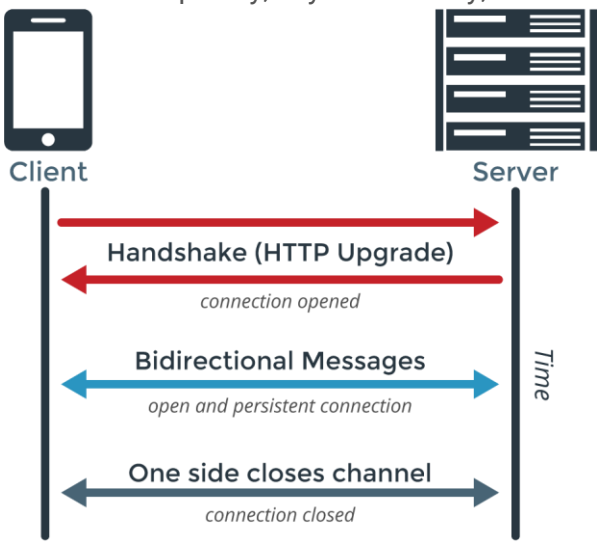
Java Architect

What is WebSocket?

WebSocket is a transport protocol defined by a persistent, bidirectional communication channel between server and client that takes place over a single TCP socket connection.
It was designed to overcome the limitations of HTTP's basic request/response mechanism (i.e. constant opening and closing of connections and header overhead), which is not suitable for real time applications that rely on sustained user interaction or continuously streamed updates.
Fully HTML5-compliant and built-in to all major web browsers, WebSocket enables full-duplex communication between server and client for use in applications such as chat, multiplayer gaming, multi-user collaboration, and live-streamed charts or scoreboards (e.g. sports matches in progress).
Because a WebSocket connection is held open for the duration of a session, messages can flow back and forth between participating endpoints with little overhead and low latency. This eliminates the need for client-server polling and is especially useful for any JavaScript-based application that processes data frequently, asynchronously, and in real time.



How WebSocket Works

To initiate a WebSocket connection, the client and server must negotiate an Upgrade handshake from HTTP, and from that point onwards, asynchronous WebSocket rules apply. Specifically, bulky HTTP headers are replaced with message frames only a few bytes in size, and both the server and client can simultaneously send new data without the other asking for it.

WebSocket is similar to Server-Sent Events (SSE), however Server-Sent Events is applicable only in unidirectional communication (i.e. server to client). Meanwhile, HTTP/2 Server Push is also related to WebSocket in concept, but Server Push is only able to push data from the server to the browser cache, and does not allow data to be pushed inside the application layer, as in WebSocket.

What is SockJS?

SockJS is a JavaScript library that provides WebSocket-like connectivity, delivering low-latency communication between web browsers and servers, and even applications that do not utilize WebSocket programming. Under the hood, SockJS uses native WebSockets first, and on failure, falls back to a variety of browser-specific transport protocols, and delivers them through WebSocket-like abstractions.

Benefits of SockJS

- Enhances third-party server implementation as it addresses various behavior patterns for most server implementations.
- All information is found in its URL, which means there is no need to use cookie-based sticky sessions.

What is Comet (Programming)?

Comet is a web application design paradigm that describes a continuous, two-way interaction between a server and a web browser using native HTTP methods. It was conceived in order to provide a conceptual model for designing responsive and highly interactive web UIs without having to resort to browser plugins like Java applets.

The Comet approach turns the typical HTTP client-server model upside-down to enable a kind of reverse Ajax functionality, wherein the web server initiates a persistent HTTP connection with a client browser and actively pushes out data instead of waiting to serve responses. In other words, Comet represents a “Push-Style” or “Server-Push” mechanism in contrast with HTTP’s “Request/Response” or “Get/Pull” mechanism.

Comet Techniques

A number of HTTP techniques exist to achieve this event-driven interaction, including Ajax Push, HTTP Streaming, and HTTP Server Push. These techniques mainly rely on JavaScript to handle streamed events on the client side, while the messaging format is often in JSON or XML. A more modern example is WebSocket, a sophisticated transport protocol that borrows from the principles of Comet yet which offers a cleaner and more standardized realization of the idea. However, WebSocket stands on its own as a distinct protocol and is not generally considered to be a form of Comet.

Comet Model

The Comet model is often used for web apps that involve multi-user collaboration where latency must be kept as low as possible for a shared realtime experience. Traditionally, it has been used to provide support for collaborative document editing and multi-protocol chat.
To apply Comet in a web application, two broad methods are available:

- Streaming: Events are pushed from server to client over a single persistent connection, and are usually processed inside a hidden iframe on the page or via XMLHttpRequest.
- Long Polling: The browser polls the server for new events with a persistent request that is held open until it gets a response. When new data is available, the server sends the response and a new long polling request is made by the browser for further events. Rinse and repeat for sustained interaction.

Sharding the Hibernate Way

To scale you are supposed to partition your data. Sounds good, but how do you do it? When you actually sit down to work out all the details it's not that easy. Hibernate Shards to the rescue! Hibernate shards is: an extension to the core Hibernate product that adds facilities for horizontal partitioning. If you know the core Hibernate API you know the shards API. No learning curve at all. Here is what a few members of the core group had to say about the Hibernate Shards open source project. Although there are some limitations, from the sound of it they are doing useful stuff in the right way and it's very much worth looking at, especially if you use Hibernate or some other ORM layer.

What is Hibernate Shards?

1. Shard: splitting up data sets. If data doesn't fit on one machine then split it up into pieces, each piece is called a shard.
2. Sharding: the process of splitting up data. For example, putting employees 1-10,000 on shard1 and employees 10,001-20,000 on shard2.
3. Sharding is used when you have too much data to fit in one single relational database. If your database has a JDBC adapter that means Hibernate can talk to it and if Hibernate can talk to it that means Hibernate Shards can talk to it.
4. Most people don't want to shards because it makes everything complex. But when you have too much data, when you fill your database up, you need another solution, which can be to shard the data across multiple relational databases. The complexity arises because your application has to have the smarts to access multiple databases and that's where Hibernate Shards tries to help.
5. Structure of the data is identical from server to server. The same schema is used across all databases (MySQL, etc).
6. Hibernate was chosen because it's a good ORM tool used internally at Google, but to Google Scale (really really big), sharding needed to be added because Hibernate didn’t support that sort of scale out of the box.
7. The learning curve for a Hibernate user is zero because the Hibernate API is the same. The shard implementation hasn’t violated the API (yet). Sharded versions of Session, Critieria, and Factory are available so the programmer doesn't need to change code. Query isn't implemented yet because features like aggregation and grouping are very difficult to implement across databases.
8. How does it compare to MySQL's horizontal partitioning? Shards is for situations where you have too much data to fit in a single database. MySQL partitioning may allow you to delay when you need to shard, but it is still a single database and you’ll eventually run into limits.

Distributed DBMS - Commit Protocols

In a local database system, for committing a transaction, the transaction manager has to only convey the decision to commit to the recovery manager. However, in a distributed system, the transaction manager should convey the decision to commit to all the servers in the various sites where the transaction is being executed and uniformly enforce the decision. When processing is complete at each site, it reaches the partially committed transaction state and

waits for all other transactions to reach their partially committed states. When it receives the message that all the sites are ready to commit, it starts to commit. In a distributed system, either all sites commit or none of them does.

