# Java Concepts

<http://www.tutorialspoint.com/java/java_interview_questions.htm>

What is volatile **variable** in Java?

**volatile** is a special modifier, which can only be used with instance variables. In concurrent Java programs, changes made by multiple threads on instance variables is not visible to other in absence of any synchronizers e.g. synchronized keyword or locks.

**Volatile** variable guarantees that a write will happen before any subsequent read

What is thread-safety? is Vector a thread-safe class? (Yes, see details)

Thread-safety is a property of an object or code which guarantees that if executed or used by multiple thread in any manner e.g. read vs write it will behave as expected. For example, a thread-safe counter object will not miss any count if same instance of that counter is shared among multiple threads. Apparently, you can also divide collection classes in two category, thread-safe and non-thread-safe. Vector is indeed a thread-safe class and it achieves thread-safety by synchronizing methods which modifies state of Vector, on the other hand, its counterpart ArrayList is not thread-safe.

What is race condition in Java? Given one example?

Race condition are cause of some subtle programming bugs when Java programs are exposed to concurrent execution environment. As name suggests, race condition occurs due to race between multiple threads, if a thread which is supposed to execute first lost the race and executed second, behaviour of code changes, which surface as non-deterministic bugs. This is one of the hardest bugs to find and re-produce because of random nature of racing between threads. One example of race condition is out-of-order processing, see this answer for some more example of race conditions in Java programs.

**Race conditions occurs** when two thread operate on same object without proper synchronization and there operation interleaves on each other. Classical example of Race condition is incrementing a counter

Multithreading interview questions:

<http://javarevisited.blogspot.hk/2014/07/top-50-java-multithreading-interview-questions-answers.html>

Recursion to Iteration

<http://stackoverflow.com/questions/18883696/converting-an-iterative-method-to-a-recursive-one-java>

Immutable vs mutable

In object-oriented and functional programming, an immutable object is an object whose state cannot be modified after it is created. This is in contrast to a mutable object, which can be modified after it is created. In some cases, an object is considered immutable even if some internally used attributes change but the object's state appears to be unchanging from an external point of view. For example, an object that uses memorization to cache the results of expensive computations could still be considered an immutable object.

Immutable objects are often useful because they are inherently thread-safe. Other benefits are that they are simpler to understand and reason about and offer higher security than mutable objects.

Immutable class, with members by reference ?? 1. Make a copy of the reference; 2. Exclude set()

Immutable classes are those class, whose object can not be modified once created, it means any modification on immutable object will result in another immutable object.

Immutable object

no setter methods, no leaking of reference, keeping separate copy of mutable object to create Immutable object

best example to understand immutable and mutable objects are, String and StringBuffer

like java.util.Date, despite storing Date into final field it can be modified internally, if internal date is returned to the client. In order to preserve immutability in such cases, its advised to return copy of original object, which is also one of the Java best practice.

Inheritance vs Composition

Inheritance

When you establish an inheritance relationship between two classes, you get to take advantage of dynamic binding and polymorphism

Dynamic binding means the JVM will decide at runtime which method implementation to invoke based on the class of the object.

Polymorphism means you can use a variable of a superclass type to hold a reference to an object whose class is the superclass or any of its subclasses.

Composition

Given that the inheritance relationship makes it hard to change the interface of a superclass. When your goal is code reuse, composition provides an approach that yields easier-to-change code.

When to use inheritance, when to use Composition? For method overriding, code reusability

It is easier to add new subclasses (inheritance) than it is to add new front-end classes (composition), because inheritance comes with polymorphism. If you have a bit of code that relies only on a superclass interface, that code can work with a new subclass without change. This is not true of composition, unless you use composition with interfaces. Used together, composition and interfaces make a very powerful design tool

Composition allows you to delay the creation of back-end objects until (and unless) they are needed, as well as changing the back-end objects dynamically throughout the lifetime of the front-end object. With inheritance, you get the image of the superclass in your subclass object image as soon as the subclass is created, and it remains part of the subclass object throughout the lifetime of the subclass

With both composition and inheritance, changing the implementation (not the interface) of any class is easy. The ripple effect of implementation changes remain inside the same class.

If it is a permanent / constant “is-a” relationship, inheritance is preferred. Otherwise, impermanent “is-a” relationships should be modeled with composition

When to use **abstract class** and **interface** in Java

An abstract class is good if you think you will plan on using inheritance since it provides a common base class implementation to derived classes; they force your subclasses to provide others.

An abstract class is also good if you want to be able to declare non-public members. In an interface, all methods must be public.

If you think you will need to add methods in the future, then an abstract class is a better choice. Because if you add new method headings to an interface, then all of the classes that already implement that interface will have to be changed to implement the new methods. That can be quite a hassle.

Interfaces are a good choice when you think that the API will not change for a while.

Interfaces are also good when you want to have something similar to multiple inheritances, since you can implement multiple interfaces.

**Garbage collections**

<http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html>

<http://www.cubrid.org/blog/dev-platform/understanding-java-garbage-collection/>

<http://javarevisited.blogspot.hk/2011/04/garbage-collection-in-java.html>

<http://javarevisited.blogspot.hk/2012/10/10-garbage-collection-interview-question-answer.html>

Java Heap Space -

When a Java program started Java Virtual Machine gets some memory from Operating System. JVM uses this memory for all its need and part of this memory is call java heap memory. Heap in Java generally located at bottom of address space and move upwards. whenever we create object using new operator or by any another means object is allocated memory from Heap and When object dies or garbage collected ,memory goes back to Heap space in Java

1) Objects are created on heap in Java irrespective of there scope e.g. local or member variable. while its 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 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 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 minimum of command line tuning.

**Summary on GC**

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

2) New objects are created into young generation and subsequently moved to 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.

