**Why use Inner class.**  
I would say Inner class is never mandatory or necessary, there is always workaround  
But at some places they are convinience like in Map.Entry, Iterator inside List, or  
listener in GUI.  
The use is to increase encapsulation and hide some implementation details.  
Classes like Semaphore use internal objects for locking, etc.  
  
**Serialization**  
We are writing the Java object in a file, and again creating the java object by reading bytes form the file.  
Above is a kind of a persistence mechanism, but Serialization purpose is not persistence, there are DB's for that.  
Use of serialization is to transfer objects over network.  
We can write the Java object as byte array in memory, and later again construct the object from byte array in memory,  
OR we can write the object over socket, data is sent over wire to receiving socket, and there it would be constructed again.  
OR we convert Java objects in XML instances and later construct back from XML in SOAP context.  
Serialization is the mechanism used by RMI to pass objects between JVMs,  
either as arguments in a method invocation from a client to a server or as return values from a method invocation.  
We should know how to make class Serializable, and also about interface Externlizable.  
  
 **Printing alternate numbers where arraylist is getting modified**  
We can get a iterator, and we might get COncurrentModificationException.  
May be we should have used CopyOnWriteArrayList used to avoid that.  
  
  
**How to implement a Hospital scenario, where doctor is examining patients, and emergency cases,**  
**and informing doctor of emergency cases, scheduling, etc.**  
We can use a priority Queue.  
  
  
**How to ensure only one instance in Singleton class.**  
class Singleton  
{  
  private static Singleton instance = new Singleton();  
  
  private Singleton()  
  {  
    //...  
  }  
  
  public static Singleton getInstance()  
  {  
    return instance;  
  }  
}  
Refer - http://www.ibm.com/developerworks/java/library/j-dcl/index.html  
  
**Synchronization -**  
Inside the Java virtual machine, each thread is awarded a Java stack, which contains data no other thread can access,  
including the local variables, parameters, and return values of each method the thread has invoked.  
All objects reside on the heap.There is only one heap inside the JVM, and all threads share it.  
Besides the Java stack and the heap, the other place data may reside in the JVM is the method area,  
which contains all the class (or static) variables used by the program.  
To coordinate shared data access among multiple threads, the Java virtual machine associates a lock with each object and class.  
A lock is like a privilege that only one thread can "possess" at any one time.  
If a thread wants to lock a particular object or class, it asks the JVM.  
When the thread no longer needs the lock, it returns it to the JVM. If another thread has requested the same lock, the JVM passes the lock to that thread.  
The JVM uses locks in conjunction with monitors. A monitor is basically a guardian in that it watches over a sequence of code,  
making sure only one thread at a time executes the code.  
  
Each monitor is associated with an object reference. When a thread arrives at the first instruction in a block of code that is under the watchful eye of a monitor,  
the thread must obtain a lock on the referenced object.  
In Java language terminology, the coordination of multiple threads that must access shared data is called synchronization.  
Two opcodes, monitorenter and monitorexit, are used for synchronization blocks within methods.  
When monitorenter is encountered by the Java virtual machine, it acquires the lock for the object referred to by objectref  
on the stack. If the thread already owns the lock for that object, a count is incremented. Each time monitorexit is executed  
for the thread on the object, the count is decremented. When the count reaches zero, the monitor is released.  
- See more at: http://www.javaworld.com/javaworld/jw-07-1997/jw-07-hood.html?page=2#sthash.VWVX3ZYW.dpuf  
  
**Is Synchronization Enough?**  
One major purpose of using synchronization is to prevent the race condition that can cause data to be found in intermediate or inconsistent state. But when applying synchronization blocks, we need to take care of few important things -   
1. Threads might be racing prior to the synchronized section of code.   
2. Order in which threads execute the synchronized code is determined by order in which lock is granted.  
3. Can we design code so that it is threadsafe with minimal synchronization.   
Atomic Classes is answer to point number 3.   
  
And next question can be -   
  
**Why use Atomic Classes?**  
1. Simple class to perform atomic operations, like AtomicInteger allows integer value to incremented atmically.   
2. Build a complex code with no synchronization, using Atomic Classes  
But note Atomic Classes not capable of providing notification's like we have in synchronization: wait/notify.   
  