The different distributed commit protocols are −

- One-phase commit
- Two-phase commit
- Three-phase commit

Distributed One-phase Commit

Distributed one-phase commit is the simplest commit protocol. Let us consider that there is a controlling site and a number of slave sites where the transaction is being executed. The steps in distributed commit are −

- After each slave has locally completed its transaction, it sends a “DONE” message to the controlling site.
- The slaves wait for “Commit” or “Abort” message from the controlling site. This waiting time is called **window of vulnerability**.
- When the controlling site receives “DONE” message from each slave, it makes a decision to commit or abort. This is called the commit point. Then, it sends this message to all the slaves.
- On receiving this message, a slave either commits or aborts and then sends an acknowledgement message to the controlling site.

Distributed Two-phase Commit

Distributed two-phase commit reduces the vulnerability of one-phase commit protocols. The steps performed in the two phases are as follows −

Phase 1: Prepare Phase

- After each slave has locally completed its transaction, it sends a “DONE” message to the controlling site. When the controlling site has received “DONE” message from all slaves, it sends a “Prepare” message to the slaves.
- The slaves vote on whether they still want to commit or not. If a slave wants to commit, it sends a “Ready” message.
- A slave that does not want to commit sends a “Not Ready” message. This may happen when the slave has conflicting concurrent transactions or there is a timeout.

Phase 2: Commit/Abort Phase

- After the controlling site has received “Ready” message from all the slaves −
    - The controlling site sends a “Global Commit” message to the slaves.
    - The slaves apply the transaction and send a “Commit ACK” message to the controlling site.

    - When the controlling site receives “Commit ACK” message from all the slaves, it considers the transaction as committed.
- After the controlling site has received the first “Not Ready” message from any slave −
    - The controlling site sends a “Global Abort” message to the slaves.
    - The slaves abort the transaction and send a “Abort ACK” message to the controlling site.
    - When the controlling site receives “Abort ACK” message from all the slaves, it considers the transaction as aborted.

Distributed Three-phase Commit

The steps in distributed three-phase commit are as follows −

Phase 1: Prepare Phase

The steps are same as in distributed two-phase commit.

Phase 2: Prepare to Commit Phase

- The controlling site issues an “Enter Prepared State” broadcast message.
- The slave sites vote “OK” in response.

Phase 3: Commit / Abort Phase

The steps are same as two-phase commit except that “Commit ACK”/”Abort ACK” message is not required.

Horizontal / Vertical partitioning

Horizontal partitioning involves putting different rows into different tables. Perhaps customers with ZIP codes less than 50000 are stored in CustomersEast, while customers with ZIP codes greater than or equal to 50000 are stored in CustomersWest. The two partition tables are then CustomersEast and CustomersWest, while a view with a union might be created over both of them to provide a complete view of all customers.

Vertical partitioning involves creating tables with fewer columns and using additional tables to store the remaining columns. Normalization also involves this splitting of columns across tables, but vertical partitioning goes beyond that and partitions columns even when already normalized.

Load Balancing - Availability and scalability

Server load balancing distributes service requests across a group of real servers and makes those servers look like a single big server to the clients. Often dozens of real servers are behind a URL that implements a single virtual service.

How does this work? In a widely used server load balancing architecture, the incoming request is directed to a dedicated server load balancer that is transparent to the client. Based on parameters such as availability or current server load, the load balancer decides which server should handle the request and forwards it to the selected server. To provide the load balancing algorithm with the required input data, the load balancer also retrieves information about the servers' health and load to verify that they can respond to traffic. Figure 1 illustrates this classic load balancer architecture.

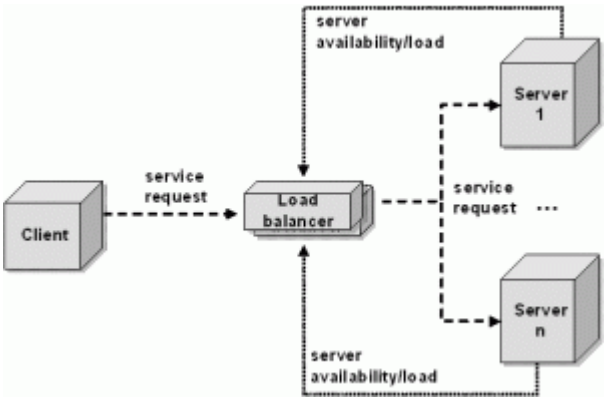


Figure 1. Classic load balancer architecture (load dispatcher)

The load-dispatcher architecture illustrated in Figure 1 is just one of several approaches. To decide which load balancing solution is the best for your infrastructure, you need to consider availability and scalability.

Availability is defined by uptime -- the time between failures. (Downtime is the time to detect the failure, repair it, perform required recovery, and restart tasks.) During uptime the system must respond to each request within a predetermined, well-defined time. If this time is exceeded, the client sees this as a server malfunction. High availability, basically, is redundancy in the system: if one server fails, the others take over the failed server's load transparently. The failure of an individual server is invisible to the client.

Scalability means that the system can serve a single client, as well as thousands of simultaneous clients, by meeting quality-of-service requirements such as response time. Under an increased load, a high scalable system can increase the throughput almost linearly in proportion to the power of added hardware resources.

In the scenario in Figure 1, high scalability is reached by distributing the incoming request over the servers. If the load increases, additional servers can be added, as long as the load balancer does not become the bottleneck. To reach high availability, the load balancer must monitor the

servers to avoid forwarding requests to overloaded or dead servers. Furthermore, the load balancer itself must be redundant too. I'll discuss this point later in this article.

Server load balancing techniques

In general, server load balancing solutions are of two main types:

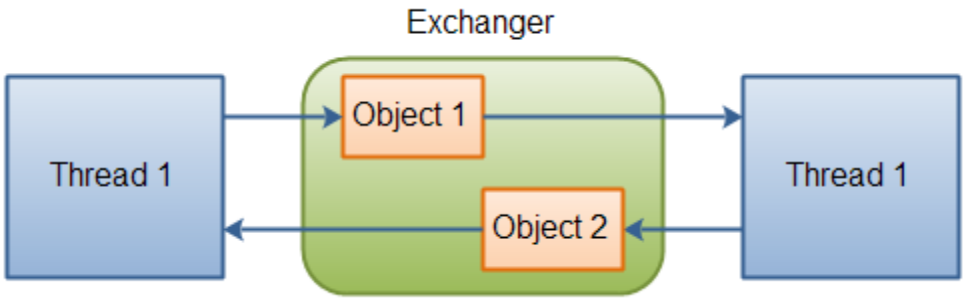
- **Transport-level load balancing** -- such as the DNS-based approach or TCP/IP-level load balancing -- acts independently of the application payload.
- **Application-level load balancing** uses the application payload to make load balancing decisions.

Load balancing solutions can be further classified into software-based load balancers and hardware-based load balancers. Hardware-based load balancers are specialized hardware boxes that include application-specific integrated circuits (ASICs) customized for a particular use. ASICs enable high-speed forwarding of network traffic without the overhead of a general-purpose operating system. Hardware-based load balancers are often used for transport-level load balancing. In general, hardware-based load balancers are faster than software-based solutions. Their drawback is their cost.

