Detailed Technical Screening

### Design Patterns

* List and define three fundamental principles of Object Orientated Programming.
* Composition and inheritance - which should you choose, when and why?
* What is open – closed principle?
* What design patterns have you used before? Can you explain one to me? Singleton, Adapter, Proxy, Strategy, Decorator
* How do you ensure the code you develop is up to standard?
* What does refactoring mean to you? Why is it important?
* Describe the design of the current system you are working on. Use whiteboard if possible. (Test of communication skills)
* .Singleton Design pattern (Ways of creating singleton)
* What is difference between proxy pattern and decorator pattern

### Java Basics

* What is the difference between final, finally & finalize?
* List the methods on class Object.
* Describe Copy-on-write List in Java - why would you use it? An ArrayList that has copy-on-write semantics. This means that no synchronization is needed. Iterators cannot throw ConcurrentModificationException.  Expensive for mutative operations.
* Exception Hierarchy - Throwable, Error, Exception, RuntimeException
* Exception thrown from try block; exception thrown from finally block: which one propagates?
* Checked vs unchecked exceptions? How to catch exception without using try catch block.
* What happens if a class implements two parameterized interfaces that both define a method with the same name? ([http://www.angelikalanger.com](http://www.angelikalanger.com/GenericsFAQ/FAQSections/ProgrammingIdioms.html#What%20happens%20if%20a%20class%20implements%20two%20parameterized%20interfaces%20that%20define%20the%20same%20method?) )
* What is the difference in usage between double and BigDecimal? What are their respective disadvantages and limitations?
* Difference between StringBuffer and StringBuilder? Why would you choose to use a StringBuilder object instead of simple String concatenation?
* What is the difference between an abstract class and an interface? What problems do they solve, or what benefits does their use provide?
* How do you make an object immutable? What are the advantages of using immutable objects?
* *Describe generics in Java* - Generics is a term denoting a set of language features related to the definition and use of generic types and methods. Generic types or methods differ from regular types and methods in that they have type parameters.
* *Are there any types that cannot have type parameters in Java and why?*

The following types cannot be generic:   
*Anonymous inner classes:* They can implement a parameterized interface or extend a parameterized class, but they cannot themselves be generic classes.  A generic anonymous class would be nonsensical.  Anonymous classes do not have a name, but the name of a generic class is needed for declaring an instantiation of the class and providing the type arguments.  Hence, generic anonymous classes would be pointless.   
*Exception types:* A generic class must not directly or indirectly be derived from class Throwable .  Generic exception or error types are disallowed because the exception handling mechanism is a runtime mechanism and the Java virtual machine does not know anything about Java generics.  The JVM would not be capable of distinguishing between different instantiations of a generic exception type. Hence, generic exception types would be pointless.   
*Enum types:* Enum types cannot have type parameters.  Conceptually, an enum type and its enum values are static.  Since type parameters cannot be used in any static context, the parameterization of an enum type would be pointless.