**Difference between Semaphore and Lock ?**  
1. Lock can have condition and only thread can acquire lock, semaphore has totally different use. Semaphore allows one or more threads to grab permits, and thus only dictates the number of threads working in parallel.  
2. Lock is used where we want a piece of code to be executed by only thread. Semaphore is used when we want to limit the number of threads running in parallel and accessing some resources like for example we want only 10 threads to run at any time in parallel no matter if they share any code or data and no matter any condition; or we want to limit the number of connections, etc.   
3. No Condition variables available in semaphore.  
4. Locks can be nested like once thread enters synchronized block it can call any other synchronized method on same object. In semaphore multiple acquisitions from same thread results in multiple permits.    
5. Semaphore constructor requires the number: no of permits. Semaphore has methods to return the number of total and free permits.   
6. Semaphore with number of permits equal to 1, is similar to lock.   
  
**Difference between Cyclic Barrier and Countdown Latch ?**  
1. Important thing in barrier is number(which is increment till it reaches its limit) and in latch is count(which is decremented till it reaches 0).   
2. Any thread can lower the count of latch by calling countDown(), i.e. without waiting, this can't happen in barrier.   
3. Latch can't be reused.   
4. In barrier we can have a BarrierAction, nothing similar is in Latch.   
  
**Why use Thread pool?**  
For better throughput.   
For eg., if 3 threads with equal priority run in time sliced manner and each one takes 3 secs to complete. Then each one will finish probably in 9 secs. But if we run these three sequentially then one will finish in 3 secs, next in 6 and last in 9 secs. And if threads are running in time sliced manner and number of threads keep on increasing, then each thread will take more time may be 30 secs for each thread if there are 10 threads. Keeping things sequential+parallel i.e., by controlling order of execution and maximum no of threads we can control delay and increase throughput  
  
**JGroup**  
JGroups is a toolkit for reliable multicast communication.  
It can be used to create groups of processes whose members can send messages to each other  
The most powerful feature of JGroups is its flexible protocol stack

**Garbage Collection -**

http://shekup.blogspot.in/2011/11/java-runtime-memory-management.html  
  
  
 **When are class garbage collected, when are classes unloaded.**  
The only way that a Class can be unloaded is if the Classloader used is garbage collected.   
 **How to ensure that instance is never garbage collected.**  
A singleton kind of pattern.   
There's a static reference to a singleton, so it won't be eligible for garbage collection until the   
classloader is eligible for garbage collection.  
  
**How is HashMap internally implemented?**  
pair is stored in HashMap. Pair is an entry to hashMap. Entries are stored in an array. So we have an array of entries.   
Each Entry has a key. We calculate hash(key.hashcode()), which determines the index in array. hash method shortens the hashcode value to a valid int index.   
The value at any index in array is called bucket, which holds the Entry.   
If for two Entries index comes out to be same, while storing. This is possible if two keys have same hashCode value. Then those two entries are stored in same bucket. Bucket is a linkedlist ate any index in array, and bucket holds the Entries. So we can have multiple entries in same bucket.   
When get(Key) operation is called. We do following -   
1. Calculate the hash(key.hashcode), so that we know the index of the bucket in array.   
2. If bucket holds more than one Key, use equals(on key) to return the exact match.   
  
**How to make existing code with Maps thread safe?**  
Collections.synchronizedMap might help  
http://docs.oracle.com/javase/7/docs/api/java/util/Collections.html#synchronizedMap(java.util.Map)   
  
  
**How to implement a polling mechanism in Java. How if I want to poll something. How if client is sending some requests to server and if request doesn't arrives in within time limit or if no requests for like 2 minutes, stop expecting more requests and suspend operation/listening/waiting ?**  
Use Class ArrayBlockingQueue  
Also, an useful method of ArrayBlockingQueue -

public [E](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ArrayBlockingQueue.html) poll(long timeout,

[TimeUnit](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/TimeUnit.html) unit)

throws [InterruptedException](http://docs.oracle.com/javase/7/docs/api/java/lang/InterruptedException.html)

**Difference between String, StringBuffer, and StringBuilder**

1) String is immutable while **StringBuffer and StringBuilder is mutable** object.

2) **StringBuffer is**[**synchronized**](http://www.blogger.com/goog_1642539054)while **StringBuilder is not** which makes StringBuilder faster than StringBuffer.

3) Concatenation operator "+" is internal implemented using either StringBuffer or StringBuilder.

4) Use String if you require [immutability](http://javarevisited.blogspot.sg/2010/10/why-string-is-immutable-in-java.html), use Stringbuffer in java if you need mutable + [thread-safety](http://javarevisited.blogspot.sg/2012/01/how-to-write-thread-safe-code-in-java.html) and use StringBuilder in Java if you require mutable + without thread-safety.