In contrast to hardware load balancers, software-based load balancers run on standard operating systems and standard hardware components such as PCs. Software-based solutions runs either within a dedicated load balancer hardware node as in Figure 1, or directly in the application.

Exchanger

The java.util.concurrent.Exchanger class represents a kind of rendezvous point where two threads can exchange objects. Here is an illustration of this mechanism:



Two threads exchanging objects via an Exchanger.

Exchanging objects is done via one of the two exchange() methods. Here is an example:

```
Exchanger exchanger = new Exchanger();

ExchangerRunnable exchangerRunnable1 =
      new ExchangerRunnable(exchanger, "A");

ExchangerRunnable exchangerRunnable2 =
      new ExchangerRunnable(exchanger, "B");

new Thread(exchangerRunnable1).start();
new Thread(exchangerRunnable2).start();
```

Here is the ExchangerRunnable code:

```
public class ExchangerRunnable implements Runnable{

  Exchanger exchanger = null;
  Object    object    = null;

  public ExchangerRunnable(Exchanger exchanger, Object object) {
    this.exchanger = exchanger;
    this.object = object;
  }

  public void run() {
    try {
      Object previous = this.object;

      this.object = this.exchanger.exchange(this.object);

      System.out.println(
            Thread.currentThread().getName() +
            " exchanged " + previous + " for " + this.object
      );
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }
}
```

This example prints out this:

```
Thread-0 exchanged A for B
Thread-1 exchanged B for A
```

Overriding Equals, HashCode and CompareTo in Java

In this Java program, we will override all three method for a Person class, which contains a String name, an integer id and a Date for date of birth. In order to override equals, you need to follow certain checks, e.g. checking null, checking type of object etc, Also your equals() method, when compared with null, should return false instead of throwing NullPointerException.

See this post for detailed tips on overriding equals method in Java. For overriding hashCode, you need to choose a prime, usually 31, but you can also choose other prime numbers e.g. 37, 17 etc. The reason for choosing these prime numbers are to generate a uniquely distributed hashcode, which eventually helps to avoid collision, when used in hash based collections like Hashtable and HashMap.

Another worth noting thing is using all variables, used in equals, in hashCode method to keep equals and hashCode consistent, and adhere to rules of overriding hashCode in Java. I have already discussed about Comparable and compareTo method, while sorting list of objects in Java.

A simple implementation of compareTo must return negative number if current object is lesser than provided object, positive if current object is greater in order than provided and zero if both objects are equal. Another worth noting point is that, compareTo() and equals() must be consistent with each other, otherwise your object will not behave properly on Collection classes, which uses compareTo() for sorting e.g. TreeSet or TreeMap. Anyway, here is a simple example of overriding equals, hashcode and compareTo in Java.

Sample implementation of Equals, HashCode and CompareTo in Java

In this sample example of overriding equals, hashcode and compareTo method, we will use a class named Person which has 3 properties String name, int id and Date to represent date of birth. We will also use Generics along with Comparable to provide a type safe implementation. Remember we have used getClass() method instead of instanceof operator, which means a Person class cannot be equal to its subclass, this could create problem if you are using this class in EJB or any application server, where there is a chance that same class is loaded by two separate class loader. On those cases it's better to use instanceof operator because it will allow a Class to be equal to its subclass if rest of properties matched. This is also true for framework like Hibernate, which provides proxy implementation, which is essentially sub class of entity classes. In short, use instanceof if your class can be loaded by multiple class loader or it can be used by framework to create proxies.

```
/**
```

```java
/**
 * Simple Java Class to represent Person with name, id and date of birth.
 *
 * @author Javin Paul
 */
public class Person implements Comparable<Person>{
  private String name;
  private int id;
  private Date dob;

   public Person(String name, int id, Date dob) {
      this.name = name;
      this.id = id;
      this.dob = dob;
   }

  @Override
  public boolean equals(Object other){
     if(this == other) return true;

     if(other == null || (this.getClass() != other.getClass())){
        return false;
     }

     Person guest = (Person) other;
     return (this.id == guest.id) &&
           (this.name != null && name.equals(guest.name)) &&
           (this.dob != null && dob.equals(guest.dob));
  }

   @Override
  public int hashCode(){
     int result = 0;
     result = 31*result + id;
     result = 31*result + (name !=null ? name.hashCode() : 0);
     result = 31*result + (dob  !=null ? dob.hashCode() : 0);

     return result;
  }

   @Override
   public int compareTo(Person o) {
      return this.id - o.id;
   }
}
```

Sample JUnit test case for testing Equals and HashCode

Here is simplest of simple test case to verify equals and hashCode in Java. Remember this unit test is not enough, if you want to verify all properties of equals and hashCode. If you should write test cases to check if it verify key properties of equals and hashcode method e.g. if a equal b then b should also be equal to a, or if a a equals b and b equals c then a and c should also be equal to each other. You must also verify whether compareTo() return 0 for equal object and return non zero value for non equal objects.

```
import java.util.Date;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 * Simple example of using equals and hashCode method
 * @author Javin Paul
 */
public class EqualsAndHashCodeTest {


  @Test
  public void testEquals(){
     Person james = new Person("James", 21, new Date(1980,12, 1));
     Person same = new Person("James", 21, new Date(1980,12, 1));
     Person similar = new Person("Harry", 21, new Date(1981,12,1));

     assertTrue(james.equals(same));
     assertTrue(james.hashCode() == same.hashCode());

     assertFalse(james.equals(null));
     assertFalse(james.equals(similar));
     assertTrue(james.hashCode() != similar.hashCode());
  }

}
```

That's all on this **simple example of overriding equals, hashcode and compareTo method in Java**. As I said, these test are just to verify that equals and hashCode methods are working, you can add more test to verify compareTo, and other behaviors of all three methods.

Related **Java Programming Articles and Tutorials** from Java67

Core Java Interview Questions for 2 to 4 years experienced Programmers
10 Java Design Pattern Interview Questions with Answers
When to use ArrayList and LinkedList in Java
Difference between Vector and ArrayList in Java
Difference between ConcurrentHashMap and HashMap in Java

1) What is String in Java? Is String is data type?
String in Java is not a primitive data type like int, long or double. The string is a class or in more simple term a user defined type. This is confusing for someone who comes from C background. String is defined in java.lang package and wrappers its content in a character array. String provides equals() method to compare two String and provides various other methods to operate on String like toUpperCase() to convert String into upper case, replace() to replace String contents, substring() to get substring, split() to split long String into multiple String.

2) Why is String final in Java?
The string is final by design in Java, some of the points which make sense why String is final is Security, optimization and to maintain a pool of String in Java. for details on each of this point see Why String is final in Java.

3) What is the difference between String and StringBuffer in Java?
This is probably the most common question on String I have seen in Java interviews. Though String and StringBuffer are two different class they are used in the context of concatenating two Strings, Since String is immutable in Java every operation which changes String produces new String, which can be avoided by using StringBuffer. See String vs StringBuffer for more details.