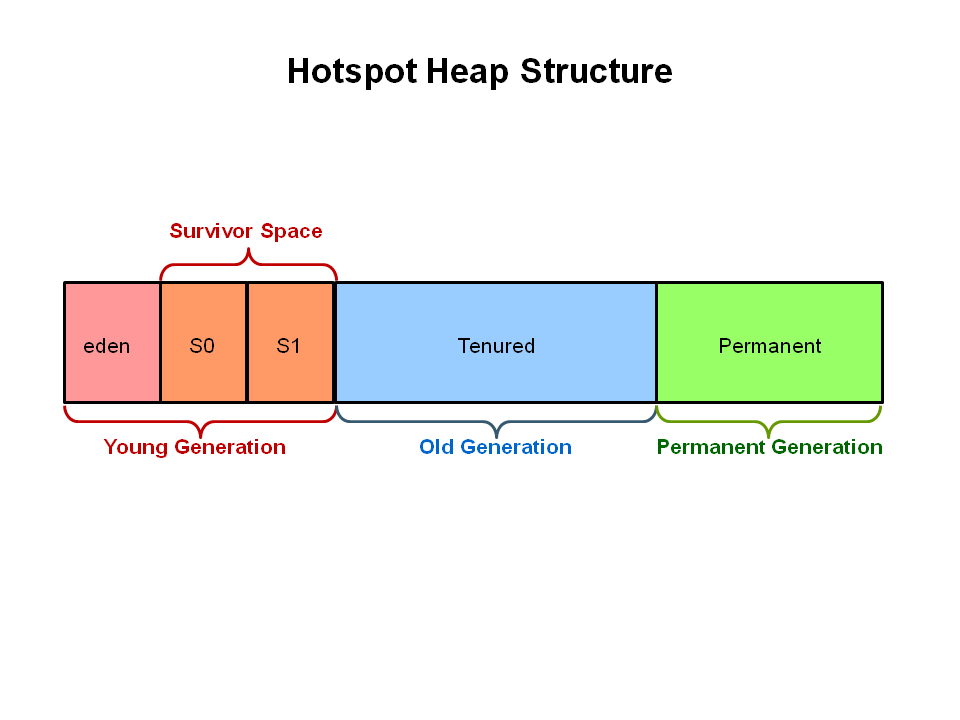
4) Minor garbage collection is used to move object from eden space to survivor 1 and survivor 2 space and major collection is used to move object from young to tenured generation.

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

6) There are few performance improvement 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. Ideal ratio of this parameter is either 1:1 or 1:1.5 based upon 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.

[](http://1.bp.blogspot.com/-MoOVJsk6fEc/VJmLWYxfgEI/AAAAAAAACQE/jggByJMXWhw/s1600/Java+Heap.PNG)

**Difference between Stack vs Heap in Java**

1) Main difference between heap and stack is that stack memory is used to store [local variables](http://javarevisited.blogspot.com/2012/02/difference-between-instance-class-and.html) and function call, while heap memory is used to store objects in Java. No matter, where object is created in code e.g. as member variable, local variable or class variable, they are always created inside heap space in Java.

2) Each [Thread in Java](http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html) has there own stack which can be specified using -Xss JVM parameter, similarly you can also specify heap size of Java program using JVM option -Xms and -Xmx where -Xms is starting size of heap and -Xmx is maximum size of java heap. to learn more about JVM options see my post [10 JVM option Java programmer should know](http://javarevisited.blogspot.com/2011/11/hotspot-jvm-options-java-examples.html).

3) If there is no memory left in stack for storing function call or local variable, JVM will throw java.lang.StackOverFlowError, while if there is no more heap space for creating object, JVM will throw java.lang.OutOfMemoryError: Java Heap Space. Read more about how to deal with java.lang.OutOfMemoryError in my post [2 ways to solve OutOfMemoryError in Java](http://javarevisited.blogspot.com/2011/09/javalangoutofmemoryerror-permgen-space.html).

4) If you are using [Recursion](http://javarevisited.blogspot.com/2012/12/recursion-in-java-with-example-programming.html), on which method calls itself, You can quickly fill up stack memory. Another difference between stack and heap is that size of stack memory is lot lesser than size of heap memory in Java.

5) Variables stored in stacks are only visible to the owner Thread, while objects created in heap are visible to all thread. In other words stack memory is kind of private memory of Java Threads, while heap memory is shared among all threads.

BFS – Find shortest path. Explore neighbor nodes first, before moving to the next level neighbors. Use Queue

DFS – Find shortest path. Use Stack

**JMS**

Persistence vs Durable

A "durable" JMS message is applicable only to publish/subscribe paradigm and the "persistent" JMS message is applicable to both "Point-to-Point" & "publish/subscribe" paradigms.   
  
A " **durable message** " is a message where the JMS server will hold on to a message if the subscriber is temporarily unavailable. So the durability is defined by the relationship between a "Topc Subscriber" and the "JMS Server". Durability is applicable only to publish/Subscribe paradigm. For this to happen subscribers need to register themselves with a unique " **client id** ".   
  
A " **persistent message** " is a message that defines the relationship between a "Message Producer" and the "JMS Server". This can be established for both point-to-point and publish/subscribe. This has to do with the guaranteed once only delivery of the message by persisting the message after it has been received from the message producer.

**Enhancement in Java 7**

* [Binary Literals](http://docs.oracle.com/javase/7/docs/technotes/guides/language/binary-literals.html) - In Java SE 7, the integral types (byte, short, int, and long) can also be expressed using the binary number system. To specify a binary literal, add the prefix 0b or 0B to the number.
* [Underscores in Numeric Literals](http://docs.oracle.com/javase/7/docs/technotes/guides/language/underscores-literals.html) - Any number of underscore characters (\_) can appear anywhere between digits in a numerical literal. This feature enables you, for example, to separate groups of digits in numeric literals, which can improve the readability of your code.
* [Strings in switch Statements](http://docs.oracle.com/javase/7/docs/technotes/guides/language/strings-switch.html) - You can use the String class in the expression of a switch statement.
* [Type Inference for Generic Instance Creation](http://docs.oracle.com/javase/7/docs/technotes/guides/language/type-inference-generic-instance-creation.html) - You can replace the type arguments required to invoke the constructor of a generic class with an empty set of type parameters (<>) as long as the compiler can infer the type arguments from the context. This pair of angle brackets is informally called the diamond.
* [Improved Compiler Warnings and Errors When Using Non-Reifiable Formal Parameters with Varargs Methods](http://docs.oracle.com/javase/7/docs/technotes/guides/language/non-reifiable-varargs.html) - The Java SE 7 complier generates a warning at the declaration site of a varargs method or constructor with a non-reifiable varargs formal parameter. Java SE 7 introduces the compiler option -Xlint:varargs and the annotations @SafeVarargs and @SuppressWarnings({"unchecked", "varargs"}) to supress these warnings.
* [The try-with-resources Statement](http://docs.oracle.com/javase/7/docs/technotes/guides/language/try-with-resources.html) - The try-with-resources statement is a try statement that declares one or more resources. A resource is an object that must be closed after the program is finished with it. The try-with-resources statement ensures that each resource is closed at the end of the statement. Any object that implements the new java.lang.AutoCloseable interface or the java.io.Closeable interface can be used as a resource. The classes java.io.InputStream, OutputStream, Reader, Writer, java.sql.Connection, Statement, and ResultSet have been retrofitted to implement the AutoCloseable interface and can all be used as resources in a try-with-resources statement.
* [Catching Multiple Exception Types and Rethrowing Exceptions with Improved Type Checking](http://docs.oracle.com/javase/7/docs/technotes/guides/language/catch-multiple.html) - A single catch block can handle more than one type of exception. In addition, the compiler performs more precise analysis of rethrown exceptions than earlier releases of Java SE. This enables you to specify more specific exception types in the throws clause of a method declaration.

