**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. New Features of Java 8. [Lambda expression, Streaming api, G1GC etc.]
2. How to sort a Map<Key, Value> on the values in Java?
3. Optimistic & Pessimistic Locking
4. What will happen with ConcurrentHashMap, if three threads are iterating on CHM and one thread will come and try to remove all items from CHM.
5. Concurrent API to communicate between threads.
6. Difference between Runnable and Callable?
7. How to decide numbers of threads for a Thread-Pool?
8. **Pool Size = [Runtime.getRuntime().availableProcessors(); + 1]**
9. **Pool Size = N \* U \* (1 + W/C)**
10. **Max Performance = 1/ [ (1-P) + P/N ]**
11. Types of Thread Pool.
    1. **[Fixed Thread Pool, Cache Thread Pool, Scheduled Thread Pool, WorkStealPool (Java8)]**
12. Fork-Join Thread Pool
13. Thread Dump Analysis
14. Collection hierarchy
    1. **Set [HashSet, LinkedHashSet, TreeSet]**
    2. **List [ArrayList, Vector, LinkedList]**
    3. **Map [HashMap, LinkedHashMap, TreeMap, WeakHashMap, IdentityHashMap, HashTable]**
15. How to create a class thread safe?
16. java.nio package
17. When would be the need of Custom ClassLoader in Java.
18. Memory Barriers in JVM to handle volatile and atomicity
    1. **[http://java.dzone.com/articles/memory-barriersfences]**
    2. **[http://gee.cs.oswego.edu/dl/jmm/cookbook.html]**
19. What is new HashMap implementation in Java8?
    1. **[Key Objects should implement Comparable interface]**
    2. **[After collision LinkedList Data Structure is now replaced with Sorted Tree]**
20. Object class source code.
21. HashMap source code for custom serialization of object (writeObject() & readObject() methods)
22. sun.misc.Unsafe class in Java
23. How String pool implemented and String.intern() method in Java 6, Java 7 and Java 8.
    1. **http://java-performance.info/string-intern-in-java-6-7-8/**
24. CompletionService vs ExecutorService
25. How to get count of live objects of particular class on JVM. (Hint – With the help of Java PhantomReference and ReferenceQueue classes)
26. Access modifier – Public, Private, Protected and Default for classes, fields and methods.
27. CopeOnWriteArrayList source code implementation.
28. Significance of Inner Classes.
29. Can inner class extends other class?
30. Significance of Anonymous Classes.
31. What is Skeleton and Stub?
32. Design Pattern used in ExecutorService.
33. How PriorityQueue works? Uses Heap Data structure.
34. Java Strong, Soft, Weak and Phantom Reference [<https://javapapers.com/core-java/java-weak-reference/>]

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Spring\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. Method Injection (Prototype scope reference within Singleton scope Class) [Spring Boot using @Lookup annotation for this]
2. Type of advice calls on AOP
   1. **[before, after, around, after-return, after-throw]**
3. Transaction Manager in spring and Propagation Types and Isolation Level

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Hibernate\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. Second level Cache
2. inverse and cascade
3. get() vs load()
4. update() vs merge()
5. update() vs lock()
6. What evict() method do on Hibernate session.
   1. **[ evict() evicts a single object from the session. clear() evicts all the objects in the session. Calling clear() is like calling evict() on every object associated with the session. ]**
7. Native Query

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SQL\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. **Phantom Read**
   1. [ Phantom read occurs where in a transaction same query executes twice, and the second result set includes rows that weren't visible in the first result set. This situation is caused by another transaction inserting new rows between the execution of the two queries. ]
2. **Dirty Read**
   1. [ Dirty read occurs wherein one transaction is changing the tuple/record, and a second transaction can read this tuple/record before the original change has been committed or rolled back. This is known as a dirty read scenario because there is always the possibility that the first transaction may rollback the change, resulting in the second transaction having read an invalid value. ]
3. **Non-Repetable Read**
   1. [ Non Repeatable Reads happen when in a same transaction same query yields different results. This happens when another transaction updates the data returned by other transaction. ]
4. Optimistic & Pessimistic Locking
5. ACID properties of Transaction
6. Normalization Forms
7. Index (Clustered and Non-Clustered Indices )
8. Cursor
9. Execution Plan of Query
10. How to know indexing is getting used query.
11. Aggregated functions
12. Trigger
13. View
14. Materialized View
15. Partition by Range, Partition by Hash, Partition by List (Table partition in Oracle)
    1. **[** [**https://www.youtube.com/watch?v=\_7ZTjeWKel4**](https://www.youtube.com/watch?v=_7ZTjeWKel4) **]**
16. CAP Theorem