4) What is the difference in String on C and Java?
If you have mentioned C in your resume, then you are likely to face this String interview question. Well, C String and Java String are completely different to each other, C String is a null terminated character array while String in Java is an Object. Also, String is more feature rich in Java than C.

5) Why char array is better than String for storing password?
This String interview question is debatable and you might not agree with interviewer but this is also a chance to show that how deep and differently you can think of. One of the reasons which people give Why you should store a password in char array over String is related to immutability since it's not possible to erase contents of String but you can erase contents of a char array. See Why char array preferred over String for a password for a complete discussion.

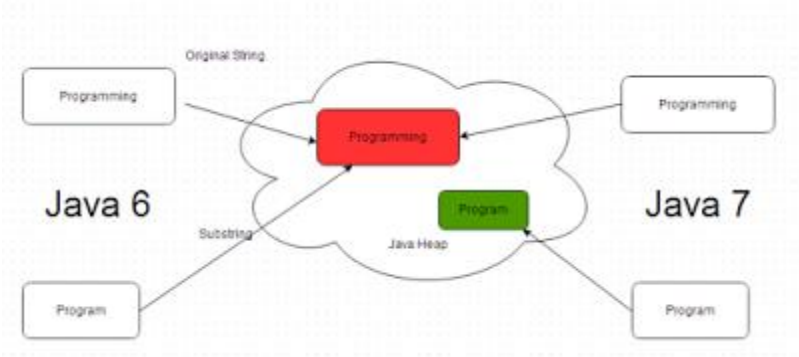
6) How do you compare two String in Java?
This is another common String interview question which appears on fresher level interviews. There are multiple ways to compare two String like equals() method, equalsIgnoreCase() etc, You can also see 4 ways to compare String in Java for more examples. The main thing which interviewer checks is that whether candidate mentioned equality operator or not "==", comparing String with equality operator is a common mistake which works in some case and doesn't work in other. next String interview question is follow-up up of this.

7) Can we compare String using == operator? What is the risk?
As discussed in previous String question, You can compare String using equality operator but that is not suggested or advised because equality operator is used to compare primitives and equals() method should be used to compare objects. As we have seen in the pitfall of autoboxing in Java that how equality operator can cause a subtle issue while comparing primitive to Object, anyway String is free from that issue because it doesn't have a corresponding primitive type and not participate in autoboxing.

Almost all the time comparing String means comparing contents of String i.e. characters and equals() method is used to perform character-based comparison. equals() return true if two String points to the same object or two String has same contents while == operator returns true if two String object points to the same object but return false if two different String object contains same contents. That explains why sometimes it works and sometimes it doesn't.

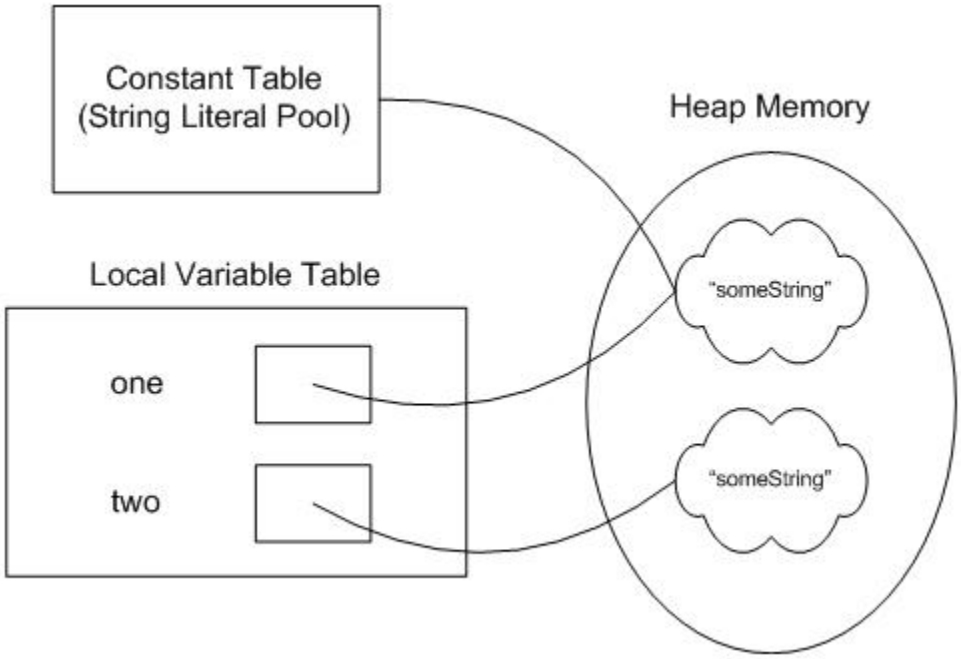
In short always use equals method in Java to check equality of two String object.

8) How does substring method work in Java?
This is one of the tricky Java question relate to String and until you are familiar with the internals of String class, it's difficult to answer. Substring shares same character array as original String which can create a memory leak if original String is quite big and not required to retain in memory but unintentionally retained by substring which is very small in size and prevents large array from begin claimed during Garbage collection in Java.



9) What is String pool in Java?

Another tough Java question asked in String interview. String pool is a special storage area in Java heap, mostly located on PerGen space, to store String literals like "ABC". When Java program creates a new String using String literal, JVM checks for that String in the pool and if String literal is already present in the pool than the same object is returned instead of creating a whole new object. String pool check is only performed when you create String as literal, if you create String using new() operator, a new String object will be created even if String with the same content is available in the pool.

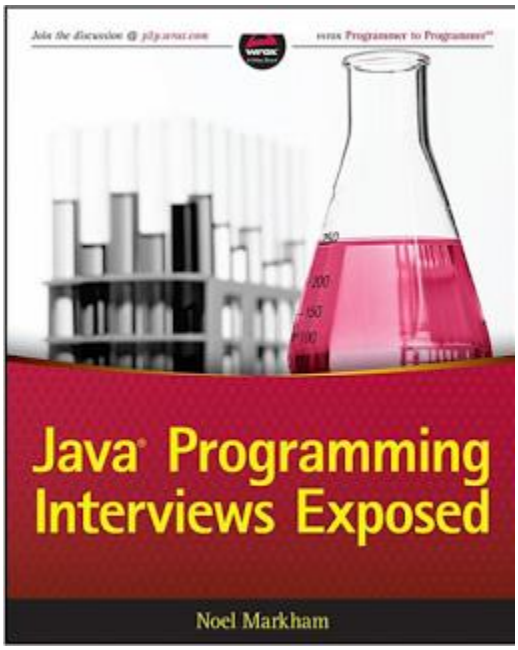


10) What does intern() method do in Java?

As discussed in previous String interview question, String object created by new() operator is by default not added in String pool as opposed to String literal. The intern method allows putting a String object into a pool.

11) Is string thread-safe in Java?
If you are familiar with the concept of immutability and thread-safety you can easily answer this String interview question in Java. Since String is immutable, it is thread-safe and it can be shared between multiple threads without external synchronization.

If you are seriously preparing for Java interviews and do not want to leave any stone unturned, I strongly suggest you go through questions given in Java Programming Interview Exposed, one of the rare book which covers all important topics for Java interviews.