**Enhancement in Java 8**

* [**Lambda Expressions**](http://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html) enable you to encapsulate a single unit of behavior and pass it to other code. You can use a lambda expressions if you want a certain action performed on each element of a collection, when a process is completed, or when a process encounters an error. Lambda expressions are supported by the following features:
  + [**Method References**](http://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html) are compact, easy-to-read lambda expressions for methods that already have a name.
  + [**Default Methods**](http://docs.oracle.com/javase/tutorial/java/IandI/defaultmethods.html) enable you to add new functionality to the interfaces of your libraries and ensure binary compatibility with code written for older versions of those interfaces. They are interface methods that have an implementation and the default keyword at the beginning of the method signature. In addition, you can define static methods in interfaces.
  + [**New and Enhanced APIs That Take Advantage of Lambda Expressions and Streams in Java SE 8**](http://docs.oracle.com/javase/8/docs/technotes/guides/language/lambda_api_jdk8.html) describe new and enhanced classes that take advantage of lambda expressions and streams.
* Improved Type Inference - The Java compiler takes advantage of target typing to infer the type parameters of a generic method invocation. The target type of an expression is the data type that the Java compiler expects depending on where the expression appears. For example, you can use an assignment statement's target type for type inference in Java SE 7. However, in Java SE 8, you can use the target type for type inference in more contexts. The most prominent example is using a method invocation's target types to infer the data types of its arguments.

Consider the following example:

List<String> stringList = new ArrayList<>();

stringList.add("A");

stringList.addAll(Arrays.asList());

Disregarding generics for the moment, the method addAll expects a Collection instance as its argument, and the method Arrays.asList returns a List instance. This works because List is a subtype of Collection.

Now considering generics, the target type of addAll is Collection<? extends String>, and Arrays.asList returns a List<T> instance. In this example, the Java SE 8 compiler can infer that the value of the type variable T is String. The compiler infers this from the target type Collection<? extends String>.

Compilers from Java SE 7 and earlier do not accept this code because they do not use target typing to infer types for method call arguments. For example, the Java SE 7 compiler generates an error message similar to the following:

error: no suitable method found for addAll(List<Object>) ...  
method List.addAll(Collection<? extends String>) is not applicable (actual argument List<Object> cannot be converted to Collection<? extends String> by method invocation conversion)

Consequently, in situations like this where the Java compiler cannot infer types, you must explicitly specify values for type variables with type witnesses. For example, the following works in Java SE 7:

List<String> stringList = new ArrayList<>();

stringList.add("A");

stringList.addAll(Arrays.<String>asList());

* Annotations on Java Types - It is now possible to apply an annotation anywhere a type is used. Used in conjunction with a pluggable type system, this allows for stronger type checking of your code. For more information, see [**Type Annotations and Pluggable Type Systems**](http://docs.oracle.com/javase/tutorial/java/annotations/type_annotations.html) in the new [**Annotations**](http://docs.oracle.com/javase/tutorial/java/annotations/) lesson in the Java Tutorial.
* Repeating Annotations - It is now possible to apply the same annotation type more than once to the same declaration or type use. For more information, see [**Repeating Annotations**](http://docs.oracle.com/javase/tutorial/java/annotations/repeating.html) in the new [**Annotations**](http://docs.oracle.com/javase/tutorial/java/annotations/) lesson in the Java Tutorial.
* Method Parameter Reflection - You can obtain the names of the formal parameters of any method or constructor with the method [**java.lang.reflect.Executable.getParameters**](http://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Executable.html#getParameters--). (The classes [**Method**](http://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Method.html) and [**Constructor**](http://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Constructor.html) extend the class [**Executable**](http://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Executable.html) and therefore inherit the method Executable.getParameters.) However, .class files do not store formal parameter names by default. To store formal parameter names in a particular .class file, and thus enable the Reflection API to retrieve formal parameter names, compile the source file with the -parameters option of the [**javac**](http://docs.oracle.com/javase/8/docs/technotes/guides/javac/index.html) compiler. See [**Obtaining Names of Method Parameters**](http://docs.oracle.com/javase/tutorial/reflect/member/methodparameterreflection.html) in the Java Tutorials.

Javascript library (jQuery, ExtJS, Dojo, YUI)

# Multithreading

Advantage of Java Multithreading

1) It doesn't block the user because threads are independent and you can perform multiple operations at same time.

2) You can perform many operations together so it saves time.

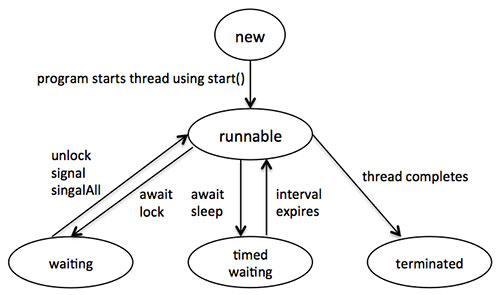
3) Threads are independent so it doesn't affect other threads if exception occur in a single thread.

A thread can only start once, otherwise will throw IllegalThreadStateException

The join() waits for a thread to die and then the later thread completes its task

**Life Cycle of a Thread:**

A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. Following diagram shows complete life cycle of a thread.



Above-mentioned stages are explained here:

* **New:** A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.
* **Runnable:** After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
* **Waiting:** Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task.A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
* **Timed waiting:** A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
* **Terminated:** A runnable thread enters the terminated state when it completes its task or otherwise terminates.

**ThreadPool**

Java Thread pool represents a group of worker threads that are waiting for the job and reuse many times.

In case of thread pool, a group of fixed size threads are created. A thread from the thread pool is pulled out and assigned a job by the service provider. After completion of the job, thread is contained in the thread pool again.

Advantage: Better performance – saves time because there is no need to create new thread and the threads are managed by the pool.

