Cultivate skills to create your own path...

This e-book covers following topics

1. Core Java (including Java 8 lambda expression & stream api)
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, Infosys, TCS, Expedia, etc

3rd edition, 2015

About Author
Munish Chandel (मुनीश चंदेल)

Munish is Java developer having 9+ years of experience working for investment banks, consulting and product 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 8, 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 an experienced Java Developer in his 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. And there could definitely be more efficient ways to solve the given problem compared to what is mentioned in this book. The approach shown here is limited to the knowledge of the author.
- Collection of Questions in Core Java 8, 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. Be a knowledge seeker for lifetime and keep enjoying new learnings!

Munish Chandel
cancerian0684@gmail.com
http://linkedIn.com/munish.chandel
August 2014

This page is Intentionally left blank.

Contents

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

Concurrency in Java 127

Object Oriented Design 196

IT Industry and Psyche of Developers

My Opinion in Indian Context

Question: 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.

Question: 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.

Question: 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.

Question: 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.

Question: 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.

Question: 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.

Question: What are advantages of acquiring 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.

Question: 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.

Question: 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.

Question: 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.

Question: Quality work never reaches India, is that right ?

Its true for many companies, even in 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 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 changes is in progress.

Chapter 1

Core Concepts, Spring & Hibernate

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

1. Understand the requirement and the business, asks questions to resolve ambiguities.
2. 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 very good test coverage (End to End and Unit Tests) keeps a developer away from last minute stress at production deployment time.
3. Automate all non-productive mundane tasks related to deployment, e.g. Salt Stack or AIDS for Dev Ops
4. Take a software change request only if it is really required. If there is no value addition to the customer then don't do it.
5. 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.
6. 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.
7. Document interfaces and reasons instead of implementation.
8. 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
9. Use pair programming when bringing someone new up to speed and when tackling particularly tricky problems which are hard to KT otherwise. This also leads to smooth landing for the new employees.
10. 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)
11. Work in small steps with frequent feedback and correction to avoid the last minute surprises (Agile).
12. Continuous Integration environment is must for rapid bug free, coordinated development. Tools like TeamCity, Hudson, Jenkins, etc are can be leveraged for Continuous Integration.
13. 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!!!

Q 2. What are good books for reference on Java Programming ?

Core Java	Framework	Design Patterns, Algorithms & Misc.
1. OCJP Oracle Certified Programmer for Java 2. Java 8 for the Really Impatient, Cay S. Horstmann 3. Java Concurrency in Practice by Brian Goetz 4. Head First Java 5. Effective Java, 2nd Edition	6. Spring in Action by Craig Walls, 4th Edition 7. Java Persistence with Hibernate 8. Rest In Action, Manning 9. jQuery in Action	10. Head First Design Patterns, Kathy Sierra 11. Algorithms, Robert Sedgewick, Kevin 12. Data Structures and Algorithms Made Easy In Java, Narasimha Karumanchi 13. Cracking The Coding Interviews, 150 Programming Questions and Solutions by Gayle, Careercup 14. Programming Pearls, 2nd Edition 15. http://www.geeksforgeeks.org/

Q 3. What is Growth Road Map for a Java Developer?

A deep understanding of basic software components is must for a developer who wants to write Scalable Enterprise/Internet Applications from scratch using Java. 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 (using Test Driven Development, Pair Programming and Continuous Integration), SOA, etc
6	**Web Tier**	Servlet basics, HTTP Basics, JavaScript & JQuery, RESTful web services, MVC Design Pattern, Spring, etc
5	**Database**	SQL, PLSQL, Database Indexes, 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.
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, Tree Insertion, Tree balancing using left & right rotation, Sorting (quick, merge, external merge, bucket, counting), Binary Search, BFS, DFS, Topological Sorting using Graph, Dijkstra's algorithm, Dynamic Programing & Sorting Algorithms
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) Familiarity with Version Control System, Continuous Integration, Design Patterns (Abstract Factory, Builder, Singleton, Observer, Template, Strategy, Visitor, Decorator, Facade, Flyweight, etc) Memory Management (Heap, Stack, Memory Generations in Java) Logarithm, Big-O notation, Bitwise Manipulation & Recursion Number System (2's complement, binary, hexadecimal, etc)

Skills Matrix for a Java Developer

Q 4. Why should I choose Java for Software Development? What are Pros and Cons of Java 8 ?

