1. Compare & contrast immutability of String & Date classes.  
   - Date mutable
2. What is abstraction?

What is abstraction?  
When do you use Abstract classes and when do you use Interfaces?  
**What is composition vs aggregation?**

* Aggregation is a simple ‘has-a’ relationship where the composed object can have its independent lifecycle. Composition is stricter relationship where the child entity has no existence without the parent entity

**Explain performance of a hashMap put & get methods**

In ideal scenario, HashMap is expected to give constant time performance irrespective of number of entries in it. This performance depends on several factors like implementation of hashcode & equals implementation of its keys and frequency of rehashing performed in the map. This rehashing frequency depends on initial capacity(buckets) and load factor of the map.

In extreme cases, when hashcode implementation always returns the same value, get performance degrades majorly as it needs to perform equality check on all the entries of the map. On the other end, unique values of the hashmap will turn the result in to linked list.

**How do you take a thread dump? How do you analyze a thread dump?**

**What are the operations which can put a thread in blocked or waiting state?**

* Waiting for an object monitor
* Explicit call to wait() method
* Join()
* Latch, barrier, semaphore, blocking queue
* LockSupport.park() static method
* IO block

**Block vs wait TODO**

**How do you implement your thread pool ? explain your design**

Execution policies are a resource management tool, and the optimal policy depends on the available computing resources and your quality of service requirements. By limiting the number of concurrent tasks, you can ensure that the application does not fail due to resource exhaustion or suffer performance problems due to contention for scarce resources. Separating the specification of execution policy from task submission makes it practical to select an execution policy at deployment time that is matched to the available hardware.

* Data structure for maintaining Thread pool
  + Core pool size
  + Max pool size
  + Strategies to change pool size as per task queue
* Data structure for maintaining task queue
  + LinkedBlockingQueue or ArrayBlockingQueue
* Provide an interface to submit/execute new tasks

ConcurrentHashMap vs synchronized collections  
When do you receive a ConcurrentModificationException?  
Fail fast vs fail safe iterators  
What is hash collision – how hashMap resolves a hashing collision

**How is the java heap organized? What are the areas? What is Perm Gen space?**

 The complete address space reserved for object memory can be divided into the young and tenured generations.

The young generation consists of eden and two survivor spaces. Most objects are initially allocated in eden. One survivor space is empty at any time, and serves as the destination of any live objects in eden and the other survivor space during the next copying collection. Objects are copied between survivor spaces in this way until they are old enough to be tenured (copied to the tenured generation).

A third generation closely related to the tenured generation is the permanent generation which holds data needed by the virtual machine to describe objects that do not have an equivalence at the Java language level. For example objects describing classes and methods are stored in the permanent generation.

1. *Throughput* is the percentage of total time not spent in garbage collection, considered over long periods of time. Throughput includes time spent in allocation (but tuning for speed of allocation is generally not needed).
2. *Pauses* are the times when an application appears unresponsive because garbage collection is occurring.

*-Xms – min heap size*

*-Xmx – max heap size*

*-XX:NewRatio=3 – young generation/tenured generation space(1/4 is young – of total heap)*

*-XX:NewSize, MaxNewSize – Similar to –Xms & -Xmx for granular level control of young gen. space*

-XX:SurvivorRatio=6 – eden/survivor space ratio (1/8 is survivor of total young gen. not 1/7 because of 2 survivor spaces)

*Biger Young Generation –* Less frequent minor GC, so higher throughput. Smaller old generations, so frequent Full GC hence more pauses.

The command line option -verbose:gc causes information about the heap and garbage collection to be printed at each collection.

**What are different garbage collectors?**

1. SerialGC – efficient for single processor machines or apps with small datasets (100mb)
2. Parallel GC – runs minor gc in young gen & full gc is run by single thread
3. ParallelOldGC (*Parallel Compaction*)– runs full gc with multiple threads –
4. ConcMarkAndSweep – Minimizes pause time at cost of throughput, performs GC concurrently. If response time is more important than overall throughput and garbage collection pauses must be kept shorter than approximately one second.

**Complexity of bubble sort algorithm O(n2)**

**What is immutability? What are the benefits of immutability?**

– Immutable objects cannot be modified after they are constructed. Benefit is that they can be passed around to any methods/classes/threads without worrying about modification. Their references can be cached / they are thread-safe / They are excellent keys of Maps / It is also useful to cut down memory usage i.e. same immutable fine-grained object can be referenced multiple times (very large number of times) for example each character in a document say ‘a’ needs only one immutable instance ...This is example of Flyweight pattern.  
  
They can be shared between multiple threads without worrying about multithreading issues.  
Under which cases Singleton may not behave as singleton? 1. Improper synchronization during creation 2. Multiple class loaders 3. During de-serialization

**How to avoid creation of multiple instances of Singleton class during deserialization process?**

readResolve is used for replacing the object read from the stream. The only use I've ever seen for this is enforcing singletons; when an object is read, replace it with the singleton instance. This ensures that nobody can create another instance by serializing and deserializing the singleton.