--------------------------------------------------------------------------------

java.util.concurrent.ArrayBlockingQueue;

java.util.concurrent.ThreadPoolExecutor;

java.util.concurrent.TimeUnit;

ThreadPoolExecutor executor = new ThreadPoolExecutor(MAX\_NUM\_OF\_THREADS, POOL\_SIZE, KEEP\_ALIVE\_TIME, TimeUnit.SECONDS, new ArrayBlockingQueue<Runnable>(noOfBatches));

executor.execute(Runnable);

executor.shutdown();

executor.awaitTermination(MAX\_TIMEOUT\_SECONDS, TimeUnit.SECONDS);

// Blocks until all tasks have completed execution after a shutdown request, or the timeout occurs, or the current thread is interrupted, whichever happens first.

**Deadlock**

Deadlock is a condition in which two threads wait for each other to take action which allows them to move further.

Easiest way to avoid deadlock is to prevent Circular wait, and this can be done by acquiring locks in a particular order and releasing them in reverse order, so that a thread can only proceed to acquire a lock if it held the other one

**Granular lock**

The granularity is a measure of the amount of data the lock is protecting. In general, choosing a coarse granularity (a small number of locks, each protecting a large segment of data) results in less **lock overhead** when a single process is accessing the protected data, but worse performance when multiple processes are running concurrently

**Synchronization**

Capability to control the access of multiple threads to any shared resource

Used to prevent thread interference and prevent consistency problem

Synchronized method – used to lock an object for any shared resource and release when the thread completes its task

Synchronized block – used to perform sync. on any specific resource of the mehod, scope of the sync. block is smaller than the method

Static synchronization – the lock will be on the class not on object

Synchronize(this) vs synchronize(class)

this - is the reference to particular this instance of class, and

MyClass.class - is the reference to MyClass description object.

This synchronization blocks differs in a way, that first will synchronize all threads, which deal concretely with this instance of MyClass, and second one will synchronize all threads independently on which object this method was called.

# SQL

<http://www.dwbiconcepts.com/tutorial/24-interview-questions/190-top-20-sql-interview-questions-with-answers.html>

What is inner join and outer join?

<http://www.datamartist.com/sql-inner-join-left-outer-join-full-outer-join-examples-with-syntax-for-sql-server>

Isolation level

<http://en.wikipedia.org/wiki/Isolation_(database_systems)>

<http://www.oracle.com/technetwork/issue-archive/2005/05-nov/o65asktom-082389.html>

A lower isolation level increases the ability of many users to access data at the same time, but increases the number of concurrency effects (such as dirty reads or lost updates) users might encounter. Conversely, a higher isolation level reduces the types of concurrency effects that users may encounter, but requires more system resources and increases the chances that one transaction will block another.

In n-tier systems (such as multiple websites attempting to book the last seat on a flight) a combination of stored procedures and transaction management is required to commit the booking and confirm to the customer.

**Phantom reads**

A *phantom read* occurs when, in the course of a transaction, two identical queries are executed, and the collection of rows returned by the second query is different from the first. (This can occur when [*range locks*](http://en.wikipedia.org/w/index.php?title=Range_locks&action=edit&redlink=1) are not acquired)

**Dirty reads**

A *dirty read* (aka *uncommitted dependency*) occurs when a transaction is allowed to read data from a row that has been modified by another running transaction and not yet committed.

|  |  |  |  |
| --- | --- | --- | --- |
| **Isolation level** | **Dirty reads** | **Non-repeatable reads** | **Phantoms** |
| Read Uncommitted | may occur | may occur | may occur |
| Read Committed | - | may occur | may occur |
| Repeatable Read | - | - | may occur |
| Serializable | - | - | - |

# Data Structures and Algorithms

**ConcurrentHashMap** – thread-safe, accessed by single thread at a time. Does not allow null values. Read is very fast while write is done with a lock. Uses multitude of locks. Does not require locking at object level. Use it when need very high concurrency. Using Fail-safe iterator, which copies the map to a temp space and make iteration. Thus, it will not throw ConcurrentModicationException, but the data will not reflect any update during the iteration.

Higher scalability than SynchonrizedHashMap

**HashMap** – not thread-safe, but can be thread-safe by using synchronizedMap(HashMap), which is equivalent to HashTable. Allow only 1 null key. No ordering on keys or values. Using fail-fast iterator, thus it will throw ConcurrentModicationException if there are more than 1 thread doing update on the map.

**SynchronizedHashMap** – synchronization at object level. Every read/write needs to acquire lock which is a performance overhead. Gives access to only one thread to the entire map and blocks all the other threads.

**Hashtable** is synchronized, but with overhead. Do not allow null keys and null values

TreeMap

<http://javahungry.blogspot.com/2014/06/how-treemap-works-ten-treemap-java-interview-questions.html>

Treemap class is like HashMap which stores key-value pairs. The major difference is that Treemap sorts the key in ascending order (natural ordering or by Comparator). Using Red-Black tree algorithm. Runtime = O(log n).

SortedMap is the Interface, while TreeMap is the class implementing SortedMap

TreeMap.clone() is a shallow copy. Both object A and B shares the same elements, but the keys and values are not cloned.

HashMap vs TreeMap

HashMap is faster while TreeMap is sorted.

HashCode() vs Equals()

two object, which are not equal by equals() method can still return same hashCode

HashMap principle

HashMap works on principle of hashing, we have put(key, value) and get(key) method for storing and retrieving Objects from HashMap. When we pass Key and Value object to put() method on Java HashMap, HashMap implementation calls hashCode method on Key object and applies returned hashcode into its own hashing function to find a bucket location for storing Entry object, important point to mention is that HashMap in Java stores both key and value object as Map.Entry in bucket which is essential to understand the retrieving logic. They will be stored in same bucket but no next node of linked list. And keys equals () method will be used to identify correct key value pair in HashMap.

Red-black Tree

Red–black trees offer worst-case guarantees for insertion time, deletion time, and search time. Not only does this make them valuable in time-sensitive applications such as [real-time applications](http://en.wikipedia.org/wiki/Real-time_computing), but it makes them valuable building blocks in other data structures which provide worst-case guarantees

Binary Tree

A tree data structure in which each note has at most two children

**ArrayList vs Vector**

Vector is synchronized while ArrayList is not synchronized. Synchronization and thread safe means at a time only one thread can access the code .In Vector class all the methods are synchronized. Thats why the Vector object is already synchronized when it is created.

Vector is slow as it is thread safe. In comparison ArrayList is fast as it is non-synchronized. Thus in ArrayList two or more threads can access the code at the same time, while Vector is limited to one thread at a time.

Other than Hashtable, Vector is the only other class which uses both Enumeration and Iterator .While ArrayList can only use Iterator for traversing an ArrayList .