* *Why do instantiations of a generic type share the same runtime type? -* a.k.a. describe what is Type erasure in Java Generics - The compiler translates generic and parameterized types by a technique called type erasure .  Basically, it elides all information related to type parameters and type arguments. For instance, the parameterized type List<String> is translated to type List , which is the so-called raw type .  The same happens for the parameterized type List<Long> ; it also appears as List in the bytecode.   
  After translation by type erasure, all information regarding type parameters and type arguments has disappeared. As a result, all instantiations of the same generic type share the same runtime type, namely the raw type.
* *Is List<Object> a supertype of List<String>?* - No, different instantiations of the same generic type for different concrete type arguments have no type relationship.   
  It is sometimes expected that a List<Object> would be a supertype of a List<String> , because Object is a supertype of String. This expectation stems from the fact that such a type relationship exists for arrays:  Object[] is a supertype of String[], because Object is a super type of String . (This type relationship is known as covariance.)  The super-subtype-relationship of the component types extends into the corresponding array types. No such a type relationship exists for instantiations of generic types. (Parameterized types are not covariant.)
* Why might object reflection be useful in Java?
* References in Java (Strong, Weak, Soft, Phantom) How does a Weak Reference differ from ordinary Java references?
* Coding Style - What is defensive coding?
* *Can you rethrow an exception from a catch block without losing the original stack trace? Can you do this if the method you have caught the exception in does not support throwing the exception type if it is a checked exception? -* Yes. You can choose to throw the original caught exception. You would need to wrap the original exception in a new Exception - either a different typed checked if the method supports it or a runtime if no exception are declared. We must use the form that accepts the original exception and not just grab its message.
* *Describe how Java finds a suitable exception handler in a parent method if a child method throws an exception.* - When a Java method completes normally, the JVM pops the current method's stack frame from the stack and continues processing in the previous method where it left off. When an exception condition occurs, however, the JVM must find a suitable exception handler. It first checks to see if the current method catches the exception or one of its parent exceptions. If so, execution will continue in that catch clause. If the current method doesn't provide a catch clause to handle the exception raised, the JVM will start popping method frames off the call stack until it finds a handler for the exception or one of its parent exceptions. Eventually, if it pops all the way back to the main( ) method and still doesn't find a handler for the exception, the thread will terminate. If that thread is the main thread and there are no other non-daemon threads running, the application itself will terminate. If the JVM does find an exception handler along the way, that method frame will become the top of the stack and execution will continue from there.  We might want to talk about checked an unchecked exceptions here as a discussion point?
* *Given above situation, why is Try/catch a bad idea for controlling program flow and should the handler be as close the exception as possible?* - In general, wrapping your Java code with try/catch blocks doesn't have a significant performance impact on your applications. Only when exceptions actually occur is there a negative performance impact, which is due to the lookup the JVM has to perform to locate the proper handler for the exception. If the catch block for the exception is located in the same method, the impact is not so bad. However, the further down the call stack the JVM has to go to find the exception handler, the greater the impact becomes
* What does the stack trace contain in a exception? - A stack trace provides information on the execution history of the current thread and lists the names of the classes and methods that were called at the point when the exception occurred. A stack trace is a useful debugging tool that you'll normally take advantage of when an exception has been thrown.
* Wrapper classes in java
* What are the different design patterns that you have used other than factory, singleton, abstract factory? And then design class diagrams with one example and use case. Also state where has those design patterns been used in Java APIs.
* What is IllegalMonitorStateException

### Java 8 New Features

Five features in java that will change how you code.

https://dzone.com/articles/5-features-java-8-will-change

* Lambda Expressions /Closures - <http://groovy.codehaus.org/Closures+-+Informal+Guide>
* Basically a way to assign a function to a variable and pass it around. Useful for modifying something in place.
* Parallel Operations
* Java + Java Script
* New Date / Time APIs
* Concurrent Accumulators

### Collections

* HashMap – Implementation, Serialization in case of HashMap, as all fields and Entity class in HashMap are transient, so how does it work and why these fields are marked as transient.
* Equals and Hashcode – General Contract of both and then if an object is going to be used as a key in HashMap than what all things you need to do to make it as a good key (they will ask for three things “equals()”, “hashcode()” & “immutability”.
* What if the generated hashcode is not lying in range of buckets or integer value (i.e how u will check the range of generated hashCode())
* Describe the difference between ArrayList and Linkedlist. Which one is better for what?
* What will be the result if hashcode is not implemented and similar data are inserted in the HashSet?
* Describe how the HashMap is implemented in Java –

Array of buckets. Each key is hashed to a bucket (may want to discuss why this is an integer and how it is used to determine a bucket using mod (n) functions ). Each bucket is a Linked List of entries. Collisions produce a list of entries in a bucket. (Optional ask whether this affects the constant time (O(1)) access for the Map. what other key collision handling options are available (linear probing/secondary hashing etc for open addressing)

* Describe how you would sort a collection in Java - Possible answers here are algorithms such as :

*Merge (average & worst  n log n)*sort takes advantage of the ease of merging already sorted lists into a new sorted list. It starts by comparing every two elements (i.e., 1 with 2, then 3 with 4...) and swapping them if the first should come after the second. It then merges each of the resulting lists of two into lists of four, then merges those lists of four, and so on; until at last two lists are merged into the final sorted list  
   
*\_Quicksort\_ (average n log n - worst n ^2^)* is a divide and conquer algorithm which relies on a partition operation: to partition an array, we choose an element, called a pivot, move all smaller elements before the pivot, and move all greater elements after it. This can be done efficiently in linear time and in-place.  
   
