSet, rest will be taken care by HashSet. But we need to override equals and hashcode based on Tic# so as to remove the duplicate TIC# entries.

Finding Top 10 price ranked Stocks

Now for finding top 10 feeds based on prices, we can use PriorityQueue (min heap) of fixed size 10, with a Stock Price comparator. While iterating over the HashSet entries we will check if the price of the feed is greater than the peek entry of PriorityQueue, if yes then poll the entry and offer the new one. This way we will get a list of top 10 priced feeds.



Java Source

```
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

/**
 * Created by Munish on 4/19/2015.
 */
public class PricingService {
    Random rand = new Random(System.currentTimeMillis());
    /**
     * Generates & Returns a Random Feed of Stock Information with variant Price information.
     * @return
     */
    public List<Stock> getPriceFeed() {
        int[] instrTics = {100, 1001, 1002, 1003};
        String[] names = {"Reliance", "TCS", "Airtel", "BSNL"};
        Random randomStraem = new Random(System.currentTimeMillis());
        double[] prices = randomStraem.doubles(100.0, 200.0).limit(50).toArray();
        List<Stock> results = new ArrayList<>();
        for (int i = 0; i < 10; i++) {
            final int randInt = randInt(0, instrTics.length-1);
            results.add(StockBuilder.aStock().withInstrTic(instrTics[randInt]).withPrice(prices[randInt(0, prices.length-1)]).
withName(names[randInt]).build());
        }
        return results;
    }


    public int randInt(int min, int max) {
        return rand.nextInt((max - min) + 1) + min;
    }
}


public class Stock {
    private String name;
    private int instrTic;
    private double price;

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Stock stock = (Stock) o;
        return instrTic == stock.instrTic;
    }

    @Override
    public int hashCode() {
        return instrTic;
    }

    //getters and setters not shown
```

```
    @Override
    public String toString() {
        return "Stock{" +
                "price=" + price +
                ", instrTic=" + instrTic +
                ", name='" + name + '\'' +
                '}';
    }
}

import java.util.*;
import java.util.function.Consumer;

public class StockFeedConsumerService {
    public void process(List<Stock> stockList, Consumer<TopOccurrence> stockConsumer){
        Set<Stock> uniqueStocks = new HashSet<>();
        uniqueStocks.addAll(stockList);
        System.out.println("uniqueStocks = " + uniqueStocks);
        TopOccurrence topOccurrence = new TopOccurrence(2);
        uniqueStocks.forEach(stock -> topOccurrence.add(stock));
        stockConsumer.accept(topOccurrence);
    }

    public static void main(String[] args) {
        PricingService pricingService = new PricingService();
        StockFeedConsumerService consumerService = new StockFeedConsumerService();
        consumerService.process(pricingService.getPriceFeed(), topOccurrence -> System.out.println(topOccurrence));
    }

    static class TopOccurrence {
        private final PriorityQueue<Stock> minHeap;
        private final int maxSize;

        public TopOccurrence(int maxSize) {
            this.maxSize = maxSize;
            this.minHeap = new PriorityQueue<>(Comparator.comparing(Stock::getPrice));
//          This constructs a min heap (when order of elements is natural i.e. ascending order).
//          We are using Natural order for integers (wc.count)
//          In order to create a max-heap, we just need to provide reversed comparator i.e. that sorts in descending order,
as shown below
//          this.minHeap = new PriorityQueue<WordCount>(Comparator.comparingInt((WordCount wc) -> wc.count).
reversed());
        }

        public void add(Stock data) {
            if (minHeap.size() < maxSize) { // size() is Big O(1)
                minHeap.offer(data); // Big O(log(k)) where k is the number of top occurrences required
            } else if (minHeap.peek().getPrice() < data.getPrice()) { // peek() is Big O(1)
                minHeap.poll(); // Big O(log(k))
                minHeap.offer(data); // Big O(log(k))
            }
        }

        @Override
```

```
        public String toString() {
            return "TopOccurrence{" +
                    "minHeap=" + minHeap +
                    ", maxSize=" + maxSize +
                    '}';
        }
    }
}
```

Q 228. Design a parking lot where cars and motorcycles can be parked. What data structure to use for finding free parking spot in Parking Lot program? Assume there are million of parking slots.

Approach

1. Create model classes to OO map Vehicle, vehicleType, Slot, SlotSize, etc.

```
public class Vehicle {
    private String id;
    private String owner;
    private long mobile;
    private String inTime;
    private String outTime;
    private VehicleType vehicleType;
    //create the getters and setters for the above fields
}

public class SlotSize {
    private final int size;

    public SlotSize(int size) {
        this.size = size;
    }
}

public enum VehicleType {
    BIKE(new SlotSize(1), "Motor Bike"),
    CAR(new SlotSize(2), "Car"),
    TRUCK(new SlotSize(4), "Truck");

    private final ParkingSlotSize slotSize;
    private final String vehicleName;

    private VehicleType(SlotSize slotSize, String vehicleName) {...}
}

public class Slot {
    int slotid;
    int floorNo;
    boolean occupied;
    SlotSize size;
}
```

2. Create parking manager which will track the free parking slots using a Queue for fast retrieval, and occupied vehicle mapping will be stored using a HashMap for fast O(1) retrieval. Whenever a slot gets free, remove it from the HashMap and add it into Queue, and if new vehicle comes in then pick slot from the head of queue and store the mapping in hashmap. Separate Queue & HashMap could be used for Motor Bike, Truck & Car vehicle type.

```
public class ParkingManager {
   private Queue<Slot> slots = new PriorityQueue<>();
   private HashMap<Vehicle, Slot> parkDetail = new HashMap<Vehicle, Slot>();
   public Slot getVehicle(String id) {...}        //Logic for searching a vehicle using vehicle Id or slot id.
   public Slot park(Vehicle vehicle) {...}        //save the <vehicle - slot> mapping inside a hash table
}
```

Notes

A free list is a data structure used in a scheme for dynamic memory allocation. It operates by connecting unallocated regions of memory together in a linked list, using the first word of each unallocated region as a pointer to the next. It's most suitable for allocating from a memory pool, where all objects have the same size.

Free lists make the allocation and deallocation operations very simple. To free a region, one would just link it to the free list. To allocate a region, one would simply remove a single region from the end of the free list and use it. If the regions are variable-sized, one may have to search for a region of large enough size, which can be expensive.

Maintain a PriorityQueue for free parking space, use hashmap for the filled spaces. In this manner it would be easier to find the free space and to find the parked object. Assume that the parking space on the lower floor gets more priority than the parking space on the higher floor, when a new car comes in just pick the top most entry from the parking queue and park the object

Q 229. There is three file contains flight data, write a standalone program to search flight detail from all files depend on criteria? Write JUnit to demonstrate the working.

Exact requirement is stated below -
There is three file contains flight data. File data has in csv format.
1)Write a standalone program to search flight detail from all files depend on criteria
2)Criteria would be departure location,arrival location, flight date.
3)Program should follow\Oops principle. And right unit test case also.
4)Result should be in Ascending or descending order.
5)Data separated with pipe | .

File A has data like below:
FLIGHT_NUM|DEP_LOC|ARR_LOC|VALID_TILL|FLIGHT_TIME|FLIGHT_DURN|FARE
AF299|FRA|LHR|20-11-2010|0600|4.10|480
AF118|DUB|MUC|21-12-2010|1410|5.40|580
AF371|AMS|MAD|30-11-2010|1210|3.45|320

File B has data like below:
FLIGHT_NUM|DEP_LOC|ARR_LOC_VALID_TILL|FLIGHT_TIME|FLIGHT_DURN|FARE
BA123|DEL|AMS|12-10-2010|0050|8.00|950

BA412|BOS|CDG|31-12-2010|0210|7.50|800
BA413|BOS|AMS|30-11-2010|1530|7.00|750

File C has data like below:
FLIGHT_NUM|DEP_LOC|ARR_LOC_VALID_TILL|FLIGHT_TIME|FLIGHT_DURN|FARE
LH348|DEL|AMS|30-11-2010|2325|11.00|1050
LH201|LHR|MEL|21-11-2010|0230|15.30|1400
LH342|VIE|JFK|20-10-2010|1130|14.20|980