**Why use Generics -**  
Benefits of using Generics -  
1. Stronger type checks at compile time. And thus avoid casting errors at run-time.  
2. No need of casting. No extra code for casting like String s = (String) list.get(0).  
3. Better, Intelligent, Generic, and good looking programming.

**J2EE -**  
**------**  
  
**Lightweight J2EE web application**  
A typical architecture for J2ee web application is  
Web layer - Business Layer - Database  
JSP - Servlet - Session Bean - Entity Bean  
which later evolved to  
JSF UI - JSF Backing Bean - EJB3 Session Bean - JPA  
Or using some implementation like ADF instead of JSF  
ADF - EJB3 Session Bean - JPA/Toplink  
OR, if using Struts -  
Struts - Business layer  
OR  
Spring MVC - JPA/Hibernate  
OR even  
JSF - Spring WebFlow - JPA/Hibernate  
Some used below  
Struts - Spring - Hibernate  
There was a recent survey done over web frameworks, and result is Spring MVC is most adopted framework  
(http://www.infoq.com/research/jvm-web-frameworks?utm\_source=infoqresearch&utm\_campaign=rr-content)  
Now remember and note there is one very imprtant layer in the front Client Technologies  
Client Technologies + Web Layer - Business layer - Persistence layer  
Client Technologies include HTML, CSS, Javascript, JQuery, Ajax, JSON, DOJO, REST, FLEX, etc.  
  
If I go more back and look into evolution  
  
HTML(high performance poor UI)  
 -> CGI(dynamic content, low performance, still bit poor UI)  
 -> JavaScript  
 -> AJAX  
and somewhere Applets also came.  
But, Google changed the architecture of web applications, esp. when intorduced web applications like  
Google Map using JavaScript, AJAX, JSON,which is a web application but looks like Desktop with amazing UI.  
Google Maps used JavaScript extensively, where when user drags the map and grid squares are downloaded from the server.  
And Web application architecture started thinking of making CLIENT more rich.  
So, Web Architecture, which was  
CLIENT(HTML) - Server(View generation - Controllers - Service layer/Business layer - Repository)  
has changed to  
CLIENT(HTML & JS also DOM, controllers, objects) - Server(Service layer/Business layer - Repository)  
And there is a need to break big service layer into independent services,  
And may be independent services are dependent on other servics coming from platform or cloud.  
And now we consider where ever we can use REST.  
  
**We have a very huge message, that we want to break into smaller parts and send. While sending we have to take care of hundereds of clients. How we can design this using JMS?**  
Hundreds of clients means hundreds of consumers. We have to publish the message which will be consumed by many consumers. So, we will use a topic for publishing message.   
Now since we are going to break the message in multiple parts and send them one by one. Consumer should be able to add all those parts received one by one, and constitute a final message. What we want is that consumer should be able to co-relate all those messages, and in JMS way to co-relate different messages is using CorrelationID.   
  
**How to handle security in Java/J2EE applications?**  
This is a huge topic, but I have summarized some points which are most important ones in my below post -   
<http://shekup.blogspot.in/2014/04/how-to-handle-security-in-javaj2ee.html>  
  
**Web Services**  
  
**Different protocols for web services -**   
  
**XML-RPC** (RPC) - Its a simple, portable way to make remote procedure calls over HTTP.   
An XML-RPC message is an HTTP-POST request. The body of the request is in XML.   
A procedure executes on the server and the value it returns is also formatted in XML.  
Procedure parameters can be scalars, numbers, strings, dates, etc.; and can also be complex record and list structures.  
HTTP request is send. No WSDL.    
(http://tldp.org/HOWTO/XML-RPC-HOWTO/xmlrpc-howto-intro.html)

**JSON-RPC**(RPC) - All transfer types are single objects, serialized using JSON. https://code.google.com/p/json-rpc/wiki/Usage  
JSON-RPC is lightweight remote procedure call protocol similar to XML-RPC  
  
**SOAP**(SOA, needs WSDL)  
SOAP like XML-RPC, but lot more features, requires WSDL.   
  
**REST**  
SOAP uses interfaces and named operations to expose business logic. REST uses (generally) URI and methods like (GET, PUT, POST, DELETE) to expose resources.  
  
