**Core java interview questions series : Part 1**

How to create a immutable object in Java? Count all benefits?

Is Java Pass by Reference or Pass by Value?

What is the use of the finally block? Is finally block in Java guaranteed to be called?

When finally block is NOT called?

Why there are two Date classes; one in java.util package and another in java.sql?

What is Marker interface?

Why main() in java is declared as public static void main?

What is the difference between creating String as new() and literal?

How does substring() inside String works?

Explain the working of HashMap.

When do you override hashCode and equals() ?

### ****How to create a immutable object in Java? Count all benefits?****

An immutable class is one whose state cannot be changed once created. Here, state of object essentially means the values stored in instance variable in class whether they are primitive types or reference types.

To make a class immutable, below steps needs to be followed:

1. Don’t provide “setter” methods or methods that modify fields or objects referred to by fields. Setter methods are meant to change the state of object and this is what we want to prevent here.
2. Make all fields final and private. Fields declared private will not be accessible outside the class and making them final will ensure the even accidentally you can not change them.
3. Don’t allow subclasses to override methods. The simplest way to do this is to declare the class as final. Final classes in java can not be overridden.
4. Always remember that your instance variables will be either mutable or immutable. Identify them and return new objects with copied content for all mutable objects (object references). Immutable variables (primitive types) can be returned safely without extra effort.

Also, you should memorize following benefits of immutable class. You might need them during interview. Immutable classes --

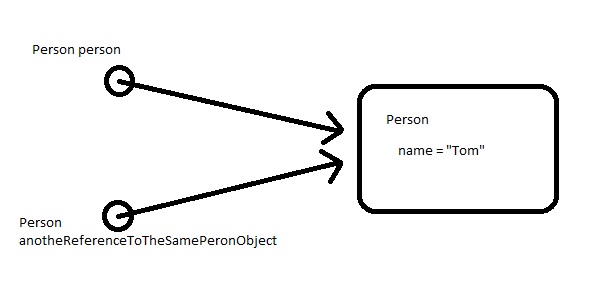
* are simple to construct, test, and use
* are automatically thread-safe and have no synchronization issues
* do not need a copy constructor
* do not need an implementation of clone
* allow hashCode to use lazy initialization, and to cache its return value
* do not need to be copied defensively when used as a field
* make good Map keys and Set elements (these objects must not change state while in the collection)
* have their class invariant established once upon construction, and it never needs to be checked again
* always have “failure atomicity” (a term used by Joshua Bloch) : if an immutable object throws an exception, it’s never left in an undesirable or indeterminate state.

Take a look an example written in [**this post**](http://howtodoinjava.com/2012/10/28/how-to-make-a-java-class-immutable/).

### ****Is Java Pass by Reference or Pass by Value?****

The Java Spec says that ***everything in Java is pass-by-value***. There is no such thing as “pass-by-reference” in Java. These terms are associated with method calling and passing variables as method parameters. Well, primitive types are always pass by value without any confusion. But, the concept should be understood in context of method parameter of complex types.

In java, when we pass a reference of complex types as any method parameters, always the memory address is copied to new reference variable bit by bit. See in below picture:

[](http://howtodoinjava.files.wordpress.com/2013/03/pass-by-value.jpg)In above example, address bits of first instance are copied to another reference variable, thus resulting both references to point a single memory location where actual object is stored. Remember, making another reference to null will not make first reference also null. But, changing state from either reference variable have impact seen in other reference also.

Read in detail here: [**http://stackoverflow.com/questions/40480/is-java-pass-by-reference**](http://stackoverflow.com/questions/40480/is-java-pass-by-reference)

### ****What is the use of the finally block? Is finally block in Java guaranteed to be called? When finally block is NOT called?****

The finally block always executes when the try block exits. This ensures that the finally block is executed even if an unexpected exception occurs. But finally is useful for more than just exception handling — it allows having cleanup code accidentally bypassed by a return, continue, or break. Putting cleanup code in a finally block is always a good practice, even when no exceptions are anticipated.

If the JVM exits while the try or catch code is being executed, then the finally block may not execute. Likewise, if the thread executing the try or catch code is interrupted or killed, the finally block may not execute even though the application as a whole continues.

### ****Why there are two Date classes; one in java.util package and another in java.sql?****

A java.util.Date represents date and time of day, a java.sql.Date only represents a date. The complement of java.sql.Date is java.sql.Time, which only represents a time of day.  
The java.sql.Date is a subclass (an extension) of java.util.Date. So, what changed in java.sql.Date:

-- toString() generates a different string representation: yyyy-mm-dd  
-- a static valueOf(String) methods to create a Date from a String with above representation  
-- the getters and setter for hours, minutes and seconds are deprecated

The java.sql.Date class is used with JDBC and it was intended to not have a time part, that is, hours, minutes, seconds, and milliseconds should be zero… but this is not enforced by the class.

### ****Explain marker interfaces?****

The marker interface pattern is a design pattern in computer science, used with languages that **provide run-time type information about objects**. It provides **a means to associate metadata with a class where the language does not have explicit support for such metadata.** In java, it is used as interfaces with no method specified.

A good example of use of marker interface in java is [Serializable](http://howtodoinjava.com/2012/11/21/a-mini-guide-for-implementing-serializable-interface-in-java/) interface. A class implements this interface to indicate that its non-transient data members can be written to a byte steam or file system.

A major problem with marker interfaces is that an interface defines a contract for implementing classes, and that contract is inherited by all subclasses. This means that you cannot “un-implement” a marker. In the example given, if you create a subclass that you do not want to serialize (perhaps because it depends on transient state), you must resort to explicitly throwing NotSerializableException.

### ****Why main() in java is declared as public static void?****

Why public? main method is public so that it can be accessible everywhere and to every object which may desire to use it for launching the application. Here, i am not saying that JDK/JRE had similar reasons because java.exe or javaw.exe (for windows) use Java Native Interface (JNI) calls to invoke method, so they can have invoked it either way irrespective of any access modifier.

Why static? Lets suppose we do not have main method as static. Now, to invoke any method you need an instance of it. Right? Java can have overloaded constructors, we all know. Now, which one should be used and from where the parameters for overloaded constructors will come.

Why void? Then there is no use of returning any value to JVM, who actually invokes this method. The only thing application would like to communicate to invoking process is: normal or abnormal termination. This is already possible using  System.exit(int). A non-zero value means abnormal termination otherwise everything was fine.

### ****What is the difference between creating String as new() and literal?****

When we create string with new() it’s created in heap and also added into string pool, while String created using literal are created in String pool only which exists in Perm area of heap.

Well you really need to know the concept of string pool very deeply to answer this question or similar questions. My advise.. “Study Hard” about [string class and string pool](http://howtodoinjava.com/2012/10/28/interview-stuff-about-string-class-in-java/).

### ****How does substring () inside String works?****

String in java are like any other programming language, a sequence of characters. This is more like a utility class to work on that char sequence. This char sequence is maintained in following variable:

/\*\* The value is used for character storage. \*/  
**private final char value[];**

To access this array in different scenarios, following variables are used:

/\*\* The offset is the first index of the storage that is used. \*/  
**private final int offset;**

/\*\* The count is the number of characters in the String. \*/  
**private final int count;**

Whenever we create a substring from any existing string instance, substring() method only set’s the new values of offset and count variables. The internal char array is unchanged. This is a possible source of memory leak if substring() method is used without care. [Read more here](http://howtodoinjava.com/2012/10/28/interview-stuff-about-string-class-in-java/)

### ****Explain the working of HashMap. How duplicate collision is resolved?****

Most of you will agree that HashMap is most favorite topic for discussion in interviews now-a-days. If anybody asks me to describe “How HashMap works?”, I simply answer: “On principles of Hashing”. As simple as it is.

Now, Hashing in its simplest form, is a way to assigning a unique code for any variable/object after applying any formula/ algorithm on its properties.

A map by definition is : “An object that maps keys to values”. Very easy.. right? So, HashMap has an inner class Entry, which looks like this:

static class Entry implements Map.Entry  
{  
final K key;  
V value;  
Entry next;  
final int hash;  
…//More code goes here  
}

When, someone tries to store a key value pair in a HashMap, following things happen:

-- First of all, key object is checked for null. If key is null, value is stored in table[0] position. Because hash code for null is always 0.

-- Then on next step, a hash value is calculated using key’s hash code by calling its hashCode() method. This hash value is used to calculate index in array for storing Entry object.  JDK designers well assumed that there might be some poorly written hashCode() functions that can return very high or low hash code value. To solve this issue, they introduced another hash() function, and passed the object’s hash code to this hash() function to bring hash value in range of array index size.

-- Now indexFor(hash, table.length) function is called to calculate exact index position for storing the Entry object.

-- Here comes the main part. Now, as we know that two unequal objects can have same hash code value, how two different objects will be stored in same array location [called bucket].  
Answer is LinkedList. If you remember, Entry class had an attribute “next”. This attribute always points to next object in chain. This is exactly the behavior of LinkedList.

So, in case of collision, Entry objects are stored in LinkedList form. When an Entry object needs to be stored in particular index, HashMap checks whether there is already an entry?? If there is no entry already present, Entry object is stored in this location.

If there is already an object sitting on calculated index, its next attribute is checked. If it is null, and current Entry object becomes next node in LinkedList. If next variable is not null, procedure is followed until next is evaluated as null.

What if we add the another value object with same key as entered before. Logically, it should replace the old value.  How it is done? Well, after determining the index position of Entry object, while iterating over LinkedList on calculated index, HashMap calls equals method on key object for each Entry object. All these Entry objects in LinkedList will have similar hash code but equals() method will test for true equality. If key.equals(k) will be true then both keys are treated as same key object. This will cause the replacing of value object inside Entry object only.

In this way, HashMap ensure the uniqueness of keys.

### ****Difference between interfaces and abstract classes?****

This is very common question if you are appearing interview for junior level programmer. Well, most noticeable differences are as below:

* Variables declared in a Java interface is by default final. An  abstract class may contain non-final variables.
* Java interface are implicitly abstract and cannot have implementations. A Java abstract class can have instance methods that implements a default behavior.
* Members of a Java interface are public by default. A Java abstract class can have the usual flavors of class members like private, protected.
* Java interface should be implemented using keyword “implements”; A Java abstract class should be extended using keyword “extends”.
* A Java class can implement multiple interfaces but it can extend only one abstract class.
* Interface is absolutely abstract and cannot be instantiated; A Java abstract class also cannot be instantiated, but can be invoked if a main() exists.
* Abstract class are slightly faster than interface because interface involves a search before calling any overridden method in Java. This is not a significant difference in most of cases but if you are writing a time critical application than you may not want to leave any stone unturned.

### ****When do you override hashCode() and equals()?****

hashCode() and equals() methods have been defined in Object class which is parent class for java objects. For this reason, all java objects inherit a default implementation of these methods.

hashCode() method is used to get a unique integer for given object. This integer is used for determining the bucket location, when this object needs to be stored in some HashTable like data structure. By default, Object’s hashCode() method returns and integer representation of memory address where object is stored.  
equals() method, as name suggest, is used to simply verify the equality of two objects.  Default implementation simply check the object references of two objects to verify their equality.

Note that it is generally necessary to override the hashCode method whenever this method is overridden, so as to maintain the general contract for the hashCode method, which states that equal objects must have equal hash codes.

equals()  must define an equality relation (it must be reflexive, symmetric, and transitive). In addition, it must be consistent (if the objects are not modified, then it must keep returning the same value). Furthermore, o.equals(null) must always return false.

hashCode()  must also be consistent (if the object is not modified in terms of equals(), it must keep returning the same value).

The relation between the two methods is:

Whenever a.equals(b), then a.hashCode() must be same as b.hashCode().

Why finalize() method should be avoided?

Why HashMap should not be used in multithreaded environment? Can it cause infinite loop as well?

Explain abstraction and encapsulation? How are they related?

Difference between interfaces and abstract classes?

How StringBuffer save the memory?

Why wait and notify is declared in Object class instead of Thread ?

Write Java program to create deadlock in Java and fix it ?

What happens if your Serializable class contains a member which is not  serializable? How do you fix it?

Explain transient and volatile keywords in java?

Difference between Iterator and ListIterator?

### ****Why finalize() method should be avoided?****

We all know the basic statement that finalize() method is called by garbage collector thread before reclaiming the memory allocated to the object. See [this program](http://howtodoinjava.com/2012/10/31/why-not-to-use-finalize-method-in-java/) which prove that finalize() invocation is not guaranteed at all. Other reasons can be:

1) finalize() methods do not work in chaining like constructors. It means like when you call a constructor then constructors of all super classes will be invokes implicitly. But, in case of finalize methods, this is not followed. Super class’s finalize() should be called explicitly.

2) Any Exception thrown by finalize method is ignored by GC thread and it will not be propagated further, in fact it will not be logged in your log files. So bad, isn’t it?

3) Also, There is some performance penalty when finalize() in included in your class. In Effective java (2nd edition ) Joshua bloch says,

“Oh, and one more thing: there is a severe performance penalty for using finalizers. On my machine, the time to create and destroy a simple object is about 5.6 ns.  
Adding a finalizer increases the time to 2,400 ns. In other words, it is about 430 times slower to create and destroy objects with finalizers.”

### ****Why HashMap should not be used in multithreaded environment? Can it cause infinite loop as well?****

We know that HashMap is non-synchronized collection where as its synchronized counter-part is HashTable. So, when you are accessing the collection in multithreaded environment and all threads are accessing a single instance of collection, then its safer to use HashTable for various obvious reasons e.g. to avoid dirty reads and to maintain data consistency. In worst case, this mutithreaded environment can result in infinite loop as well.

Yes, it is true. HashMap.get() can cause an infinite loop. Lets see how??

If you look at the source code HashMap.get(Object key) method, it looks like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | public Object get(Object key) {      Object k = maskNull(key);      int hash = hash(k);      int i = indexFor(hash, table.length);      Entry e = table[i];      while (true) {          if (e == null)              return e;          if (e.hash == hash &amp;&amp; eq(k, e.key))              return e.value;          e = e.next;      }  } |

while(true) can always be a victim of infinite loop at runtime in multithreaded environment, IF, somehow e.next can point to itself. This will result in infinite loop. But, how e.next will point to itself (i.e. e).