Approach To solve the Problem -

- Write a parser to read the flight details from the flat file using pipe (|) as the delimiter. Class FlightDataReader does this Job as shown in the coming pages.
- Create a Data Transfer Object (FlightSearchDTO) to store the flight related information in a collection, then load all the flight data into arraylist.
- Index all flight data into HashMap using <src, dest> pair as the key of hashing data structure, it will allow constant time Big O (1) search for a given src destination pair.
- Create a Fare comparator to sort flight search details based on fair details in ascending order.
- Write a Junit Test to run the demonstration of Program.
-

```
import java.io.IOException;
import java.nio.file.Paths;
import java.util.*;

/**
 * Created by Munish on 4/19/2015.
 */
public class FlightSearchService {
    private Map<String, List<FlightSearchDTO>> srcDstFlightIndex = new HashMap<>();
    private FlightDataReader flightDataReader = new FlightDataReader();
    private List<FlightSearchDTO> allFlightData;

    public void loadFlightData() throws IOException {
        allFlightData = flightDataReader.getAll(Paths.get("D:\\workspace\\CrackingJavaInterview\\src\\main\\java\\
org\\shunya\\interview\\example4"));
        allFlightData.forEach(flightSearchDTO -> srcDstFlightIndex.computeIfAbsent(flightSearchDTO.
getDepartLocation() + flightSearchDTO.getArrivalLocation(), s -> new ArrayList<>()).add(flightSearchDTO));
    }

    public List<FlightSearchDTO> searchFlights(FlightSearchCriteria searchCriteria) {
        String srcDestKey = searchCriteria.getDepartLocation() + searchCriteria.getArrivalLocation();
        final List<FlightSearchDTO> flightSearchDTOs = srcDstFlightIndex.get(srcDestKey);
        if (searchCriteria.isSortByFare())
            Collections.sort(flightSearchDTOs, Comparator.comparing(FlightSearchDTO::getFare));
        return flightSearchDTOs;
    }
}


import java.io.IOException;
import java.nio.file.Files;
```

```
import java.nio.file.Path;
import java.nio.file.Paths;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.List;

import static java.util.stream.Collectors.toList;

/**
 * Created by Munish on 4/19/2015.
 */
public class FlightDataReader {

    List<FlightSearchDTO> getAll(Path dir) throws IOException {
        List<FlightSearchDTO> flightSearchDTOs = new ArrayList<>();
        final SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy");
        Files.list(dir).filter(path -> path.toString().endsWith(".txt")).forEach(path1 -> {
            try {
                flightSearchDTOs.addAll(Files.lines(path1).skip(1).map(s -> {
                    FlightSearchDTO flightSearchDTO = new FlightSearchDTO();
                    final String[] split = s.split("[|]");
                    flightSearchDTO.setFlightNo(split[0]);
                    flightSearchDTO.setDepartLocation(split[1]);
                    flightSearchDTO.setArrivalLocation(split[2]);
                    try {
                        flightSearchDTO.setValidTillDate(dateFormat.parse(split[3]));
                    } catch (ParseException e) {
                        e.printStackTrace();
                    }
                    flightSearchDTO.setFlightTine(Integer.parseInt(split[4]));
                    flightSearchDTO.setDuration(Float.parseFloat(split[5]));
                    flightSearchDTO.setFare(Integer.parseInt(split[6]));
                    return flightSearchDTO;
                }).collect(toList()));
            } catch (IOException e) {
                e.printStackTrace();
            }
        });
        System.out.println("All flightSearchDTOs = " + flightSearchDTOs);
        return flightSearchDTOs;
    }

    public static void main(String[] args) throws IOException {
        FlightDataReader dataReader = new FlightDataReader();
        dataReader.getAll(Paths.get("D:\\workspace\\CrackingJavaInterview\\src\\main\\java\\org\\shunya\\
interview\\example4"));
    }
}


public class FlightSearchCriteria {
    private String departLocation;
    private String arrivalLocation;
    private boolean sortByFare;
```

```
    public String getDepartLocation() {
        return departLocation;
    }

    public void setDepartLocation(String departLocation) {
        this.departLocation = departLocation;
    }

    public String getArrivalLocation() {
        return arrivalLocation;
    }

    public void setArrivalLocation(String arrivalLocation) {
        this.arrivalLocation = arrivalLocation;
    }

    public boolean isSortByFare() {
        return sortByFare;
    }

    public void setSortByFare(boolean sortByFare) {
        this.sortByFare = sortByFare;
    }
}


import java.util.Date;

public class FlightSearchDTO {
    private String flightNo;
    private String departLocation;
    private String arrivalLocation;
    private Date validTillDate;
    private int flightTine;
    private float duration;
    private int fare;

    public String getFlightNo() {
        return flightNo;
    }

    public void setFlightNo(String flightNo) {
        this.flightNo = flightNo;
    }

    public String getDepartLocation() {
        return departLocation;
    }

    public void setDepartLocation(String departLocation) {
        this.departLocation = departLocation;
    }

    public String getArrivalLocation() {
```

```
        return arrivalLocation;
    }

    public void setArrivalLocation(String arrivalLocation) {
        this.arrivalLocation = arrivalLocation;
    }

    public Date getValidTillDate() {
        return validTillDate;
    }

    public void setValidTillDate(Date validTillDate) {
        this.validTillDate = validTillDate;
    }

    public int getFlightTine() {
        return flightTine;
    }

    public void setFlightTine(int flightTine) {
        this.flightTine = flightTine;
    }

    public float getDuration() {
        return duration;
    }

    public void setDuration(float duration) {
        this.duration = duration;
    }

    public int getFare() {
        return fare;
    }

    public void setFare(int fare) {
        this.fare = fare;
    }

    @Override
    public String toString() {
        return "FlightSearchDTO{" +
                "flightNo='" + flightNo + '\'' +
                ", departLocation='" + departLocation + '\'' +
                ", arrivalLocation='" + arrivalLocation + '\'' +
                ", validTillDate=" + validTillDate +
                ", flightTine=" + flightTine +
                ", duration=" + duration +
                ", fare=" + fare +
                '}'+"\n";
    }
}
```

```
import org.junit.Before;
import org.junit.Test;

import java.util.List;

import static org.hamcrest.Matchers.equalTo;
import static org.junit.Assert.*;

public class FlightSearchServiceTest {
    FlightSearchService flightSearchService = new FlightSearchService();

    @Before
    public void setUp() throws Exception {
        flightSearchService.loadFlightData();
    }

    @Test
    public void testSearchFlights() throws Exception {
        FlightSearchCriteria flightSearchCriteria = new FlightSearchCriteria();
        flightSearchCriteria.setDepartLocation("DEL");
        flightSearchCriteria.setArrivalLocation("AMS");
        flightSearchCriteria.setSortByFare(true);
        final List<FlightSearchDTO> flightSearchDTOs = flightSearchService.searchFlights(flightSearchCriteria);
        assertThat(flightSearchDTOs.size(), equalTo(2));
        System.out.println("flightSearchDTOs Search = " + flightSearchDTOs);
    }
}
```

Q 230. Implement the classes to model two pieces of furniture (Desk and Chair) that can be constructed of one of two kinds of materials (Steel and Oak). The classes representing every piece of furniture must have a method getIgnitionPoint() that returns the integer temperature at which its material will combust. The design must be extensible to allow other pieces of furniture and other materials to be added later. Do not use multiple inheritance to implement the classes.

Design

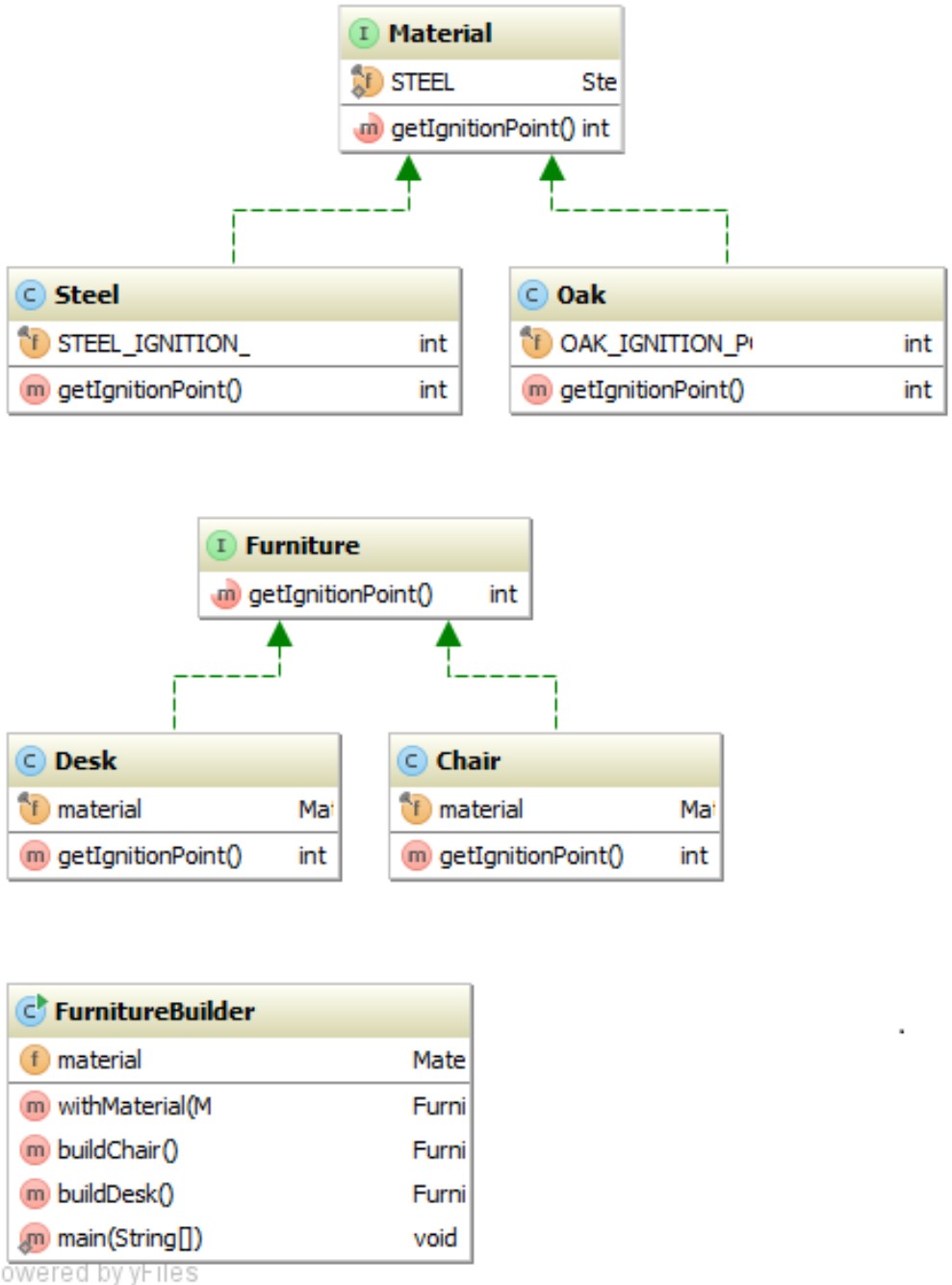
Abstract Factory along with Bridge Pattern can solve this problem.

```
public interface Furniture {
   public int getIgnitionPoint();
}

public interface Material {
   final Material STEEL = new Steel();
   final Material OAK = new Oak();
   int getIgnitionPoint();
}

public class Steel implements Material {
   private final int STEEL_IGNITION_POINT = 1500;
   @Override
   public int getIgnitionPoint() {return STEEL_IGNITION_POINT;}
}

class Oak implements Material {
   private final int OAK_IGNITION_POINT = 900;
   @Override
   public int getIgnitionPoint() { return OAK_IGNITION_POINT; }
}

public class Desk implements Furniture {
   private final Material material;
   public Desk(Material material) {
      this.material = material;
   }
   @Override
   public int getIgnitionPoint() {return material.getIgnitionPoint();}
}

public class FurnitureFactory {
   public Furniture createChair(Material material){
      return new Chair(material);
   }

   public Furniture createDesk(Material material){
      return new Desk(material);
   }
   public static void main(String[] args) {
      FurnitureFactory factory = new FurnitureFactory();
      Furniture desk = factory.createDesk(Material.STEEL);
      int ignitionPoint = desk.getIgnitionPoint();
      System.out.println("ignitionPoint = " + ignitionPoint);
```

```
    }
}
```