**Managing Session in Web Services**  
Normally, a JAX-WS Web service is stateless:   
that is, none of the local variables and object values that you set in the Web service object are saved from one invocation   
to the next.   
Even sequential requests from a single client are treated each as independent, stateless method invocations.  
Enabling stateful support in a JAX-WS Web service requires a minimal amount of coding on both the client and server.  
And there can be several approaches.   
Like what we can do is - Inject WebServiceCOntext in Web service Impl class.   
We can access the HttpSessionObject from WebServiceContext.   
Save objects in HttpSession using setAttribute, and can later retrieved using getAttribute.   
(Check for more details - http://docs.oracle.com/cd/E17904\_01/web.1111/e13734/stateful.htm)  
  
  
**EJB**  
  
***How can EJB/business layer client can execute a use case's business logic in one transaction and in one call.***  
EJB Session Facade pattern.   
Wrap Entity beans in a layer of session beans, and clients accessing sesison beans.   
Some precautions must be taken while designing the session facde -   
1. Avoid very big facade.  
2. Avoid duplicate code in facade methods.   
I remember a design where we first wrote a Java class(AFacade) with all the implementation code.   
AFacade implements a interface ASession. and we have a session bean implementation class ASessionImpl.  
AsessionImpl extends AFacade. Generally we were writing very few lines in the ASessionImpl,  
since all the implementation code was already there in AFacade.   
Benefits, the business logic was also going in a separate application, a jar file.  
Jar file was a batch implementation. Some clients which do not want to make remote lookup,  
or are not running in the app server, were directly using the jar file.   
Same code when becomes part of EJB, there are capabilities added -   
1. Its exposed for remote lookup  
2. It can exposed as Web Service  
3. transaction capabilities are added  
4. Low Coupling  
5. Clear separation of other layers(like presentation). It represents athe business layer.   
App Server provides multiple options like pooling, caching so using EJB is not a performance issue.   
  
**How client can make a asynchronous call along with transaction capabilities.**  
Message Driven Beans  
A Message Driven Bean can also act as a facade.   
Client send a message to a JMS Queue, MDB's onMessage is invoked.   
onMessage can be in transaction, and if it fails, the message can be rolled back and sent back to Queue.   
Some precautions must be taken whiloe using MDB -   
1. The request, should be properly constructed. We can objects also in reuqest.   
   There are several options for request.   
2. Prevent Message loss.   
3. Transaciton handling.   
4. Exception handling, because actual exeption wont be propagated automatically to client.   
5. Response to client, because there is no automatic response send.   
  
**How to prevent Message loss in Message Driven Bean -**  
Please read a small post   
http://shekup.blogspot.in/2012/02/prevent-message-loss-in-message-driven\_21.html   
  
**Session Bean and pooling -**  
When Server is first started, several Stateless session bean instances are created and placed in the Ready pool. More instances might be created by the container as needed by the EJB container.  When a bean instance is in the ready state, it can service client requests; that is, execute component methods. When a client invokes a business method, the EJB container assigns an available bean instance to execute the business method. Once execution has finished, the session bean instance is ready to execute another business method.  When the EJB container decides to reduce the number of session bean instances in the ready pool, it makes the bean instance ready for garbage collection. Just prior to doing this, it calls the callback method ejbRemove. If your session bean needs to execute some cleanup action prior to garbage collection, you can implement it using this callback method. The callback method is not tied to the remove method invoked by a client. For a stateless session bean, calling the remove method invalidates the reference to the bean instance already in the ready pool, but it does not move a bean instance from the ready to the does not exist state, as the management of stateless session bean instances is fully done by the EJB container.    
If all the beans form pool are active, any new client comes, it would be blocked if transaction time outs or time out occurs, ***Server would throw RemoteException to remote clients and EJBException to local clients.***  
For stateful session bean, When a client invokes a create method on a stateful session bean, the EJB container creates a new instance.  To more optimally manage resources, the EJB container might passivate an inactive stateful session bean instance by moving it from the ready state to the passive state. When a session bean instance is passivated, its (non-transient) data is serialized and written to disk, after which the bean instance is purged from memory, and if cache is full instance would be passivated to disk. For session bean there is algo to remove the bean, like time out, where bean is totally discarded, when bean is in passive state for time longer than session time out. Unlike anonymous stateless sesison beans, stateful have identity, which is used to bound it to client.   
***One important point we should know is that pool is use for stateless(either EJB or thread or anything), and cache is used for statefull(either EJB or entity or anything).***  
***We always prefer stateless session bean over stateful.***  
  