This can happen in void transfer(Entry[] newTable) method, which is invoked at time the HashMap resizing is done.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | do {      Entry next = e.next;      int i = indexFor(e.hash, newCapacity);      e.next = newTable[i];      newTable[i] = e;      e = next;  } while (e != null); |

This piece of code is prone to produce above condition, if resizing happen and at the same time other threads tried to modify the map instance.

Only way to avoid this scenario is to use synchronization in code, or better, used synchronized collection.

### ****Explain abstraction and encapsulation? How are they related?****

In simple words: “***Abstraction***captures only those details about an object that are relevant to the current  perspective.”  
In object-oriented programming theory, abstraction involves the facility to define objects that represent abstract “actors” that can perform work, report on and change their state, and “communicate” with other objects in the system.

Abstraction in any programming language works in many ways. It can be seen from creating subroutines to defining interfaces for making low level language calls. Some abstractions try to limit the breadth of concepts a programmer needs, by completely hiding the abstractions they in turn are built on, e.g. design patterns.

Typically abstraction can be seen in two ways:

**Data abstraction** is the way to create complex data types and exposing only meaningful operations to interact with data type, where as hiding all the implementation details from outside works. **Control abstraction** is the process of identifying all such statements and expose them as a unit of work. We normally use this feature when we create a function to perform any work.

**Wrapping data and methods within classes in combination with implementation hiding (through access control) is often called encapsulation.** The result is a data type with characteristics and behaviors. Encapsulation essentially has both i.e. information hiding and implementation hiding.

“***Whatever changes, encapsulate it***“. It has been quoted as a famous design principle. For that matter in any class, changes can happen in data in runtime and changes in implementation can happen in future releases. So, encapsulation applies to both i.e. data as well as implementation.

SO, they can relate like following :

-- Abstraction is more about ‘What‘ a class can do. [Idea]  
-- Encapsulation is more about ‘How‘ to achieve that functionality. [Implementation]

### ****Difference between interfaces and abstract classes?****

Basic differences can be counted as follows:

* An interface cannot implement any methods, whereas an abstract class can
* A class can implement many interfaces but can have only one superclass (abstract or not)
* An interface is not part of the class hierarchy. Unrelated classes can implement the same interface

You should remember that : “When you can fully describe the concept in terms of “**what it does**” without needing to specify any of “**how it does**“, then you should use an interface.  If you need to include some implementation details, then you will need to represent your concept in an abstract class.”

Also, if i talk differently : Are there many classes that can be “grouped together” and described by one noun? If so, have an abstract class by the name of this noun, and inherit the classes from it. For example Cat and Dog can both inherit from abstract class Animal, and this abstract base class will implement a method void Breathe() which all animals will thus do in exactly the same fashion.

What kinds of verbs can be applied to my class, that might in general also be applied to others? Create an interface for each of these verbs. For example, All animals can be fed, so I will create an interface called IFeedable and have Animal implement that. Only Dog and Horse are nice enough though to implement ILikeable, but some are not.

As said by someone: the main difference is where you want your implementation. By creating an interface, you can move your implementation to any class that implements your interface. By creating an abstract class, you can share implementation for all derived classes in one central place, and avoid lots of bad things like code duplication.

### ****How StringBuffer save the memory?****

A String is implemented as an immutable object; that is, when you initially decide to put something into a String object, the JVM allocates a fixed-width array of exactly the size of your initial value. This is then treated as a constant inside the JVM, which allows for very significant performance savings in the case where the String’s value is not changed. However, if you decide to change the String’s contents in any way, what the JVM then essentially does is copy the contents of the original String into a temporary space, make your changes, then save those changes into a whole new memory array. Thus, making changes to a String’s value after initialization is a fairly expensive operation.

StringBuffer, on the other hand, is implemented as a dynamically -- growable array inside the JVM, which means that any change operation can occur on the existing memory location, with new memory allocated only as-needed. However, there is no opportunity for the JVM to make optimizations around the StringBuffer, since its contents are assumed to be changeable at any instance.

### ****Why wait and notify is declared in Object class instead of Thread?****

Wait , notify , notifyAll methods are only required when you want your threads to access a shared resource and a shared resource could be any java object which is on the heap. So, these methods are defined on the core Object class so that each object has control of allowing Threads to wait on it’s monitor. Java doesn’t have any special object which is used for sharing a common resource. No such data structure is defined.So, onus is given on the Object class to be able to become shared resource providing it will helper methods like wait(),notify() and notifyAll().

Java is based on Hoare’s monitors idea. In Java all object has a monitor. Threads waits on monitors so, to perform a wait, we need 2 parameters:

-- a Thread  
-- a monitor (any object)

In the Java design, the thread can not be specified, it is always the current thread running the code. However, we can specify the monitor (which is the object we call wait on). This is a good design, because if we could make any other thread to wait on a desired monitor, this would lead to an “intrusion”, posing difficulties on designing /programming concurrent programs. Remember that in Java all operations that are intrusive in another thread’s execution are deprecated (e.g. stop()).

### ****Write Java program to create deadlock in Java and fix it ?****

In java, a deadlock is a situation where minimum two threads are holding lock on some different resource, and both are waiting for other resource to complete its task. And, none is able to leave the lock on resource it is holding.

To create a deadlock situation, and to know the solution : read full post “[Writing a deadlock and resolving in java](http://howtodoinjava.com/2012/10/16/writing-a-deadlock-and-resolving-in-java/)“.

### ****What happens if your Serializable class contains a member which is not serializable? How do you fix it?****

In this case, **NotSerializableException** will be thrown at runtime. To fix this issue, a very simple solution is to mark such fields transient. It means these fields will not be serialized. If you want to save the state of these fields as well then you should consider reference variables which already implements serializable interface.

You also might need to use readResolve() and writeResolve() methods. Lets summarize this:

* First, make your non-serialisable field transient.
* In writeObject, first call defaultWriteObject on the stream to store all the non-transient fields, then call other methods to serialise the individual properties of your non-serialisable object.
* In readObject, first call defaultReadObject on the stream to read back all the non-transient fields, then call other methods (corresponding to the ones you added to writeObject) to deserialise your non-serialisable object.

Also, i will highly recommend to read [**full guide on serialization in java**](http://howtodoinjava.com/2012/11/21/a-mini-guide-for-implementing-serializable-interface-in-java/).

### ****Explain transient and volatile keywords in java?****

“The***transient***keyword in Java is used to indicate that a field should not be serialized.” According to language specification: Variables may be marked transient to indicate that they are not part of the persistent state of an object. For example, you may have fields that are derived from other fields, and should only be done so programmatically, rather than having the state be persisted via serialization.

For example, in class BankPayment.java fields like principal and rate can be serialized while interest can be calculated any time even after de-serialization.

If we recall, each thread in java has its own local memory space as well and it does all read/write operations in its local memory. Once all operations are done, it write back the modified state of variable in main memory from where all threads access this variable. Normally, this is the default flow inside JVM. But, the volatile modifier tells the JVM that a thread accessing the variable must always reconcile its own private copy of the variable with the master copy in memory. It means every time thread want to read the state of variable, it must flush its local memory state and update the variable from main memory.

**Volatile** is most useful in lock-free algorithms. You mark the variable holding shared data as volatile when you are not using locking to access that variable and you want changes made by one thread to be visible in another, or you want to create a “happens-after” relation to ensure that computation is not re-ordered, again, to ensure changes become visible at the appropriate time.

The volatile should be used to safely publish immutable objects in a multi-threaded Environment. Declaring a field like public volatile ImmutableObject foo secures that all threads always see the currently available instance reference.

### ****Difference between Iterator and ListIterator?****

We can use Iterator to traverse a Set or a List or a Map. But ListIterator can only be used to traverse a List only. Other differences can be listed as below.

You can

1. iterate backwards.
2. obtain the index at any point.
3. add a new value at any point.
4. set a new value at that point.

Deep copy and shallow copy?

What is synchronization? Class level locking and object level locking?

Difference between sleep() and wait()?

Can you assign null to this reference variable?

What if the difference between && and &??

How to override equals and hashCode() methods?

Explain all access modifiers?

What is garbage collection? Can we enforce it?

What is native keyword?

What is serialization? Explain the catches?

### ****Deep copy and shallow copy?****

A clone is an exact copy of the original. In java, it essentially means the ability to create an object with similar state as the original object. The clone() method provides this functionality.

Shallow copies duplicate as little as possible.  By default, java cloning is shallow copy or ‘field by field copy’ i.e. as the Object class does not have idea about the structure of class on which clone() method will be invoked. So, JVM when called for cloning, do following things:

1) If the class has only primitive data type members then a completely new copy of the object will be created and the reference to the new object copy will be returned.

2) If the class contains members of any class type then only the object references to those members are copied and hence the member references in both the original object as well as the cloned object refer to the same object.

Deep copies duplicate everything. A deep copy of a collection is two collections with all of the elements in the original collection duplicated. Here, we want a clone which is independent of original and making changes in clone should not affect original.

Deep cloning requires satisfaction of following rules.

1. No need to separately copy primitives.
2. All the member classes in original class should support cloning and in clone method of original class in context should call super.clone() on all member classes.
3. If any member class does not support cloning then in clone method, one must create a new instance of that member class and copy all its attributes one by one to new member class object. This new member class object will be set in cloned object.

[Read more about cloning here](http://howtodoinjava.com/2012/11/08/a-guide-to-object-cloning-in-java/).

### ****What is synchronization? Object level locking and class level locking?****

***Synchronization***refers to multi-threading. A synchronized block of code can only be executed by one thread at a time. Java supports multiple threads to be executed. This may cause two or more threads to access the same fields or objects. Synchronization is a process which keeps all concurrent threads in execution to be in synch. Synchronization avoids memory consistence errors caused due to inconsistent view of shared memory. When a method is declared as synchronized; the thread holds the monitor for that method’s object If another thread is executing the synchronized method, your thread is blocked until that thread releases the monitor.

Synchronization in java is achieved using synchronized keyword. You can use synchronized keyword in your class on defined methods or blocks. Keyword can not be used with variables or attributes in class definition.

***Object level locking*** is mechanism when you want to synchronize a non-static method or non-static code block such that only one thread will be able to execute the code block on given instance of the class. This should always be done to make instance level data thread safe.

***Class level locking*** prevents multiple threads to enter in synchronized block in any of all available instances on runtime. This means if in runtime there are 100 instances of  DemoClass, then only one thread will be able to execute demoMethod() in any one of instance at a time, and all other instances will be locked for other threads. This should always be done to make static data thread safe.

[Read more about synchronization here.](http://howtodoinjava.com/2013/03/08/thread-synchronization-object-level-locking-and-class-level-locking/)

### ****Difference between sleep() and wait()?****

sleep() is a method which is used to hold the process for few seconds or the time you wanted but in case of wait() method thread goes in waiting state and it won’t come back automatically until we call the notify() or notifyAll().

The major difference is that wait() releases the lock or monitor while sleep() doesn’t releases any lock or monitor while waiting. Wait is used for inter-thread communication while sleep is used to introduce pause on execution, generally.

Thread.sleep() sends the current thread into the “Not Runnable” state for some amount of time. The thread keeps the monitors it has aquired — i.e. if the thread is currently in a synchronized block or method no other thread can enter this block or method. If another thread calls t.interrupt() it will wake up the sleeping thread. Note that sleep is a static method, which means that it always affects the current thread (the one that is executing the sleep method). A common mistake is to call t.sleep() where t is a different thread; even then, it is the current thread that will sleep, not the t thread.

object.wait() sends the current thread into the “Not Runnable” state, like sleep(), but with a twist. Wait is called on a object, not a thread; we call this object the “lock object.” Before lock.wait() is called, the current thread must synchronize on the lock object; wait() then releases this lock, and adds the thread to the “wait list” associated with the lock. Later, another thread can synchronize on the same lock object and call lock.notify(). This wakes up the original, waiting thread. Basically, wait()/notify() is like sleep()/interrupt(), only the active thread does not need a direct pointer to the sleeping thread, but only to the shared lock object.

[Read the difference in detail here.](http://howtodoinjava.com/2013/03/08/difference-between-sleep-and-wait/)

### ****Can you assign null to this reference variable?****

NO. You can’t. In java, left hand side of an assignment statement must be a variable. ‘this’ is a special keyword which represent the current instance always. This is not any variable.

Similarly, null can not be assigned to ‘super’ or any such keyword for that matter.

### ****What if the difference between && and &??****

& is bitwise and && is logical.

* & evaluates both sides of the operation.
* && evaluates the left side of the operation, if it’s true, it continues and evaluates the right side.

[Read here for deep understanding.](http://en.wikipedia.org/wiki/Bitwise_operation)

### ****How to override equals and hashCode() methods?****

hashCode() and equals() methods have been defined in Object class which is parent class for java objects. For this reason, all java objects inherit a default implementation of these methods.

hashCode() method is used to get a unique integer for given object. This integer is used for determining the bucket location, when this object needs to be stored in some HashTable like data structure. By default, Object’s hashCode() method returns and integer representation of memory address where object is stored.

equals() method, as name suggest, is used to simply verify the equality of two objects.  Default implementation simply check the object references of two objects to verify their equality.

Below are the important points to keep remember while overriding these functions.

1. Always use same attributes of an object to generate hashCode() and equals() both. As in our case, we have used employee id.
2. equals() must be consistent (if the objects are not modified, then it must keep returning the same value).
3. Whenever a.equals(b), then a.hashCode() must be same as b.hashCode().
4. If you override one, then you should override the other.

[Read more interesting facts and how to guide here.](http://howtodoinjava.com/2012/10/09/working-with-hashcode-and-equals-methods-in-java/)

### ****Explain all access modifiers?****

Java classes, fields, constructors and methods can have one of four different access modifiers:

***private***

If a method or variable is marked as private, then only code inside the same class can access the variable, or call the method. Code inside subclasses cannot access the variable or method, nor can code from any external class.  
If a class is marked as private then no external class an access the class. This doesn’t really make so much sense for classes though. Therefore, the access modifier private is mostly used for fields, constructors and methods.

***default***

The default access level is declared by not writing any access modifier at all. Default access levels means that code inside the class itself + code inside classes in the same package as this class, can access the class, field, constructor or method. Therefore, the default access modifier is also sometimes called a package access modifier.

Subclasses cannot access methods and member variables in the superclass, if they have default accessibility declared, unless the subclass is located in the same package as the superclass.