String based Coding Questions
These questions are most based upon Java's implementation of String and you can only answer them well if you have good knowledge of java.lang.String class. But, String is very general data structure and you will find it in almost all programming and script language e.g. C, C++, C#, Python, Perl or Ruby. That's why I am going to share some more String based coding question, which is not Java specific. You can solve these question in any programming language as they are mostly logic based programming question.

1) Write a Java program to reverse String in Java without using any API? (solution)
This means you can not use StringBuffer's reverse() method or any of String utility method, all you can have is a character array for reversing contents.

2) Write a Program to check if a String is a palindrome or not? (solution)
For example, a String e.g. "madam" is a palindrome but "book" is not a palindrome. You also need to solve this question without taking any help from Java String API.

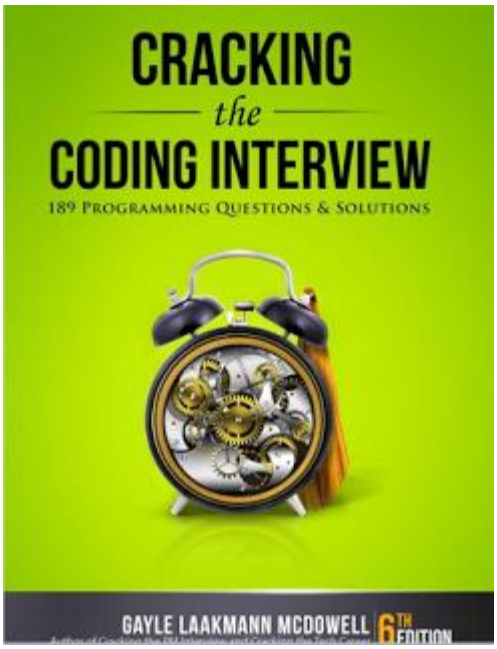
3) Write a Java program to check if two String are Anagram or not? (solution)
You need to write method e.g. isAnagram(String first, String second) which will return true if second String is an anagram of the first string. An anagram must contain the same number of characters and exactly same characters but in different order e.g. top and pot, or army and mary.

4) Write a method in Java to remove any character from String? (solution)
For example, you need to write method remove(String word, char removeThis), this method should return a String without character, which is asked to remove. you can use indexOf(), substring() and similar methods from String class, but your method must handle corner cases e.g. passing null or empty String, String containing just one character etc.

5) Write a method to split a comma separated String in Java? (solution)

6) Write Java program to print all permutations of a String e.g. passing "ABC" will print all permutations like "BCA", "CBA" etc (solution)

If you are hungry for more String based coding question, you can also check the Cracking the Coding Interview book, a collection of 189 programming questions and solutions from various programming job interviews of reputed tech companies like Amazon, Google, Facebook, and Microsoft.



That's all about **Java String interview question and answers**. In Summary, there are a lot of specifics about String which needs to be known for anyone who has started Java programming and these String question will not just help to perform better on Java Interviews but also opens the new door of learning about String. I didn't know much String related concepts until I come across these question which motivated to research and learns more about String in Java.

Read more: http://javarevisited.blogspot.com/2012/10/10-java-string-interview-question-answers-top.html#ixzz4R58zwaPe

Important points about Garbage Collection in Java

This article is in continuation of my previous articles How Classpath works in Java and How to write Equals method in Java and before moving ahead let's recall few important points about garbage collection in Java.

1) Objects are created on the heap in Java irrespective of their scope e.g. local or member variable. while it's worth noting that class variables or static members are created in method area of Java memory space and both heap and method area is shared between different thread.

2) Garbage collection is a mechanism provided by Java Virtual Machine to reclaim heap space from objects which are eligible for Garbage collection.

3) Garbage collection relieves Java programmer from memory management which is an essential part of C++ programming and gives more time to focus on business logic.

4) Garbage Collection in Java is carried by a daemon thread called Garbage Collector.

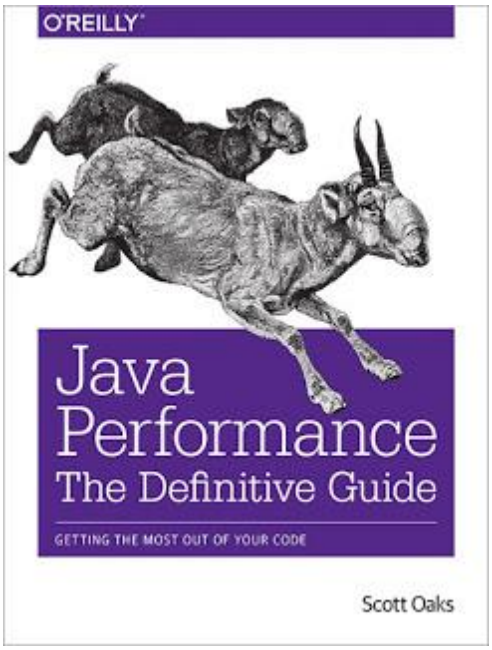
5) Before removing an object from memory garbage collection thread invokes finalize() method of that object and gives an opportunity to perform any sort of cleanup required.

6) You as Java programmer can not force garbage collection in Java; it will only trigger if JVM thinks it needs a garbage collection based on Java heap size.

7) There are methods like System.gc() and Runtime.gc() which is used to send request of Garbage collection to JVM but it's not guaranteed that garbage collection will happen.

8) If there is no memory space for creating a new object in Heap Java Virtual Machine throws OutOfMemoryError or java.lang.OutOfMemoryError heap space

9) J2SE 5(Java 2 Standard Edition) adds a new feature called Ergonomics goal of ergonomics is to provide good performance from the JVM with a minimum of command line tuning. See Java Performance The Definitive Guide for more details on garbage collection tuning.



When an Object becomes Eligible for Garbage Collection

An object becomes eligible for Garbage collection or GC if it's not reachable from any live threads or by any static references. In other words, you can say that an object becomes eligible for garbage collection if its all references are null. Cyclic dependencies are not counted as the reference so if object A has a reference to object B and object B has a reference to Object A and they don't have any other live reference then both Objects A and B will be eligible for Garbage collection.