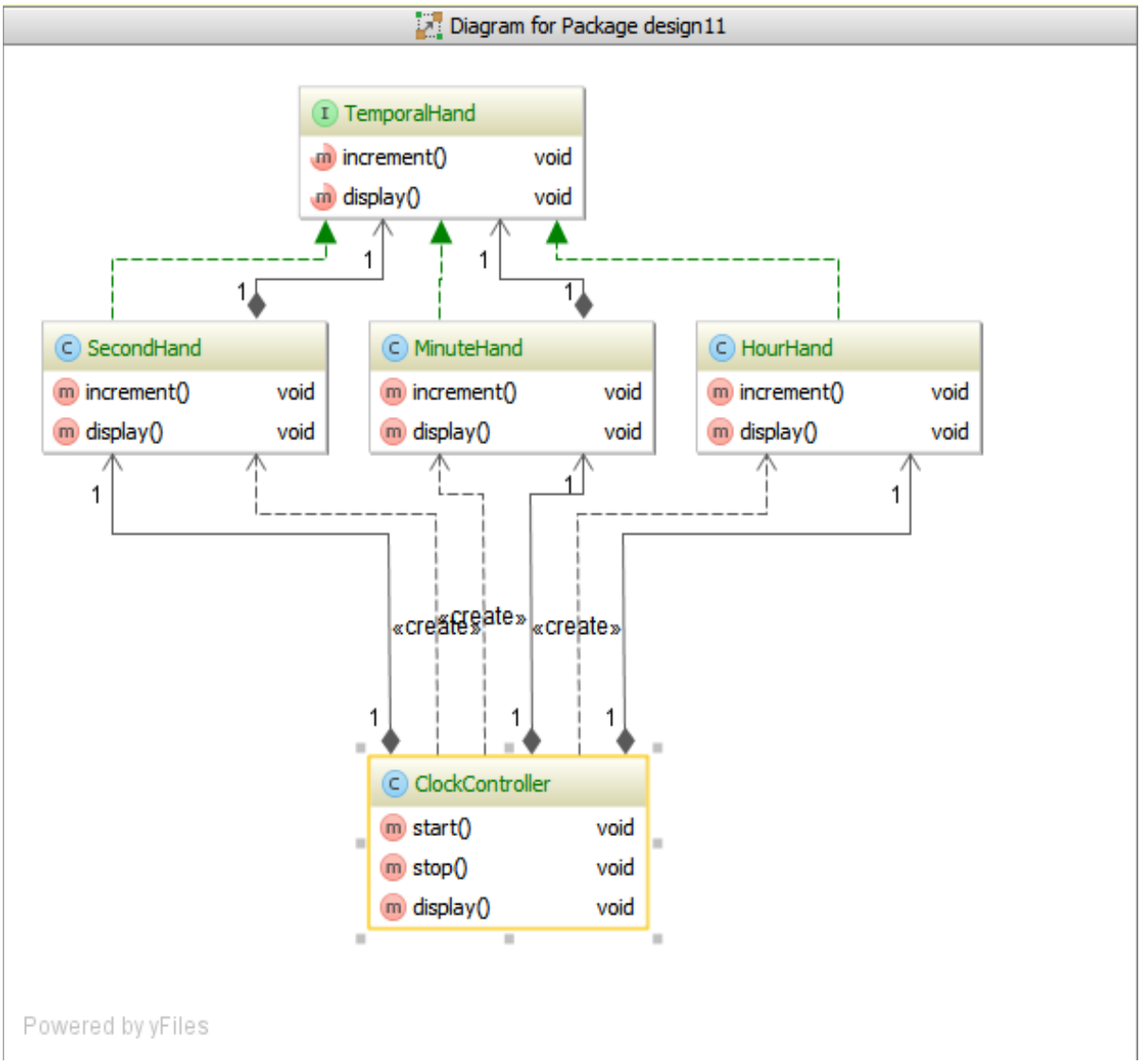
Notes

Class Diagram For the Solution



Q 231. How would you simulate a digital Clock in Object Oriented Programming Language?

The simplistic design consists of creating a second hand, which when completes 60 seconds, advances the minute hand by 1 minute. Similarly Hour hand is advanced by 1 hour when minute hand completes its 60 minutes. This can be emulated in software by registering MinuteHand as an Observer to SecondHand, and HourHand as a observer to MinuteHand. The only real subject in this case is the Second Hand which keeps ticking once every second.



ClockController is the class that manages the overall state of the clock and provide us the option to start & stop the clock.

Let's now see the very basic implementation of this clock.

```
public class ClockController {
    HourHand hourHand = new HourHand();
    MinuteHand minuteHand = new MinuteHand(hourHand);
    SecondHand secondHand = new SecondHand(minuteHand);
    private volatile boolean start =true;

    public void start() throws InterruptedException {
        while(start){
            secondHand.increment();
            Thread.sleep(1000);
        }
    }

    public void stop(){
        start = false;
    }

    public void display(){
        hourHand.display();
        minuteHand.display();
        secondHand.display();
    }
}


public class SecondHand implements TemporalHand{
    private int count = 0;
    private final TemporalHand observer;

    public SecondHand(TemporalHand observer) {this.observer = observer;}

    @Override
    public void increment() {
        count++;
        if(count>=60){
            count=0;
            observer.increment();
        }
    }

    @Override
    public void display(){
        System.out.println("seconds = " + count);
    }
}

public class MinuteHand implements TemporalHand {
    private int count =0;
    private final TemporalHand observer;

    public MinuteHand(TemporalHand observer) {this.observer = observer;}

    @Override
    public void increment() {
        count++;
```

```
        if(count>=60){
            count=0;
            observer.increment();
        }
    }

    @Override
    public void display() {
        System.out.println("minutes = " + count);
    }
}


public class HourHand implements TemporalHand {
    private int count =0;

    @Override
    public void increment() {
        count++;
        if(count>=24){
            count=0;
        }
    }

    @Override
    public void display() {
        System.out.println("hours = " + count);
    }
}

public interface TemporalHand {
    void increment();

    void display();
}
```

Q 232. How would you design an elevator system for multi story building? Provide with request scheduling algorithm & Class diagram for the design.

For a single elevator system, normally two different queues are used to store the requests. One queue is used to store upward requests and other queue used to store the downward requests. Queue implementation used is the BlockingPriorityQueue which maintains the priority of its requests based on the floor numbers. For upward motion, the lower floor number has the higher priority and opposite for the downward motion of elevator. A 2 bit flag can be used to store the current direction of the elevator where 00 represents Idle, 01 for upward motion, 11 for the downward motion.