***protected***

The protected acces modifier does the same as the default access, except subclasses can also access protected methods and member variables of the superclass. This is true even if the subclass is not located in the same package as the superclass.

***public***

The public access modifier means that all code can access the class, field, constructor or method, regardless of where the accessing code is located.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Modifiers** | Same Class | Same Package | Subclass | Other packages |
| public | Y | Y | Y | Y |
| protected | Y | Y | Y | N |
| default | Y | Y | N | N |
| private | Y | N | N | N |

### ****What is garbage collection? Can we enforce it?****

Garbage collection is an automatic memory management feature in many modern programming languages, such as Java and languages in the .NET framework. Languages that use garbage collection are often interpreted or run within a virtual machine like the JVM. In each case, the environment that runs the code is also responsible for garbage collection. A GC has two goals: any unused memory should be freed, and no memory should be freed unless the program will not use it anymore.

Can you force garbage collection?? Nope, System.gc() is as close as you can get. Your best option is to call System.gc() which simply is a hint to the garbage collector that you want it to do a collection. There is no way to force and immediate collection though as the garbage collector is non-deterministic. Also, under the documentation for OutOfMemoryError it declares that it will not be thrown unless the VM has failed to reclaim memory following a full garbage collection. So if you keep allocating memory until you get the error, you will have already forced a full garbage collection.

[Read more about garbage collection here.](http://howtodoinjava.com/2012/11/11/revisiting-memory-management-and-garbage-collection-mechanisms-in-java/)

### ****What is native keyword? Explain in detail?****

The native keyword is applied to a method to indicate that the method is implemented in native code using JNI. It marks a method, that it will be implemented in other languages, not in Java.

Native methods were used in the past to write performance critical sections but with Java getting faster this is now less common. Native methods are currently needed when

* You need to call a library from Java that is written in other language.
* You need to access system or hardware resources that are only reachable from the other language (typically C). Actually, many system functions that interact with real computer (disk and network IO, for instance) can only do this because they call native code.

The downsides of using native code libraries are also significant:

1. JNI / JNA have a tendency to destabilize the JVM, especially if you try to do something complicated. If your native code gets native code memory management wrong, there’s a chance that you will crash the JVM. If your native code is non-reentrant and gets called from more than one Java thread, bad things will happen … sporadically. And so on.
2. Java with native code is harder to debug than pure Java or pure C/C++.
3. Native code can introduce significant platform dependencies / issues for an otherwise platform independent Java app.
4. Native code requires a separate build framework, and that may have platform / portability issues as well.

### ****What is serialization? Explain the catches?****

In computer science, in the context of data storage and transmission, serialization is the process of translating data structures or object state into a format that can be stored  and “resurrected” later in the same or another computer environment.  When the resulting series of bits is reread according to the serialization format, it can be used to create a semantically identical clone of the original object.

Java provides automatic serialization which requires that the object be marked by implementing the java.io.Serializable interface. Implementing the interface marks the class as “okay to serialize,” and Java then handles serialization internally. There are no serialization methods defined on the Serializable interface, but a serializable class can optionally define methods with certain special names and signatures that if defined, will be called as part of the serialization/deserialization process.

Once an object is serialized, changes in its class break the de-serialization process. To identify the future changes in your class which will be compatible and others which will prove incompatible, please read the[**full guide here**](http://howtodoinjava.com/2012/11/21/a-mini-guide-for-implementing-serializable-interface-in-java/). In short, I am listing down here:

**Incompatible changes**

* Deleting fields
* Moving classes up or down the hierarchy
* Changing a non-static field to static or a non-transient field to transient
* Changing the declared type of a primitive field
* Changing the writeObject or readObject method so that it no longer writes or reads the default field data
* Changing a class from Serializable to Externalizable or vice-versa
* Changing a class from a non-enum type to an enum type or vice versa
* Removing either Serializable or Externalizable
* Adding the writeReplace or readResolve method to a class

**Compatible changes**

* Adding fields
* Adding/ Removing classes
* Adding writeObject/readObject methods [defaultReadObject or defaultWriteObject should be called first]
* Removing writeObject/readObject methods
* Adding java.io.Serializable
* Changing the access to a field
* Changing a field from static to non-static or transient to non transient

# Useful java collection interview questions

General questions

[1) What is the Java Collections API? List down its advantages?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#what_is_collection_in_java)

[2) Explain Collections hierarchy?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#collections_hierarchy)

[3) Why Collection interface does not extend Cloneable and Serializable interface?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#why_collection_not_extend_cloneable_serializable)

[4) Why Map interface does not extend Collection interface?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#why_map_not_extend_collection)

List interface related

[5) Why we use List interface? What are main classes implementing List interface?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#what_is_list_in_java)

[6) How to convert an array of String to ArrayList?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#convert_array_of_String_to_ArrayList)

[7) How to reverse the list?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#reverse_list)

Set interface related

[8) Why we use Set interface? What are main classes implementing Set interface?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#what_is_set_in_java)

[9) How HashSet store elements?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#how_set_store_elements)

[10) Can a null element added to a TreeSet or HashSet?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#null_in_set)

Map interface related

[11) Why we use Map interface? What are main classes implementing Map interface?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#what_is_map_in_java)

[12) What are IdentityHashMap and WeakHashMap?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#identityHashMap_weakHashMap_differences)

[13) Explain ConcurrentHashMap? How it works?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#how_concurrentHashMap_works)

[14) How hashmap works?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#how_hashmap_works)

[15) How to design a good key for hashmap?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#good_key_for_hashmap)

[16) What are different Collection views provided by Map interface?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#different_collection_views)

[17) When to use HashMap or TreeMap?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#hashmap_or_treemap)

Tell the difference questions

[18) Difference between Set and List?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#difference_set_and_list)

[19) Difference between List and Map?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#difference_map_and_list)

[20) Difference between HashMap and HashTable?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#difference_hashmap_and_hashtable)

[21) Difference between Vector and ArrayList?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#difference_vector_and_arraylist)

[22) Difference between Iterator and Enumeration?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#difference_iterator_and_enumerator)

[23) Difference between HashMap and HashSet?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#difference_hashmap_and_hashset)

[24) Difference between Iterator and ListIterator?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#difference_iterator_and_listiterator)

[25) Difference between TreeSet and SortedSet?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#difference_treeset_and_sortedset)

[26) Difference between ArrayList and LinkedList?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#difference_arraylist_and_linkedlist)

More questions

[27) How to make a collection read only?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#read_only_collection)

[28) How to make a collection thread safe?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#thread_safe_collection)

[29) Why there is not method like Iterator.add() to add elements to the collection?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#why_no_iterator_add)

[30) What are different ways to iterate over a list?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#different_ways_to_iterate_list)

[31) What do you understand by iterator fail-fast property?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#fail_fast_iterator)

[32) What is difference between fail-fast and fail-safe?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#fail_safe_vs_fail_fast)

[33) How to avoid ConcurrentModificationException while iterating a collection?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#avoid_ConcurrentModificationException)

[34) What is UnsupportedOperationException?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#UnsupportedOperationException)

[35) Which collection classes provide random access of it’s elements?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#random_access_collections)

[36) What is BlockingQueue?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#BlockingQueue)

[37) What is Queue and Stack, list their differences?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#queue_and_stack_difference)

[38) What is Comparable and Comparator interface?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#Comparable_and_Comparator)

[39) What are Collections and Arrays class?](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#Collections_and_arrays)

[40) Recommended resources](http://howtodoinjava.com/2013/07/09/useful-java-collection-interview-questions/#resources)

Without wasting time, let dig into concepts.

### ****General question****

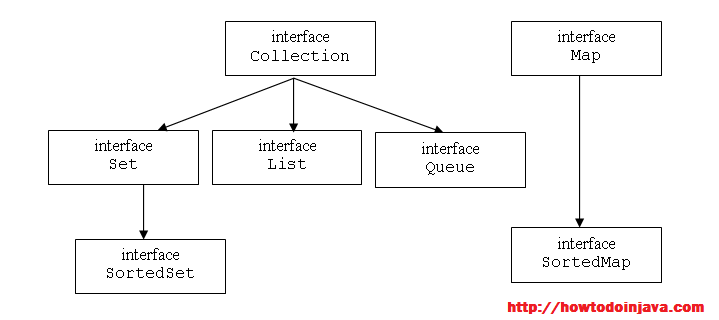
##### **1) What is the Java Collection framework? List down its advantages?**

By definition, a collection is **an object that represents a group of objects**. Like in set theory, a set is group of elements. Easy enough !!  
Prior to JDK 1.2, JDK has some utility classes such as Vector and HashTable, but there was no concept of Collection framework. Later from JDK 1.2 onwards, JDK felt the need of having a consistent support for reusable data structures. Finally, the collections framework was designed and developed primarily by Joshua Bloch, and was**introduced in JDK 1.2**.

Its most **noticeable advantages** can be listed as:

* Reduced programming effort due to ready to use code
* Increased performance because of high-performance implementations of data structures and algorithms
* Provides interoperability between unrelated APIs by establishing a common language to pass collections back and forth
* Easy to learn APIs by learning only some top level interfaces and supported operations

##### **2) Explain Collection’s hierarchy?**



*Java Collection Hierarchy*

As shown in above image, collection framework has one interface at top i.e. **Collection**. It is **extended by Set, List and Queue interfaces**. Then there are loads of other classes in these 3 branches which we will learn in following questions.

Remember the signature of Collection interface. It will help you in many question.

|  |  |
| --- | --- |
| 1  2  3 | public interface Collection extends Iterable {  //method definitions  } |

Framework also consist of Map interface, which is part of collection framework. but it does not extend Collection interface. We will see the reason in 4th question in this question bank.

##### **3) Why Collection interface does not extend Cloneable and Serializable interface?**

Well, simplest answer is “**there is no need to do it**“. Extending an interface simply means that you are creating a subtype of interface, in other words a more specialized behavior and Collection interface is not expected to do what Cloneable and Serializable interfaces do.

Another reason is that not everybody will have a reason to have Cloneable collection because if it has very large data, then every **unnecessary clone operation will consume a big memory**. Beginners might use it without knowing the consequences.

Another reason is that **Cloneable and Serializable are very specialized behavior** and so should be implemented only when required. For example, many concrete classes in collection implement these interfaces. So if you want this feature. use these collection classes otherwise use their alternative classes.

##### **4) Why Map interface does not extend Collection interface?**

A good answer to this interview question is “**because they are incompatible**“. Collection has a method add(Object o). Map can not have such method because it need key-value pair. There are other reasons also such as Map supports keySet, valueSet etc. Collection classes does not have such views.

Due to such big differences, Collection interface was not used in Map interface, and it was build in separate hierarchy.

### ****List interface related****

##### **5) Why we use List interface? What are main classes implementing List interface?**

A java list is a **“ordered” collection of elements**. This ordering is a **zero based index**. It does not care about duplicates. Apart from methods defined in Collection interface, it does **have its own methods** also which are largely to manipulate the collection **based on index location of element**. These methods can be grouped as search, get, iteration and range view. All above operations support index locations.

The main classes implementing List interface are: **Stack, Vector, ArrayList and LinkedList**. Read more about them in java documentation.

##### **6) How to convert an array of String to arraylist?**

This is more of a programmatic question which is seen at beginner level. The intent is to check the knowledge of applicant in Collection utility classes. For now, lets learn that there are two utility classes in Collection framework which are mostly seen in interviews i.e. **Collections and Arrays**.

Collections class provides some static functions to perform specific operations on collection types. And Arrays provide utility functions to be performed on array types.

|  |  |
| --- | --- |
| 1  2  3  4  5 | //String array  String[] words = {&quot;ace&quot;, &quot;boom&quot;, &quot;crew&quot;, &quot;dog&quot;, &quot;eon&quot;};  //Use Arrays utility class  List wordList = Arrays.asList(words);  //Now you can iterate over the list |

Please not that this function is not specific to String class, it will return List of element of any type, of which the array is. e.g.

|  |  |
| --- | --- |
| 1  2  3  4 | //String array  Integer[] nums = {1,2,3,4};  //Use Arrays utility class  List numsList = Arrays.asList(nums); |

##### **7) How to reverse the list?**

This question is just like above to test your knowledge of **Collections** utility class. Use it **reverse()** method to reverse the list.

|  |  |
| --- | --- |
| 1 | Collections.reverse(list); |

### ****Set interface related****

##### **8) Why we use Set interface? What are main classes implementing Set interface?**

It **models the mathematical set in set theory**. Set interface is like List interface but with some differences. First, it is**not ordered collection**. So no ordering is preserved while adding or removing elements. The main feature it does provide is “**uniqueness of elements**“. It does not support duplicate elements.

Set also adds a stronger contract on the behavior of the equals and hashCode operations, allowing Set instances to be compared meaningfully even if their implementation types differ. Two Set instances are equal if they contain the same elements.

Based on above reasons, it **does not have operations based on indexes of elements like List**. It only has methods which are inherited by Collection interface.

Main classes implementing Set interface are :**EnumSet, HashSet, LinkedHashSet, TreeSet**. Read more on related java documentation.

##### **9) How HashSet store elements?**

You must know that HashMap store key-value pairs, with one condition i.e. keys will be unique. HashSet uses Map’s this feature to ensure uniqueness of elements. In HashSet class, a map declaration is as below:

|  |  |
| --- | --- |
| 1  2  3  4 | private transient HashMap<E,Object> map;    //This is added as value for each key  private static final Object PRESENT = new Object(); |

So **when you store a element in HashSet, it stores the element as key in map and “PRESENT” object as value**. (See declaration above).

|  |  |
| --- | --- |
| 1  2  3 | public boolean add(E e) {  return map.put(e, PRESENT)==null;  } |

I will highly suggest you to read this post: [**How HashMap works in java?**](http://howtodoinjava.com/2012/10/09/how-hashmap-works-in-java/) This post will help you in answering all the HashMap related questions very easily.

##### **10) Can a null element added to a TreeSet or HashSet?**

As you see, There is no null check in add() method in previous question. And HashMap also allows one null key, so**one “null” is allowed in HashSet**.

TreeSet uses the same concept as HashSet for internal logic, but uses NavigableMap for storing the elements.

|  |  |
| --- | --- |
| 1  2  3  4 | private transient NavigableMap<E,Object> m;    // Dummy value to associate with an Object in the backing Map  private static final Object PRESENT = new Object(); |

NavigableMap is subtype of SortedMap which does not allow null keys. So essentially,**TreeSet also does not support null keys**. It will throw NullPointerException if you try to add null element in TreeSet.