*\_Heap Sort\_ (average & worst  n log n)* It works by determining the largest (or smallest) element of the list, placing that at the end (or beginning) of the list, then continuing with the rest of the list, but accomplishes this task efficiently by using a data structure called a heap, a type of binary tree  
   
*Or use of Collections.sort: -* this uses Merge Sort - (may want to ask why used instead of quicksort - REASON guaranteed running time and stable sort)

* Describe how you would search a collection for an item - Possible answers:

Can use scanning/binary search (if binary search this require collection to be sorted - might want to ask why?)  
or collections binary search - get candidate to describe algorithm.

* Compare and contrast HashSet/TreeSet and why you would use a tree set –

*TreeSet - log(n)* time for operation, provides ordering, navigable in an ordered manner, ordering must be consistent with equals - so the Set interface is defined in terms of the equals operation, but a TreeSet instance performs all element comparisons using its compareTo (or compare) method, so two elements that are deemed equal by this method are, from the standpoint of the set, equal.  
Hashset - constant time operations, no guaranteed ordering, uniqueness via equals  
   
\_If strong on collections may want to ask about priority queue:\_

* Name some important data structures.

List, Set, Map, Array/Vector, Tree - perhaps understand when you would use them

* How does a Hash-Map work? (equals & hashcode methods )? What mechanism does it use internally to store and retrieve the objects placed in it? If you create a custom class whose instances are to be used as the key in a HashMap, what methods of the Object class should you strongly consider overriding?
* Difference between HashMap and TreeMap, when to use one over the other?
* How would you increase the size of an array?
* Describe performance of map using big 0 notation - Should correctly say O(1) for put/get. Should be able to discuss impact of resize.
* What is the best object to contain a large group of objects space-wise? - *LinkedList*
* What is the best object to contain a large group of objects so as to enable the fastest access? *HashMap*
* If you were to iterate multiple times through a HashMap, what is the best way to preserve the order which the objects are iterated? *LinkedHashMap*
* How do the ArrayList and LinkedList implementations differ, and what are the relative advantages/disadvantages of both?
* How would you search a Collection for an item?
* Explain merge sort
* What is the performance of Binary Search in big O notation, is this always the case?
* What data structure could you use to record URLs that have been visited?
* What's a suffix tree and what is it used for?
* Describe how to avoid concurrent issues using synchronize and notify
* What would you do if you wanted to use a map in multiple threads? - Discuss ConcurrentHashMap. How is concurrency achieved? Multiple partitions which can be locked independently. (16 by default) Using concurrent Locks operations for thread safety instead of synchronized. Has thread safe Iterators. Synchronized Collection's iterators are not thread safe.
* (ConcurrentHashMap does not throw ConcurrentModificationException if the ConcurrentHashMap is changed while being iterated. The Iterator is not designed to be used by more than one thread) Does not expose the internal locks. Synchronized Collection does.
* Internal structure of ConcurrentHashMap and its performance
* Write your own implementation of hashmap
* Comparable vs. comparator
* Which collections can be used to implement combining the cache and the most recently used object feature?
* What are  bounded and unbounded wildcards in java generics
* How to handle exception in multi-threading
* How to design a class which object never got garbage collected
* If an hashCode always return 1 then what happen inside HashMap and how it disturb HashMap functionality ?
* Implement custom Linked hashmap
* What is garbage collection ? How this works ? If i have finally block in my code which is somehow not executed and in this piece of code, i am de-referencing the object, how those object will be garbage collected ? if the system gets exit, i wont be able to call the finalize method as well in this case.
* What all are the scenarios where finally block in not executed
* How compare and equals method work, if present together, works in camparator/comparable ?
* I have two list, list1 (1,2,4) list2 (4,2,1) list1.equals(list2) ? What this will return ?