Classes that need to designate a replacement when an instance of it is read from the stream should implement this special method with the exact signature.

ANY-ACCESS-MODIFIER Object readResolve() throws ObjectStreamException;

This readResolve method follows the same invocation rules and accessibility rules as writeReplace.

**public** **final** **class** MySingleton {

**private** MySingleton() { }

**private** **static** **final** MySingleton INSTANCE = **new** MySingleton();

**public** **static** MySingleton getInstance() { **return** INSTANCE; }

**private** Object readResolve() **throws** ObjectStreamException {

// instead of the object we're on,

// return the class variable INSTANCE

**return** INSTANCE;

}

}

**Hibernate**  
Different types of relationships  
If student entity has 1 to many relationship with subject & it is mentioned as lazy=”true” in association. If the subject is accessed for one student in detached state – it throws exception.

How to overcome this do we add any filter in web.xml to handle this?  
Have u used Criteria query API? How to build criteria query? Which pattern does it use?  
Do you know versioning in Hibernate? Explain.  
  
  
Round 2 Topics given by HR  
  
OO: inheritance vs composition, loose coupling -   
Design patterns: above average. decorator, visitor and usage.    
design principles and examples  
Java  
                                                number of stacks in jvm, analyzing thread dumps,thread states, sizing Thread pool  
                                                Thread safety, multi-threading -   
**Atomic classes -**  
                                                Immutability  
                                                Volatile -   
                                                Concurrenthashmap -   
                                                Mentioned advanced locks -   
**Semaphore -**  
                                                java 5 improvements –

* Concurrent API
* Generics
* Annotations

Java concurrency package

**Java nio vs blocking IO**  
Periodic GC on a singleton  
**Describe ideal GC**Problem Solving:   
JMS:   
DLQ - loss -   
Spring:    
**Hibernate/JPA: n+1 select problem**Lazy init exception  
Joined-subclass  
Inverse  
Dealing with Concurrent modification:    
Database  
Data modeling -   
Queries -     
Linux:    
Web development:  
NoSQL:   
Security:   
Asymm encryption   
Nonce  
XSS  
Sql injection  
  
  
  
  
Round 2 Questions  
  
OO: inheritance vs composition, loose coupling –   
  
**What would you prefer Inheritance or Composition? Why?**1.    You have a Person, Teacher & Student entities. How would you model your classes?  
2.    If a Teacher can also be a Student & a Student can also be a teacher – how would you model it?  
  
**Design patterns: above average. decorator, visitor and usage.**    
1.    Which is your favourite design patter? Explain the class diagram of that design pattern. (Explained Decorator pattern)  
2.    If you have a Pricing engine and it could use different pricing calculation algorithms. Which design pattern would you use? Strategy  
**design principles and examples**  
Java: -  
analyzing thread dumps,

thread states,

1.    How many stacks in JVM  
2.    How will you size your threadpool. What are the key parameters while creating a threadpool  
3.    How many threads should you create in your thread pool? Thread safety, multi-threading –   
1.    What are the different ways you can achieve thread safety?  
Atomic classes –   
1. Which Atomic classes you have used? What is the benefit of using Atomic classes as opposed to synchronized blocks?  
2. How do Atomic classes work? For example the operations provided by AtomicInteger  
                                                Immutability  
                What is an immutable class? What are the benefits of Immutability?  
                                                Volatile –   
                What happens when you declare a variable volatile?      
                                                Concurrenthashmap -   
                                                Mentioned advanced locks –   
              
1.    What is reentrantreadwritelock – how is it different from synchronized block?  
                                                Semaphore -   
                                                java 5 improvements –   
1.    What are the new features in Java 5?   
2.    What is the difference in working with a collection used with generics & without generics? What will be the difference in the bytecode that is generated?  
                                                Java concurrency package –   
1.    Which of the java.util.concurrent package classes you have used? Locks/ReentrantReadWriteLock, Atomic classes,   
                                                 Java nio vs blocking IO -   
                                                Periodic GC on a singleton –   
1.    How can we trigger garbage collection of a singleton periodically? By having another public static method which just sets instance = null.  
                                                Describe ideal GC -   
Problem Solving:   
1.    A folder has many files. The folder may have some files which are duplicate (contents exactly same but different file name). How to find out the duplicate files? The files could be very large each file could be few GBs.  
JMS:   
DLQ -   
No message loss –   
1.    How can we achieve zero message loss with JMS? Use persistent queues, use guaranteed message delivery through acknowledgement (message acknowledgement – AUTO\_ACKNOWLEDGE or DUPS\_OK\_ACKNOWLEDGE mode), use transactions - message will not be removed from the queue until the transaction is committed by the consumer.  
  