### ****Map interface related****

##### **11) Why we use Map interface? What are main classes implementing Map interface?**

Map interface is a special type of collection which is **used to store key-value pairs**. It does not extend Collection interface for this reason. This interface provides methods to add, remove, search or iterate over various views of Map.

Main classes implementing Map interface are:**HashMap, Hashtable, EnumMap, IdentityHashMap, LinkedHashMap and Properties.**

##### **12) What are IdentityHashMap and WeakHashMap?**

**IdentityHashMap** is similar to HashMap except that**it uses reference equality when comparing elements**. IdentityHashMap class is not a widely used Map implementation. While this class implements the Map interface, it intentionally violates Map’s general contract, which mandates the use of the equals() method when comparing objects. IdentityHashMap is designed for use only in the rare cases wherein reference-equality semantics are required.

**WeakHashMap** is an implementation of the Map interface **that stores only weak references to its keys**. Storing only weak references allows a key-value pair to be garbage collected when its key is no longer referenced outside of the WeakHashMap. This class is intended primarily for use with key objects whose equals methods test for object identity using the == operator. Once such a key is discarded it can never be recreated, so it is impossible to do a look-up of that key in a WeakHashMap at some later time and be surprised that its entry has been removed.

##### **13) Explain ConcurrentHashMap? How it works?**

Taking from java docs:

**A hash table supporting full concurrency of retrievals and adjustable expected concurrency for updates**. This class obeys the same functional specification as Hashtable, and includes versions of methods corresponding to each method of Hashtable. However, even though all operations are thread-safe, retrieval operations do not entail locking, and there is not any support for locking the entire table in a way that prevents all access. This class is fully interoperable with Hashtable in programs that rely on its thread safety but not on its synchronization details.

Read more about how [**concurrent hashmap works and related interview questions**](http://howtodoinjava.com/2013/06/14/popular-hashmap-and-concurrenthashmap-interview-questions/).

##### **14) How hashmap works?**

The **most important question** which is most likely to be seen in every level of job interviews. You must be very clear on this topic., not only because it is most asked question but also it will open up your mind in further questions related to collection APIs.

Answer to this question is very large and you should read it my post: [**How HashMap works?**](http://howtodoinjava.com/2012/10/09/how-hashmap-works-in-java/) For now, lets remember that HashMap works **on principle of Hashing**. A map by definition is : “An object that maps keys to values”. To store such structure, **it uses an inner class Entry**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | static class Entry implements Map.Entry  {  final K key;  V value;  Entry next;  final int hash;  ...//More code goes here  } |

Here key and value variables are used to store key-value pairs. Whole entry object is stored in an array.

|  |  |
| --- | --- |
| 1  2  3  4 | /\*\*  \* The table, re-sized as necessary. Length MUST Always be a power of two.  \*/  transient Entry[] table; |

The index of array is calculated on basis on hashcode of Key object. Read more of linked topic.

##### **15) How to design a good key for hashmap?**

Another good question usually followed up after answering how hashmap works. Well, the most important constraint is **you must be able to fetch the value object back in future**. Otherwise, there is no use of having such a data structure. If you understand the working of hashmap, you will find it largely depends on hashCode() and equals() method of Key objects.

So a good key object**must provide same hashCode() again and again**, no matter how many times it is fetched. Similarly, same keys**must return true when compare with equals() method and different keys must return false**.

For this reason,**immutable classes are considered best candidate for HashMap keys**.

Read more : [**How to design a good key for HashMap?**](http://howtodoinjava.com/2013/05/02/how-to-design-a-good-key-for-hashmap/)

##### **16) What are different Collection views provided by Map interface?**

Map interface provides 3 views of key-values pairs stored in it:

* key set view
* value set view
* entry set view

All the views can be navigated using iterators.

##### **17) When to use HashMap or TreeMap?**

HashMap is well known class and all of us know that. So, I will leave this part by saying that it is used to store key-value pairs and allows to perform many operations on such collection of pairs.

TreeMap is special form of HashMap. **It maintains the ordering of keys** which is missing in HashMap class. This ordering is **by default “natural ordering”**. The default ordering can be override by providing an instance of Comparator class, whose compare method will be used to maintain ordering of keys.

Please note that **all keys inserted into the map must implement the Comparable interface** (this is necessary to decide the ordering). Furthermore, all such keys must be mutually comparable: k1.compareTo(k2) must not throw a ClassCastException for any keys k1 and k2 in the map. If the user attempts to put a key into the map that violates this constraint (for example, the user attempts to put a string key into a map whose keys are integers), the put(Object key, Object value) call will throw a ClassCastException.

### ****Tell the difference questions****

##### **18) Difference between Set and List?**

The most noticeable differences are :

* Set is unordered collection where List is ordered collection based on zero based index.
* List allow duplicate elements but Set does not allow duplicates.
* List does not prevent inserting null elements (as many you like), but Set will allow only one null element.

##### **19) Difference between List and Map?**

Perhaps most easy question. **List is collection of elements where as map is collection of key-value pairs**. There is actually lots of differences which originate from first statement. They have**separate top level interface, separate set of generic methods, different supported methods and different views of collection**.

I will take much time hear as answer to this question is enough as first difference only.

##### **20) Difference between HashMap and HashTable?**

There are several differences between HashMap and Hashtable in Java:

* Hashtable is synchronized, whereas HashMap is not.
* Hashtable does not allow null keys or values. HashMap allows one null key and any number of null values.
* The third significant difference between HashMap vs Hashtable is that Iterator in the HashMap is a fail-fast iterator while the enumerator for the Hashtable is not.

##### **21) Difference between Vector and ArrayList?**

Lets note down the differences:

* All the methods of Vector is synchronized. But, the methods of ArrayList is not synchronized.
* Vector is a Legacy class added in first release of JDK. ArrayList was part of JDK 1.2, when collection framework was introduced in java.
* By default, Vector doubles the size of its array when it is re-sized internally. But, ArrayList increases by half of its size when it is re-sized.

##### **22) Difference between Iterator and Enumeration?**

Iterators differ from enumerations in three ways:

* Iterators allow the caller to remove elements from the underlying collection during the iteration with its remove() method. You can not add/remove elements from a collection when using enumerator.
* Enumeration is available in legacy classes i.e Vector/Stack etc. whereas Iterator is available in all modern collection classes.
* Another minor difference is that Iterator has improved method names e.g. Enumeration.hasMoreElement() has become Iterator.hasNext(), Enumeration.nextElement() has become Iterator.next() etc.

##### **23) Difference between HashMap and HashSet?**

HashMap is collection of key-value pairs whereas HashSet is un-ordered collection of unique elements. That’s it. No need to describe further.

##### **24) Difference between Iterator and ListIterator?**

There are three Differences are there:

* We can use Iterator to traverse Set and List and also Map type of Objects. But List Iterator can be used to traverse for List type Objects, but not for Set type of Objects.
* By using Iterator we can retrieve the elements from Collection Object in forward direction only whereas List Iterator, which allows you to traverse in either directions using hasPrevious() and previous() methods.
* ListIterator allows you modify the list using add() remove() methods. Using Iterator you can not add, only remove the elements.

##### **25) Difference between TreeSet and SortedSet?**

SortedSet is an interface which TreeSet implements. That’ it !!

##### **26) Difference between ArrayList and LinkedList?**

* LinkedList store elements within a doubly-linked list data structure. ArrayList store elements within a dynamically resizing array.
* LinkedList allows for constant-time insertions or removals, but only sequential access of elements. In other words, you can walk the list forwards or backwards, but grabbing an element in the middle takes time proportional to the size of the list. ArrayLists, on the other hand, allow random access, so you can grab any element in constant time. But adding or removing from anywhere but the end requires shifting all the latter elements over, either to make an opening or fill the gap.
* LinkedList has more memory overhead than ArrayList because in ArrayList each index only holds actual object (data) but in case of LinkedList each node holds both data and address of next and previous node.

### ****More questions****

##### **27) How to make a collection read only?**

Use following methods:

* Collections.unmodifiableList(list);
* Collections.unmodifiableSet(set);
* Collections.unmodifiableMap(map);

These methods takes collection parameter and return a new read-only collection with same elements as in original collection.

##### **28) How to make a collection thread safe?**

Use below methods:

* Collections.synchronizedList(list);
* Collections.synchronizedSet(set);
* Collections.synchronizedMap(map);

Above methods take collection as parameter and return same type of collection which are synchronized and thread safe.

##### **29) Why there is not method like Iterator.add() to add elements to the collection?**

The sole purpose of an Iterator is to enumerate through a collection. All collections contain the add() method to serve your purpose. There would be no point in adding to an Iterator because the **collection may or may not be ordered**. And **add() method can not have same implementation for ordered and unordered collections**.

##### **30) What are different ways to iterate over a list?**

You can iterate over a list using following ways:

* Iterator loop
* For loop
* For loop (Advance)
* While loop

Read more : <http://www.mkyong.com/java/how-do-loop-iterate-a-list-in-java/>

##### **31) What do you understand by iterator fail-fast property?**

**Fail-fast Iterators fail as soon as they realized that structure of Collection has been changed since iteration has begun**. Structural changes means adding, removing or updating any element from collection while one thread is Iterating over that collection.

Fail-fast behavior is implemented by keeping a modification count and if iteration thread realizes the change in modification count it throws ConcurrentModificationException.

##### **32) What is difference between fail-fast and fail-safe?**

You have understood fail-fast in previous question. **Fail-safe iterators** are just opposite to fail-fast. **They never fail if you modify the underlying collection on which they are iterating**, because they work on clone of Collection instead of original collection and that’s why they are called as fail-safe iterator.

Iterator of CopyOnWriteArrayList is an example of fail-safe Iterator also iterator written by ConcurrentHashMap keySet is also fail-safe iterator and never throw ConcurrentModificationException.

##### **33) How to avoid ConcurrentModificationException while iterating a collection?**

You should first try to **find another alternative iterator which are fail-safe**. For example if you are using List and you can use ListIterator. If it is legacy collection, you can use enumeration.

If above options are not possible then you can use one of three changes:

* If you are using JDK1.5 or higher then you can use ConcurrentHashMap and CopyOnWriteArrayList classes. It is the recommended approach.
* You can convert the list to an array and then iterate on the array.
* You can lock the list while iterating by putting it in a synchronized block.

Please note that last two approaches will cause a performance hit.

##### **34) What is UnsupportedOperationException?**

This exception is thrown **on invoked methods which are not supported by actual collection type**.

For example, if you make a read-only list list using “Collections.unmodifiableList(list)” and then call add() or remove() method, what should happen. It should clearly throw UnsupportedOperationException.

##### **35) Which collection classes provide random access of it’s elements?**

ArrayList, HashMap, TreeMap, Hashtable classes provide random access to it’s elements.

##### **36) What is BlockingQueue?**

**A Queue that additionally supports operations that wait for the queue to become non-empty when retrieving an element, and wait for space to become available in the queue when storing an element.**

BlockingQueue methods come in four forms: one throws an exception, the second returns a special value (either null or false, depending on the operation), the third blocks the current thread indefinitely until the operation can succeed, and the fourth blocks for only a given maximum time limit before giving up.

Read the example usage of blocking queue in post : [**How to use blocking queue?**](http://howtodoinjava.com/2012/10/20/how-to-use-blockingqueue-and-threadpoolexecutor-in-java/)

##### **37) What is Queue and Stack, list down their differences?**

**A collection designed for holding elements prior to processing.** Besides basic Collection operations, queues provide additional insertion, extraction, and inspection operations.  
**Queues typically, but do not necessarily, order elements in a FIFO (first-in-first-out) manner.**

**Stack is also a form of Queue but one difference, it is LIFO (last-in-first-out).**

Whatever the ordering used, the head of the queue is that element which would be removed by a call to remove() or poll(). Also note that Stack and Vector are both synchronized.

**Usage:** Use a queue if you want to process a stream of incoming items in the order that they are received.Good for work lists and handling requests.  
Use a stack if you want to push and pop from the top of the stack only. Good for recursive algorithms.

##### **38) What is Comparable and Comparator interface?**

In java. all collection which have feature of automatic sorting, uses compare methods to ensure the correct sorting of elements. For example classes which use sorting are TreeSet, TreeMap etc.

**To sort the data elements a class needs to implement Comparator or Comparable interface**. That’s why all Wrapper classes like Integer,Double and String class implements Comparable interface.

**Comparable helps in preserving default natural sorting, whereas Comparator helps in sorting the elements in some special required sorting pattern.** The instance of comparator if passed usually as collection’s constructor argument in supporting collections.

##### **39) What are Collections and Arrays classes?**

**Collections and Arrays classes are special utility classes to support collection framework core classes.** They provide utility functions to get read-only/ synchronized collections, sort the collection on various ways etc.

Arrays also helps array of objects to convert in collection objects. Arrays also have some functions which helps in copying or working in part of array objects.

##### **40) Recommended resources**

Well it is not interview question.. :-). This is only for fun. But you should really read my blog for more posts on collection framework knowledge.

I hope these java collection interview questions will help in in your next interview. Further, I will suggest you to read more on above questions apart from this post. A more knowledge will only help you.

# Popular HashMap and ConcurrentHashMap interview questions

Topics covered in this post:

1. How you will design a good key for HashMap?
2. Difference between HashMap and ConcurrentHashMap?
3. Difference between HashMap and Collections.synchronizedMap(HashMap)?
4. Difference between ConcurrentHashMap and Collections.synchronizedMap(HashMap)?
5. Difference between HashMap and HashTable?
6. Difference between HashTable and Collections.synchronized(HashMap)?
7. Impact of random/fixed hashCode() value for key?
8. Using HashMap in non-synchronized code in multi-threaded application?

Lets start the discussion without wasting time on unnecessary things.

### ****1) How to design a good key for HashMap****

The very basic need for designing a good key is that “we should be able to retrieve the value object back from the map without failure”, right?? Otherwise no matter how fancy data structure you build, it will be of no use. To decide that we have created a good key, we MUST know that “[**how HashMap works?**](http://howtodoinjava.com/2012/10/09/how-hashmap-works-in-java/)“. I will leave, how hashmap works, part on you to read from linked post, but in summary it works on principle of Hashing.