Java Pros

1. Java is free, download it and start creating your own applications
2. Plenty of third party libraries, frameworks & IDE for faster development (spring, hibernate, intelliJ, etc)
3. Platform independent, write once run on most modern platform (Unix, Windows, Mac, 32/64 bit Hardware)
4. Supports Object Oriented Programming, easy to model real life scenarios into object model
5. 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 (Work Stealing Algorithm), non-blocking algorithm using CAS, Stream API, Parallel Streams, CompletableFuture, Parallel Array Operations, Atomic Values, LongAccumulator, etc.
6. Very good support for Internationalization
7. Memory management is automatic by use of garbage collector (G1, Concurrent Mark Sweep, parallel scavenge garbage collector, etc)
8. Pure Java Byte code running on 32 bit JVM works perfectly fine on 64 bit platform
9. 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 ay C/C++ written application
2. Normal Java is not good for real time systems because of "stop the world garbage collector pauses".

Q 5. 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.*

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

Q 6. What are four basic principles of OOP?

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.

Question: What is aggregation, how is it different from composition ?

Both of these are special type of association and differ only in weight of relationship.
Composition is stronger form of "is part of" relationship compared to aggregation "has a".
In composition, the member object can not exist outside the enclosing class while same is not true for Aggregation.

Q 7. What are the key paradigms for Developing the Clean Object Oriented Code?

1. Program to an Interface (or the Super Type) not the implementation.
2. Interacting Classes should be loosely coupled among themselves.
3. Code should implement tight encapsulation. Use of public and static variables should be avoided whereever possible, they introduce coupling and make testing of classes tough. Avoid the Singleton Design pattern whereever possible.
4. Always reuse the code using Inheritance, Composition and Utility Methods. Strictly follow the Do not Repeat Yourself (DRY) principle.
5. Has-A relationship is better than Is-A relationship because it offer more flexibility, see Decorator Design Pattern for more details.
6. In case of multi-threaded applications, use immutable objects to represent the state.
7. Make proper use of Design Patterns wherever possible.
8. Use up to date software dependencies & make best use of latest technology available to us.

Q 8. 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 (Interview scenario, hypothetical)

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 9. What is Logarithm? Why is it relevant in Software Development?

A logarithm[1] 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.

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.
- 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.

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

Q 10. What do you understand by Big O notation, why is it important in software development ?

Big O Notation[1] is a mechanism used to measure the relative efficiencies of Algorithms in terms of **Space** and **Time**. 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

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

O(1) for put operation

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

Q 11. How would you determine the 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 12. 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 232 elements, i.e., we need, to hold a count value up to 232-1, 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 13. Why Prime Numbers are given much importance in writing certain algorithms like hashcode()?

Prime numbers are very useful for generating hashcode, RSA algorithms, random number generators. String class's hashcode method multiplies its hash value by prime number 31 :

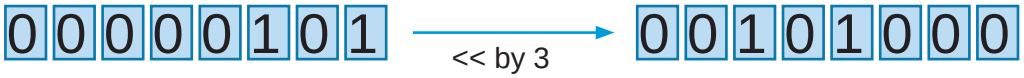
```
int hash =0;
 for (char ch : str.toCharArray()) {
    hash = 31 * hash + ch;
 }
```

A number is either prime number or a composite number (can be factorized into prime numbers). Prime numbers are always unique and can not be divided by any other number except 1. 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 14. What is left shift <<, right shift >> and Unsigned right shift >>> operator in Java? How are these useful?