# Design Patterns

Give an example where you prefer abstract class over interface?

This is common but yet tricky design interview question. both interface and abstract class follow "writing code for interface than implementation" design principle which adds flexibility in code, quite important to tackle with changing requirement. here are some pointers which help you to answer this question:

1. In Java you can only extend one class but implement multiple interface. So if you extend a class you lost your chance of extending another class.

2. Interface are used to represent adjective or behavior e.g. Runnable, Clonable, Serializable etc, so if you use an abstract class to represent behavior your class can not be Runnable and Clonable at same time because you can not extend two class in Java but if you use interface your class can have multiple behavior at same time.

**3. On time critical application prefer abstract class is slightly faster than interface.**

4. If there is a genuine common behavior across the inheritance hierarchy which can be coded better at one place than abstract class is preferred choice. Some time interface and abstract class can work together also where defining function in interface and default functionality on abstract class.

1. What is design patterns? Have you used any design pattern in your code ?

Design patterns are tried and tested way to solve particular design issues by various programmers in the world. Design patterns are extension of code reuse.

2. Can you name few design patterns used in standard JDK library?

**Decorator** design pattern which is used in various Java IO classes;

**Singleton** pattern which is used in Runtime , Calendar and various other classes;

**Factory** pattern which is used along with various Immutable classes likes Boolean e.g. Boolean.valueOf

**Observer** pattern which is used in Swing and many event listener frameworks.

3. What is Singleton design pattern in Java? write code for thread-safe singleton in Java

Singleton pattern focus on sharing of expensive object in whole system. Only one instance of a particular class is maintained in whole application which is shared by all modules. Java.lang.Runtime is a classical example of Singleton design pattern. You can also see my post 10 questions on Singleton pattern in Java for more questions and discussion. From Java 5 onwards you can use enum to thread-safe singleton.

4. What is main benefit of using factory pattern? Where do you use it?

Factory pattern’s main benefit is increased level of encapsulation while creating objects. If you use Factory to create object you can later replace original implementation of Products or classes with more advanced and high performance implementation without any change on client layer. See my post on Factory pattern for more detailed explanation and benefits.

5. What is observer design pattern in Java

Observer design pattern is based on communicating changes in state of object to observers so that they can take there action. Simple example is a weather system where change in weather must be reflected in Views to show to public. Here weather object is Subject while different views are Observers. Look on this article for complete example of Observer pattern in Java.

6. Give example of decorator design pattern in Java? Does it operate on object level or class level?

Decorator pattern enhances capability of individual object. Java IO uses decorator pattern extensively and classical example is Buffered classes like BufferedReader and BufferedWriter which enhances Reader and Writer objects to perform Buffer level reading and writing for improved performance.

**Abstract Factory Pattern**

"Provide an interface for creating families of related or dependent objects without specifying their concrete classes."

The abstract factory pattern provides a way to encapsulate a group of individual factories that have a common theme without specifying their concrete classes. In normal usage, the client software creates a concrete implementation of the abstract factory and then uses the generic interface of the factory to create the concrete objects that are part of the theme. The client doesn't know (or care) which concrete objects it gets from each of these internal factories, since it uses only the generic interfaces of their products. This pattern separates the details of implementation of a set of objects from their general usage and relies on object composition, as object creation is implemented in methods exposed in the factory interface.

The client has no need to specify the type, since it has already been specified in the configuration file

**Singleton**

a [design pattern](http://en.wikipedia.org/wiki/Design_pattern_(computer_science)) that restricts the [instantiation](http://en.wikipedia.org/wiki/Instantiation_(computer_science)) of a class to one [object](http://en.wikipedia.org/wiki/Object-oriented_programming). This is useful when exactly one object is needed to coordinate actions across the system

* Singletons are often preferred to global variables because:
  + They do not pollute the global namespace (or, in languages with namespaces, their containing namespace) with unnecessary variables.[[4]](http://en.wikipedia.org/wiki/Singleton_pattern#cite_note-4)
  + They permit [lazy](http://en.wikipedia.org/wiki/Lazy_evaluation) allocation and initialization, whereas global variables in many languages will always consume resources.

**Initialization-on-demand holder**

It takes advantage of language guarantees about class initialization, and will therefore work correctly in all Java-compliant compilers and virtual machines. The nested class is referenced no earlier (and therefore loaded no earlier by the class loader) than the moment that getInstance() is called. Thus, this solution is [thread-safe](http://en.wikipedia.org/wiki/Thread_safety) without requiring special language constructs

**double-checked locking** -

which should not be used prior to [J2SE 5.0](http://en.wikipedia.org/wiki/Java_Platform,_Standard_Edition), as it is vulnerable to subtle bugs. The problem is that an out-of-order write may allow the instance reference to be returned before the Singleton constructor is executed

**Enum**

The best way to implement a singleton, for any Java that supports enums. No drawbacks regarding serializable objects. The drawback is that the enum type is somewhat inflexible.