Key’s hash code is used primarily in conjunction to its equals() method, for putting a key in map and then searching it back from map. So if hash code of key object changes after we have put a key-value pair in map, then its almost impossible to fetch the value object back from map. It is a case of memory leak. To avoid this, map **keys should be immutable**. These are few things to [**create an immutable of class**](http://howtodoinjava.com/2012/10/28/how-to-make-a-java-class-immutable/).

This is the main reason why immutable classes like String, Integer or other wrapper classes are a good key object candidate.

But remember that **immutability is recommended and not mandatory**. If you want to make a mutable object as key in hashmap, then you have to make sure that **state change for key object does not change the hash code of object**. This can be done by overriding the hashCode() method. Also, key class must honor the [**hashCode() and equals() methods contract**](http://howtodoinjava.com/2012/10/09/working-with-hashcode-and-equals-methods-in-java/) to avoid the undesired and surprising behavior on run time. Read more about this contract in linked post.

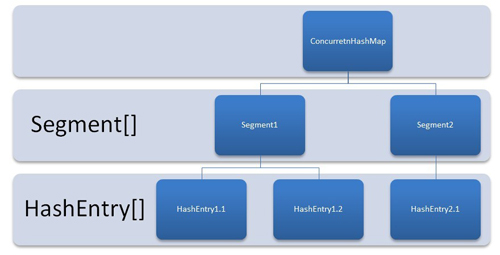
A more detailed information is available in [**here**](http://howtodoinjava.com/2013/05/02/how-to-design-a-good-key-for-hashmap/).

### ****2) Difference between HashMap and ConcurrentHashMap****

To better visualize the ConcurrentHashMap, let it consider as a group of HashMaps. To get and put key-value pairs from hashmap, you have to calculate the hashcode and look for correct bucket location in array of Collection.Entry. Rest you have read on previous related article on how hashmap works.

In concurrentHashMap, the **difference lies in internal structure to store these key-value pairs**. ConcurrentHashMap has an addition concept of segments. It will be easier to understand it you think of one segment equal to one HashMap [conceptually]. A concurrentHashMap is divided into number of segments [default 16] on initialization. ConcurrentHashMap allows similar number (16) of threads to access these segments concurrently so that each thread work on a specific segment during high concurrency.

This way, if when your key-value pair is stored in segment 10; code does not need to block other 15 segments additionally. This structure provides a very high level of concurrency.



*ConcurrentHashMap Internal Structure*

In other words, **ConcurrentHashMap uses a multitude of locks, each lock controls one segment of the map**. When setting data in a particular segment, the lock for that segment is obtained. So essentially **update operations are synchronized**.  
**When getting data, a volatile read is used** without any synchronization. If the volatile read results in a miss, then the lock for that segment is obtained and entry is again searched in synchronized block.

I will go further deeper into this concept in my coming post. I will suggest you to subscribe email updates to get notified.

### ****3) Difference between HashMap and Collections.synchronizedMap(HashMap)****

It’s easy question, right !! HashMap is non-synchronized and Collections.synchronizedMap() returns a wrapped instance of HashMap which has all get, put methods synchronized.

Essentially, **Collections.synchronizedMap() returns the reference of internally created inner-class “SynchronizedMap”**, which contains key-value pairs of input HashMap, passed as argument.

This instance of inner class has nothing to do with original parameter HashMap instance and is completely independent.

### ****4) Difference between ConcurrentHashMap and Collections.synchronizedMap( HashMap )****

This one is slightly tougher. Both are synchronized version of HashMap, with difference in their core functionality and internal structure.

As stated above, ConcurrentHashMap is consist of internal segments which can be viewed as independent HashMaps, conceptually. All such segments can be locked by separate threads in high concurrent executions. In this way, **multiple threads can get/put key-value pairs from ConcurrentHashMap without blocking/waiting for each other**.

In Collections.synchronizedMap(), we get a synchronized version of HashMap and **it is accessed in blocking manner**. This means if multiple threads try to access synchronizedMap at same time, they will be allowed to get/put key-value pairs one at a time in synchronized manner.

### ****5) Difference between HashMap and HashTable****

It is also very easy question. The major difference is that **HashTable is synchronized and HashMap is not**.

If asked for other reasons, tell them, **HashTable is legacy class** (part of JDK 1.0) which was promoted into collections framework by implementing Map interface later. It still has some **extra features like Enumerator** with it, which HashMap lacks.

Another minor reason can be: **HashMap supports null key** (mapped to zero bucket), HashTable does not support null keys and throws NullPointerException on such attempt.

### ****6) Difference between HashTable and Collections.synchronized(HashMap)****

So far you must have got the core idea of the similarities between them. Both are synchronized version of collection. Both have synchronized methods inside class. Both are blocking in nature i.e. multiple threads will need to wait for getting the lock on instance before putting/getting anything out of it.

So what is the difference. Well, **NO major difference** for above said reasons. Performance is also same for both collections.

Only thing which separates them is the fact **HashTable is legacy** class promoted into collection framework. It got its own extra features like enumerators.

### ****7) Impact of random/fixed hashcode() value for key****

The impact of both cases (fixed hashcode or random hashcode for keys) will have same result and that is “**unexpected behavior**“. The very basic need of hashcode in HashMap is to identify the bucket location where to put the key-value pair, and from where it has to be retrieved.

If the hashcode of key object changes every time, the exact location of key-value pair will be calculated different, every time. This way, one object stored in HashMap will be lost forever and there will be very minimum possibility to get it back from map.

For this same reason, key are suggested to be immutable, so that they return a unique and same hashcode each time requested on same key object.

### ****8) Using HashMap in non-synchronized code in multi-threaded application****

In normal cases, it **can leave the hashmap in inconsistent state** where key-value pairs added and retrieved can be different. Apart from this, other surprising behavior like NullPointerException can come into picture.

In worst case,**It can cause infinite loop**. YES. You got it right. It can cause infinite loop. What did you asked, How?? Well, here is the reason.

HashMap has the concept of rehashing when it reaches to its upper limit of size. This rehashing is the process of creating a new memory area, and copying all the already present key-value pairs in new memory are. Lets say Thread A tried to put a key-value pair in map and then rehashing started. At the same time, thread B came and started manipulating the buckets using put operation.

Here while rehashing process, there are chances to generate the cyclic dependency where one element in linked list [in any bucket] can point to any previous node in same bucket. This will result in infinite loop, because rehashing code contains a “while(true) { //get next node; }” block and in cyclic dependency it will run infinite.

To watch closely, look art source code of transfer method which is used in rehashing:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | public Object get(Object key) {      Object k = maskNull(key);      int hash = hash(k);      int i = indexFor(hash, table.length);      Entry e = table[i];        //While true is always a bad practice and cause infinite loops        while (true) {          if (e == null)              return e;          if (e.hash == hash && eq(k, e.key))              return e.value;          e = e.next;      }  } |

I will write a more detailed article on this in future.

# Top Spring Core Interview Questions with Answers

**Table of Contents**

[1) What is Spring Framework? What are it's main modules?](http://howtodoinjava.com/2015/02/26/top-spring-core-interview-questions-with-answers/#spring_framework)

[2) What are the benefits of using Spring Framework?](http://howtodoinjava.com/2015/02/26/top-spring-core-interview-questions-with-answers/#spring_benefits)

[3) What is Inversion of Control (IoC) and Dependency Injection?](http://howtodoinjava.com/2015/02/26/top-spring-core-interview-questions-with-answers/#ioc_di)

[4) Explain IoC in Spring Framework?](http://howtodoinjava.com/2015/02/26/top-spring-core-interview-questions-with-answers/#ioc_in_spring)

[5) Difference between BeanFactory and ApplicationContext?](http://howtodoinjava.com/2015/02/26/top-spring-core-interview-questions-with-answers/#beanfactory_vs_applicationcontext)

[6) In how many ways, you can configure Spring into our application?](http://howtodoinjava.com/2015/02/26/top-spring-core-interview-questions-with-answers/#spring_configuration_types)

[7) What is Spring XML-Based Configuration?](http://howtodoinjava.com/2015/02/26/top-spring-core-interview-questions-with-answers/#xml_based_configuration)

[8) What is Spring Java-Based Configuration?](http://howtodoinjava.com/2015/02/26/top-spring-core-interview-questions-with-answers/#java_based_configuration)

[9) What is Spring Annotation-based Configuration?](http://howtodoinjava.com/2015/02/26/top-spring-core-interview-questions-with-answers/#annotation_based_configuration)

[10) Explain Spring Bean lifecycle?](http://howtodoinjava.com/2015/02/26/top-spring-core-interview-questions-with-answers/#bean_lifecycle)

[11) What are different Spring Bean Scopes?](http://howtodoinjava.com/2015/02/26/top-spring-core-interview-questions-with-answers/#bean_scopes)

[12) What are inner beans in Spring?](http://howtodoinjava.com/2015/02/26/top-spring-core-interview-questions-with-answers/#inner_beans)

[13) Are Singleton beans thread safe in Spring Framework?](http://howtodoinjava.com/2015/02/26/top-spring-core-interview-questions-with-answers/#singleton_bean_threadsafe)

[14) How can you inject a Java Collection in Spring? Give example?](http://howtodoinjava.com/2015/02/26/top-spring-core-interview-questions-with-answers/#inject_collection)

[15) How to inject a java.util.Properties into a Spring Bean?](http://howtodoinjava.com/2015/02/26/top-spring-core-interview-questions-with-answers/#inject_properties)

[16) Explain Spring Bean Autowiring?](http://howtodoinjava.com/2015/02/26/top-spring-core-interview-questions-with-answers/#bean_autowiring)

[17) Explain different modes of bean autowiring?](http://howtodoinjava.com/2015/02/26/top-spring-core-interview-questions-with-answers/#autowiring_modes)

[18) How do you turn on annotation based autowiring?](http://howtodoinjava.com/2015/02/26/top-spring-core-interview-questions-with-answers/#enable_autowiring)

[19) Explain @Required annotation with example?](http://howtodoinjava.com/2015/02/26/top-spring-core-interview-questions-with-answers/#required_annotation)

[20) Explain @Autowired annotation with example?](http://howtodoinjava.com/2015/02/26/top-spring-core-interview-questions-with-answers/#autowired_annotation)

[21) Explain @Qualifier annotation with example?](http://howtodoinjava.com/2015/02/26/top-spring-core-interview-questions-with-answers/#qualifier_annotation)

[22) Difference between constructor injection and setter injection?](http://howtodoinjava.com/2015/02/26/top-spring-core-interview-questions-with-answers/#constructor_vs_setter_injection)

[23) What are the different types of events in spring framework?](http://howtodoinjava.com/2015/02/26/top-spring-core-interview-questions-with-answers/#applicationcontext_events)

[24) Difference between FileSystemResource and ClassPathResource?](http://howtodoinjava.com/2015/02/26/top-spring-core-interview-questions-with-answers/#filesystemresource_vs_classpathresource)

[25) Name some of the design patterns used in Spring Framework?](http://howtodoinjava.com/2015/02/26/top-spring-core-interview-questions-with-answers/#design_patterns_used_in_spring)

## 1) What is Spring Framework? What are it’s main modules?

The Spring Framework is a Java platform that provides comprehensive infrastructure support for developing Java applications. Spring handles the infrastructure part so you can focus on your application part. Inside itself, Spring Framework codifies formalized design patterns as first-class objects that you can integrate into your own application(s) without worrying too much about how they work in backend.

At present, Spring Framework consists of features organized into about 20 modules. These modules are grouped into Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation, Messaging, and Test, as shown in the following diagram.



**Read More :** [Spring Framework Tutorials](http://howtodoinjava.com/java-spring-framework-tutorials/)

## 2) What are the benefits of using Spring Framework?

Following is the list of few of the great benefits of using Spring Framework --

* With the [**Dependency Injection(DI)**](http://howtodoinjava.com/2013/03/19/inversion-of-control-ioc-and-dependency-injection-di-patterns-in-spring-framework-and-related-interview-questions/) approach, dependencies are explicit and evident in constructor or JavaBean properties.
* IoC containers tend to be lightweight, especially when compared to EJB containers, for example. This is beneficial for developing and deploying applications on computers with limited memory and CPU resources.
* Spring does not reinvent the wheel instead, it truly makes use of some of the existing technologies like several ORM frameworks, logging frameworks, JEE, Quartz and JDK timers, other view technologies.
* Spring is organized in a modular fashion. Even though the number of packages and classes are substantial, you have to worry only about ones you need and ignore the rest.
* [**Testing an application**](http://howtodoinjava.com/2013/04/19/how-to-unit-test-spring-security-authentication-with-junit/) written with Spring is simple because environment-dependent code is moved into this framework. Furthermore, by using JavaBean-style POJOs, it becomes easier to use dependency injection for injecting test data.
* Spring’s web framework is a well-designed web MVC framework, which provides a great alternative to web frameworks such as Struts or other over engineered or less popular web frameworks.
* Spring provides a consistent transaction management interface that can scale down to a local transaction (using a single database, for example) and scale up to global transactions (using JTA, for example).

## 3) What is Inversion of Control (IoC) and Dependency Injection?

In software engineering, **inversion of control** (IoC) is a programming technique in which object coupling is bound at run time by an assembler object and is typically not known at compile time using static analysis. In traditional programming, the flow of the business logic is determined by objects that are statically assigned to one another. With inversion of control, the flow depends on the object graph that is instantiated by the assembler and is made possible by object interactions being defined through abstractions. The binding process is achieved through “dependency injection”.

Inversion of control is a design paradigm with the goal of giving more control to the targeted components of your application, the ones that are actually doing the work.

Dependency injection is a pattern used to create instances of objects that other objects rely on without knowing at compile time which class will be used to provide that functionality. Inversion of control relies on dependency injection because a mechanism is needed in order to activate the components providing the specific functionality. Otherwise how will the framework know which components to create if it is no longer in control?

In Java, dependency injection may happen through 3 ways:

1. A constructor injection
2. A setter injection
3. An interface injection

## 4) Explain IoC in Spring Framework?

The org.springframework.beans and org.springframework.context packages provide the basis for the Spring Framework’s IoC container. The BeanFactory interface provides an advanced configuration mechanism capable of managing objects of any nature. The ApplicationContext interface builds on top of the BeanFactory (it is a sub-interface) and adds other functionality such as easier integration with [**Spring’s AOP features**](http://howtodoinjava.com/category/frameworks/java-spring-tutorials/spring-aop/), [**message resource handling**](http://howtodoinjava.com/2015/02/10/spring-mvc-internationalization-i18n-and-localization-i10n-example/) (for use in internationalization), event propagation, and application-layer specific contexts such as theWebApplicationContext for use in web applications.