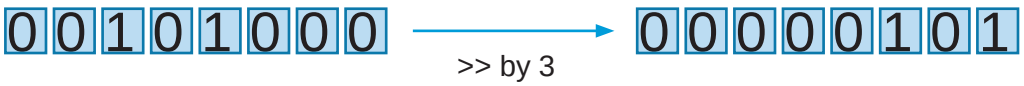
All Integer in Java are of signed type (negative numbers are represented in 2's complementary notation), hence Java provides both signed and unsigned bit shift operators to support signed and unsigned shift of bits.

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 15. What is 2's complement notation system for Binary Numbers?

It is a number format for storing negative numbers in Binary[1]. 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

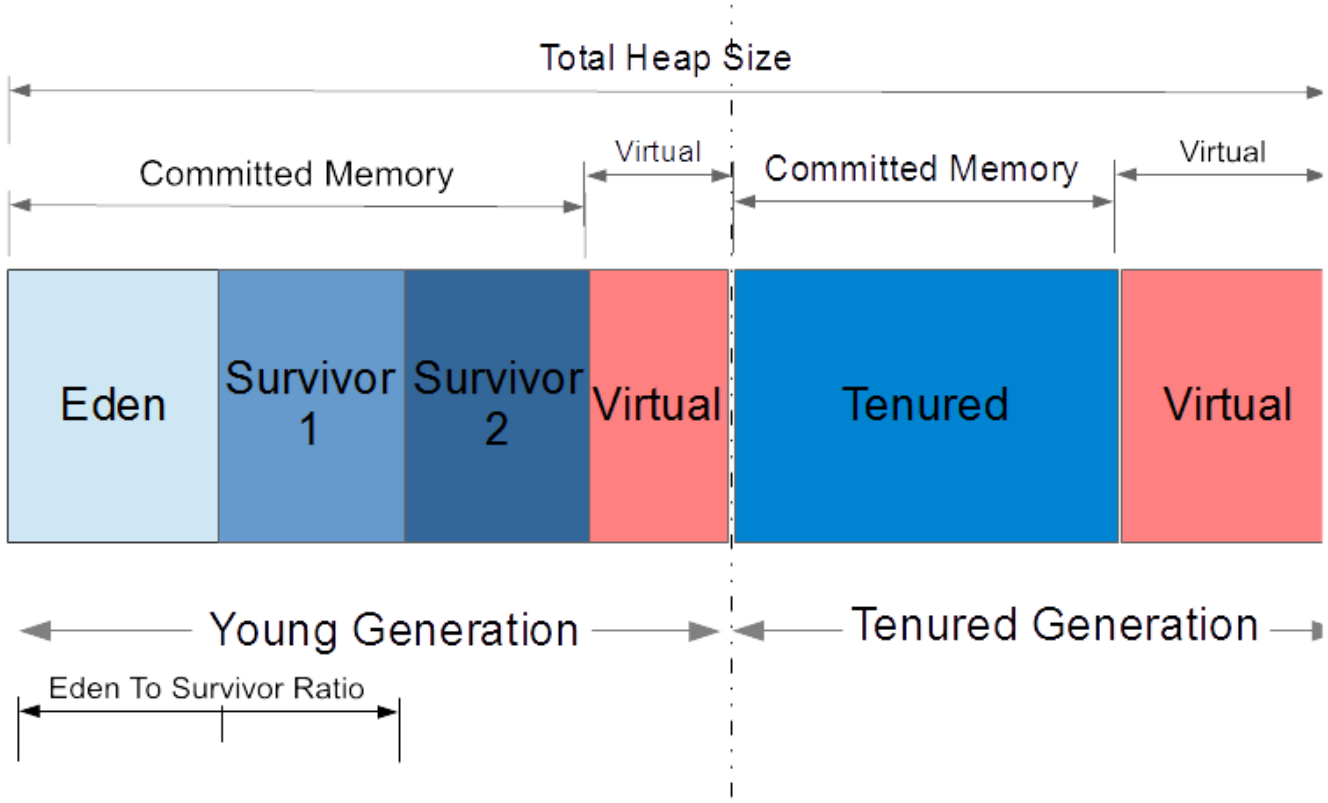
That is +5.