**EJB Transactions -**   
  
EJB provides some options for setting the transaction. We can set the transactionattribute -   
***REQUIRED***- Methods executed within a transaction. If client provides transaction, it is used; if not, new transaction generated. Commit at end of method. Default attribute set by WebSphere Studio. Well-suited for EJB Sessions.  
***MANDATORY***- Client of this EJB must create a transaction in which this method operates, otherwise an error. Well-suited for EJB Entitys.  
***REQUIRES\_NEW***- Methods executed within a transaction. If client provides transaction, it is suspended. A new transaction is generated, regardless. Commit at end of method.  
***SUPPORTS***- Transactions optional.  
***NOT\_SUPPORTED***- Transactions not supported; if provided, ignored.  
Transaction attributes help us create boundaries  but they don't solve the common problems faced, obsolete data or concurrent modification.   
Some common problems related to data are -   
***Dirty Reads -*** A transaction reads data written by another transaction that has not been committed yet. Because this data is uncommitted, a transaction failure would roll back these read changes.  
Occurs when one transaction (T2) reads data that has been modified by previously started transaction (T1), but not committed  
What happens if the T1 rolls back? T1 has incorrect data, thus "dirty read".  
***Nonrepeatable reads -*** A transaction rereads data it has previously read and finds that data has been modified by another committed transaction in the meantime.

Occurs when one transaction (T1) reads same data twice, while another transaction (T2) modifies the data between the two reads by T1.

T1 gets different value between the first and the second read, thus "nonrepeatable read".

***Phantom reads -*** A transaction reexecutes a query returning a set of rows that satisfy a search condition and finds that the set of rows satisfying the condition has changed due to another committed transaction in the meantime.

Occurs when one transaction begins reading data and another inserts to or deletes data from the table being read.

Above mentioned problems are solved by setting the isolation level, which can be provided by App server or DB.

**Web Development -**  
  
**Difference between JQuery and AJAX -**  
Both are different. JQuery is a Javascript llibrary. AJAX is a combination of technologies , its a functionality. In order to use AJAX, you would need a client side scripting language that allows you to detect the actions of the user and modify elements on the page accordingly. jQuery does that exactly, that is why both are often used together to present web pages that a user can interact with easily without repetitive loading.  jQuery gives you an AJAX command to retrieve whichever data you need from the server.  
  
**A modern Web Application in 2014-**ORM, Spring framework, REST Services, Javascript MVC framework, and probably MEAN stack.

### Does BlockingQueue is thread-safe? (Yes, take() and put() method of this class guarantees thread-safety, no need to externally synchronize this class for adding and retrieving objects, here is an example of this class to solve [producer consumer problem in Java](http://javarevisited.blogspot.sg/2012/02/producer-consumer-design-pattern-with.html)) SOAP versus RESTful Web service -- comparison

Recently I attended an interview with a large investment bank, and I was quizzed on SOAP versus RESTful web service. The interview questions were targeted at ascertaining my understanding of the differences, pros and cons of each, and when to use what.   
  
Web services are very popular and widely used to integrate similar (i.e. Java applications) and disparate systems (i.e. legacy applications and applications written in .Net etc). It is imperative to understand thedifferences, pros, and cons between each approach.