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*WEB\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. HTTP vs HTTPS
2. Session Management Types and Handling
3. Get, Post and Delete Methods in JSP or Servlet
4. How to unload servlet?
5. HTML5
6. What new in HTTP2.0?
7. HTTPS with SSL

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Design Model\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. Open and Close model
2. Advantages of static factory method
3. Difference between Coupling and Cohesion?
4. Favor composition over inheritance.
5. Composition vs Aggregation
6. SOLID principles
7. Abstraction, Encapsulation

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Software Development Process \*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. SDLC (Software Development Life Cylce)
2. Waterfall Model.
3. Evolutionary Prototyping Model.
4. Spiral Method (SDM)
5. Iterative and Incremental Method.
6. Extreme programming (Agile development)
7. Imp Link: <https://melsatar.wordpress.com/2012/03/15/software-development-life-cycle-models-and-methodologies/>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Data Structures & Algorithms\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. Sort 8GB text file with 2GB RAM
2. Graph Traversal (BFS and DFS)
3. Tree Traversal without recursion
4. LCA of Binary Tree
5. Threaded Binary Trees
6. Red Black Trees and AVL Trees
7. Reverse a linked list
8. Dynamic Programming
9. String matching
10. Big O Complexity (1, N, 2N, N^2, N^N, LogN, 2LogN)
11. Big O notation for Algorithms.
12. Data structure used in Heap (Tree Based Data Structure)
13. Suitable data structure for many to many relationship between classes. (Student and Course)

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Unit Testing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. What is mocking & stubbing? What are key differences between them? Did you use any mocking framework?
2. What's the difference between faking, mocking, and stubbing?
3. How will you create a mock of class if it is final class?
4. What is test-driven development (TDD)?
5. Logging Writing Reading.
6. How to compare two arrays by JUnit.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*XML\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. XPath

**########################## OLD ##########################**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. #Java: Reflection - (How Its Works)
2. #Java: Comparable vs Comparator (Implementation)
3. #Java: Collection Utility
4. #Java: Class Loader Mechanism in JVM (Bootstrap Loader, Extension Loader, System Loader)
5. #Java: java.util.WeakHashMap
6. #Java: HashMap implementation
7. #Java: Best way to Override HashCode and Equal Methods. (Tips to override these methods.)
8. #Java: Enums
9. #Java: Java4, Java5, Java6, Java7 (Differences)
10. #Java: User Defined Marker Interface
11. #Java: Types of Garbage Collectors (Customization of Garbage Collectors and Heap Generation space)
12. #Java: G1 Garbage Collector
13. #Java: Finalize Method (Requirement)
14. #Java: Collections Class
15. #Java: Java SE 6 HotSpot[tm] Virtual Machine Garbage Collection Tuning.
16. #Java: One Application has its individual JVM or one JVM can run multiple applications concurrently?
17. #Java: How can I start JVM and Pass JVM -Options at startup to JVM?
18. #Java: How ClassPath works in Java?
19. #Java: JIT Compiler
20. #Java: Overloading rules
21. #Java: Overriding rules (return types, arguments, thrown exceptions )
22. #Java: Why HashTable does not allow null as key and values.
23. #Java: User Defined Exception.
24. #Java: Which interface is necessary to implement to use any class as a key in TreeMap ( Comparable or Comparator )
25. #Java: Which class implementation of Map assures ordering? (Hint - LinkedHashMap)
26. #Java: LinkedHashMap Implementation
27. #Java: Difference between Class.forName and ClassLoader.loadClass
28. #Java: what is the difference between collections.synchronized and synchronized block used?
29. #Java: difference between super and extends in java generics
30. #Java: Implementations of Stack in Collection framework (Stack is itself a class)
31. #Java: Difference between NoClassFoundException and ClassNotFoundError
32. #Java: Difference between synchronization and Locks in Java.
33. #Java: Types of Semaphore
34. #Java: Use of Semaphore
35. #Java: How annotation works in Java?
36. #Java: Custom annotation in Java
37. #Java: OOPs Terms in Java : Association, Aggregation, Composition, Abstraction, Generalization, Realization and Dependency
38. #Java: Assert in Java
39. #Java: How can we stop creating instance inside constructor (I guess assert can help in this)
40. #Java: Java generics - Bridge method.
41. #Java: JDBC
42. #Java: Java Cloning types (Deep Copy & Shallow Copy)
43. #Java: How to perform search on specific attribute on Object collection. (Data Structure Used)
44. #Java: RMI (Remote Method Invocation) in Java
45. #Java: Socket Programing in Java
46. #Java: Cache implementation in Java application.
47. #Java: String vs StringBuffer vs StringBuilder
48. #Java: Error vs Exception
49. #Java Checked Exception vs Unchecked Exception