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

Q 16. How Heap space 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 33

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

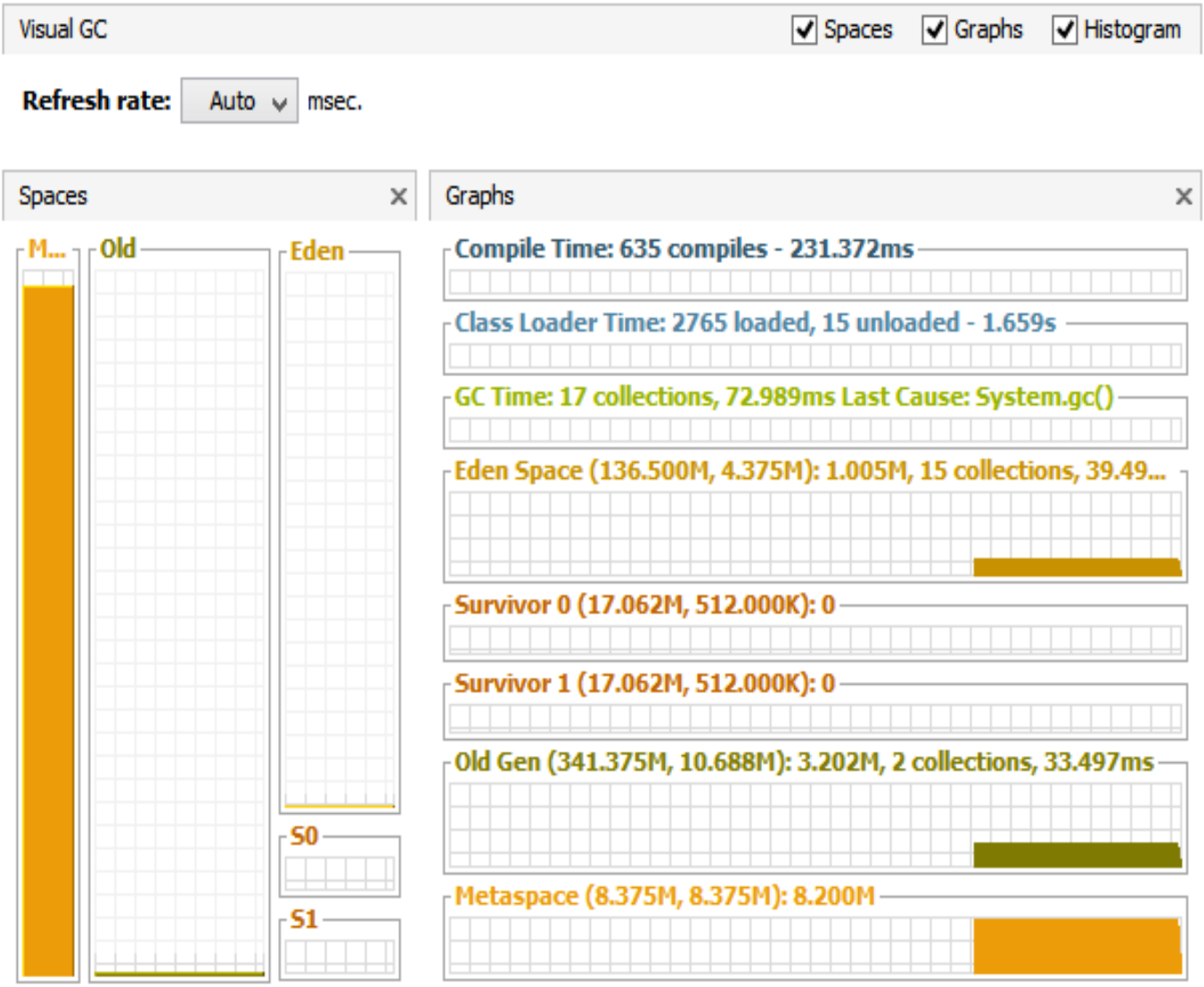
Question: Does GC collects memory from Perm Gen Space ?
Solution : The PermGen space is garbage collected like the other parts of the heap. PermGen contains meta-data of classes and objects (pointers to heap memory allocation). It also includes ClassLoaders that need to be manually destroyed at the end 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 17. 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 18. What is a Binary Tree? Where and why is this used in Java Programs?