Spring:    
1.    What do you have to say about code which is using appContext.getBean(“xyz”) or beanFactory.getBean(“xyz”)?  
Hibernate/JPA: n+1 select problem  
Lazy init exception –   
1.    When do you get LazyInitializationException? How to avoid it?  
Joined-subclass:   
1.    What is joined sub-class?  
Inverse:  
What is the meaning of inverse=”true” in hibernate mapping  
Dealing with Concurrent modification:    
How do you deal with concurrent modification in hibernate?  
Database:   
Data modeling -   
Queries -     
Linux:    
Web development:  
NoSQL:   
Security:   
Asymm encryption   
Nonce  
XSS  
Sql injection  
  
  
  
  
  
  
  
  
  
  
Design Principles & Patterns  
  
  
Abstraction is the concept of describing something in simpler terms, i.e abstracting away the details, in order to focus on what is important (This is also seen in abstract art, for example, where the artist focuses on the building blocks of images, such as colour or shapes). The same idea translates to OOP by using an inheritance hierarchy, where more abstract concepts are at the top and more concrete ideas, at the bottom, build upon their abstractions. At its most abstract level there is no implementation details at all and perhaps very few commonalities, which are added as the abstraction decreases.  
  
As an example, at the top might be an interface with a single method, then the next level, provides several abstract classes, which may or may not fill in some of the details about the top level, but branches by adding their own abstract methods, then for each of these abstract classes are concrete classes providing implementations of all the remaining methods.  
  
Encapsulation is a technique. It may or may not be for aiding in abstraction, but it is certainly about information hiding and/or organisation. It demands data and functions be grouped in some way - of course good OOP practice demands that they should be grouped by abstraction. However, there are other uses which just aid in maintainability etc.  
  
  
--------------------------------------------------------------------------------------------------------------------------------------------------------------------  
  
  
**1. Strategy**   ---  defines a family of algorithms, encapsulates each one, and makes them interchangeable.Strategy lets the algorithm vary independently from clients that use it  
  
Car has break & accelerator  
  
both break behavior & accelerator behavior can be encapsulated as strategy and can be easily replaced at run-time with different behaviours.  
  
  
**2. Observer**  --  Defines a one to many dependency between objects so that when one object changes state, all of its dependents are notified and updated automatically  
  
Observer Observable in jdk  
HttpSessionBindingListener  
HttpSessionAttributeListener  
  
**3. Decorator**  -- attaches additional responsibilities to an object dynamically. Decorators provide flexible alternative to subclassing for extending functionality.  
  
Java IO API   
  
  
InputStream - Component  
  
FileInputStream - ConcreteComponent  
  
FilterInputStream - AbstractDecorator  
  
  
  
**4. Factory method**  -  defines an interface for creating an object, but lets subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.  
  
java.util.Calendar#getInstance();  
java.util.ResourceBundle#getBundle()  
  
**Abstract factory**  - provides an interface for creating families of related or dependent objects without specifying their concrete classes.  
  
javax.xml.parsers.DocumentBuilderFactory#newInstance()  
javax.xml.transform.TransformerFactory#newInstance()  
javax.xml.xpath.XPathFactory#newInstance()  
  
**5. Singleton** -- ensures a class has only one instance and provides a global point of access to it.  
  
java.lang.Runtime#getRuntime()  
  
  
**6. Command**  
  
**7. Visitor**

The visitor [design pattern](http://en.wikipedia.org/wiki/Software_design_pattern) is a way of separating an [algorithm](http://en.wikipedia.org/wiki/Algorithm) from an object structure on which it operates. A practical result of this separation is the ability to add new operations to existing object structures without modifying those structures. It is one way to follow the [open/closed principle](http://en.wikipedia.org/wiki/Open/closed_principle).

Often collections contain objects of different types and in those cases some operations have to be performed on all the collection elements without knowing the type. A possible approach to do it would be to use a marker interface type to categories all items and when different operation is required on specific types, use ‘instanceof’ checks. This is not an object-oriented way to do it.

**8. Flyweight**

A flyweight is an [object](http://en.wikipedia.org/wiki/Object_(computer_science)) that minimizes [memory](http://en.wikipedia.org/wiki/Computer_memory) use by sharing as much data as possible with other similar objects; it is a way to use objects in large numbers when a simple repeated representation would use an unacceptable amount of memory.

**9. Builder**

**Adapter & façade**

**Template method**

**Iterator & composite**

**State**

**Proxy**

**Bridge**  -- allow to implementations to   
  
chain or responsibilities

**Interpreter**

**Mediator**  
**Memento**  
**Prototype**

Principles  
----------------------------------------------------------------------------------------------------------------------  
  
Encapsulate what varies      -- strategy  
  
Program to an interface, not an implementation -- strategy  
  
Favour composition over inheritance   -- strategy  (ability to change behavior at runtime - define behavior at runtime rather than compile time)  
  
Program to interfaces - not implementations  
  
Strive for loosely coupled design between objects that interact    -                       --- (Observer)  
  
Classes should be open for extension but closed for modification   - Open Closed Principle --- (Decorator, Visitor)  
  
Depend on abstractions, do not depend on concrete classes          - Dependency Inversion  --- (Factory Method)  
  
Only talk to your friends  
  
Don't call us, we'll call you                       - Template Method Pattern  
  
A class should have only one reason to change