|  |  |  |
| --- | --- | --- |
| [Creational patterns](http://en.wikipedia.org/wiki/Creational_pattern) | | |
| **Name** | **Description** |
| [Abstract factory](http://en.wikipedia.org/wiki/Abstract_factory_pattern) | Provide an interface for creating *families* of related or dependent objects without specifying their concrete classes. |
| [Builder](http://en.wikipedia.org/wiki/Builder_pattern) | Separate the construction of a complex object from its representation, allowing the same construction process to create various representations. |
| [Factory method](http://en.wikipedia.org/wiki/Factory_method_pattern) | Define an interface for creating a *single* object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses ([dependency injection](http://en.wikipedia.org/wiki/Dependency_injection)[[15]](http://en.wikipedia.org/wiki/Software_design_pattern#cite_note-15)). |
| [Lazy initialization](http://en.wikipedia.org/wiki/Lazy_initialization) | Tactic of delaying the creation of an object, the calculation of a value, or some other expensive process until the first time it is needed. This pattern appears in the GoF catalog as "virtual proxy", an implementation strategy for the [Proxy](http://en.wikipedia.org/wiki/Proxy_pattern) pattern. |
| [Multiton](http://en.wikipedia.org/wiki/Multiton_pattern) | Ensure a class has only named instances, and provide global point of access to them. |
| [Object pool](http://en.wikipedia.org/wiki/Object_pool_pattern) | Avoid expensive acquisition and release of resources by recycling objects that are no longer in use. Can be considered a generalisation of [connection pool](http://en.wikipedia.org/wiki/Connection_pool) and [thread pool](http://en.wikipedia.org/wiki/Thread_pool) patterns. |
| [Prototype](http://en.wikipedia.org/wiki/Prototype_pattern) | Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype. |
| [Resource acquisition is initialization](http://en.wikipedia.org/wiki/Resource_Acquisition_Is_Initialization) | Ensure that resources are properly released by tying them to the lifespan of suitable objects. |
| [Singleton](http://en.wikipedia.org/wiki/Singleton_pattern) | Ensure a class has only one instance, and provide a global point of access to it. |
|  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| [Structural patterns](http://en.wikipedia.org/wiki/Structural_pattern) | | | |
| **Name** | **Description** | |
| [Adapter](http://en.wikipedia.org/wiki/Adapter_pattern) or Wrapper or Translator. | Convert the interface of a class into another interface clients expect. An adapter lets classes work together that could not otherwise because of incompatible interfaces. The enterprise integration pattern equivalent is the translator. | |
| [Bridge](http://en.wikipedia.org/wiki/Bridge_pattern) | Decouple an abstraction from its implementation allowing the two to vary independently. | |
| [Composite](http://en.wikipedia.org/wiki/Composite_pattern) | Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly. | |
| [Decorator](http://en.wikipedia.org/wiki/Decorator_pattern) | Attach additional responsibilities to an object dynamically keeping the same interface. Decorators provide a flexible alternative to subclassing for extending functionality. | |
| [Facade](http://en.wikipedia.org/wiki/Facade_pattern) | Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use. | |
| [Flyweight](http://en.wikipedia.org/wiki/Flyweight_pattern) | Use sharing to support large numbers of similar objects efficiently. | |
| [Front Controller](http://en.wikipedia.org/wiki/Front_Controller_pattern) | The pattern relates to the design of Web applications. It provides a centralized entry point for handling requests. | |
| [Module](http://en.wikipedia.org/wiki/Module_pattern) | Group several related elements, such as classes, singletons, methods, globally used, into a single conceptual entity. | |
| [Proxy](http://en.wikipedia.org/wiki/Proxy_pattern) | Provide a surrogate or placeholder for another object to control access to it. | |
| Twin [[17]](http://en.wikipedia.org/wiki/Software_design_pattern#cite_note-17) | Twin allows modeling of multiple inheritance in programming languages that do not support this feature. | |
|  |  | |
|  |  | |
| [Behavioral patterns](http://en.wikipedia.org/wiki/Behavioral_pattern) | | | |
| **Name** | | **Description** |
| Blackboard | | Generalized observer, which allows multiple readers and writers. Communicates information system-wide. |
| [Chain of responsibility](http://en.wikipedia.org/wiki/Chain_of_responsibility_pattern) | | Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it. |
| [Command](http://en.wikipedia.org/wiki/Command_pattern) | | Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations. |
| [Interpreter](http://en.wikipedia.org/wiki/Interpreter_pattern) | | Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language. |
| [Iterator](http://en.wikipedia.org/wiki/Iterator_pattern) | | Provide a way to access the elements of an [aggregate](http://en.wikipedia.org/wiki/Aggregate_pattern) object sequentially without exposing its underlying representation. |
| [Mediator](http://en.wikipedia.org/wiki/Mediator_pattern) | | Define an object that encapsulates how a set of objects interact. Mediator promotes [loose coupling](http://en.wikipedia.org/wiki/Loose_coupling) by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently. |
| [Memento](http://en.wikipedia.org/wiki/Memento_pattern) | | Without violating encapsulation, capture and externalize an object's internal state allowing the object to be restored to this state later. |
| [Null object](http://en.wikipedia.org/wiki/Null_Object_pattern) | | Avoid null references by providing a default object. |
| [Observer](http://en.wikipedia.org/wiki/Observer_pattern) or [Publish/subscribe](http://en.wikipedia.org/wiki/Publish/subscribe) | | Define a one-to-many dependency between objects where a state change in one object results in all its dependents being notified and updated automatically. |
| [Servant](http://en.wikipedia.org/wiki/Design_pattern_Servant) | | Define common functionality for a group of classes |
| [Specification](http://en.wikipedia.org/wiki/Specification_pattern) | | Recombinable [business logic](http://en.wikipedia.org/wiki/Business_logic) in a [Boolean](http://en.wikipedia.org/wiki/Boolean_algebra) fashion |
| [State](http://en.wikipedia.org/wiki/State_pattern) | | Allow an object to alter its behavior when its internal state changes. The object will appear to change its class. |
| [Strategy](http://en.wikipedia.org/wiki/Strategy_pattern) | | Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it. |
| [Template method](http://en.wikipedia.org/wiki/Template_method_pattern) | | Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure. |
| [Visitor](http://en.wikipedia.org/wiki/Visitor_pattern) | | Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates. |
|  | |  |