A Bit Vector can be used to map the floor numbers, and if someone presses the floor button then the corresponding bit can be set to true, and a request is pushed to the queue. This will solve the duplicate request problem from outside the elevator at the same floor. As soon as the floor request is served, the corresponding bit is cleared and the request is removed from the queue.

The actual software application for handling elevator requires lot of interaction with the hardware and is out of scope for this book.

For further reading
http://thought-works.blogspot.in/2012/11/object-oriented-design-for-elevator-in.html

Q 233. Given two log files, each with a billion username (each username appended to the log file), find the username existing in both documents in the most efficient manner?

Hashing technique could be utilized to solve this problem.

Pseudo Code

for 1st file
read each line,
hash into their abc..xyz buckets depending on the start of the letter of the word. (26 buckets for A to Z to form something like a 26 by xxx table)
then sort each 26 rows in the hash table and delete duplicates

for 2nd file
sort and delete duplicates
for each line/name, find match in the hashtable created earlier.
if match found, output to another file name.

B+ sorting is surely another possible solution we can look for.
http://en.wikipedia.org/wiki/B%2B_tree

Q 234. Design DVD renting system, database table, class and interface.

In order to make a Online DVD rental Store we would require a database, web server (Jetty), hibernate layer to access the database, Restful Webservices (Jersey), HTML and JavaScript for the GUI.

```
@Entity
@Table(name = "DVD")
class DVD {
   final int charge;
   final String name;
   final String id;
   String category;
}

class RentalFacade{
   void rentDVD(DVD dvd) {}
   void returnDVD(DVD dvd) {}
   int calculateRent(DVD dvd) {return 0;}

   public List<String> searchByCategory(String category){
      List<String> categoryList = null;
      try {
         org.hibernate.Transaction tx = session.beginTransaction();
         Query q = session.createQuery("from DVD where category like '%" + category + "%')");
         categoryList = (List<Category>) q.list();
      } catch (Exception e) {
         e.printStackTrace();
      }
      return categoryList;
   }
}

@Path("/dvd")
public class MachineResource extends AbstractResource {
   @GET
   @Path("getAll")
   @Produces({APPLICATION_JSON})
   public Response getAllDVD() {
      return Response.ok(fromContext(DAO_FACADE, RentalFacade.class).searchByCategory("")).build();
   }

   protected<T> T fromContext(String key, Class<T> type) {
      return  type.cast(context.getAttribute(key));
   }
..
}
```

For further reading -
https://netbeans.org/kb/docs/web/hibernate-webapp.html

Chapter 6

Puzzles & Misc

Q 235. Why man holes are round in shape?

There are multiple reasons for that
1. Round shape is easy to machine compared to any other shape, thus it reduces the effort and cost.
2. Round objects are easy to move by rolling them, thus labour is reduced.
3. Round shape objects can't fell into the hole, for other shapes its not true.

Q 236. Solve two egg problem?

What we need is a solution that minimizes our maximum regret. The examples above hint towards what we need is a strategy that tries to make solutions to all possible answers the same depth (same number of drops). The way to reduce the worst case is to attempt to make all cases take the same number of drops.

As I hope you can see by now, if the solution lays somewhere in a floor low down, then we have extra-headroom when we need to step by singles, but, as we get higher up the building, we've already used drop chances to get there, so there we have less drops left when we have to switch to going floor-by-floor.

Let's break out some algebra.
Imagine we drop our first egg from floor n, if it breaks, we can step through the previous (n-1) floors one-by-one.

If it doesn't break, rather than jumping up another n floors, instead we should step up just (n-1) floors (because we have one less drop available if we have to switch to one-by-one floors), so the next floor we should try is floor n + (n-1)
Similarly, if this drop does not break, we next need to jump up to floor n + (n-1) + (n-2), then floor n + (n-1) + (n-2) + (n-3) …

We keep reducing the step by one each time we jump up, until that step-up is just one floor, and get the following equation for a 100 floor building:

x + (x-1) + (x-2) + (x-3) + (x-4) + … 3 + 2 + 1 >= no. of floors
x (x+1)/2 = no. of floors

This summation, as many will recognize, is the formula for triangular numbers (which kind of makes sense, since we're reducing the step by one each drop we make) and can be simplified to:
x (x+1) / 2 >= 100
This is a quadratic equation, with the positive root of 13.651

If the number of floors = 10 then x =4.

Q 237. There are 100 doors all closed initially. A person iterates 100 times, toggling the state of door in each iteration i.e. closed door will be opened & vice versa. 1st iteration opens all doors (1x multiplier) 2nd iteration closes 2,4,6,8 .. doors (2x multiplier) 3rd iteration opens 3,6,9,12 ... doors (3x multiplier) and so on. In the end of 100 iterations, which all doors will be in open state?

A door will be left closed if even number of door toggling iterations for that particular door. For all odd number toggling, the door state will be open.

Now let's see how we can approach towards the solution.
In mathematics the number of factors that can divide a given number can easily decide the solution. The doors will be ultimately closed where the number of factors of that number are even.

For example,

The factors of 1 are 1 and itself. It has one factor - odd number of factors
The factors of 2, 3, and 5 are 1 and themselves. They have two factors - even number of factors
The factors of 4 are 1 and itself, as well as 2. 4 has three factors - odd number of factors

After analysis we can find that every number has even number factors except the number which are perfect squares (and thus have odd number of factors).

1. Factors of 1 = 1x1; (no. of factors is 1) - door will be toggled only 1 time in very 1st iteration.
2. Factors of 4=1x4, 2x2, 4x1; (no. of factors is 3) - door will be toggled in 1st, 2nd and 4th iteration.
3. Factors of 9=1x9, 3x3, 9x1; (no. of factors is 3) - door will be toggled in 1st, 3rd and 9th iteration.
4. Factors of 16 = 1x16, 2x8, 4x4, 8x2, 16x1; (no. of factors is 5) - door will be toggled in 1st, 2nd, 4th, 8th and 16th iteration.

So in case of perfect squares, both the factors are same number, that trick will solve this problem.
So we can conclude that odd number of iterations will happen for all the numbers which are perfect square numbers and those doors will be in Closed state (initially opened)
1,4,9,16,25,36,49,64,81,100

For every other room, door will remain Open after 100 iterations.

Q 238. What is probability of a daat, hitting closer to centre of a circle rather than circumference?

Answer is 25%

Let's understand this question using the figure shown here. Daat will hit closer to centre of circle than the circumference when daat hits in a area whose radius is half the area of circle. The area of interest is shown in blue color in the given figure, and total area is the blue + yellow area.

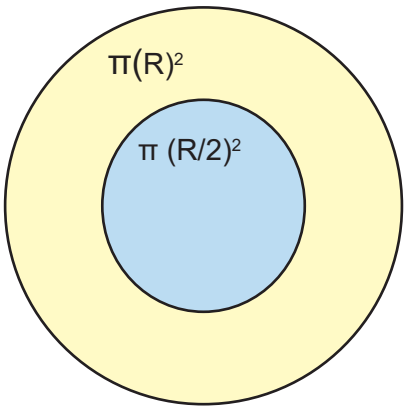
Probability of hitting the daat closer to circle

= (expected area) / (total area)

= π x $(R/2)^2$ / (π x R^2)

=1/4

=25%



Q 239. What is financial Instrument, Bond, equity, Asset, future, option, swap and stock with example each?

Bond
A bond is a debt security under which the issuer owes the holders a debt and depending on the terms of the bond, is obliged to pay them interest/coupon and to repay the principal at a later date, termed as maturity.

Stock
Constitutes the equity stake of its owners.

Equity
Equity is the residual claim or interest of the most junior class of investors in asset after all liabilities are paid.

Asset
Assets are economic resources. Anything tangible or intangible that is capable of being owned/controlled to produce value and that is held to have positive economic value is considered an asset. In other words, Asset represents value of ownership that can be converted into cash.
Capital = Assets - Liabilities

Coupling
Coupling is the degree to which each program module relies on each one of other module in the software application.

Q 240. Toolkit & Resources for a Java Developer.

Essential Tool kit for Java Developer

IntelliJ IDE (Free Community Edition- http://www.jetbrains.com/idea/download/)
H2 or Java DB for in memory operations & testing.
Twitter Bootstrap & bootstrap extra CSS library
Jquery for Ajax & Java Script
Freemarker for generating the HTML
Struts 2 as the MVC framework with Rest Plugin
Hibernate as JPA Provider
Spring Framework
Servlets
Restful WebServices - Spring/Jersey Implementation
Tortoise SVN for Version Control System, or Git
Apache Web Server
Tomcat Java EE server
Jetty Server
HTML 5
Firebug extension for Firefox
Cygwin for Unix simulation
Logback, SLF4j, Log4j api for logging

Books

Design Patterns in Java - Head First
Concurrency In Practice by Brian Goetz
Effective Java 2nd Edition by Joshua Bloch
Algorithms 4th edition : http://algs4.cs.princeton.edu/home/
Cracking the Coding Interview

Technology Forums

http://www.geeksforgeeks.org/fundamentals-of-algorithms/
http://www.careercup.com
http://www.stackoverflow.com

Great Tutorials Articles

Few articles on Java 6 at IBM website
http://www.ibm.com/developerworks/views/java/libraryview.jsp?search_by=5+things+you+did
Concurrency In Practice by Brian Goetz - http://www.briangoetz.com/pubs.html
Java Articles : http://www.oracle.com/technetwork/articles/java/index.html
http://www.oracle.com/technetwork/articles/java/index.html
Java SE Tutorial : http://docs.oracle.com/javase/tutorial/index.html
Java 7 docs: http://docs.oracle.com/javase/7/docs/

Video Tutorials on the web

http://www.youtube.com/user/nptelhrd

Q 241. Unsolved Interview Questions.