|  |  |  |
| --- | --- | --- |
| **Key Area** | **SOAP based Web service** | **RESTful Web service** |
| Specification/Platform Fundamentals (SF/PF) | Transport is platform & protocol neutral. Supports multiple protocols like HTTP(S), Messaging, TCP, UDP, SMTP, etc.  Permits only XML data format, hence language neutral.  You define operations, which tunnels through the **POST**or**GET**. The focus is on accessing the named operations and exposing the application logic as a service.  Defines the contract via **WSDL**. | Transport is protocol specific. Supports only HTTP or HTTPS protocols.  Permits multiple data formats like XML, JSON data, text, HTML, atom, RSS, etc.  Any browser can be used because the REST approach uses the standard **GET, PUT, POST,**and **DELETE** web operations. The focus is on accessing the named resources and exposing the data as a service.  Traditionally, the big drawback of REST was the lack of contract for the web service. This has changed with WSDL 2.0 defining non SOAP bindings and the emergence of **WADL**.  Simpler to implement. REST has Ajax support. It can use the *XMLHttpRequest* object.  Good for stateless CRUD (Create, Read, Update, and Delete) operations. |
| Performance Consideration (PC) | SOAP based reads cannot be cached. The application that uses SOAP needs to provide cacheing. | REST based reads can be cached. Performs and scales better. |
| Security (SE) | Supports both **SSL security**and **WS-security**, which adds some enterprise security features. Supports identity through intermediaries, not just point to point SSL.  WS-Security maintains its encryption right up to the point where the request is being processed.  WS-Security allows you to secure parts (e.g. only credit card details) of the message that needs to be secured. Given that encryption/decryption is not a cheap operation, this can be a performance boost for larger messages.  It is also possible with WS-Security to secure different parts of the message using different keys or encryption algorithms. This allows separate parts of the message to be read by different people without exposing other, unneeded information.  SSL security can only be used with HTTP. WS-Security can be used with other protocols like UDP, SMTP, etc. | Supports only point-to-point **SSL security**.  The basic mechanism behind SSL is that the client encrypts all of the requests based on a key retrieved from a third party. When the request is received at the destination, it is decrypted and presented to the service. This means the request is only encrypted while it is traveling between the client and the server. Once it hits the server (or a proxy which has a valid certificate), it is decrypted from that moment on.  The SSL encrypts the whole message, whether all of it is sensitive or not. |
| Transaction Management (TM) | Has comprehensive support for both**ACID**based transaction management for short-lived transactions and **compensation** based transaction management for long-running transactions. It also supports two-phase commit across distributed resources. | REST supports transactions, but it is neither ACID compliant nor can provide two phase commit across distributed transactional resources as it is limited by its HTTP protocol. |
| Quality of Service (QoS) | SOAP has success or retry logic built in and provides end-to-end reliability even through SOAP intermediaries. | REST does not have a standard messaging system, and expects clients invoking the service to deal with communication failures by retrying. |
| Best Practice (BP) | In general, a REST based web service is preferred due to its simplicity, performance, scalability, and support for multiple data formats. SOAP is favored where service requires comprehensive support for security, transactional reliability and stricter contract. | |

**Q.**Differentiatebetween**SOA** (Service Oriented Architecture) versus **WOA** (Web Oriented Architecture)?  
**A**. WOA extends SOA to be a light-weight architecture using technologies such as REST and POX (Plain Old XML). POX compliments REST. **JSON** is a variant for data returned by REST Web Services. It consumes less bandwidth and is easily handled by web developers mastering the Javascript language  
  
**SOA** and **WOA** differ in terms of the layers of abstraction. SOA is a **system-level architectural style**that tries to expose business capabilities so that they can be consumed by many applications. **WOA** is an interface-level architectural style that focuses on the means by which these service capabilities are exposed to consumers. You can start out with a WOA and then grow into SOA.

How will you go about ensuring that you build a more robust application? or How do you improve quality of your application?  
**A3**. This question is very popular with the interviewers because they want to hire candidates who write good quality application. Here are some useful links to prepare for this question.

* [Java interview questions and answers on code quality.](http://java-success.blogspot.com/2011/10/judging-java-experience-questions-and.html) Gives a high level over view as to things you can do and tools you can use to improve code quality. A common popular job interview question answered is -- **What is the difference between fake objects, mock objects, and stubs?**
* [Unit testing with mock objects interview questions and answers](http://java-success.blogspot.com/2011/10/unit-testing-with-mock-objects.html). Tutorial that explains how to write **JUnit**tests with mock objects using the **Mockito**and **Power mock** frameworks.
* [Java Interview Questions and Answers - performance testing your Java application](http://java-success.blogspot.com/2012/03/java-interview-questions-and-answers.html). A high level overview as to how you can go about monitoring performance. Many applications face performance issues and this is a very popular question with the interviewers.

**Q**6. How would you go about designing a car parking station?   
**A6**.   
  
**Map out the requirements:**

* The car park needs to cater for different types of car parks like regular, handicapped, and compact.
* It should keep track of empty and filled spaces.
* It should also cater for valet parking.

**Map out the classes that would be required.** Use a [UML class diagram.](http://java-success.blogspot.com/2012/07/uml-diagrams-interview-questions-and.html) Here are some points to get started.

* A CarPark class to represent a parking station.
* A ParkingSpace can be an abstract class or an interface to represent a parking space, and RegularParkingSpace, HandicappedParkingSpace, CompactParkingSpace, etc are subtypes of a ParkingSpace. This means a RegularParkingSpace **is a**ParkingSpace.
* A CarPark **has a** (i.e. composition) finite number of ParkingSpaces. A CarPark also keeps track of all the parking spaces and a separate list of all the vacant parking spaces.
* A Vehicle class **uses a** (i.e. delegation) ParkingSpace.  The Vehicle class will hold attributes using enum classes like  VehicleType and ParkingType. The vehicle types could be Compact, Regular, and Handicapped. The parking types could be Self or Valet. Depending on the requirements, the self or valet types could be designed as subtypes of the Vehicle class.