**The org.springframework.beans.factory.BeanFactory is the actual representation of the Spring IoC container that is responsible for containing and otherwise managing the aforementioned beans. The BeanFactory interface is the central IoC container interface in Spring.**

## 5) Difference between BeanFactory and ApplicationContext?

A BeanFactory is like a factory class that contains a collection of beans. The BeanFactory holds Bean Definitions of multiple beans within itself and then instantiates the bean whenever asked for by clients. BeanFactory is able to create associations between collaborating objects as they are instantiated. This removes the burden of configuration from bean itself and the beans client. BeanFactory also takes part in the life cycle of a bean, making calls to custom initialization and destruction methods.

On the surface, an application context is same as a bean factory.Both load bean definitions, wire beans together, and dispense beans upon request. But it also provides:

1. A means for resolving text messages, including support for internationalization.
2. A generic way to load file resources.
3. Events to beans that are registered as listeners.

The three commonly used implementations of ApplicationContext are:

1. **ClassPathXmlApplicationContext** : It Loads context definition from an XML file located in the classpath, treating context definitions as classpath resources. The application context is loaded from the application’s classpath by using the code.

|  |  |
| --- | --- |
| 1 | ApplicationContext context = new ClassPathXmlApplicationContext(“bean.xml”); |

1. FileSystemXmlApplicationContext : It loads context definition from an XML file in the filesystem. The application context is loaded from the file system by using the code.

|  |  |
| --- | --- |
| 1 | ApplicationContext context = new FileSystemXmlApplicationContext(“bean.xml”); |

1. XmlWebApplicationContext : It loads context definition from an XML file contained within a web application.

## 6) In how many ways, you can configure Spring into our application?

You can configure spring into your application in 3 ways:

1. XML Based Configuration
2. Annotation-based configuration
3. Java-based configuration

## 7) What is Spring XML-Based Configuration?

In Spring framework, dependencies and the services needed by beans are specified in configuration files, which are typically in an XML format. These configuration files usually start with <beans> tag and contain a lot of bean definitions AND application specific configuration options.

The main goal of Spring XML Configuration is to have all the Spring components configured by using xml files.  
This means that there will not be present any other type of Spring Configuration (like annotations or configuration via Java classes).

A Spring XML Configuration uses Spring namespaces to make available the sets of XML tags used in the configuration; the main Spring namespaces are: context, beans, jdbc, tx, aop, mvc, aso.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <beans>        <!-- JSON Support -->      <bean name="viewResolver" class="org.springframework.web.servlet.view.BeanNameViewResolver"/>      <bean name="jsonTemplate" class="org.springframework.web.servlet.view.json.MappingJackson2JsonView"/>        <bean id="restTemplate" class="org.springframework.web.client.RestTemplate"/>    </beans> |

And simplest web.xml file to make your application load configuration file and configure the runtime components for you is below where you configure only **DispatcherServlet**.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | <web-app>    <display-name>Archetype Created Web Application</display-name>      <servlet>          <servlet-name>spring</servlet-name>              <servlet-class>                  org.springframework.web.servlet.DispatcherServlet              </servlet-class>          <load-on-startup>1</load-on-startup>      </servlet>        <servlet-mapping>          <servlet-name>spring</servlet-name>          <url-pattern>/</url-pattern>      </servlet-mapping>    </web-app> |

## 8) What is Spring Java-Based Configuration?

The central artifacts in Spring’s new Java-configuration support are @Configuration annotated classes and @Beanannotated methods.

The @Bean annotation is used to indicate that a method instantiates, configures and initializes a new object to be managed by the Spring IoC container. @Bean annotation plays the same role as the <bean/> element.

Annotating a class with @Configuration indicates that its primary purpose is as a source of bean definitions. Furthermore, @Configuration classes allow inter-bean dependencies to be defined by simply calling other @Beanmethods in the same class. The simplest possible @Configuration class would read as follows:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | @Configuration  public class AppConfig  {      @Bean      public MyService myService() {          return new MyServiceImpl();      }  } |

The equivalent XML configuration for above java config would be:

|  |  |
| --- | --- |
| 1  2  3 | <beans>      <bean id="myService" class="com.howtodoinjava.services.MyServiceImpl"/>  </beans> |

To instantiate such config, you will need the help of AnnotationConfigApplicationContext class.

|  |  |
| --- | --- |
| 1  2  3  4  5 | public static void main(String[] args) {      ApplicationContext ctx = new AnnotationConfigApplicationContext(AppConfig.class);      MyService myService = ctx.getBean(MyService.class);      myService.doStuff();  } |

To enable component scanning, just annotate your @Configuration class as follows:

|  |  |
| --- | --- |
| 1  2  3  4  5 | @Configuration  @ComponentScan(basePackages = "com.howtodoinjava")  public class AppConfig  {      ...  } |

In the example above, the com.acme package will be scanned, looking for any @Component annotated classes, and those classes will be registered as Spring bean definitions within the container.

If you are using above configuration in a web application then you will be usingAnnotationConfigWebApplicationContext class. This implementation may be used when configuring the SpringContextLoaderListener servlet listener, Spring MVC DispatcherServlet etc.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49 | <web-app>      <!-- Configure ContextLoaderListener to use AnnotationConfigWebApplicationContext          instead of the default XmlWebApplicationContext -->      <context-param>          <param-name>contextClass</param-name>          <param-value>              org.springframework.web.context.support.AnnotationConfigWebApplicationContext          </param-value>      </context-param>        <!-- Configuration locations must consist of one or more comma- or space-delimited          fully-qualified @Configuration classes. Fully-qualified packages may also be          specified for component-scanning -->      <context-param>          <param-name>contextConfigLocation</param-name>          <param-value>com.howtodoinjava.AppConfig</param-value>      </context-param>        <!-- Bootstrap the root application context as usual using ContextLoaderListener -->      <listener>          <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>      </listener>        <!-- Declare a Spring MVC DispatcherServlet as usual -->      <servlet>          <servlet-name>dispatcher</servlet-name>          <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>          <!-- Configure DispatcherServlet to use AnnotationConfigWebApplicationContext              instead of the default XmlWebApplicationContext -->          <init-param>              <param-name>contextClass</param-name>              <param-value>                  org.springframework.web.context.support.AnnotationConfigWebApplicationContext              </param-value>          </init-param>          <!-- Again, config locations must consist of one or more comma- or space-delimited              and fully-qualified @Configuration classes -->          <init-param>              <param-name>contextConfigLocation</param-name>              <param-value>com.howtodoinjava.web.MvcConfig</param-value>          </init-param>      </servlet>        <!-- map all requests for /app/\* to the dispatcher servlet -->      <servlet-mapping>          <servlet-name>dispatcher</servlet-name>          <url-pattern>/app/\*</url-pattern>      </servlet-mapping>  </web-app> |

## 9) What is Spring Annotation-based Configuration?

Starting from Spring 2.5 it became possible to configure the dependency injection using annotations. So instead of using XML to describe a bean wiring, you can move the bean configuration into the component class itself by using annotations on the relevant class, method, or field declaration. Annotation injection is performed before XML injection, thus the latter configuration will override the former for properties wired through both approaches.

Annotation wiring is not turned on in the Spring container by default. So, before we can use annotation-based wiring, we will need to enable it in our Spring configuration file. So consider to have following configuration file in case you want to use any annotation in your Spring application.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <beans>       <context:annotation-config/>     <!-- bean definitions go here -->    </beans> |

Once <context:annotation-config/> is configured, you can start annotating your code to indicate that Spring should automatically wire values into properties, methods, and constructors.

Few important annotations which you will be using in this type of configuration are :

1. **@Required** : The @Required annotation applies to bean property setter methods.
2. **@Autowired** : The @Autowired annotation can apply to bean property setter methods, non-setter methods, constructor and properties.
3. **@Qualifier** : The @Qualifier annotation along with @Autowired can be used to remove the confusion by specifiying which exact bean will be wired.
4. **JSR-250 Annotations** : Spring supports JSR-250 based annotations which include @Resource, @PostConstruct and @PreDestroy annotations.

## 10) Explain Spring Bean lifecycle?

The life cycle of a Spring bean is easy to understand. When a bean is instantiated, it may be required to perform some initialization to get it into a usable state. Similarly, when the bean is no longer required and is removed from the container, some cleanup may be required.

Spring bean factory is responsible for managing the life cycle of beans created through spring container. The life cycle of beans consist of **call back methods which can be categorized broadly in two groups**:

1. Post initialization call back methods
2. Pre destruction call back methods

Spring framework provides following **4 ways for controlling life cycle events** of bean:

* InitializingBean and DisposableBean callback interfaces
* Other Aware interfaces for specific behavior
* Custom init() and destroy() methods in bean configuration file
* @PostConstruct and @PreDestroy annotations

For example, customInit() and customDestroy() methods are example of life cycle method.

|  |  |
| --- | --- |
| 1  2  3  4 | <beans>      <bean id="demoBean" class="com.howtodoinjava.task.DemoBean"              init-method="customInit" destroy-method="customDestroy"></bean>  </beans> |

**Read More:** [Spring Bean Life Cycle](http://howtodoinjava.com/2013/05/07/spring-bean-life-cycle/)

## 11) What are different Spring Bean Scopes?

The beans in spring container can be created in **five scopes**. All the scope names are self-explanatory but lets make them clear so that there will not be any doubt.

1. **singleton**: This bean scope is default and it enforces the container to have only one instance per spring container irrespective of how much time you request for its instance. This singleton behavior is maintained by bean factory itself.
2. **prototype**: This bean scope just reverses the behavior of singleton scope and produces a new instance each and every time a bean is requested.
3. **request**: With this bean scope, a new bean instance will be created for each web request made by client. As soon as request completes, bean will be out of scope and garbage collected.
4. **session**: Just like request scope, this ensures one instance of bean per user session. As soon as user ends its session, bean is out of scope.
5. **global-session**: global-session is something which is connected to Portlet applications. When your application works in Portlet container it is built of some amount of portlets. Each portlet has its own session, but if your want to store variables global for all portlets in your application than you should store them in global-session. This scope doesn’t have any special effect different from session scope in Servlet based applications.

**Read More :** [Spring Bean Scopes](http://howtodoinjava.com/2013/05/07/spring-bean-scopes/)

## 12) What are inner beans in Spring?

In Spring framework, whenever a bean is used for only one particular property, it’s advise to declare it as an inner bean. And the inner bean is supported both in setter injection ‘**property**‘ and constructor injection ‘**constructor-arg**‘.

For example, let’s say we one Customer class having reference of Person class. In our application, we will be creating only one instance of Person class, and use it inside Customer.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | public class Customer  {      private Person person;        //Setters and Getters  } |
| 1  2  3  4  5  6  7  8 | public class Person  {      private String name;      private String address;      private int age;        //Setters and Getters  } |

Now inner bean declaration will look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <bean id="CustomerBean" class="com.howtodoinjava.common.Customer">      <property name="person">          <!-- This is inner bean -->          <bean class="com.howtodoinjava.common.Person">              <property name="name" value="lokesh" />              <property name="address" value="India" />              <property name="age" value="34" />          </bean>      </property>  </bean> |

## 13) Are Singleton beans thread safe in Spring Framework?

Spring framework does not do anything under the hood concerning the multi-threaded behavior of a [**singleton**](http://howtodoinjava.com/2012/10/22/singleton-design-pattern-in-java/) bean. It is the developer’s responsibility to deal with concurrency issue and [**thread safety**](http://howtodoinjava.com/2014/06/02/what-is-thread-safety/) of the singleton bean.

While practically, most spring beans have no mutable state (e.g. Service and DAO clases), and as such are trivially thread safe. But if your bean has mutable state (e.g. View Model Objects), so you need to ensure thread safety. The most easy and obvious solution for this problem is to change bean scope of mutable beans from “**singleton**” to “**prototype**“.

## 14) How can you inject a Java Collection in Spring? Give example?

Spring offers four types of collection configuration elements which are as follows:

**<list>** : This helps in wiring ie injecting a list of values, allowing duplicates.  
**<set>** : This helps in wiring a set of values but without any duplicates.  
**<map>** : This can be used to inject a collection of name-value pairs where name and value can be of any type.  
**<props>** : This can be used to inject a collection of name-value pairs where the name and value are both Strings.

Let’s see example of each type.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46 | <beans>       <!-- Definition for javaCollection -->     <bean id="javaCollection" class="com.howtodoinjava.JavaCollection">          <!-- java.util.List -->        <property name="customList">          <list>             <value>INDIA</value>             <value>Pakistan</value>             <value>USA</value>             <value>UK</value>          </list>        </property>         <!-- java.util.Set -->       <property name="customSet">          <set>             <value>INDIA</value>             <value>Pakistan</value>             <value>USA</value>             <value>UK</value>          </set>        </property>         <!-- java.util.Map -->       <property name="customMap">          <map>             <entry key="1" value="INDIA"/>             <entry key="2" value="Pakistan"/>             <entry key="3" value="USA"/>             <entry key="4" value="UK"/>          </map>        </property>          <!-- java.util.Properties -->      <property name="customProperies">          <props>              <prop key="admin">admin@nospam.com</prop>              <prop key="support">support@nospam.com</prop>          </props>      </property>       </bean>    </beans> |

## 15) How to inject a java.util.Properties into a Spring Bean?

First way is to use **<props>** tag as below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <bean id="adminUser" class="com.howtodoinjava.common.Customer">        <!-- java.util.Properties -->      <property name="emails">          <props>              <prop key="admin">admin@nospam.com</prop>              <prop key="support">support@nospam.com</prop>          </props>      </property>    </bean> |

You can use “**util:**” namespace as well to create properties bean from properties file, and use bean reference for setter injection.

|  |  |
| --- | --- |
| 1 | <util:properties id="emails" location="classpath:com/foo/emails.properties" /> |

## 16) Explain Spring Bean Autowiring?

In spring framework, setting bean dependencies in configuration files is a good practice to follow, but the spring container is also able to autowire relationships between collaborating beans. This means that it is possible to automatically let Spring resolve collaborators (other beans) for your bean by inspecting the contents of the BeanFactory. [**Autowiring**](http://howtodoinjava.com/2013/05/08/spring-beans-autowiring-concepts/) is specified per bean and can thus be enabled for some beans, while other beans will not be autowired.