Question : How would you implement a optimistic database locking using various techniques? How would you avoid non-repeatable reads?
Hint: https://blogs.oracle.com/carolmcdonald/entry/jpa_2_0_concurrency_and
https://blogs.oracle.com/enterprisetechtips/entry/preventing_non_repeatable_reads_in
Question : What is the best way to implement PriorityQueue?
Question : How does quick sort works?
Question : How would you print rhyming words from a stream using PipedStreams?
list of words -> reverse -> sort -> reverse
http://www.cs.nccu.edu.tw/~linw/javadoc/tutorial/java/io/streampairs.html#PIPED
http://www.dca.fee.unicamp.br/projects/sapiens/calm/References/Java/tutorial/essential/io/pipedstreams.html
Question : How would redesign executors framework by reducing the thread contention?
http://today.java.net/article/2011/06/14/method-reducing-contention-and-overhead-worker-queues-multithreaded-java-applications
Question : How would you find nth highest salary from a table using SQL?
"Select * From Employee E1 Where N = (Select Count(Distinct E2.Salary) From Employee E2 Where E2.Salary >= E1.Salary)"
Question : How would you delete duplicate rows from a table?
Question : How will you find the Lowest Common Ancestor in a Binary Search Tree?
Question : Your are give a file with milions of numbers in it. Find the duplicate numbers in it.
Question : Quickest way to Find the missing number in an array of 1 to n. slot in array is blank, Find 2 missing number..
Hint : Priority queue with distance from the current floor as the inverse of priority
Question : How would you traverse Binary Search tree using iterations?
Question : what do you understand by MVC design pattern?
Question : What is dependency Injection design pattern?
Question : TreeMap vs ConcurrentSkipListMap?
Question : Inline sorting of an array where max element in the array is equal to size of the array itself? and most of the elements are unique.
Question : InOrder traversal of a binary search tree results in Ascending order sorting of elements.
Question : Find the top 10 most frequent words in a file.
Hint: use hashmap to store word occurrence count and for every 10th occurrence update the values in a binary heap (PriorityQueue) with the maximum count. PriorityQueue can just store the top 10 elements
http://stackoverflow.com/questions/742125/how-to-find-high-frequency-words-in-a-book-in-an-environment-low-on-memory
http://stackoverflow.com/questions/358240/space-efficient-data-structure-for-storing-a-word-list
http://en.wikipedia.org/wiki/Bloom%5Ffilter
Question : How would you lower thread contention using Dequeue.
http://today.java.net/article/2011/06/14/method-reducing-contention-and-overhead-worker-queues-multithreaded-java-applications
Question: What is huffman encoding technique?
http://en.wikipedia.org/wiki/Huffman_coding
Question : Use Stack to implement a Queue… you can use 2 stacks
Question : Use 2 stacks to implement getMin() function with O(1) complexity.
Question : BFS and DFS of a Tree
Question :3 ants are at 3 vertices of a triangle. They randomly start moving towards another vertex. What is the probability that they don't collide?
Hint - 6/8, total scenarios = 2x2x2 =8, collision will occur for 6 (2 scenarios will not encounter collision)
Question :You have 2 ropes which burns in 1 hr each..how will you measure 45 min of time?

Continued on 253

Question :"given 1000 bottles of juice, one of them contains poison and tastes bitter. Spot the spoiled bottle in minimum sips?"
Question : How would you detect a circular loop inside a linked list?
Question : How would you calculate size of a linked list having circular loop in it?
Circular Linked List - take 2 pointers
increment one by +1;
increment other by +2
they will first meet in N iterations
length of stem = n/2;
Question : There is a sorted Array of Integer but the array is rotated. How would you fix it using binary search? why choose binary search?
http://leetcode.com/2010/04/searching-element-in-rotated-array.html
http://www.careercup.com/question?id=2800
http://xorswap.com/questions/77-implement-binary-search-for-a-sorted-integer-array-that-has-been-rotated
Question: How would you mirror a binary tree?
Question : I want to implement 2 different display score boards for the IPL cricket match, one specific to IPL another for T20. Which design pattern will rescue you in this case?
Question : What is contract between equals() and hashcode() method?
Question : How would you write a hashcode() method for a class having two fields? Can we multiply hashcode with a random number?
Question : Explain Fork and Join with concrete example?
http://www.oracle.com/technetwork/articles/java/fork-join-422606.html
http://www.oracle.com/technetwork/articles/java/trywithresources-401775.html
http://docs.oracle.com/javase/tutorial/essential/concurrency/forkjoin.html
http://fahdshariff.blogspot.in/2012/08/java-7-forkjoin-framework-example.html
http://www.javabeat.net/2012/06/simple-introduction-to-fork-join-framework-in-java-7/
http://www.vogella.com/articles/JavaConcurrency/article.html
Question : http://tech-queries.blogspot.in/2008/11/sort-array-containing-0-and-1.html
Question : Why wait() and notify() are at Object level rather than Thread level?
Answer : http://javarevisited.blogspot.in/2012/02/why-wait-notify-and-notifyall-is.html
Question : Why not to choose static factory method in place of singleton design pattern?
Question: There is an JPA entity having lazy loading items. You want to use this entity to render a view page which will display this entity. What all options do you have to overcome the lazy loading in this case?
http://wiki.eclipse.org/Introduction_to_Mappings_(ELUG)#Indirection_.28Lazy_Loading.29
http://java.dzone.com/articles/jpa-lazy-loading
http://www.javacodegeeks.com/2012/07/four-solutions-to-lazyinitializationexc_05.html
Question : find median of two sorted array
Hint :http://www.geeksforgeeks.org/median-of-two-sorted-arrays/
Question : When no direct method is found, the most specific method is chosen by the JVM for a method call? example?
Question : What are various techniques for achieving thread safety in Java : Immutable, ThreadLocal, Synchronized access, non-blocking algo using CAS.
Question : http://www.geeksforgeeks.org/find-the-two-repeating-elements-in-a-given-array/
Question : Write a print method for printing Top N numbers from an array?
Question : Given a collection of Trades. Write an algorithm to remove duplicates based on Tic# and sorting based on NAV.
Question : Design a vending machine.
Question : How would you implement a BoundedBuffer using Lock and Condition?
http://www.baptiste-wicht.com/2010/09/java-concurrency-part-5-monitors-locks-and-conditions/
Question : Give an example of timed locking using explicit locking in Java.

continued on 254

Hint : http://codeidol.com/java/java-concurrency/Explicit-Locks/Lock-and-ReentrantLock/
Question : Fibonacci Series using various techniques - recursive, iterative, Big O(1)
Question : Left Outer Join vs Right Outer Join?
Hint : http://en.wikipedia.org/wiki/Join_(SQL)#Inner_join
Question : Angle between Minute Hand and Hour hand of a clock?
Question : How is a TreeSet implemented in Java?
Question : How does google analytics works without causing a extra load on your web server?
Hint : Javascript is used to hit a google server with the required identifier then the new site is visited.
Question: How would you avoid data corruption by a web page which allows a update a database row, and 2 users try to update the same row simultaneously.
Question : How does batch update works in Java?
Question : Find the first common ancestor of two given nodes in a binary tree in o log n space complexity and O(n) time complexity.
Hint - 1. do DFS, 2. during DFS if you find one of the nodes store the stack contents (path from the root) repeat the same process for the second node. this requires 2nlogn space. 3. Now compare both of these paths from the root, the last common node in the path is the first common ancestor. this takes logn time avg case and n in worst case
Question : What is a BloomFilter? How is it better than hashmap in certain casess?
http://en.wikipedia.org/wiki/Bloom%5Ffilter
"Sort An Array Containing '0' And '1'
Sort An Array Containing '0','1' And '2'
http://tech-queries.blogspot.in/2008/11/sort-array-containing-0-and-1.html
Dutch flag algo."
http://en.wikipedia.org/wiki/Dutch_national_flag_problem
Discuss the numbering system.
traceroute command and nslookup
Natural ordering in search operation and stable search
TRIE
Sorting and Searching
Atomic package and CAS, non blocking algorithms
Database indexes clustered indexes, query plan etc, Outer and Inner Join
UNIX stuff cut grep ps etc, piping the output
Question : How would you implement a Trie in Java. Suppose you want to implement a Auto-suggest functionality using Java where user presses a letter and all the words starting with that letter are listed in the suggestion box. Which datastructure would you choose for this scenario?
Question: How would you implement ThreadPool in Java 1.4?
Question: How would you implement StringBuffer class, so that it doesn't create un-necessary imtermediate string objects upon modifications?
Question : Write a method to count the size of a LinkedList given a Node. There could be a circular loop inside the list, hence your method should be smart enough to handle such situation gracefully.
Hint : http://crackinterviewtoday.wordpress.com/2010/03/16/loop-in-a-linked-list/
Question : How would you design a FileSystem for showing just the hierarchy of files? File Interface & then File and DIR as the subclasses.
How would you map department and employee table into Java Objects? What kind of relationship would you put there? Lazy loading?
Question : Which object construction mechanism you prefer in Spring DI - constructor based or setter based?
Hint- setter based injection is preferred mechanism for injecting dependencies, but at times constructor based injection is preferred when mandatory dependencies need to be injected.
Question - We have list of One million numbers on which some mathematical function needs to be applied, How would you make algorithm concurrent?