**Q7**: Can you design the classes that represent a restaurant?   
**A7**: Very popular. Good understanding of **OO design** is vital to job interview success. Learn the **SOLID**design principles, know why you**favor composition over inheritance**, and**don't under estimate the power of coding to interface**. 

* [Java OO Interview Questions and Answers](http://java-success.blogspot.com/2011/09/java-oo-interview-questions-and-answers.html). A step-by-step tutorial explaining how you would go about designing your classes. The "Core Java Career Essentials" PDF covers more examples on SOLID design principles with working code.

How would you go about identifying **thread safety issues** in an application? How would you go about identifying **memory leaks** issues? How would you go about identifying **performance issues**?, etc to judge your experience. I am yet to work on a project that didn't face problems relating to thread safety, memory leaks, and performance issues. Hence, it really pays to market your skills and experience in fixing issues relating to these common challenges.   
  
  
**Q8**. How will you go about fixing memory leaks in Java?  
**A8**. Again, a very common problem, and a thorough answer will go a long way in securing your next Java job.

* [Detecting and fixing memory leak issues in Java](http://java-success.blogspot.com/2012/05/memory-leak-in-java.html). Tools and techniques to detect memory leaks in Java.
* [Memory profiling in Java](http://java-success.blogspot.com/2012/05/java-memory-profiling-questions-and.html). Tutorial on memory profiling a Java application.

**Q9**: How will you go about fixing performance issues in Java?  
**A9**:

* [Detecting performance issues in Java](http://java-success.blogspot.com/2012/05/java-performance-questions-and-answers.html). A basic tutorial covering -- how to detect performance issues.
* Also, refer to the JMeter tutorials in this blog as to learn how you can put load on your application to simulate concurrent users.

**Q10**: How will you go about detecting and fixing thread-safety issues in Java?  
**A10**: Debugging concurrency issues and fixing any **thread starvation,** **dead lock**, and **contention**requires skills and experience to identify and reproduce these hard to resolve issues. Here are some techniques to detect concurrency issues.

* Manually reviewing the code for any obvious thread-safety issues. There are static analysis tools like Sonar, ThreadCheck, etc for catching concurrency bugs at compile-time by analyzing their byte code.
* List all possible causes and add extensive log statements and write test cases to prove or disprove your theories.
* Thread dumps are very useful for diagnosing synchronization problems such as deadlocks. The trick is to take 5 or 6 sets of thread dumps at an interval of 5 seconds between each to have a log file that has 25 to 30 seconds worth of runtime action. For thread dumps, use kill -3 in Unix and CTRL+BREAK in Windows. There are tools like Thread Dump Analyzer (TDA), Samurai, etc. to derive useful information from the thread dumps to find where the problem is. For example, Samurai colors idle threads in grey, blocked threads in red, and running threads in green. You must pay more attention to those red threads.
* There are tools like JDB (i.e. Java DeBugger) where a “watch” can be set up on the suspected variable. When ever the application is modifying that variable, a thread dump will be printed.
* There are dynamic analysis tools like jstack and JConsole, which is a JMX compliant GUI tool to get a thread dump on the fly. The JConsole GUI tool does have handy features like “detect deadlock” button to perform deadlock detection operations and ability to inspect the threads and objects in error states. Similar tools are available for other languages as well.

**Q**. What is the difference between mutex and semaphores? What problem do they solve?  
**A**.   
  
**Example 1**: A trading system creates a buy order and places it in a queue, and a separate consumer thread picks up the order and sends it to the stock exchange. A typical producer and consumer scenario.  
  
**Example 2**: If you want to recursively traverse through nested folders and spawn a number of worker threads to move the html files found to a destination folder and increment the semaphore permits with a release( ) method call. A separate thread will wait with the acquire(numberOfPermits) to acquire all the released permits before start archiving (i.e. zipping up) those files.   
  
**Example 3**: A counter that keeps track of the number of active logged in users by incrementing the count when users log in and decrementing the count when the users log out.  
  
[Mutex and semaphores are used to solve producer and consumer synchroniztion](http://java-success.blogspot.com/2012/03/java-multi-threading-interview.html) scenarios outline above..    
  