The following excerpt from the XML configuration file shows a bean being autowired by name.

|  |  |
| --- | --- |
| 1 | <bean id="employeeDAO" class="com.howtodoinjava.EmployeeDAOImpl" autowire="byName" /> |

Apart from the autowiring modes provided in bean configuration file, autowiring can be specified in bean classes also using @Autowired annotation. To use @Autowired annotation in bean classes, you must first enable the annotation in spring application using below configuration.

|  |  |
| --- | --- |
| 1 | <context:annotation-config /> |

Same can be achieved using AutowiredAnnotationBeanPostProcessor bean definition in configuration file.

|  |  |
| --- | --- |
| 1 | <bean class ="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor"/> |

Now, when annotation configuration has been enables, you are free to autowire bean dependencies using@Autowired, the way you like.

|  |  |
| --- | --- |
| 1  2  3  4 | @Autowired  public EmployeeDAOImpl ( EmployeeManager manager ) {      this.manager = manager;  } |

## 17) Explain different modes of bean autowiring?

There are **five auto wiring modes** in spring framework. Lets discuss them one by one.

1. **no**: This option is default for spring framework and it means that autowiring is OFF. You have to explicitly set the dependencies using tags in bean definitions.
2. **byName**: This option enables the dependency injection based on bean names. When autowiring a property in bean, property name is used for searching a matching bean definition in configuration file. If such bean is found, it is injected in property. If no such bean is found, a error is raised.
3. **byType**: This option enables the dependency injection based on bean types. When autowiring a property in bean, property’s class type is used for searching a matching bean definition in configuration file. If such bean is found, it is injected in property. If no such bean is found, a error is raised.
4. **constructor**: Autowiring by constructor is similar to byType, but applies to constructor arguments. In autowire enabled bean, it will look for class type of constructor arguments, and then do a autowire by type on all constructor arguments. Please note that if there isn’t exactly one bean of the constructor argument type in the container, a fatal error is raised.
5. **autodetect**: Autowiring by autodetect uses either of two modes i.e. constructor or byType modes. First it will try to look for valid constructor with arguments, If found the constructor mode is chosen. If there is no constructor defined in bean, or explicit default no-args constructor is present, the autowire byType mode is chosen.

## 18) How do you turn on annotation based autowiring?

To enable @Autowired, you have to register AutowiredAnnotationBeanPostProcessor, and you can do it in two ways.

1. Include <context:annotation-config > in bean configuration file.

|  |  |
| --- | --- |
| 1  2  3 | <beans>      <context:annotation-config />  </beans> |

2. Include AutowiredAnnotationBeanPostProcessor directly in bean configuration file.

|  |  |
| --- | --- |
| 1  2  3 | <beans>      <bean class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor"/>  </beans> |

## 19) Explain @Required annotation with example?

In a production-scale application, there may be hundreds or thousands of beans declared in the IoC container, and the dependencies between them are often very complicated. One of the shortcomings of setter injection is that it’s very hard for you to check if all required properties have been set or not. To overcome this problem, you can set “**dependency-check**” attribute of <bean> and set one of four attributes i.e. none, simple, objects or all (none is default option).

In real life application, you will not be interested in checking all the bean properties configured in your context files. Rather you would like to check if particular set of properties have been set or not in some specific beans only. Spring’s dependency checking feature using “**dependency-check**” attribute, will not able to help you in this case. So solve this problem, you can use @Required annotation.

To Use the @Required annotation over setter method of bean property in class file as below:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | public class EmployeeFactoryBean extends AbstractFactoryBean<Object>  {      private String designation;        public String getDesignation() {          return designation;      }        @Required      public void setDesignation(String designation) {          this.designation = designation;      }        //more code here  } |

RequiredAnnotationBeanPostProcessor is a spring bean post processor that checks if all the bean properties with the @Required annotation have been set. To enable this bean post processor for property checking, you must register it in the Spring IoC container.

|  |  |
| --- | --- |
| 1 | <bean class="org.springframework.beans.factory.annotation.RequiredAnnotationBeanPostProcessor" /> |

If any properties with @Required have not been set, a BeanInitializationException will be thrown by this bean post processor.

## 20) Explain @Autowired annotation with example?

The @Autowired annotation provides more fine-grained control over where and how autowiring should be accomplished. The @Autowired annotation can be used to autowire bean on the setter method just like @Requiredannotation, constructor, a property or methods with arbitrary names and/or multiple arguments.

E.g. You can use @Autowired annotation on setter methods to get rid of the <property> element in XML configuration file. When Spring finds an @Autowired annotation used with setter methods, it tries to perform **byType** autowiring on the method.

You can apply @Autowired to constructors as well. A constructor @Autowired annotation indicates that the constructor should be autowired when creating the bean, even if no <constructor-arg> elements are used while configuring the bean in XML file.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | public class TextEditor {     private SpellChecker spellChecker;       @Autowired     public TextEditor(SpellChecker spellChecker){        System.out.println("Inside TextEditor constructor." );        this.spellChecker = spellChecker;     }       public void spellCheck(){        spellChecker.checkSpelling();     }  } |

And it’s configuration without constructor arguments.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <beans>       <context:annotation-config/>       <!-- Definition for textEditor bean without constructor-arg  -->     <bean id="textEditor" class="com.howtodoinjava.TextEditor">     </bean>       <!-- Definition for spellChecker bean -->     <bean id="spellChecker" class="com.howtodoinjava.SpellChecker">     </bean>    </beans> |

## 21) Explain @Qualifier annotation with example?

@Qualifier means, which bean is qualify to autowired on a field. The qualifier annotation helps disambiguate bean references when Spring would otherwise not be able to do so.

See below example, it will autowired a “**person**” bean into customer’s person property.

|  |  |
| --- | --- |
| 1  2  3  4  5 | public class Customer  {      @Autowired      private Person person;  } |

And we have two bean definitions for Person class.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <bean id="customer" class="com.howtodoinjava.common.Customer" />    <bean id="personA" class="com.howtodoinjava.common.Person" >      <property name="name" value="lokesh" />  </bean>    <bean id="personB" class="com.howtodoinjava.common.Person" >      <property name="name" value="alex" />  </bean> |

Will Spring know which person bean should autowired? NO. When you run above example, it hits below exception :

|  |  |
| --- | --- |
| 1  2  3 | Caused by: org.springframework.beans.factory.NoSuchBeanDefinitionException:      No unique bean of type [com.howtodoinjava.common.Person] is defined:          expected single matching bean but found 2: [personA, personB] |

To fix above problem, you need @Quanlifier to tell Spring about which bean should autowired.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | public class Customer  {      @Autowired      @Qualifier("personA")      private Person person;  } |

## 22) Difference between constructor injection and setter injection?

Please find below the noticeable differences:

1. In Setter Injection, partial injection of dependencies can possible, means if we have 3 dependencies like int, string, long, then its not necessary to inject all values if we use setter injection. If you are not inject it will takes default values for those primitives. In constructor injection, partial injection of dependencies is not possible, because for calling constructor we must pass all the arguments right, if not so we may get error.
2. Setter Injection will overrides the constructor injection value, provided if we write setter and constructor injection for the same property. But, constructor injection cannot overrides the setter injected values. It’s obvious because constructors are called to first to create the instance.
3. Using setter injection you can not guarantee that certain dependency is injected or not, which means you may have an object with incomplete dependency. On other hand constructor Injection does not allow you to construct object, until your dependencies are ready.
4. In setter injection, if Object A and B are dependent each other i.e A is depends on B and vice-versa, Spring throwsObjectCurrentlyInCreationException while creating objects of A and B because A object cannot be created until B is created and vice-versa. So spring can resolve circular dependencies through setter-injection because Objects are constructed before setter methods invoked.

## 23) What are the different types of events in spring framework?

Spring’s ApplicationContext provides the functionality to support events and listeners in code. We can create beans that listen for events which are published through our ApplicationContext. Event handling in the ApplicationContext is provided through the ApplicationEvent class and ApplicationListener interface. So if a bean implements theApplicationListener, then every time an ApplicationEvent gets published to the ApplicationContext, that bean is notified.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public class AllApplicationEventListener implements ApplicationListener < ApplicationEvent >  {      @Override      public void onApplicationEvent(ApplicationEvent applicationEvent)      {          //process event      }  } |

Spring provides the following **5 standard events**:

1. **ContextRefreshedEvent** : This event is published when the ApplicationContext is either initialized or refreshed. This can also be raised using the refresh() method on the ConfigurableApplicationContext interface.
2. **ContextStartedEvent** : This event is published when the ApplicationContext is started using the start() method on the ConfigurableApplicationContext interface. You can poll your database or you can re/start any stopped application after receiving this event.
3. **ContextStoppedEvent** : This event is published when the ApplicationContext is stopped using the stop() method on the ConfigurableApplicationContext interface. You can do required house keeping work after receiving this event.
4. **ContextClosedEvent** : This event is published when the ApplicationContext is closed using the close() method on the ConfigurableApplicationContext interface. A closed context reaches its end of life; it cannot be refreshed or restarted.
5. **RequestHandledEvent** : This is a web-specific event telling all beans that an HTTP request has been serviced.

Apart from above, you can create your own custom events by extending ApplicationEvent class. e.g.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public class CustomApplicationEvent extends ApplicationEvent  {      public CustomApplicationEvent ( Object source, final String msg )      {          super(source);          System.out.println("Created a Custom event");      }  } |

To listen this event, create a listener like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | public class CustomEventListener implements ApplicationListener < CustomApplicationEvent >  {      @Override      public void onApplicationEvent(CustomApplicationEvent applicationEvent) {          //handle event      }  } |

And to publish this event, you will need the help of applicationContext instance.

|  |  |
| --- | --- |
| 1  2 | CustomApplicationEvent customEvent = new CustomApplicationEvent( applicationContext, "Test message" );  applicationContext.publishEvent ( customEvent ); |

## 24) Difference between FileSystemResource and ClassPathResource?

In FileSystemResource you need to give path of spring-config.xml (Spring Configuration) file relative to your project or the absolute location of the file.

In ClassPathResource spring looks for the file using ClassPath so spring-config.xml should be included in classpath. If spring-config.xml is in “**src**” so we can give just its name because **src** is in classpath path by default.

**In one sentence, ClassPathResource looks in the class path and FileSystemResource looks in the file system.**

## 25) Name some of the design patterns used in Spring Framework?

There are loads of different design patterns used, but there are a few obvious ones:

* **Proxy** -- used heavily in AOP, and remoting.
* **Singleton** -- beans defined in spring config files are singletons by default.
* **Template method** -- used extensively to deal with boilerplate repeated code e.g. [**RestTemplate**](http://howtodoinjava.com/2015/02/20/spring-restful-client-resttemplate-example/), JmsTemplate,JpaTemplate.
* **Front Controller** -- Spring provides DispatcherServlet to ensure an incoming request gets dispatched to your controllers.
* **View Helper** -- Spring has a number of custom JSP tags, and velocity macros, to assist in separating code from presentation in views.
* **Dependency injection** -- Center to the whole BeanFactory / ApplicationContext concepts.
* **Factory pattern** -- BeanFactory for creating instance of an object.

# Inversion of control (IoC) and dependency injection (DI) patterns in spring framework and related interview questions

**Sections in this post:**

IoC introduction

Differentiating with dependency injection

Various ways to implement IoC

IoC in spring framework

Ways to instantiate beans

Various dependency injection mechanisms

Some commonly asked interview questions

### IoC introduction

Intraditional programming, the flow of the business logic is determined by objects that are statically assigned to one another. With **inversion of control**, the flow depends on the object graph that is instantiated by the assembler and is made possible by object interactions being defined through abstractions. The binding process is achieved through**dependency injection**, although some argue that the use of a service locator also provides inversion of control.

Inversion of control as a ***design guideline*** serves the following purposes:

1. There is a decoupling of the execution of a certain task from implementation.
2. Every module can focus on what it is designed for.
3. Modules make no assumptions about what other systems do but rely on their contracts.
4. Replacing modules has no side effect on other modules.

### Differentiating with dependency injection

Inversion of control is a design paradigm with the goal of giving more control to the targeted components of your application, the ones getting the work done.  
Dependency injection is a pattern used to create instances of objects that other objects rely on without knowing at compile time which class will be used to provide that functionality. Inversion of control relies on dependency injection because a mechanism is needed in order to activate the components providing the specific functionality.  
The two concepts work together in this way to allow for much more flexible, reusable, and encapsulated code to be written. As such, they are important concepts in designing object-oriented solutions.

### Implementing inversion of control design pattern

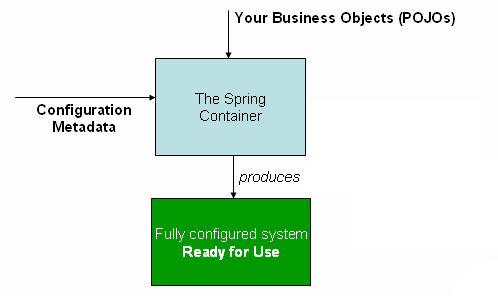
In object-oriented programming, there are several basic techniques to implement inversion of control. These are:

1. using a factory pattern
2. using a service locator pattern
3. using a **dependency injection** of any given below type:
   * a constructor injection
   * a setter injection
   * an interface injection

### IoC in spring framework

The**org.springframework.beans** and **org.springframework.contex**t packages provide the basis for the Spring Framework’s IoC container. The [**BeanFactory**](http://static.springsource.org/spring/docs/1.2.x/api/org/springframework/beans/factory/BeanFactory.html) interface provides an advanced configuration mechanism capable of managing objects of any nature. The [**ApplicationContext**](http://static.springsource.org/spring/docs/2.5.5/api/org/springframework/context/ApplicationContext.html) interface builds on top of the BeanFactory (it is a sub-interface) and adds other functionality such as easier integration with Spring’s AOP features, message resource handling (for use in internationalization), event propagation, and application-layer specific contexts such as the WebApplicationContext for use in web applications.

The **org.springframework.beans.factory.BeanFactory is the actual representation of the Spring IoC** container that is responsible for containing and otherwise managing the aforementioned beans. The BeanFactory interface is the central IoC container interface in Spring.