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Java Thread\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. #Java-Thread: Difference between process and thread?
2. #Java-Thread: Semaphore vs mutux (Purpose)
3. #Java-Thread: CyclicBarrier (Purposes)
4. #Java-Thread: CountDownLatch (Purposes)
5. #Java-Thread: Exchanger (Concurrent Package)
6. #Java-Thread: Phaser (Java 7)
7. #Java-Thread: Yeild Method
8. #Java-Thread: Join Method
9. #Java-Thread: ConcurrentHashMap (implementation, Locking mechanism,
   1. Number of Thread which can take read write lock,
   2. How Map knows about read and write lock.)
   3. Note: internally ConcurrentHashMap use default 16 HashMap, each thread acquire lock on difference HashMap.
10. #Java-Thread: CopyOnWriteArrayList (implementation, Copy mechanism, How and when copy get merged with original list.)
11. #Java-Thread: wait notify Mechanism
12. #Java-Thread: Iterator and concurrent Iterator( concurrent iterator are implemented in ConcurrentHashMap and CopyOnWriteArrayList classes, which iterate concurrently)
13. #Java-Thread: ExecutorService (purpose)
14. CompletionService vs ExecutorService
15. #Java-Thread: fail-safe vs fail fast Iterator
16. #Java-Thread: Park Method (Threading)
17. #Java-Thread: Thread Dumps (Analysis)
18. #Java-Thread: Thread-Pool (What is the need of Thread Pool In Java App)
19. #Java Thread: Atomic Variable (How they work)
20. #Java-Thread: Thread Dump Analysis(How to take Thread Dumps)
21. #Java-Thread: Is there any effect on application running on production, if thread dump are taken during production?
22. #Java Thread: How to make threads in sequence execution without using JOIN. Like Three threads T1, T2, T3. Execution must be T1 -> T2 -> T3.
23. #Java Thread: BlockingQueue
24. #Java-Thread: Printing consecutive Even and Odd using two Threads in Java
25. #Java-Thread: which thread method ensure main(String[] args) method is last method to finish execution.
26. #Java-Thread: java.util.concurrent.locks.Condition Interface (method used await(), signal(), signalAll()) (Implementation used in ArrayBlockingQueue class)
27. #Java-Thread: can we have synchronized constructor (No)
28. #Java-Thread: Can we start same thread two times (No, after a thread death you cannot start it again it will throw IlleagleThreadStateException)
29. #Java-Thread: Deployed application getting slow at customer site, how would you check what could be happened to application ( hint : process size)
30. #Java-Thread: ThreadLocal (What and how to use it)
31. #Java-Thread: What does java.lang.Thread.interrupt() method do?
32. #Java-Thread: Transaction handling in multithreading environment.
33. #Java-Thread: java.lang.ThreadGroup class purpose.
34. #Java-Thread: Exception handling in multithreading environment.
35. #Java-Thread: Multiple threads are trying print on one printer, how to manage time between thread so that in between the process of one thread of work other thread could also execute by halting running thread. (Hint: use Yield () method)

# #Java-Thread: [Memory Consistency - happens-before relationship in Java](http://stackoverflow.com/questions/16248898/memory-consistency-happens-before-relationship-in-java)

# #Java-Thread: User of volatile keyword.

# Mo

# #Java-Thread: Read & Write Lock in concurrent API

# #Java-Thread: Multithreading Issues- Atomicity, Visibility, Ordering, Race Condition, Data Race, Live Lock, Starvation, Dead Lock, Thread fairness etc.

1. #Java-Thread: When Double Check Locking got broken?[<http://www.cs.umd.edu/~pugh/java/memoryModel/DoubleCheckedLocking.html>]

# #Java-Thread: What is busy wait in synchronization and how Lock interface has overcome this problem?

# #Java-Thread: Volatile vs Atomic Classes

# #Java-Thread: RejectedExecutionHandler (Used in ThreadPoolExecutor)

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Design Patterns\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. #Design Patterns: Difference between Abstract Factory Pattern and Factory Pattern
2. #Design Patterns: Design pattern used Collections.synchronizedList() method (java Collection framework). (Decorator Pattern I guess)
3. #Design Principle: S O L I D
4. #Design Patterns: Strategy Pattern
5. Proxy Pattern
6. Adapter Pattern
7. Proxy vs Adapter Patterns
8. Command Patterns
9. Observer Patterns
10. Prototype Patterns
11. Decorator Patterns
12. Strategy Patterns
13. Builder Patterns
14. Singleton Patterns