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 19. 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 20. 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 21. 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.

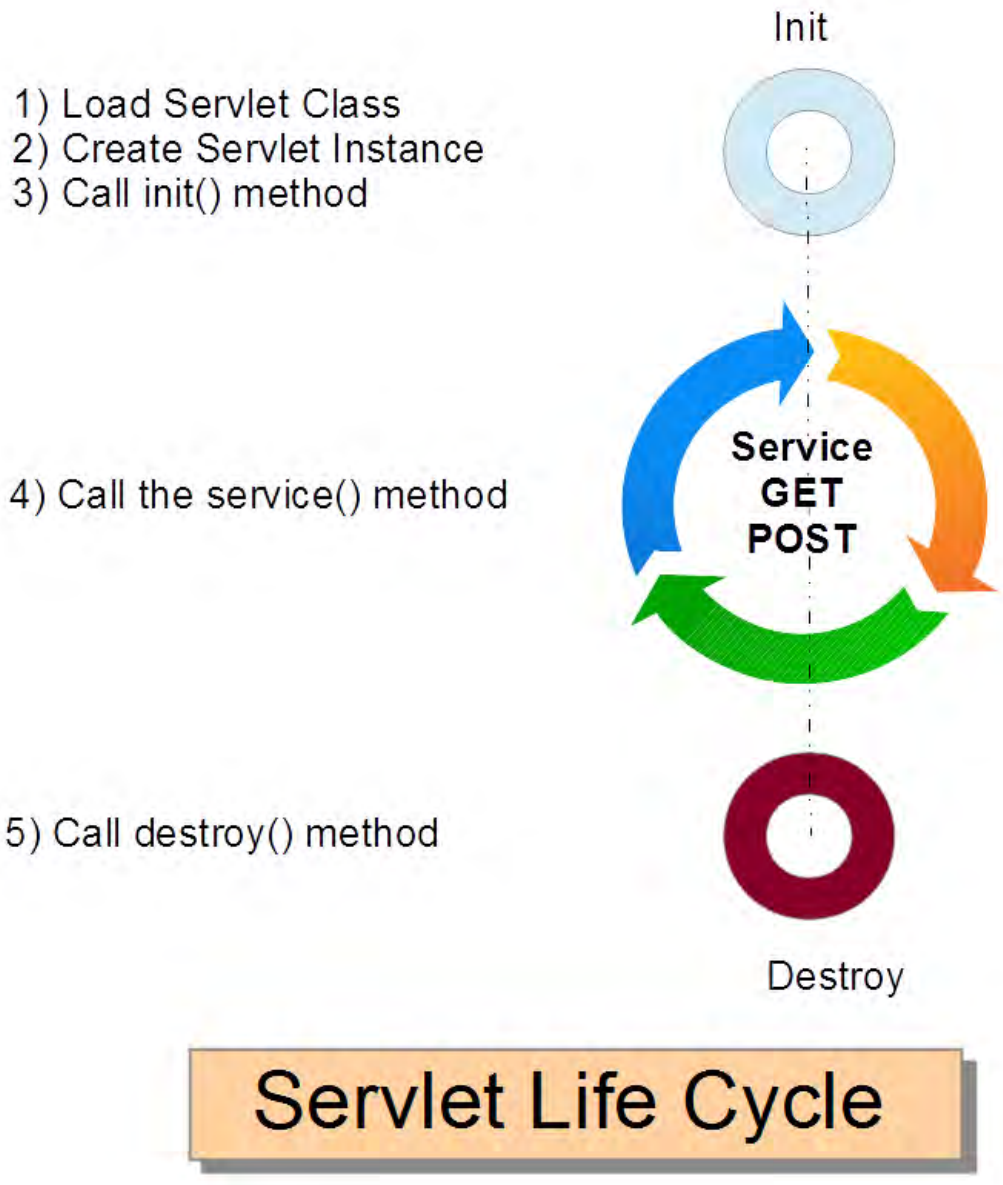
```
public interface Servlet {

public void init(ServletConfig config) throws ServletException;

public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException;

public void destroy();

}
```



Servlet Life Cycle

Q 22. 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 23. How will you write a Recursive Program?