Continued on 255

Hint - You can use Executor framework for spawning multiple workers, and use a queue to feed the one million input numbers. There could be another strategy where you divide the one million numbers into N parts and feed each of these parts to one worker. You can also think of atomic package for handling such scenario.
There is an un-ordered stack of 5 elements and we have a method nextMinimum() which returns us the subsequent next minimum element in O(1). suppose we have 2,3,1,4,5 as the elements, then first invocation will return us 1, second 2, third 3.
Hint - maintain a queue which maintains the sorted references to the underlying stack.
What is Spring bean lifecycle?
What is embeddable in JPA
How do you performance tune an application - By GC, by changing algorithm, using different data structure which is more appropriate for a given scenario.
Question: Design multi-player Chess Game using Class Diagrams.
Question: Design a Restaurant Reservation system.
Solution : http://www.careercup.com/question?id=15062886
Question: Design SkyDrive.
http://www.careercup.com/question?id=14692764
http://thought-works.blogspot.in/2012/11/object-oriented-design-for-cloud-based.html
Question: Design Online Auction Site.
Solution : http://thought-works.blogspot.in/2012/11/object-oriented-design-for-online.html
Question: Design a Train & reservation system. Give class structure and design UML
Solution :
http://www.careercup.com/question?id=3220674
Question: Write a 2 Thread application where one thread prints even number and the other thread prints odd numbers and both of them act in a synchronized manner.
Question : Security and Performance Tuning of a REST and Ajax Application
http://www.oracle.com/technetwork/articles/java/securityperf-rest-ajax-177520.html
http://www.oracle.com/technetwork/articles/javaee/jax-rs-159890.html
Question : How does Tree Balancing works? left and right rotation?
Question : How does ReentrantReadWriteLock works internally?
Question : What do you understand by volatile keyword?
http://www.ibm.com/developerworks/java/library/j-jtp06197/index.html
Question : Why would you use Suffix Tree for searching?
Write a chapter on glossary. Mention few keywords used in Java and financial word. jargons
Can you tell me with example the Usage of ThreadLocal class? Calendar class, JDBC transaction management, etc.
Question : Synchronization of getClass() in case of inheritance, will lock the actual class rather than the whole hierarchy.
Question: How would you convert a sorted integer array to height balanced binary tree?
Question: Discuss about non-blocking algorithms using CAS?
https://www.ibm.com/developerworks/java/library/j-jtp04186/
Discuss the numbering system.
traceroute command and nslookup
TRIE and Sorting and Searching
Database indexes clustered indexes, query plan etc, Outer and Inner Join
UNIX stuff cut grep ps etc, piping the output
Question : What are the ways to achieve thread-safety in a concurrent program?
Question : How will you deal with ConcurrentModificationException?
Question : How to expose a method over JMX using MBean?
Thread.interrupt() puzzle.
Discuss on External Sorting

Continued on 256

http://www.careercup.com/question?id=83696
Question : Design a solution to print employee hierarchy in Java given a employee record.
Article : Java Software in Harmony with the Hardware - Mark Thompson
http://mechanical-sympathy.blogspot.in/2012/10/compact-off-heap-structurestuples-in.html
http://mechanical-sympathy.blogspot.in/2011/07/false-sharing.html
http://mechanical-sympathy.blogspot.in/2011/12/java-sequential-io-performance.html
Question : How would you find kth highest number in a list of n Numbers?
http://en.wikipedia.org/wiki/Selection_algorithm
http://stackoverflow.com/questions/3628718/find-the-2nd-largest-element-in-an-array-with-minimum-of-comparisom
http://stackoverflow.com/questions/251781/how-to-find-the-kth-largest-element-in-an-unsorted-array-of-length-n-in-on
Question: Design Coffee maker (Vending Machine)..provide some class diagram
Solution : http://www.careercup.com/question?id=3171714
http://stackoverflow.com/questions/7067044/java-algorithm-to-solve-vendor-machine-change-giving-problem
Question: How do you represent the following expression in "class design": (5*3)+(4/2)? How would an algorithm that computes the value of this expression work?
http://www.careercup.com/question?id=65911
Question : How would you design Money class, which holds currency as well as amount of money?
Hint : http://stackoverflow.com/questions/1359817/using-bigdecimal-to-work-with-currencies
http://stackoverflow.com/questions/11434938/display-curreny-value-in-india-as-rs-100-using-java
http://www.javapractices.com/topic/TopicAction.do?Id=13
How would you perform tree rotation to balance a tree.
Question : Write a function which logs/writes messages to files asynchronously . Multiple thread should be able to write to different files concurrently e.g. if thread A want to write to a file ‘FA’ at the same time thread B wants to write to a file ‘FB’ then both threads should be able complete operation concurrently. Threads which wants to write messages to file shouldn’t block for file related i/o .
Sample interface.

```
Log{
void log(filename, message);
}
```

http://mentablog.soliveirajr.com/2013/02/inter-socket-communication-with-less-than-2-microseconds-latency/
http://mentablog.soliveirajr.com/2012/12/asynchronous-logging-versus-memory-mapped-files/
http://mentablog.soliveirajr.com/2012/11/inter-thread-communication-with-2-digit-nanosecond-latency/

Question : Design a price making system for a wholesale dealer, where user can subscribe/un-subscribe online for any product to receive the real time prices. System will internally subscribe to different vendors to get the product prices, aggregate them and return the best price to customer. Vendors may broadcast new prices every few seconds and customers would like to see all the price updates until he un-subscribe for that product. Dealer may also like to add some commission on every product price to remain in business and make some profit.
Assume system will have limited number of vendors and products but can have high number of concurrent customers requesting for product prices.
Primary concern for the customers to have minimal latency in the price updates.
Question : U are given binary search tree. How will you check whether it is balanced or not.
Question : U have UI and service. UI making 5000 request and service can handle only 500 request. I am okay with slow response. But how will make sure all 5000 requests are processed
Question : U have a tree with each node has link to its parent. You are given left most child node of the tree. How will you get right most child node of the tree.

Question : Write regular expression which checks for Any occurence of 'A' followed by two or more 'B' followed by any occurrence of 'A'
Question : Merge 2 sorted arrays in constant space and minimum time complexity.
http://www.cs.ubc.ca/~harrison/Java/MergeSortAlgorithm.java.html
http://thomas.baudel.name/Visualisation/VisuTri/inplacestablesort.html

Question: What will happen if in a try block we throw an exception but in the finally block we return a int value?

```
public class MyFinalTest {
  public int doMethod(){
      try{
         throw new Exception();
      }
      finally{
         return 10;
      }
   }
public static void main(String[] args) {
      MyFinalTest testEx = new MyFinalTest();
      int rVal = testEx.doMethod();
      System.out.println("The return Val : "+rVal);
   }
}
```

Hint - the method call will return 10 instead of throwing the exception.

Question : How would you write a simple Struts 2 Interceptor which will log the request and response to an Invocation?
http://www.dzone.com/tutorials/java/struts-2/struts-2-example/struts-2-interceptors-example-1.html
Question : What are different scopes of a Bean in Spring framework?
Answer : singleton – Return a single bean instance per Spring IoC container
prototype – Return a new bean instance each time when requested
request – Return a single bean instance per HTTP request.
session – Return a single bean instance per HTTP session.
globalSession – Return a single bean instance per global HTTP session.
Question : How to create a singleton bean in Spring by calling a custom initialization method (for eg instance())?
Answer : provide factory method attribute in the bean declaration, as shown below
<bean id="mySingleton" class="org.shunya.MySingleton" factory-method="getInstance" />

Q 242. Sample Unsolved Puzzles?

1. The Calendar Problem: A man has two cubes at his desk. Every day he arranges both cubes so that the front faces show the current day of the month. What numbers are on the faces of the cubes to allow this?
2. A 100 doors: You have 100 doors in a row that are all initially closed. You make 100 passes by the doors starting with the first door every time. First time through you visit every door and toggle the door (if the door is closed, you open it, if it's open, you close it). The second time you only visit every 2nd door (door #2, #4, #6). The third time, every 3rd door (door #3, #6, #9), etc, until you only visit the 100th door. What state are the doors in after the last pass or in general after n pass?
3. 100 Jars: There are 100 jars containing infinite number of marbles each of 10 grams except one jar which contains all marbles of 9 grams. How would you find out which jar it is in exactly one weighing? For the same problem what would be your approach if more than one jars had 9 gram marbles?