Generally, an object becomes eligible for garbage collection in Java on following cases:

1) All references to that object explicitly set to null e.g. object = null

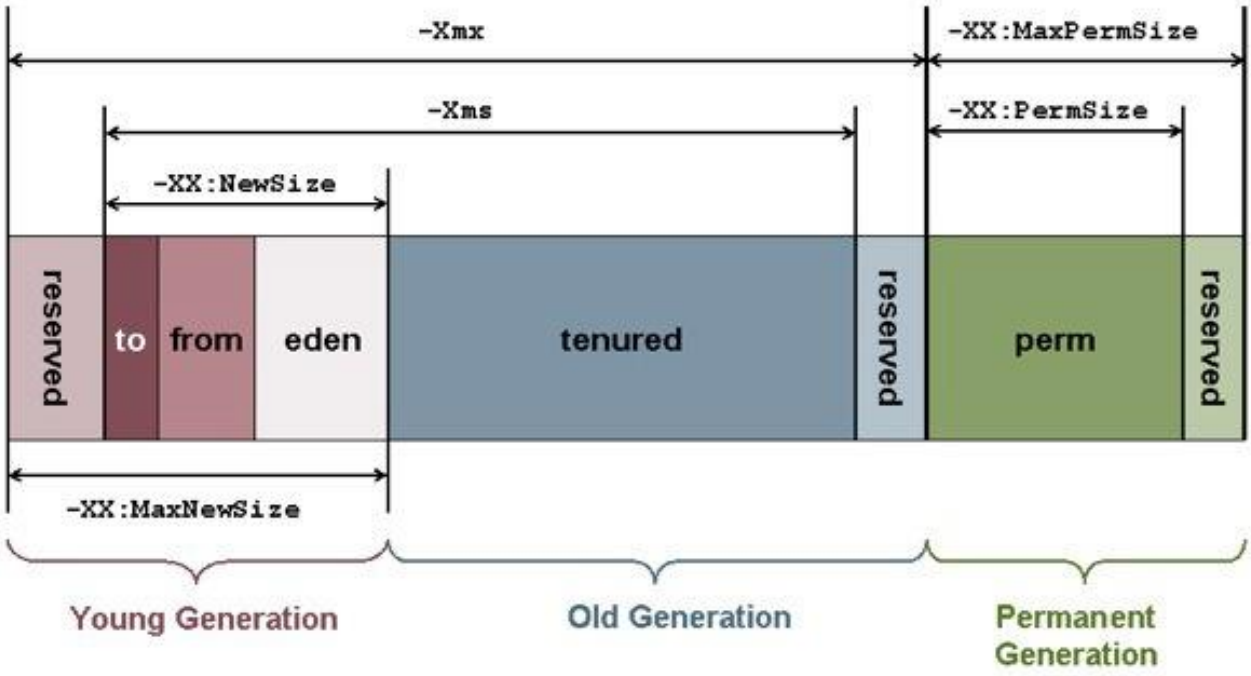
2) The object is created inside a block and reference goes out scope once control exit that block.

3) Parent object set to null if an object holds the reference to another object and when you set container object's reference null, child or contained object automatically becomes eligible for garbage collection.

4) If an object has only lived weak references via WeakHashMap it will be eligible for garbage collection.

Heap Generations for Garbage Collection in Java

Java objects are created in Heap and Heap is divided into three parts or generations for the sake of garbage collection in Java, these are called as Young generation, Tenured or Old Generation and Perm Area of the heap. New Generation is further divided into three parts known as Eden space, Survivor 1 and Survivor 2 space. When an object first created in heap its gets created in new generation inside Eden space and after subsequent minor garbage collection if an object survives its gets moved to survivor 1 and then survivor 2 before major garbage collection moved that object to old or tenured generation.



Permanent generation of Heap or Perm Area of Heap is somewhat special and it is used to store Metadata related to classes and method in JVM, it also hosts String pool provided by JVM as discussed in my string tutorial why String is immutable in Java. There are many opinions around whether garbage collection in Java happens in perm area of Java heap or not, as per my knowledge this is something which is JVM dependent and happens at least in Sun's implementation of JVM. You can also try this by just creating millions of String and watching for Garbage collection or OutOfMemoryError.

Types of Garbage Collector in Java

Java Runtime (J2SE 5) provides various types of Garbage collection in Java which you can choose based on your application's performance requirement. Java 5 adds three additional garbage collectors except serial garbage collector. Each is generational garbage collector which has been implemented to increase the throughput of the application or to reduce garbage collection pause times.

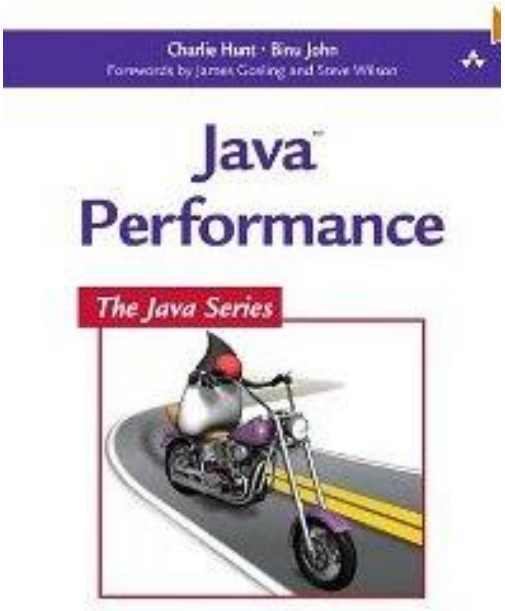
1) Throughput Garbage Collector: This garbage collector in Java uses a parallel version of the young generation collector. It is used if the -XX:+UseParallelGC option is passed to the runtime via JVM command line options . The tenured generation collector is same as the serial collector.

2) Concurrent low pause Collector: This Collector is used if the -Xingc or -XX:+UseConcMarkSweepGC is passed on the command line. This is also referred as Concurrent Mark Sweep Garbage collector. The concurrent collector is used to collect the tenured generation and does most of the collection concurrently with the execution of the application. The application is paused for short periods during the collection. A parallel version of the young generation copying collector is used with the concurrent collector. Concurrent Mark Sweep Garbage collector is most widely used garbage collector in java and it uses an algorithm to first mark object which needs to collect when garbage collection triggers.

3) The Incremental (Sometimes called train) low pause collector: This collector is used only if -XX:+UseTrainGC is passed on the command line. This garbage collector has not changed since the java 1.4.2 and is currently not under active development. It will not be supported in future releases so avoid using this and please see 1.4.2 GC Tuning document for information on this collector.

An important point to note is that -XX:+UseParallelGC should not be used with -XX:+UseConcMarkSweepGC. The argument passing in the J2SE platform starting with version 1.4.2 should only allow the legal combination of command line options for garbage collector but earlier releases may not find or detect all illegal combination and the results for illegal combination are unpredictable. It’s not recommended to use this garbage collector in java.

See Java Performance by Binu John and Charlie Hunt to learn more about different JVM options and garbage collection tuning and troubleshooting.



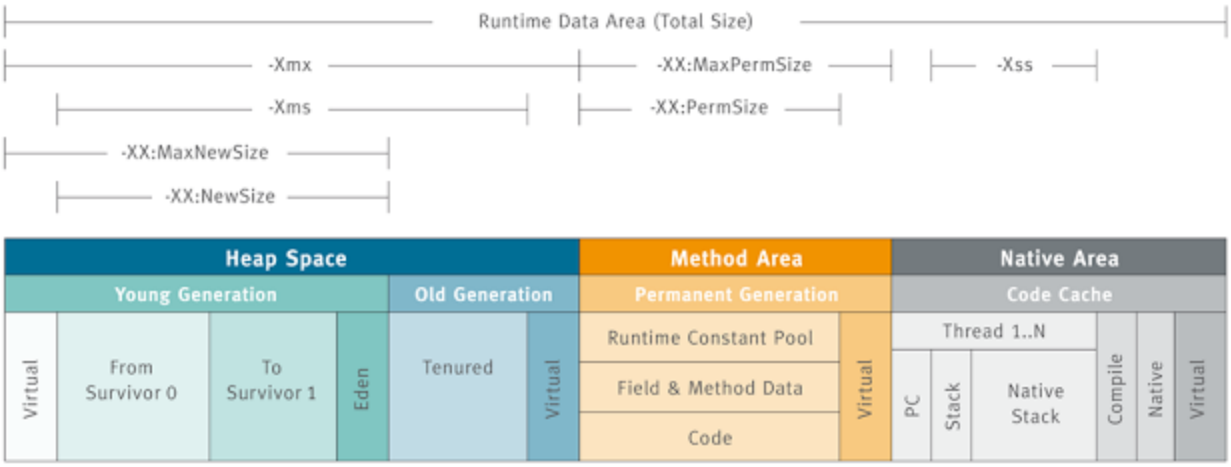
JVM Parameters for Garbage Collection in Java

Garbage collection tuning is a long exercise and requires a lot of profiling of application and patience to get it right. While working with High volume low latency Electronic trading system I have worked with some of the project where we need to increase the performance of Java application by profiling and finding what causing full GC and I found that Garbage collection tuning largely depends on application profile, what kind of object application has and what are their average lifetime etc.