A general structure of any recursion program is like this :

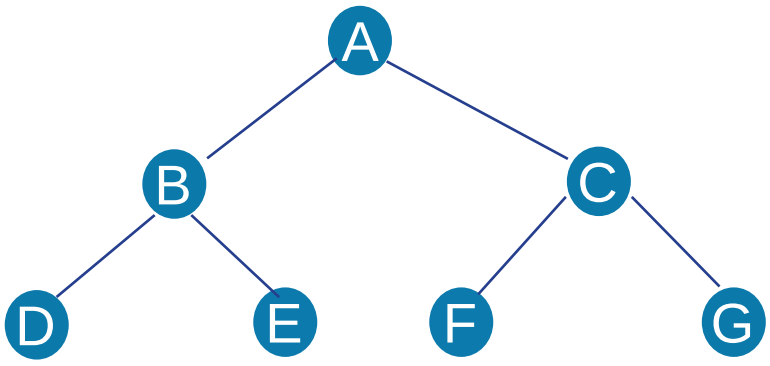
```
if (base condition...){
   // return some simple iterative expression
}
else {// recursive case
   //some work before call
   //recursive call
   //some work after call
}
```

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 24. 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 1 level tree = 1
Total Nodes in 2 level tree = 3
Total Nodes in 5 level tree = 31