4. Racing horses: There are 25 horses in a racing competition. You can have race among 5 horses in a particular race. What would be the minimum number of races that will be required to determine the 1st, 2nd and 3rd fastest horses?
5. Two persons: one always speaks truth other always speaks false. You don't know who is what, You are new to the city, you are allowed to ask exactly one question to find out the direction (for e.g. south or north), what question would you ask?
6. Three persons: one always speaks the truth, second always lies and third randomly speaks the truth or lies. You are allowed to ask each entity one or more yes-no questions. You are allowed to ask three such questions. You must deduce the identities of the three entities with the answers you get. How should you ask the three questions?
7. Cake cutting: There is a rectangular shaped cake of arbitrary size; we cut a rectangular piece (any size or orientation) from the original cake. Question is how you would cut the remaining cake into two equal halves in a straight cut of a knife. And obviously you can't cut the cake by its cross section.
8. The light bulb problem: You have three light bulbs in a sealed room. You know that initially, all three light bulbs are off. Outside the room there are three switches with a one-to-one correspondence to the light bulbs. You may flip the switches however you like and you may enter the room once. How should you flip the switches to determine which switch controls which light bulb?
9. You have a cylindrical glass with 100% full of water. You have to make it 50% (half). Condition: You are not supposed to use any scale or any type of measuring instrument.
10. A problem of probability: You are a prisoner sentenced to death. The Emperor offers you a chance to live by playing a simple game. He gives you 50 black marbles, 50 white marbles and 2 empty bowls. He then says, "Divide these 100 marbles into these 2 bowls. You can divide them any way you like as long as you use all the marbles. Then I will blindfold you and shuffle the bowls. You then may choose one bowl randomly and remove ONE marble from it. If the marble is WHITE you will live, but if the marble is BLACK... you will die." How do you divide the marbles up so that you have the greatest probability of choosing a WHITE marble?
11. Pirates on deck: Five pirates discover a chest full of 100 gold coins. The pirates are ranked by their years of service, Pirate 5 having five years of service, Pirate 4 four years, and so on down to Pirate 1 with only one year of deck scrubbing under his belt. To divide up the loot, they agree on the following:
The most senior pirate will propose a distribution of the booty. All pirates will then vote, including the most senior pirate, and if at least 50% of the pirates on board accept the proposal, the gold is divided as proposed. If not, the most senior pirate is forced to walk the plank. Then the process starts over with the next most senior pirate until a plan is approved. The pirate's preference is first to remain alive, and next to get as much gold as possible. The most senior pirate thinks for a moment and then proposes a plan that maximizes his gold, and which he knows the others will accept. How does he divide up the coins?
What plan would the most senior pirate propose on a boat full of 15 pirates?
12. What is next number in the series:
a> 1, 11, 21, 1211, 111221, 312211...
b> 1, 20, 33, 400, 505, 660, 777,8000, 9009...
13. Sneaking Spider: A rectangular room measures 7.5 meters in length and 3 meters in width. The room has a height of 3 meters. A spider sits 25 centimeters down from the ceiling at the middle of one of the short walls. A sleeping fly sits 25 centimeters up from the floor at the middle of the opposite wall. The spider wants to walk (i.e., move along the walls, floor, and ceiling only) to the fly to catch it. How can the spider reach the fly, walking just 10 meters? Is it even possible?
14. The Fuse Problem: I have a box of one hour fuses. If I set one end of a fuse on fire, I know that the fuse will burn all the way to the other end in EXACTLY one hour. However, the fuses may burn unevenly [ie - it may take 59 minutes to burn the first half of a fuse, but only 1 minute to burn the other half]. Furthermore, all of the fuses may burn unevenly at a different rate. The only thing we know for sure is that each one takes 1 HOUR to burn completely. The Question: Given 2 of these fuses and a lighter, how can I time out 45 minutes precisely?
15. Dropping eggs: There is a building of 100 floors If an egg drops from the Nth floor or above it will break If it's dropped from any floor below, it will not break You're given 2 eggs Find N, while minimizing the number of drops for the worst case.

16. MIT Mathematicians: Two MIT math grads bump into each other while shopping. They haven't seen each other in over 20 years.
First grad to the second: "How have you been?"
Second: "Great! I got married and I have three daughters now."
First: "Really? How old are they?"
Second: "Well, the product of their ages is 72, and the sum of their ages is the same as the number on that building over there..."
First: "Right, ok... Oh wait... Hmm, I still don't know."
Second: "Oh sorry, the oldest one just started to play the piano."
First: "Wonderful! My oldest is the same age!"
How old was the first grad's daughter?
17. Crazy guy on the plane: A line of 100 airline passengers is waiting to board a plane. They each hold a ticket to one of the 100 seats on that flight. (For convenience, let's say that the nth passenger in line has a ticket for the seat number n.) Unfortunately, the first person in line is crazy, and will ignore the seat number on their ticket, picking a random seat to occupy. All of the other passengers are quite normal, and will go to their proper seat unless it is already occupied. If it is occupied, they will then find a free seat to sit in, at random. What is the probability that the last (100th) person to board the plane will sit in their proper seat (#100)?
18. Escape from Alcatraz: A prisoner stays at the maximum security prison on Alcatraz Island. The prison is in shape of 4X4 cells, the prisoner stays at top right cell with all other cell having a guard, only escape from prison is from bottom left cell (see diagram for further clarification). Here are the rules for a successful escape from the prison.

- The prisoner has to escape from the prison overnight by killing all the guards.
- He can only move vertically or horizontally, no diagonal movement is allowed.
- As soon as the prisoner enters a cell he has to kill the guard.
- If he sees the dead guard again he will go mad for 24 hrs out of guilt, i.e he can't go to same cell twice.

Provide an escape route.
19. Transporting bananas: You are standing at point A with 3000 bananas and a faithful camel. Your destination is point B which is exactly 1000 kms away. The objective is to transport as many bananas as possible to point B, under the following conditions.
1. Only the camel can carry bananas.
2. The maximum load that the camel can carry at a time is 1000 bananas.
3. The camel consumes 1 banana for every km that it travels. (Irrespective of direction of travel or load)

20. There are 10 marbles of equal weight except for one which weighs a little more. Given a balance how many weighing are required to deduce the heavier marble. What would be the answer for N marbles? Your answer should consider the worst case.

21. Imagine a disk spinning like a record player turn table. Half of the disk is black and the other is white. Assume you have an unlimited number of color sensors. How many sensors would you have to place around the disk to determine the direction the disk is spinning? Where would they be placed?

22. There are 3 baskets. One of them has apples, one has oranges only and the other has mixture of apples and oranges. The labels on their baskets always lie. (i.e. if the label says oranges, you are sure that it doesn't have oranges only, it could be a mixture) The task is to pick one basket and pick only one fruit from it and then correctly label all the three baskets. How do you do it?

23. Prime pairs: Pairs of primes separated by a single number are called prime pairs. Examples are 17 and 19, 5 and 7 etc. Prove that the number between a prime pair is always divisible by 6 (assuming both numbers in the pair are greater than 6). Now prove that there are no 'prime triples'.

24. Suicidal Monks: There is a group of monks in a monastery. These monks have all taken a vow of silence.

They cannot communicate with each other, and all they do is pray in a common room during the day and sleep at night. As well, they have no mirrors in the compound. One day, the head monk calls them all together and says "Tonight while you sleep, I will place a black X on some of your foreheads. When you awaken, continue your normal activities. But once you determine that you have an X, you must wait until night, and then kill yourself". So from then on, they pray together by day, and each night some may commit suicide. The question: if there are N monks with Xes, how many days does it take for the N monks to commit suicide?

25. The Monty Hall problem: You are given a choice between three doors -- 1, 2, and 3. One of them contains a trip to Hawaii, and the other 2 are empty. You pick one. Then he opens one that you didn't pick, and it's empty. He gives you the chance to switch your choice to the other door you did not choose. Should you change your original selection?

Q 243. Few sample UNIX questions.

How to invert the grep to exclude a given search pattern?
Assume we have a book-order.txt file and we want to search all orders which do not contain word "gurgaon", then we can use the following command for searching.
>grep -v gurgaon book-orders.txt

How to count number of lines in a file in UNIX?
wc command can be used to count bytes, lines, characters, and words in a given file
>ls -ltr | wc -l
>wc -c abc.txt
>wc -l abc.txt

How would you find which UNIX operating system you are running?
Following commands can be used to know the OS details -
>uname -a
>arch
How would you find the number of cpu's in your computer?
>cat /etc/cpuinfo

How would you print the running processes?
>ps -elf | grep <identifier>

How would you search a running process in UNIX
>psg <process identifier>

How would you kill a running process in UNIX?
>kill -9 <pid>

How would you tail a log file for listening to changes?
>tail -100f <log file name>

How will you extract specific lines from a text file?
The following command will extract 4th and 5th line from <filename> and put it into output.txt
>sed -n "4,5p" <filename> | less > output.txt

How would you clean a windows file on unix for \n and \r characters?
>dostounix <filename>

Show size of a directory in UNIX?
>du -sk <dir>

How will you run a command from history?
The following commands will list all history processes and then run 12th item from history
>history
>!12

Find what process using which ports in Windows
netstat -aon | find /i "<port number>"

Q 244. Top Java Interview Questions?