**Q**. Can you write code to demonstrate deadlock in Java?   
**A**. [Creating a deadlock in Java](http://java-success.blogspot.com/2012/03/java-multi-threading-interview.html)  
  
**Q**. Can you give some examples of thread racing conditions you had experienced?  
**A**. [Thread racing conditions and atomic operations.](http://java-success.blogspot.com/2012/03/java-multi-threading-interview.html)

Three of the Java keywords that are very popular in job interviews are **1**) volatile **2**) transient **3**) const  
  
1) **volatile** is popular because it can be a bit tricky to understand and know when to use it.  
2) **transient** is popular as it is not widely used, but handy in object serialization.  
3) **const** is popular because it is a reserved keyword, but is not currently used. The final keyword  on a reference variable just means that the reference cannot be changed to reference a different object. But, the object itself can be changed if their fields are not final. The const is reserved to make the whole object not changeable, but currently not used.  
  
Let's focus now on the "volatile" keyword.  
  
**Q.**What is a **volatile** key word in Java?  
**A**. The volatile keyword is used with object and primitive variable references to indicate that a variable's value will be modified by different threads. This means

* The value of this variable will never be cached locally within the thread, and all the reads and writes must go to the main memory to be **visible** to the other threads**.**In other words the keyword volatile **guarantees visibility.**
* From JDK 5 onwards**,**writing to a volatile variable happens before reading from a volatilevariable. In other words, the volatile keyword guarantees**ordering,**and prevents compiler or JVM from reordering of the code.

**Q.**How does  **volatile** keyword differ from the **synchronized** keyword?  
**A.**

1. The **volatile** keyword is applied to variables of both primitives and objects**,**whereas the synchronized keyword is applied to only objects.
2. The volatile keyword only guarantees visibility and ordering, but not atomicity, whereas the synchronized keyword can guarantee both visibility and atomicity if done properly. So, the volatilevariable has a limited use, and cannot be used in compound operations like incrementing a variable, etc.

Wrong use of volatile in a compound operation 

[view plainprint?](http://java-success.blogspot.in/2014/08/what-is-volatile-variable-in-java-and.html?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed:+Java/JEECareerCompanion+(Java/JEE+Career+Companion))

1. **volatile** **int** counter = 0;
3. **public** **void** increment(){
4. counter++;
5. }

Right use of volatile. **Example1**:

[view plainprint?](http://java-success.blogspot.in/2014/08/what-is-volatile-variable-in-java-and.html?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed:+Java/JEECareerCompanion+(Java/JEE+Career+Companion))

1. **volatile** **boolean** status = **false**;
3. //...
5. **public** **void** process(){
6. **while**(!status){
7. //....
8. }
9. }

Or in lazy singleton. **Example2**: Double checked locking

[view plainprint?](http://java-success.blogspot.in/2014/08/what-is-volatile-variable-in-java-and.html?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed:+Java/JEECareerCompanion+(Java/JEE+Career+Companion))

1. **public** **final** Class MySingleton {
3. **private** **static** **volatile** MySingleton instance = **null**;
5. **private** MySingleton( ){}
7. **public** **static** MySingleton getInstance() {
8. **if**(instance == **null**) {
9. **synchronized** (MySingleton.**class**) {
10. **if**(instance == **null**) {
11. instance = **new** MySingleton();
12. }
13. }
14. }
16. **return** instance;
17. }
19. }

**Important**: Synchronized keyword (i.e. locking) can guarantee both visibility and atomicity, whereasvolatile variables can only guarantee visibility. A synchronized block can be used in place of volatile but the inverse is not true.  
  
So, if you are not sure where to use, then favor the "synchronized" keyword.  
  
  
**Q.** Why is locking of a method or block of code for thread safety is called "**synchronized**" and not "**lock**" or "**locked**"?  
**A.** When a method or block of code is locked with the reserved "synchronized" key word in Java, the memory (i.e. heap) where the shared data is kept is synchronized. This means,

* When a synchronized block or method is entered after the lock has been acquired by a thread, it first **reads any changes to the locked object from the main heap memory** to ensure that the thread that has the lock has the current info before start executing.
* After the synchronized  block has completed and the thread is ready to relinquish the lock, **all the changes that were made to the object that was locked is written or flushed back to the main heap memory** so that the other threads that acquire the lock next has the current info.

This is why it is called "synchronized" and not "locked". This is also the reason why the immutable objects are inherently thread-safe and does not require any synchronization. Once created, the immutable objects cannot be modified.