For example, if an application has too many short lived object then making Eden space wide enough or larger will reduce the number of minor collections. you can also control the size of both young and Tenured generation using JVM parameters for example setting -XX:NewRatio=3 means that the ratio of the young and tenured generation is 1:3 , you got to be careful on sizing this generation.

As making young generation larger will reduce the size of the tenured generation which will force the Major collection to occur more frequently which pause application thread during that duration results in degraded or reduced throughput. The parameters NewSize and MaxNewSize are used to specify the young generation size from below and above. Setting these equal to one another fixes the young generation.

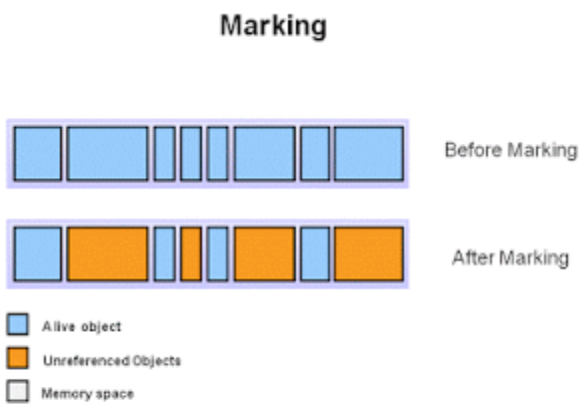
In my opinion, before doing garbage collection tuning detailed understanding of how garbage collection works in Java is a must and I would recommend reading Garbage collection document provided by Sun Microsystems for detail knowledge of garbage collection in Java. Also to get a full list of JVM parameters for a particular Java Virtual machine please refer official documents on garbage collection in Java. I found this link quite helpful though http://www.oracle.com/technetwork/java/gc-tuning-5-138395.html



Full GC and Concurrent Garbage Collection in Java

The concurrent garbage collector in java uses a single garbage collector thread that runs concurrently with the application threads with the goal of completing the collection of the tenured generation before it becomes full. In normal operation, the concurrent garbage collector is able to do most of its work with the application threads still running, so only brief pauses are seen by the application threads.

As a fallback, if the concurrent garbage collector is unable to finish before the tenured generation fills up, the application is paused and the collection is completed with all the application threads stopped. Such Collections with the application stopped are referred as full garbage collections or full GC and are a sign that some adjustments need to be made to the concurrent collection parameters.



Always try to avoid or minimize full garbage collection or Full GC because it affects the performance of Java application.

When you work in finance domain for an electronic trading platform and on a high volume low latency systems performance of Java application becomes extremely critical you definitely like to avoid full GC during the trading hours.

Summary on Garbage collection in Java

1) Java Heap is divided into three generation for the sake of garbage collection. These are a young generation, tenured or old generation, and Perm area.

2) New objects are created by young generation and subsequently moved to the old generation.

3) String pool is created in PermGen area of Heap, garbage collection can occur in perm space but depends upon JVM to JVM. By the way from JDK 1.7 update, String pool is moved to heap area where objects are created.

4) Minor garbage collection is used to move an object from Eden space to survivor 1 and survivor 2 space and major collection is used to move an object from young to tenured generation.

5) Whenever Major garbage collection occurs application threads stop during that period which will reduce application's performance and throughput.

6) There are few performance improvements has been applied in garbage collection in java 6 and we usually use JRE 1.6.20 for running our application.

7) JVM command line options –Xmx and -Xms is used to setup starting and max size for Java Heap. The ideal ratio of this parameter is either 1:1 or 1:1.5 based on my experience, for example, you can have either both –Xmx and –Xms as 1GB or –Xms 1.2 GB and 1.8 GB.

8) There is no manual way of doing garbage collection in Java, but you can use various reference classes e.g. WeakReference or SoftReference to assist garbage collector.

Read more: http://javarevisited.blogspot.com/2011/04/garbage-collection-in-java.html#ixzz4R59oPuo6

Q1. Should we create system software (e.g Operating system) in Java ?

Ans. No, Java runs on a virtual machine called JVM and hence doesn't embed well with the underlying hardware. Though we can create a platform independent system software but that would be really slow and that's what we would never need.

Q2. What are the different types of memory used by JVM ?

Ans. Class , Heap , Stack , Register , Native Method Stack.

Q3. What are the benefits of using Spring Framework ?

Ans. Spring enables developers to develop enterprise-class applications using POJOs. The benefit of using only POJOs is that you do not need an EJB container product.

Spring is organized in a modular fashion. Even though the number of packages and classes are substantial, you have to worry only about ones you need and ignore the rest.

Spring does not reinvent the wheel instead, it truly makes use of some of the existing technologies like several ORM frameworks, logging frameworks, JEE, Quartz and JDK timers, other view technologies.

Testing an application written with Spring is simple because environment-dependent code is moved into this framework. Furthermore, by using JavaBean-style POJOs, it becomes easier to use dependency injection for injecting test data.

Spring's web framework is a well-designed web MVC framework, which provides a great alternative to web frameworks such as Struts or other over engineered or less popular web frameworks.

Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate, or JDO, for example) into consistent, unchecked exceptions.

Lightweight IoC containers tend to be lightweight, especially when compared to EJB containers, for example. This is beneficial for developing and deploying applications on computers with limited memory and CPU resources.

Spring provides a consistent transaction management interface that can scale down to a local transaction

Q4. What are various types of Class loaders used by JVM ?

Ans. Bootstrap - Loads JDK internal classes, java.* packages.

Extensions - Loads jar files from JDK extensions directory - usually lib/ext directory of the JRE

System - Loads classes from system classpath.

Q5. What is PermGen or Permanent Generation ?

Ans. The memory pool containing all the reflective data of the java 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. The Permanent generation contains metadata required by the JVM to describe the classes and methods used in the application. The permanent generation is populated by the JVM at runtime based on classes in use by the application. In addition, Java SE library classes and methods may be stored here.

Q6. What is metaspace ?

Ans. The Permanent Generation (PermGen) space has completely been removed and is kind of replaced by a new space called Metaspace. The consequences of the PermGen removal is that obviously the PermSize and MaxPermSize JVM arguments are ignored and you will never get a java.lang.OutOfMemoryError: PermGen error.