### Serialization

* Java default serialization requirements
* Methods that can be implemented - writeObject/readObject, writeReplace/readResolve
* What is serialVersionUID used for?
* XML, JSON, Google protocol buffers, POF
* What happen When we deserialise the object two times

### Concurrency

* Basic questions on Multithreading, like what is sleep or wait/notify/notify all.
* What is synchronized block?
* What is Thread pool?
* What does class level lock signify and how it’s acquired internally.
* Differentiate between static synchronized and non-static synchronized blocks.
* Locks – ReentrantLock & ReentrantReadWriteLock
  1. how they work
  2. Difference between synchronized and Locks
  3. Are synchronized blocks reentrant? If yes then how and if no then how we can make them.
* Immutability –
  1. What is immutable & mutable.
  2. How to make a class immutable. (Try Explaining all the necessary conditions on immutability (Cloning – Shallow and Deep both and when and where required))
  3. Why to make classes Immutable (benefits & example of class from JAVA)
* ConcurrentHashMap (CHM) –
  1. Difference between HashMap, HashTable and CHM
  2. Working of CHM.
  3. Locking used in CHM
  4. Does read operation take lock in CHM
  5. How multiple write works in CHM(Concept of CHM)
  6. Use of UnSafe Class in CHM
  7. Is there any possibility that a CHM may give you intermediate data while parallel read and write operation?
  8. Is CHM throws ConcurrentModificationException? If yes – then how and if no implement a scenario so that it can throw the same.
* Difference between CyclicBarrier and CountdownLatch (except the difference that CyclicBarrier can be reused). They will ask for the scenario in which you will apply semaphore, CyclicBarrier & CountdownLatch.
* Describe re-entrant synchronization in Java. - A thread can aquire a lock it already owns as many times as it wants without blocking. This describes a situation where synchronized code, directly or indirectly, invokes a method that also contains synchronized code, and both sets of code use the same lock.
* What additional precaution would be needed if synchronization was not re-entrant. - The thread would have to check if it was already the owner before acquiring it again, otherwise it would block itself. Thread.holdsLock().
* Describe deadlock, livelock, starvation - *Deadlock:* describes a situation where two or more threads are blocked forever, waiting for each other  
  *Livelock:* A thread often acts in response to the action of another thread. If the other thread's action is also a response to the action of another thread, then livelock may result. As with deadlock, livelocked threads are unable to make further progress. However, the threads are not blocked ¿ they are simply too busy responding to each other to resume work. This is comparable to two people attempting to pass each other in a corridor. *Starvation:* describes a situation where a thread is unable to gain regular access to shared resources and is unable to make progress. This happens when shared resources are made unavailable for long periods by "greedy" threads.
* Describe Reentrant lock and how it differs from Java synchronization. A reentrant mutual exclusion Lock with the same basic behavior and semantics as the implicit monitor lock, but slightly different capabilities.  
  a) Does not require use of block structures. So locks can be acquired and released in different scopes.  
  b) Consequence of A is that 'lock chaining' or 'hand over hand' locking is far easier than with synchronization. e.g. acquire lock on A then B, then release A then acquire C then release B. needed for some concurrent data structures (such as trees)  
  c) ability to try the lock (using try Lock) and not block if it cannot be obtained  
  d) Ability to try lock with a timed wait.
* How would you ensure that a particular variable is thread safe?
* A discussion about what concurrency is, mentioning threadlocal, synchronized methods/objects, volatile - see <http://docs.oracle.com/javase/tutorial/essential/concurrency/>
* Follow up questions could involve how to create threads, joins, deadlock, live lock, thread starvation.
* What does it have in common with the volatile keyword? (happens-before relationships)
* How do you trace issues in a distributed system? What can go wrong in a distributed system?
* What is the difference between a thread and a process?
* What is deadlock? How do you prevent deadlocks? What is *livelock*?
* What is compare and swap and when would you use it?
* Why does ConcurrentHashMap have a *putIfAbsent* method?
* One thread put items into an array, another takes them off. Can you describe how to use synchronization primitives (pre 1.5) to handle access to the array? Looking for correct use of synchronize / wait / notify. What happens when the array is full / empty? Remember a while loop while checking condition. Follow up: What java 5+ thing does this for you? Answer: *BlockingQueue*.
* Sync primitives: synchronized, wait, notify. deadlock, *livelock*, thread starvation
* What does the volatile keyword mean? Can it be used instead of synchronization?
* *notify* vs *notifyAll*
* Fair locks
* Interruption - InterruptedException, InterruptedIOException
* Lock elision
* If you have to do some additional cleanup once an object was eligible for GC, what options are available in Java? (Finalize, Reference Queues)
* Describe the priority queue - The Priority Queue class is a priority queue based on the heap data structure. This queue orders elements according to an order specified at construction time, which can be the elements' natural ordering or the ordering imposed by an explicit Comparator. The head of the queue is the least element with respect to the specified ordering. Equal ties in ordering are broken arbitrarily.
* Difference between Runnable and Callable?
* What is Executor framework?
* Describe Producer and Consumer problem.
* What is Thread pool?
* Is submit() method in scheduled thread pool blocking or non-blocking?
* In main() method if you want to resume execution only after completion of all the tasks submitted to executor service, then how can you do that?
* How to write our own blockingQueue?
* ConcurrentHashMap : Working and How segments are created for thread safe operations.