|  |  |  |
| --- | --- | --- |
| [Concurrency patterns](http://en.wikipedia.org/wiki/Concurrency_pattern) | | |
| **Name** | **Description** |
| [Active Object](http://en.wikipedia.org/wiki/Active_object) | Decouples method execution from method invocation that reside in their own thread of control. The goal is to introduce concurrency, by using [asynchronous method invocation](http://en.wikipedia.org/wiki/Asynchronous_method_invocation) and a [scheduler](http://en.wikipedia.org/wiki/Scheduling_(computing)) for handling requests. |
| [Balking](http://en.wikipedia.org/wiki/Balking_pattern) | Only execute an action on an object when the object is in a particular state. |
| Binding properties | Combining multiple observers to force properties in different objects to be synchronized or coordinated in some way.[[19]](http://en.wikipedia.org/wiki/Software_design_pattern#cite_note-19) |
| [Double-checked locking](http://en.wikipedia.org/wiki/Double_checked_locking_pattern) | Reduce the overhead of acquiring a lock by first testing the locking criterion (the 'lock hint') in an unsafe manner; only if that succeeds does the actual locking logic proceed.  Can be unsafe when implemented in some language/hardware combinations. It can therefore sometimes be considered an [anti-pattern](http://en.wikipedia.org/wiki/Anti-pattern). |
| [Event-based asynchronous](http://en.wikipedia.org/wiki/Event-Based_Asynchronous_Pattern) | Addresses problems with the asynchronous pattern that occur in multithreaded programs.[[20]](http://en.wikipedia.org/wiki/Software_design_pattern#cite_note-PC.232008-20) |
| [Guarded suspension](http://en.wikipedia.org/wiki/Guarded_suspension) | Manages operations that require both a lock to be acquired and a precondition to be satisfied before the operation can be executed. |
| [Join](http://en.wikipedia.org/wiki/Join-pattern) | Join-pattern provides a way to write concurrent, parallel and distributed programs by message passing. Compared to the use of threads and locks, this is a high level programming model. |
| [Lock](http://en.wikipedia.org/wiki/Lock_(computer_science)) | One thread puts a "lock" on a resource, preventing other threads from accessing or modifying it.[[21]](http://en.wikipedia.org/wiki/Software_design_pattern#cite_note-21) |
| [Messaging design pattern (MDP)](http://en.wikipedia.org/wiki/Messaging_pattern) | Allows the interchange of information (i.e. messages) between components and applications. |
| [Monitor object](http://en.wikipedia.org/wiki/Monitor_(synchronization)) | An object whose methods are subject to [mutual exclusion](http://en.wikipedia.org/wiki/Mutual_exclusion), thus preventing multiple objects from erroneously trying to use it at the same time. |
| [Reactor](http://en.wikipedia.org/wiki/Reactor_pattern) | A reactor object provides an asynchronous interface to resources that must be handled synchronously. |
| [Read-write lock](http://en.wikipedia.org/wiki/Read/write_lock_pattern) | Allows concurrent read access to an object, but requires exclusive access for write operations. |
| [Scheduler](http://en.wikipedia.org/wiki/Scheduler_pattern) | Explicitly control when threads may execute single-threaded code. |
| [Thread pool](http://en.wikipedia.org/wiki/Thread_pool_pattern) | A number of threads are created to perform a number of tasks, which are usually organized in a queue. Typically, there are many more tasks than threads. Can be considered a special case of the [object pool](http://en.wikipedia.org/wiki/Object_pool) pattern. |
| [Thread-specific storage](http://en.wikipedia.org/wiki/Thread-Specific_Storage) | Static or "global" memory local to a thread. |

EES: <http://sp001.jpmchase.net/sites/spsxnqkl/Entitlements/Public%20Entitlement%20Wiki/EES%20FAQ.aspx>

XACML- Extensible Access Control Markup Language

define a core schema and corresponding namespace for the expression of authorization policies in XML against objects that are themselves identified in XML. The basic requirements of a policy language for expressing information system security policy are:

* To provide a method for combining individual rules and policies into a single policy set that applies to a particular decision request
* To provide a method for flexible definition of the procedure by which rules and policies are combined
* To provide a method for dealing with multiple subjects acting in different capacities
* To provide a method for basing an authorization decision on attributes of the subject and resource
* To provide a method for dealing with multi-valued attributes
* To provide a method for basing an authorization decision on the contents of an information resource
* To provide a set of logical and mathematical operators on attributes of the subject, resource and environment
* To provide a method for handling a distributed set of policy components, while abstracting the method for locating, retrieving and authenticating the policy components
* To provide a method for rapidly identifying the policy that applies to a given action, based upon the values of attributes of the subjects, resource and action
* To provide an abstraction-layer that insulates the policy-writer from the details of the application environment
* To provide a method for specifying a set of actions that must be performed in conjunction with policy enforcement

# Miscellaneous

What is Technical Debt – Appear when schedule is very tight and give up quality. Programming practice, technology

Code review – require discussion, different perspective, check correctness / programming practices of the logics

Refactoring

**Performance tuning**

Code optimization

Caching strategy - remove performance bottlenecks that are the result of slow access to data

Load balancing

Distributed computing

Bottlenecks

Disadvantages for adding index

* Increase disk space
* Increase insert time
* Rebuild index

**JDBC api**  
Connection conn = DriverManager.getConnection(DB\_URL, USER, PASS);

PreparedStatement stmt = conn.createStatement();

ResultSet rs1 = stmt.executeQuery(SQL\_1);

ResultSet rs2 = stmt.executeUpdate(SQL\_2);

rs.close();

stmt.close();

conn.close();

Source control – SVN

Deployment tool – Maven

<http://www.tutorialspoint.com/maven/maven_interview_questions.htm>

Maven vs Ant

The fundamental difference between Maven and Ant is that Maven's design regards all projects as having a certain structure and a set of supported task work-flows (e.g., getting resources from source control, compiling the project, unit testing, etc.)

Maven requires that this structure and the operation implementation details be defined in the POM file.

Ant is simply a toolbox whereas Maven is about the application of patterns in order to achieve an infrastructure which displays the characteristics of visibility, reusability, maintainability, and comprehensibility. It is wrong to consider Maven as a build tool and just a replacement for Ant.

POM file – Project Object Model, an XML file contains project dependencies, plugins, goals, build profiles, project versions, developers, mailing list

Maven artifact – usually a JAR that gets deployed to a Maven repository. A Maven build produces one or more artifacts such as a compiled JAR and a sources JAR.

JPM – AIM, FRS

https://confluence.uk.jpmorgan.com/confluence/dosearchsite.action?queryString=frs

Cryptography – Public key, private key, SSL, 2 factor authentication, 3DES, etc

Symmetric key vs Asymmetric key

Web service

* Use xjc to generate schema from XSD
* Use cxf to generate client classes from WSDL

**SOLID**

SOLID (Single responsibility, Open-closed, Liskov substitution, Interface segregation and Dependency inversion) is a mnemonic acronym introduced by Michael Feathers for the "first five principles" named by Robert C. Martin in the early 2000s that stands for five basic principles of object-oriented programming and design. The principles, when applied together, intend to make it more likely that a programmer will create a system that is easy to maintain and extend over time. The principles of SOLID are guidelines that can be applied while working on software to remove code smells by causing the programmer to refactor the software's source code until it is both legible and extensible. It is part of an overall strategy of agile and adaptive programming.

**S**ingle responsibility principle

a class should have only a single responsibility (i.e. only one potential change in the software's specification should be able to affect the specification of the class)

**O**pen/closed principle

“software entities … should be open for extension, but closed for modification.”

**L**iskov substitution principle

“objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.” See also design by contract.

**I**nterface segregation principle

“many client-specific interfaces are better than one general-purpose interface.”

**D**ependency inversion principle

one should “Depend upon Abstractions. Do not depend upon concretions.”

**Dependency Injection**

Dependency injection is a software design pattern that implements inversion of control (**IoC**) for software libraries. Caller delegates to an external framework the control flow of discovering and importing a service or software module specified or "injected" by the caller. Dependency injection allows a program design to follow the dependency inversion principle where modules are loosely coupled. With dependency injection, the client which uses a module or service doesn't need to know all its details, and typically the module can be replaced by another one of similar characteristics without altering the client.