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SQL\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. #SQL: SQL Indexes
2. #SQL: SQL Trigger
3. #SQL: SQL Views
4. #SQL: Complex SQL Queries
5. #SQL: Composite Indexing & Benefit (DB)
6. #SQL: Normalization
7. #SQL: Dynamic Queries

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*XML\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. #SAX Parser vs DOM Parser (XML)

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*JUnit\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. #JUnit: How to test user defined checked exception in test case?
2. #Mockito: Difference between @Mock and @InjectMocks

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Algorithms\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. #Algorithms: Linked-List Implementation
2. #Algorithms: B-Tree (Efficiency) O(log n);
3. #Algorithms: Check if two Linked-List get merge on any point.
4. #Algorithms: Heap Memory (Data Structure ) Note : A heap is a partially sorted binary tree
5. #Algorithms: All sorting algorithms and their efficiency.
6. #Algorithms: Best sorting algorithm
7. #Algorithms: Find minimum number in two dimension array using multi-threads in Java.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SPRING\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. #Spring: Dependency Injection (IOC - Inversion of Control)
2. #Spring: AOP (Aspect Oriented Programming)
3. #Spring: Boilerplate Codes Removal
4. #Spring: Activity (AOP)
5. #Spring: Difference between BeanFactory and ApplicationContext

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Hibernate\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. #Hibernate: Difference between Get and Load methods
2. #Hibernate-Link: http://www.javagyan.com/preparing-for-an-interview/hibernate-interview-questions
3. #Hibernate-Link: http://java-success.blogspot.in/2010/12/hibernate-interview-questions-q.html
4. #Hibernate-Link: http://javadecodedquestions.blogspot.in/2011/12/hibernate-interview-questions.html
5. #Hibernate-Link: http://www.javabeat.net/hibernate-interview-questions/

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*BLOGS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. BLOG: How memory management (Garbage Collection) works in Java and difference types of Garbage Collectors in Java (Done)
2. BLOG: Light on Multithreading Issues- Atomicity, Visibility, Ordering, Race Condition, Data Race, Live Lock, Starvation, Dead Lock etc. (Done)
3. BLOG: Difference between race condition and data race in Java?
   1. BLOG: Assert in Java (Done)
4. BLOG: Difference between super and extends in java generics
5. BLOG: G1GC
6. BLOG: Double Check locking is broken (http://www.cs.umd.edu/~pugh/java/memoryModel/DoubleCheckedLocking.html)
7. BLOG: java.lang.ThreadGroup class purpose.
8. BLOG: Fork-Join API
9. BLOG: Exception Handling in Thread. (Done)
10. BLOG: Java Generics - Bridge method. (Done)
11. BLOG: UnSupportedClassVersionError in Java (Done)
12. BLOG: ClassLoader in Java (Done)
13. BLOG: Java Cloning (Sallow Copy & Deep Copy) (Done)
14. BLOG: Java Access Modifiers (public, protected, default, private, static etc.)
15. BLOG: How to create a class immutable? (Done)
16. BlOG: Java "Pass By Value" Or "Pass By Reference"
17. BlOG: Intercrosses communication in Java
18. BLOG: OOPs Terms in Java # Association, Aggregation, Composition, Abstraction, Generalization, Specialization, Realization and Dependency (Done)
19. BLOG: readResolve() method

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Few Important Topics\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. #Java: Access Modifier
2. Design Patterns
3. Coding Examples
4. Serializations
5. Immutability concepts
6. ###
7. Enum
8. ClassPath
9. Reflection
10. Data Structures
11. UML
12. Concurrency Packages (Executor Service)
13. JUnit (Mocking) (Example with EasyMock)
14. SQL
15. XML
16. WebService
17. Hibernate
18. Spring
19. Agile
20. AJAX
21. HTML
22. Build
23. JMS

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Important Text\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Q1 : Why write a custom class loader?**

**ANS**: The three main reasons for creating a custom class loader:

**To allow class loading from alternative repositories.**

This is the most common case, in which an application developer might want to load classes from other locations, for example, over a network connection.

**To partition user code.**

This case is less frequently used by application developers, but widely used in servlet engines.

**To allow the unloading of classes.**

This case is useful if the application creates large numbers of classes that are used for only a finite period. Because a class loader maintains a cache of the classes that it has loaded, these classes cannot be unloaded until the class loader itself has been dereferenced. For this reason, system and extension classes are never unloaded, but application classes can be unloaded when their classloader is.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Important Links\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

#Atomicity, Visibility and Ordering Concepts

http://jeremymanson.blogspot.in/2007/08/atomicity-visibility-and-ordering.html

#Java Generics Concepts (Bridge Methods & Type Erasure etc.)

http://www.angelikalanger.com/GenericsFAQ/FAQSections/TechnicalDetails.html