### GC / Memory Model

* How does the GC work? (young vs old, copying, compacting, fragmentation)
* When does an object becomes eligible for Garbage collection in Java? Explain 2 types of garbage collection algorithms.
* How would you go about performance tuning an application?
* Describe the structure of memory in Java - The Java VM manages two kinds of memory: heap and non-heap memory, both of which are created when the Java VM starts.

Heap memory is the runtime data area from which the Java VM allocates memory for all class instances and arrays. The heap may be of a fixed or variable size. The garbage collector is an automatic memory management system that reclaims heap memory for objects.  
Non-heap memory includes a method area shared among all threads and memory required for the internal processing or optimization for the Java VM. It stores per-class structures such as a runtime constant pool, field and method data, and the code for methods and constructors. The method area is logically part of the heap but, depending on the implementation, a Java VM may not garbage collect or compact it. Like the heap memory, the method area may be of a fixed or variable size. The memory for the method area does not need to be contiguous.  
   
In addition to the method area, a Java VM may require memory for internal processing or optimization which also belongs to non-heap memory. For example, the Just-In-Time (JIT) compiler requires memory for storing the native machine code translated from the Java VM code for high performance.  
 The Java HotSpot VM defines two generations: the young generation (sometimes called the "nursery") and the old generation. The young generation consists of an "Eden space" and two "survivor spaces." The VM initially assigns all objects to the Eden space, and most objects die there. When it performs a minor GC, the VM moves any remaining objects from the Eden space to one of the survivor spaces. The 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. The permanent generation holds all the reflective data of the virtual machine itself, such as class and method objects.  
 Permanent Generation (non-heap): The pool containing all the reflective data of the virtual machine itself, such as class and method objects. With Java VMs that use class data sharing, this generation is divided into read-only and read-write areas.

* Explain how memory leaks can manifest in the JVM, and how one can avoid them. How would you go about diagnosing the cause of an OutOfMemoryException?
* Talk about the methodology behind diagnosing and analyzing a memory issue in java. Could talk about the following link: <http://www.captaindebug.com/2013/12/investigating-memory-leaks-part-2.html#.U2qGJjY6aN8>

Specifically talk about connecting jconsole up to the object (or utilize other system), taking a memory dump, using a tool to create a report, then analyzing the report