1. What do you understand by thread-safety? Why is it required? And finally, how to achieve thread-safety in Java Applications?
   Hint : discuss the need for the concurrent programming, using volatile, synchronization, Immutability & Atomic packages to address the concurrency problems. Discuss the Java Memory Model. Impact of final keyword in Java. Differences between wait and notify method in Object class.
2. What are the drawbacks of not synchronizing the getters of an shared mutable object?
3. Discuss the Singleton Design Pattern? How to make it thread-safe? Discuss the Double Check Locking?
4. Can Keys in HashMap be made Mutable? What would be the impact in that case?
5. How would you implement your own ThreadPool in Java? Discuss the designing part.
6. How would you implement a Stack or a Queue in Java? It must be synchronized.
7. Discuss Big O notation for calculating relative performance of Algorithms. How do various collection methods perform in terms of Big O Notation?
8. Implement Queue using an ArrayList.
9. What are the types of Inner classes with example of each?
10. What is a tree map? Discuss its underlying implementation i.e. red-black binary tree.
11. There are 1 million trades, you need to check if a given trade exists in those trades or not. Which Collection would you chose to store those 1 million trades and why?
    Hint : think from time complexity point of view and why HashSet could be a better data structure for storing these trades assuming we have sufficient memory to hold those items.
12. What is difference between StringBuilder and String? Which one should be preferred.
13. In a program, multiple threads are creating thousands of large temporary StringBuilder objects. Life span of all those objects is 1 GC cycle. But somehow we are getting JVM pauses in our application. How would you troubleshoot the problem?
    Hint : Think from GC tuning perspective, setting the appropriate survivor ratio for proper eden space.
14. What are memory generations in Hot Spot VM? How generational GC's work?
15. What is difference between Primary Key and Unique Key?
16. What is clustered and non-clustered index?
17. What is Outer and Inner Join?
18. What is ADT? We do not need to know how a data type is implemented in order to be able to use it.
19. Are you familiar with a messaging system i.e. MQ? What is a QueueManager? Why do you think the Queue is so important in banking world?
20. How would you make an application asynchronous? Can Message Queues help achieving this?
21. How to achieve loose coupling in your application?
22. What is TDD and how it helps Agile methodology of software development?
23. How to make a class Immutable? What purpose Immutablity solve?
24. What is difference between Callable and Runnable?
25. What are Inheritance strategies in JPA?
26. Discuss Internals of HashMap and ConcurrentHashMap?
27. What is best way to store Currency Values in Java application?
28. What is AtomicInteger and how it is useful in concurrent environment?
29. What are key principles while designing Scalable Software Applications?
30. What does Collections.unmodifiableCollection() do? is it useful in multi-threading environment?
31. How would you add an element to a Collection while iterating over it in a loop?
32. There are 3 Classes A, B and C. C extends B and B extends A, each class has a method named add() with same signature (overriding). Is it possible to call A's add() method from Class C? Reason?
33. How would you write a simple implementation for Struts 2 Interceptor which just logs the request and response of an Action?

Q 245. What is the Typical Interview Coverage for Core Java Candidate?

Java Basics
OOP prinicples, overloading, overriding, exception handling, garbage collection, Immutability, Generics
Collections
New collections introduced in the latest version of JDK, internals of HashMap, ConcurrentHashMap, time complexity of various collection methods.
Serialization
Custom serilaization using Serializable and Externalizable interfaces. Serializing legacy classes, construction invocation in serialization.
Data structure and Algorithms
List, Queue, Binary Search Tree, Time complexity of operations, sorting, seraching, etc.
Design Patterns
Singleton, thread-safe siungleton, decorator, adaptor, strategy, builder, factory, observer, etc
Database & Hibernate
Database indexes, types of algorithms for indexes, types of indexes, SQL, SQL tuning, query plan, outer and inner joins, relationships in database (OneToOne, OneToMany, ManyToMany), inheritance strategies in JPA, lazy loading, handling concurrency in database transactions.
MVC Framework
MVC design patterns, Interceptors, Dependency Injection, Servlets, Filters, Struts 2, Restful Webservices, SOA, Spring Framework etc
Misc.
Continuous Integration, Unit Testing, TDD, GC tuning, Maven , UNIX commands, Autosys Jobs, Scripting Language, etc

Q 246. What is the art of writing resume?

Resume should be small and crisp, no interviewer likes to read 5 pages long resume of the candidate. The style that i follow for writing my resume contains
Summary
Clearly showing my key skills along with my currency job profile and the profile I am looking for in my new Job.
Employment
Brief description of professional experience. It should list the employer and the projects very clearly with minimum words. Employer usually look for candidate's role and responsibilities in the project and hence these we must provide all the necessary information.
Open Source Projects
When we gain experience then our social contribution matters a lot, specifically if we are looking for long term career in programming. You should list down all the Open Source involvement providing the proper links to the hosting.
Education
Academics should be mentioned in this section, highlighting details about the masters, bachelors and higher schooling.
Skills
Detailed description about the key skills should be mentioned in this section. We should never flood this section with the each and every bit of technology that we have worked in past. This section should contain only those skills that you are comfortable working with and desiring to look in your new role.

In the next 3 pages i have shown my resume as the template, which you can follow if you like !

Q 247. Sample Questions on Swings framework.

1. Is swings multi-threaded model? No, Swings is a single Threaded Model for GUI. But you are always free to spawn more threads to manage the computation tasks. But all GUI related events are handled by Single Event Dispatch Thread also known as EDT.
2. What is event dispatch thread in swings?
3. Using swings worker for background tasks in Swings. Discuss the benefits of swing worker thread. http://docs.oracle.com/javase/tutorial/uiswing/concurrency/worker.html
4. Is swings thread-safe? No, Swing Components are not Thread Safe, you need to call update on Swing Components from within Event Dispatch Thread.
5. How would you make your swings application responsive?
6. Discuss the only thread-safe methods in swings like - repaint(), revalidate() and invalidate().
7. EventDispatchThread is not a deamon thread and hence the application does not closes upon closing the GUI.
8. What if UI is freezing or pausing for long time. How would troubleshoot such scenario? Hint - Never use Event Dispatch Thread to do heavy tasks, computation demanding tasks should be delegated to new Threads.
9. What is difference between paint() and repaint() method?
10. How would you achieve abstraction in swings code ?
11. You have a multi-threaded swings application where a Thread calculates the result of a calculation and wants to update the same in the Swings Table component. How should he achieve that ? Discuss EDT's invoke() and invokeLater() method.
12. How will you enable JTextField for multilingual characters? Hint - Just use "Arial Unicode MS" font that can display the unicode characters.

Q 248. What are the Interview questions that most candidates answer wrongly?

1. Is it required to synchronize the accessor of a shared mutable object in case of multi-threading? If yes, why?
2. In what scenario StringBuilder should be preferred over String class?
3. I am working on an application where millions of temporary StringBuilder objects are being created, due to which application is facing big system wide GC pauses, how would you rectify the problem, assuming that memory available can not be increased to great extent.
4. How Atomic updates are different from their synchronized counterpart?
5. When do we get the ConcurrentModificationException? What constitutes the structural modifications in a collection?
6. Is it possible to write a method in Java which can swap two int variable? What if we change the type from int to Integer?
7. What is difference between Class and the Instance level locking?
8. Can you prove a scenario where thread starvation occurs?
9. Is it a mendate to make all fields final inside a immutable class? If yes, why?
10. How to fix Double Check Locking?
11. What is Java Memory Model? Who should read it?
12. What is upper bound and lower bound in generics?
13. What happens when an exception is thrown from a try block which has no catch clause. But in finally block, a value is returned by the method? Discuss the scenario.

About Author

Munish Chandel (मुनीश चंदेल)

Munish is Java developer having 12+ years of experience working for investment banks, consulting and product companies. As of this writing, he was working for a MNC in Healthcare domain. By academics, he holds a degree in mechanical engineering from NIT Hamirpur in year 2005. His hobbies include trekking, biking, photography, exploring ancient Hindu Texts (वेदांत - ब्रह्मसूत्र, गीता और उपनिषद्), Ayurveda (आयुर्वेद), yog-asana (योगासन), and Developing Java Applications, etc.

Author contributes to the below mentioned Open Source Projects in Java -

- DLI Downloader for Digital Library of India (https://github.com/cancerian0684/dli-downloader)
- Punter for Dev Assistance (https://github.com/cancerian0684/Punter)
- AIDS - Autonomous Integrated Deployment Software (https://github.com/Buddh/AIDS)

Connect with Munish:
cancerian0684@gmail.com
https://www.linkedin.com/in/munishchandel/

Thank you, your feedback is most important

For feedback and suggestions, please drop me a note.

ISBN 978-1-304-69865-0
90000
9 781304 698650