An injection is the passing of a dependency (a service) to a dependent object (a client). The service is made part of the client's state. Passing the service to the client, rather than allowing a client to build or find the service, is the fundamental requirement of the pattern.

There are three common forms of dependency injection: setter-, interface- and constructor-based injection, where the responsibility of injecting the dependency lies upon the client, the service or the constructor method respectively.

Advantages

* Because dependency injection doesn't require any change in code behavior it can be applied to legacy code as a refactoring. The result is clients that are more independent and that are easier to unit test in isolation using stubs or mock objects that simulate other objects not under test. This ease of testing is often the first benefit noticed when using dependency injection.
* Dependency injection allows a client to remove all knowledge of a concrete implementation that it needs to use. This helps isolate the client from the impact of design changes and defects. It promotes reusability, testability and maintainability.
* Dependency injection can be used to externalize a system's configuration details into configuration files allowing the system to be reconfigured without recompilation. Separate configurations can be written for different situations that require different implementations of components. This includes, but is not limited to, testing.
* Dependency injection allows concurrent or independent development. Two developers can independently develop classes that use each other, while only needing to know the interface the classes will communicate through. Plugins are often developed by third party shops that never even talk to the developers who created the product that uses the plugins.
* Dependency Injection decreases coupling between a class and its dependency.

Disadvantages

* Dependency injection can make code difficult to trace (read) because it separates behavior from construction. This means developers must refer to more files to follow how a system performs.[citation needed]
* Dependency injection typically requires more lines of code to accomplish the same behavior legacy code would.

**REST**

Respresentational State Transfer (REST) is a software architecture style consisting of guidelines and best practices for creating scalable web services. REST is a coordinated set of constraints applied to the design of components in a distributed hypermedia system that can lead to a more performant and maintainable architecture.

REST has gained widespread acceptance across the Web as a simpler alternative to SOAP and WSDL-based Web services. RESTful systems typically, but not always, communicate over the Hypertext Transfer Protocol with the same HTTP verbs (GET, POST, PUT, DELETE, etc.) used by web browsers to retrieve web pages and send data to remote servers.

**Spring**

What is IOC or inversion of control?

Answer: As the name implies Inversion of control means now we have inverted the control of creating the object from our own using new operator to container or framework. Now it’s the responsibility of container to create object as required. We maintain one xml file where we configure our components, services, all the classes and their property. We just need to mention which service is needed by which component and container will create the object for us. This concept is known as dependency injection because all object dependency (resources) is injected into it by framework.

Bean Life cycle

Ans: Spring framework is based on IOC so we call it as IOC container also So Spring beans reside inside the IOC container. Spring beans are nothing but Plain old java object (POJO).

Following steps explain their life cycle inside container.

1. Container will look the bean definition inside configuration file (e.g. bean.xml).

2 using reflection container will create the object and if any property is defined inside the bean definition then it will also be set.

3. If the bean implements the BeanNameAware interface, the factory calls setBeanName() passing the bean’s ID.

4. If the bean implements the BeanFactoryAware interface, the factory calls setBeanFactory(), passing an instance of itself.

5. If there are any BeanPostProcessors associated with the bean, their post- ProcessBeforeInitialization() methods will be called before the properties for the Bean are set.

6. If an init() method is specified for the bean, it will be called.

7. If the Bean class implements the DisposableBean interface, then the method destroy() will be called when the Application no longer needs the bean reference.

8. If the Bean definition in the Configuration file contains a 'destroy-method' attribute, then the corresponding method definition in the Bean class will be called.

**Hibernate**

Difference between get and load in Hibernate?

get vs load is one of the most frequently asked Hibernate Interview question, since correct understanding of both get() and load() is require to effectively using Hibernate. Main difference between get and load is that, get will hit the database if object is not found in the cache and returned completely initialized object, which may involve several database call while load() method can return proxy, if object is not found in cache and only hit database if any method other than getId() is called. This can save lot of performance in some cases.

What is SessionFactory in Hibernate? is SessionFactory thread-safe?

Another common Interview questions related to Hibernate framework. SessionFactory as name suggest is a factory to create hibernate Session objects. SessionFactory is often built during start-up and used by application code to get session object. It acts as single data store and its also thread-safe so that multiple thread can use same SessionFactory. Usually a Java JEE application has just one SessionFactory, and individual threads, which are servicing client’s request obtain hibernate Session instances from this factory, that’s why any implementation of SessionFactory interface must be thread-safe. Also internal state of SessionFactory, which contains all meta data about Object/Relational mapping is Immutable and can not be changed once created.

What is Session in Hibernate? Can we share single Session among multiple threads in Hibernate?

This is usually asked as follow-up question of previous Hibernate Interview question. After SessionFactory its time for Session. Session represent a small unit of work in Hibernate, they maintain connection with database and they are not thread-safe, it means you can not share Hibernate Session between multiple threads. Though Session obtains database connection lazily it's good to close session as soon as you are done with it.

**Scrum**

It defines "a flexible, [holistic](http://en.wikipedia.org/wiki/Holism) product development strategy where a development team works as a unit to reach a common goal", challenges assumptions of the "traditional, sequential approach" to product development, and enables teams to self-organize by encouraging physical co-location or close online collaboration of all team members, as well as daily face-to-face communication among all team members and disciplines in the project.

Roles:

Product Owner – stakeholder, voice of the customer, ensure the team deliver values to customers, write customer-centric items (user stories), ranks and prioritize them and add them to product backlog

Scrum Master – facilitate the scrum, remove impediments to deliver the product goals, ensure Scrum processes is used as intended, enforces the rules of Scrum, act as a buffer between the team and any distracting influences

Development Team – for delivering potentially shippable increments (PSIs) at the end of each Sprint

executors   
reenterant lock

spring bean scope

我被人問過一條，我覺得幾好

question: a program using HashMap, they found that the delay spike when putting the value at around market close, the pattern is similar for everyday

why and solution

reason: observed that the HashMap spike when large amount of data stored, the spike should be due to HashMap rehash

solution: use TreeMap rather than HashMap, as TreeMap don't need rehash when full, but it try to keep the tree balance in every insert/delete

so, average time spent on insert is more, but more predictable, while HashMap insert most of the time faster, but may spike due to large amount of data

 reentrant lock 仲易

reentrant means same thread can obtain the same lock repeatly

POSIX lock is different, when same thread obtained a POSIX lock, if it try to obtain the same lock again, it blocks

lock v.s. synchronized 囉