Total Number of nodes in a tree with depth 10 will be = $2^{10}-1$ = 1023 nodes

Q 25. 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

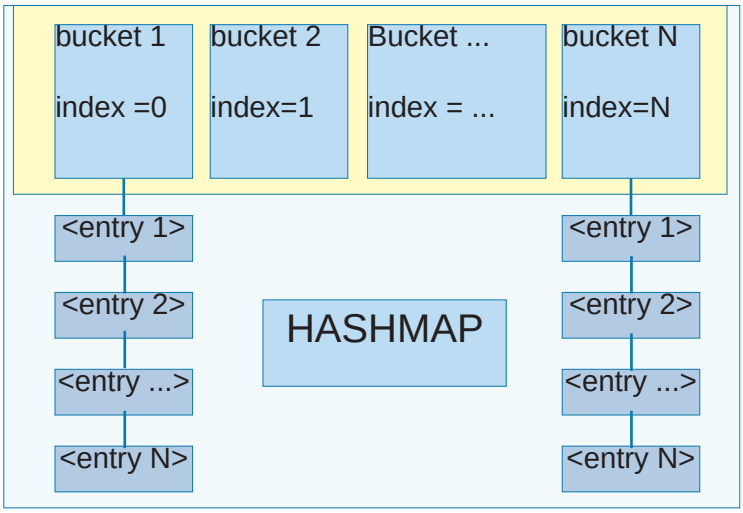
When we add a key-value pair to hashmap, it queries key's hashcode. Hashmap uses that code to calculate the bucket index in which to place the key/value. For example, if hashcode is zero then hashmap will place the key value in 0th bucket. Hashmap strips down the key's hashcode to fit into the existing 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 26. 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 27. 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();
}
```

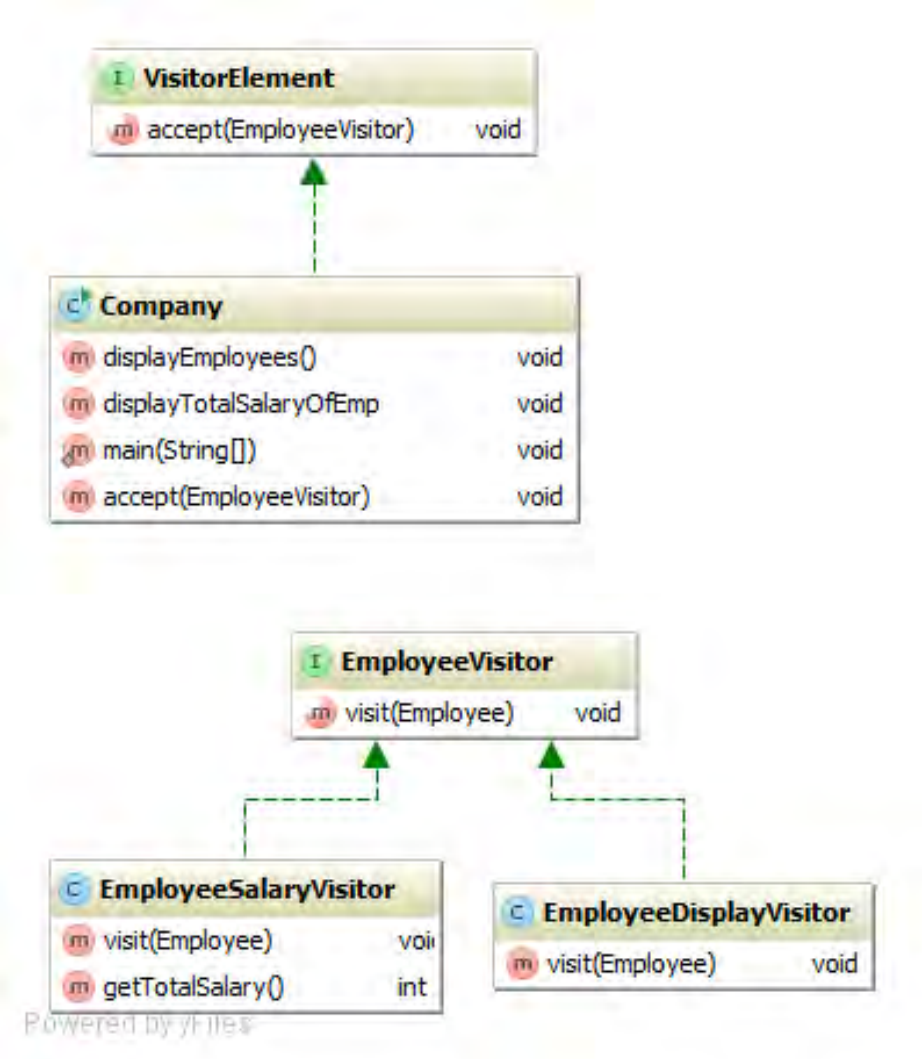
Q 28. Discuss Visitor, Template, Decorator, Strategy, Observer and Facade Design Patterns?

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 47

```
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.

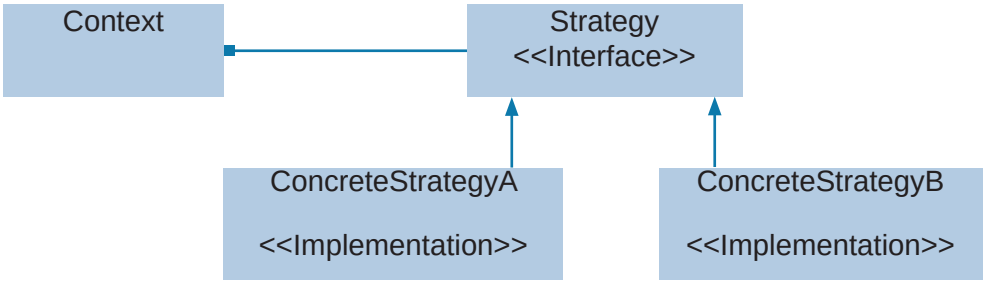
Strategy Design Pattern

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 48

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 29. What is a strong, soft, weak and Phantom reference in Java? Where are these used?

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 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

continued on 49

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 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 30. What are database transaction Isolation levels?

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.

Sample Chapters End Here.