Q7. How does volatile affect code optimization by compiler?

Ans. Volatile is an instruction that the variables can be accessed by multiple threads and hence shouldn't be cached. As volatile variables are never cached and hence their retrieval cannot be optimized.

Q8. What things should be kept in mind while creating your own exceptions in Java?

Ans. All exceptions must be a child of Throwable.

If you want to write a checked exception that is automatically enforced by the Handle or Declare Rule, you need to extend the Exception class.

You want to write a runtime exception, you need to extend the RuntimeException class.

Q9. What is the best practice configuration usage for files - pom.xml or settings.xml ?

Ans. The best practice guideline between settings.xml and pom.xml is that configurations in settings.xml must be specific to the current user and that pom.xml configurations are specific to the project.

Q10. Can you provide some implementation of a Dictionary having large number of words ?

Ans. Simplest implementation we can have is a List wherein we can place ordered words and hence can perform Binary Search.

Other implementation with better search performance is to use HashMap with key as first character of the word and value as a LinkedList.

Further level up, we can have linked Hashmaps like ,

hashmap {
a (key) -> hashmap (key-aa , value (hashmap(key-aaa,value)
b (key) -> hashmap (key-ba , value (hashmap(key-baa,value)
.....
z(key) -> hashmap (key-za , value (hashmap(key-zaa,value)
}

upto n levels (where n is the average size of the word in dictionary.

Q11. What is database deadlock ? How can we avoid them?

Ans. When multiple external resources are trying to access the DB locks and runs into cyclic wait, it may makes the DB unresponsive.

Deadlock can be avoided using variety of measures, Few listed below -

Can make a queue wherein we can verify and order the request to DB.

Less use of cursors as they lock the tables for long time.

Keeping the transaction smaller.

Q12. Why Web services use HTTP as the communication protocol ?

Ans. With the advent of Internet, HTTP is the most preferred way of communication. Most of the clients (web thin client , web thick clients , mobile apps) are designed to communicate using http only. Web Services using http makes them accessible from vast variety of client applications.

Q13. Why using cookie to store session info is a better idea than just using session info in the request ?

Ans. Session info in the request can be intercepted and hence a vulnerability. Cookie can be read and write by respective domain only and make sure that right session information is being passed by the client.

Q14. Difference between first level and second level cache in hibernate ?

Ans. 1. First level cache is enabled by default whereas Second level cache needs to be enabled explicitly.

2. First level Cache came with Hibernate 1.0 whereas Second level cache came with Hibernate 3.0.

3. First level Cache is Session specific whereas Second level cache is shared by sessions that is why First level cache is considered local and second level cache is considered global.

Q15. What are the ways to avoid LazyInitializationException ?

Ans. 1. Set lazy=false in the hibernate config file.

2. Set @Basic(fetch=FetchType.EAGER) at the mapping.

3. Make sure that we are accessing the dependent objects before closing the session.

4. Using Fetch Join in HQL.

Q16. What are new features introduced with Java 8 ?

Ans. Lambda Expressions , Interface Default and Static Methods , Method Reference , Parameters Name , Optional , Streams, Concurrency.

Q17. What things you would care about to improve the performance of Application if its identified that its DB communication that needs to be improved ?

Ans. 1. Query Optimization (Query Rewriting , Prepared Statements)
2. Restructuring Indexes.
3. DB Caching Tuning (if using ORM)
4. Identifying the problems (if any) with the ORM Strategy (If using ORM)

Q18. If you are given a choice to implement the code to either Insert a Record or Update if already exist, Which approach will you follow ?

1. Insert into the DB Table. If exception occurs, update the existing record.
2. Check if the record exists and update it if it exists, If not insert a new record.

Ans. In first case, there would be 2 DB calls in worst case and 1 in best case. In 2nd approach there will be always 2 DB calls.

Decision on the approach should depend on the following considerations -

1. How costly is the call to DB ? Are we using indices , hibernate etc

If calls to DB are costly , 1st approach should be the choice.

2. Exception Book keeping load upon exception.

The benefit of saving 1st call in approach 1 should be bigger than the Book keeping for the exception.

3. Probability of the exception in first apparoach.

If the DB Table is almost empty, it makes sense to follow Approach 1 as majority of the 1st calls will pass through without exception.

Q19. What would you do if you have to add a jar to the project using Maven ?

Ans. If its already there in Maven local repository, We can add that as a dependency in the project pom file with its Group Id, Artifact Id and version.

We can provide additional attribute SystemPath if its unable to locate the jar in the local repository.

If its not there in the local repository, we can install it first in the local repository and then can add it as dependency.

Q20. Should we create system software (e.g Operating system) in Java ?

Ans. No, Java runs on a virtual machine called JVM and hence doesn't embed well with the underlying hardware. Though we can create a platform independent system software but that would be really slow and that's what we would never need.

Q21. Which UML diagrams you usually use for design ?

Ans. Use Case Diagram, Component Diagram for High level Design and Class Diagram , Sequence Diagram for low level design.

Q22. How do you coordinate and communicate with the team developers ?

Ans. We as a team of developers , testers , analyst , lead and architect sit close to each other. Most of the time I would just jump to their seat and talk to them (if required). We have daily stand up where we discuss things that needs team attention.

Q23. What kind of software architecture your organization follow ?

Ans. We have multi tier architecture with multiple layers , We have series of web servers and applications in application tier, infrastructure libraries at middle tier and Database servers at the lower tier. We are using Oracle as Database, ESB (Enterprise service Bus) for asynchronous communication and Rest Web Services.

Q24. Difference between Proxy and Adapter Deisgn Patterns ?

Ans. Adapter object has a different input than the real subject whereas Proxy object has the same input as the real subject. Proxy object is such that it should be placed as it is in place of the real subject.

Q25. Difference between Adapter and Facade ?

Ans. The Difference between these patterns in only the intent. Adapter is used because the objects in current form cannot communicate where as in Facade , though the objects can communicate , A Facade object is placed between the client and subject to simplify the interface.

Q26. Difference between Builder and Composite ?

Ans. Builder is a creational Design Pattern whereas Composite is a structural design pattern. Composite creates Parent - Child relations between your objects while Builder is used to create group of objects of predefined types.

Q27. Difference between Factory and Strategy Design Pattern ?

Ans. Factory is a creational design pattern whereas Strategy is behavioral design pattern. Factory revolves around the creation of object at runtime whereas Strategy or Policy revolves around the decision at runtime.

Q28. Shall we use abstract classes or Interfaces in Policy / Strategy Design Pattern ?

Ans. Strategy deals only with decision making at runtime so Interfaces should be used.

Q29. Semaphores

A Semaphore is a thread synchronization construct that can be used either to send signals between threads to avoid missed signals, or to guard a critical section like you would with a lock. Java 5 comes with semaphore implementations in the `java.util.concurrent` package so you don't have to implement your own semaphores. Still, it can be useful to know the theory behind their implementation and use.

Java 5 comes with a built-in Semaphore so you don't have to implement your own. You can read more about it in the `java.util.concurrent.Semaphore` text, in my `java.util.concurrent` tutorial.