[](http://howtodoinjava.files.wordpress.com/2013/03/container-magic.png)

There are a number of implementations of the BeanFactory interface. The most commonly used BeanFactory implementation is the **XmlBeanFactory** class. Other commonly used class is **XmlWebApplicationContext**. Depending on the bean definition, the factory will return either an independent instance of a contained object (the Prototype design pattern), or a single shared instance (a superior alternative to the Singleton design pattern, in which the instance is a singleton in the scope of the factory). Which type of instance will be returned depends on the bean factory configuration: the API is the same.

Before we dive into dependency injection types, let first identify the ways of creating a bean in spring framework as it will help in understanding the things in next section.

### Ways to instantiate beans in spring

A bean definition can be seen as a recipe for creating one or more actual objects. The container looks at the recipe for a named bean when asked, and uses the configuration metadata encapsulated by that bean definition to create (or acquire) an actual object.

**Instantiation using a constructor**

When creating a bean using the constructor approach, all normal classes are usable by and compatible with Spring. That is, the class being created does not need to implement any specific interfaces or be coded in a specific fashion. Just specifying the bean class should be enough. When using XML-based configuration metadata you can specify your bean class like so:

|  |  |
| --- | --- |
| 1 | <bean id="exampleBean"/> |

**Instantiation using a static factory method**

When defining a bean which is to be created using a static factory method, along with the class attribute which specifies the class containing the static factory method, another attribute named factory-method is needed to specify the name of the factory method itself.

|  |  |
| --- | --- |
| 1 | <bean id="exampleBean" factory-method="createInstance"/> |

Spring expects to be able to call this method and get back a live object, which from that point on is treated as if it had been created normally via a constructor.

**Instantiation using an instance factory method**

In a fashion similar to instantiation via a static factory method, instantiation using an instance factory method is where the factory method of an existing bean from the container is invoked to create the new bean.

|  |  |
| --- | --- |
| 1  2  3  4 | <bean id="myFactoryBean"  class="...">    <bean id="exampleBean"  factory-bean="myFactoryBean"              factory-method="createInstance"></bean> |

### Various dependency injection mechanisms

The basic principle behind Dependency Injection (DI) is that objects define their dependencies only through constructor arguments, arguments to a factory method, or properties which are set on the object instance after it has been constructed or returned from a factory method. Then, it is the job of the container to actually inject those dependencies when it creates the bean. This is fundamentally the inverse, hence the name Inversion of Control (IoC).

**Setter injection**

Setter-based DI is realized by calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public class TestSetter {        DemoBean demoBean = null;        public void setDemoBean(DemoBean demoBean) {          this.demoBean = demoBean;      }  } |

**Constructor injection**

Constructor-based DI is realized by invoking a constructor with a number of arguments, each representing a collaborator. Additionally, calling a static factory method with specific arguments to construct the bean, can be considered almost equivalent, and the rest of this text will consider arguments to a constructor and arguments to a static factory method similarly.

**Interface injection** In this methodology we implement an interface from the IOC framework. IOC framework will use the interface method to inject the object in the main class. It is much more appropriate to use this approach when you need to have some logic that is not applicable to place in a property. Such as logging support.

|  |  |
| --- | --- |
| 1  2  3  4  5 | public void SetLogger(ILogger logger)  {    \_notificationService.SetLogger(logger);    \_productService.SetLogger(logger);  } |

### Some commonly asked interview questions

**What is difference between component and service?**

A component is a glob of software that’s intended to be used, without change, by an application that is out of the control of the writers of the component. By ‘without change’ means that the using application doesn’t change the source code of the components, although they may alter the component’s behavior by extending it in ways allowed by the component writers.

A service is similar to a component in that it’s used by foreign applications. The main difference is that a component to be used locally (think jar file, assembly, dll, or a source import). A service will be used remotely through some remote interface, either synchronous or asynchronous (eg web service, messaging system, RPC, or socket.)

**How DI is different from Service locator pattern?**

The key benefit of a Dependency Injector is that it allows to plug-in a suitable implementation of a service according to environment and usage. Injection isn’t the only way to break this dependency, another is to use a service locator. The basic idea behind a service locator is to have an object that knows how to get hold of all of the services that an application might need. It then scans all such services and store them as a singleton Registry. When asked for a service implementation, a requester can query the registry with a token and get appropriate implementation.

Mostly these registries are populated via some configuration files. The key difference is that with a Service Locator every user of a service has a dependency to the locator. The locator can hide dependencies to other implementations, but you do need to see the locator.

**Which one should be better to use i.e. service locator or dependency injection?**

Well, it as I already said that key difference is that with a Service Locator every user of a service has a dependency to the locator. It means you must know the details of service locator in terms of input and output. So, it actually becomes the deciding factor which pattern to choose from.

If it is easy and necessary to maintain registry information then go for service locator, or else simply use dependency injection as it does not bother the users of service with any per-requisites.

**Which is better constructor injection or setter injection?**

The choice between setter and constructor injection is interesting as it mirrors a more general issue with object-oriented programming -- should you fill fields in a constructor or with setters.  
Constructors with parameters give you a clear statement of what it means to create a valid object in an obvious place. If there’s more than one way to do it, create multiple constructors that show the different combinations. Another advantage with constructor initialization is that it allows you to clearly hide any fields that are immutable by simply not providing a setter. I think this is important -- if something shouldn’t change then the lack of a setter communicates this very well. If you use setters for initialization, then this can become a pain.

But If you have a lot of constructor parameters things can look messy, particularly in languages without keyword parameters. If you have multiple ways to construct a valid object, it can be hard to show this through constructors, since constructors can only vary on the number and type of parameters. Constructors also suffer if you have simple parameters such as strings. With setter injection you can give each setter a name to indicate what the string is supposed to do. With constructors you are just relying on the position, which is harder to follow.

My preference is to start with constructor injection, but be ready to switch to setter injection as soon as the problems I’ve outlined above start to become a problem.

**What is Bean Factory ?**

A BeanFactory is like a factory class that contains a collection of beans. The BeanFactory holds Bean Definitions of multiple beans within itself and then instantiates the bean whenever asked for by clients.

BeanFactory is able to create associations between collaborating objects as they are instantiated. This removes the burden of configuration from bean itself and the beans client. BeanFactory also takes part in the life cycle of a bean, making calls to custom initialization and destruction methods.

**What is Application Context?**

A bean factory is fine to simple applications, but to take advantage of the full power of the Spring framework, you may want to move up to Springs more advanced container, the application context. On the surface, an application context is same as a bean factory.Both load bean definitions, wire beans together, and dispense beans upon request. But it also provides:

* A means for resolving text messages, including support for internationalization.
* A generic way to load file resources.
* Events to beans that are registered as listeners.

**What are the common implementations of the Application Context ?**

The three commonly used implementation of ‘Application Context’ are

**1**) ClassPathXmlApplicationContext : It Loads context definition from an XML file located in the classpath, treating context definitions as classpath resources. The application context is loaded from the application’s classpath by using the code .

ApplicationContext context = new ClassPathXmlApplicationContext(“bean.xml”);

**2**) FileSystemXmlApplicationContext : It loads context definition from an XML file in the filesystem. The application context is loaded from the file system by using the code .

ApplicationContext context = new FileSystemXmlApplicationContext(“bean.xml”);

**3**) XmlWebApplicationContext : It loads context definition from an XML file contained within a web application.

**What should be used preferably BeanFactory or ApplicationContext?**

A BeanFactory pretty much just instantiates and configures beans. An ApplicationContext also does that, and it provides the supporting infrastructure to enable lots of enterprise-specific features such as transactions and AOP. In short, favor the use of an ApplicationContext.

# Top Spring AOP Interview Questions with Answers

**Table of Contents**

[Describe Spring AOP?](http://howtodoinjava.com/2015/02/27/top-spring-aop-interview-questions-with-answers/#spring_aop)

[What is the difference between concern and cross-cutting concern in Spring AOP?](http://howtodoinjava.com/2015/02/27/top-spring-aop-interview-questions-with-answers/#concerns)

[What are the available AOP implementations?](http://howtodoinjava.com/2015/02/27/top-spring-aop-interview-questions-with-answers/#aop_implementations)

[What are the different advice types in spring?](http://howtodoinjava.com/2015/02/27/top-spring-aop-interview-questions-with-answers/#advice_types)

[What is Spring AOP Proxy?](http://howtodoinjava.com/2015/02/27/top-spring-aop-interview-questions-with-answers/#proxy)

[What is Introduction?](http://howtodoinjava.com/2015/02/27/top-spring-aop-interview-questions-with-answers/#introduction)

[What is joint point and Point cut?](http://howtodoinjava.com/2015/02/27/top-spring-aop-interview-questions-with-answers/#jointpoint_pointcut)

[What is Weaving in AOP?](http://howtodoinjava.com/2015/02/27/top-spring-aop-interview-questions-with-answers/#weaving)

## Describe Spring AOP?

[**Spring AOP (Aspect Oriented Programming)**](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/aop.html) compliments [**OOPs**](http://howtodoinjava.com/category/object-oriented-principles/) in the sense that it also provides modularity. In OOPs, key unit is Objects, but **in AOP key unit is aspects** or concerns (simply assume stand-alone modules in your application). Some aspects have centralized code but other aspects may be scattered or tangled e.g. logging or transactions. **These scattered aspects are called cross-cutting concern**. A cross-cutting concern is a concern that can affect the whole application and should be centralized in one location in code as possible, such as transaction management, authentication, logging, security etc.

AOP provides the way to dynamically add the cross-cutting concern before, after or around the actual logic using simple pluggable configurations. It makes easy to maintain code in present and future as well. You can add/remove concerns without recompiling complete sourcecode simply by changing configuration files (if you are applying aspects suing XML configuration).

Spring AOP can be used by majorly 2 ways given below. But the widely used approach is Spring AspectJ Annotation Style.

**1)**[**By AspectJ annotation-style**](http://howtodoinjava.com/2015/01/30/spring-aop-aspectj-example-tutorial-using-annotation-config/) **2)**[**By Spring XML configuration-style**](http://howtodoinjava.com/2015/02/03/spring-aop-aspectj-xml-configuration-example/)

## What is the difference between concern and cross-cutting concern in Spring AOP?

**Concern is behavior which we want to have in a module of an application.** Concern may be defined as a functionality we want to implement to solve a specific business problem. E.g. in any eCommerce application different concerns (or modules) may be inventory management, shipping management, user management etc.

**Cross-cutting concern is a concern which is applicable throughout the application (or more than one module).** e.g. logging , security and data transfer are the concerns which are needed in almost every module of an application, hence they are termed as cross-cutting concerns.

## What are the available AOP implementations?

Main java based AOP implementations are listed below :

1. [AspectJ](http://eclipse.org/aspectj/)
2. Spring AOP
3. [JBoss AOP](http://jbossaop.jboss.org/)

You can find the big list of AOP implementations in [**wiki page**](http://en.wikipedia.org/wiki/Aspect-oriented_programming#Implementations).

## What are the different advice types in spring?

An advice is the implementation of cross-cutting concern which you are interested in applying on other modules of your application. Advices are of mainly 5 types :

1. **Before advice** : Advice that executes before a join point, but which does not have the ability to prevent execution flow proceeding to the join point (unless it throws an exception). To use this advice, use @Before annotation.
2. **After returning advice** : Advice to be executed after a join point completes normally. For example, if a method returns without throwing an exception. To use this advice, use @AfterReturning annotation.
3. **After throwing advice** : Advice to be executed if a method exits by throwing an exception. To use this advice, use@AfterThrowing annotation.
4. **After advice** : Advice to be executed regardless of the means by which a join point exits (normal or exceptional return). To use this advice, use @After annotation.
5. **Around advice** : Advice that surrounds a join point such as a method invocation. This is the most powerful kind of advice. To use this advice, use @Around annotation.

## What is Spring AOP Proxy?

A proxy is a well-used design pattern. To put it simply, **a proxy is an object that looks like another object, but adds special functionality behind the scene**.

Spring AOP is proxy-based. AOP proxy is an object created by the AOP framework in order to implement the aspect contracts in runtime.

Spring AOP defaults to using standard JDK dynamic proxies for AOP proxies. This enables any interface (or set of interfaces) to be proxied. Spring AOP can also use CGLIB proxies. This is necessary to proxy classes, rather than interfaces.

**CGLIB is used by default if a business object does not implement an interface.**

## What is Introduction?

**Introductions enable an aspect to declare that advised objects implement any additional interface(s) which they don’t have in real**, and to provide an implementation of that interface on behalf of those objects.

An introduction is made using the @DeclareParents annotation.

Read more about [**introductions**](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/aop.html#aop-introductions).

## What is Joint point and Point cut?

Join point is a point of execution of the program, such as the execution of a method or the handling of an exception. In Spring AOP, a **join point always represents a method execution**. For example, all the methods defined inside yourEmployeeManager interface cab be considered joint points if you apply any cross-cutting concern of them.

**Pointcut is a predicate or expression that matches join points.** Advice is associated with a pointcut expression and runs at any join point matched by the pointcut (for example, expression “execution(\* EmployeeManager.getEmployeeById(..))” to match getEmployeeById() the method in EmployeeManagerinterface). The concept of join points as matched by pointcut expressions is central to AOP, and Spring uses the AspectJ pointcut expression language by default.

## What is Weaving?

**The Spring AOP framework supports only limited types of AspectJ pointcuts and allows aspects to apply to beans declared in the IoC container. If you want to use additional pointcut types or apply your aspects “to objects created outside the Spring IoC container“, you have to use the AspectJ framework in your Spring application and use it’s weaving feature.**

Weaving is the process of linking aspects with other outsider application types or objects to create an advised object. This can be done at compile time (using the AspectJ compiler, for example), load time, or at runtime. Spring AOP, like other pure Java AOP frameworks, performs weaving at runtime only. In contrast, the AspectJ framework supports both compile-time and load-time weaving.

AspectJ compile-time weaving is done through a special AspectJ compiler called ajc. It can weave aspects into your Java source files and output woven binary class files. It can also weave aspects into your compiled class files or JAR files. This process is known as post-compile-time weaving. You can perform compile-time and post-compile-time weaving for your classes before declaring them in the Spring IoC container. Spring is not involved in the weaving process at all. For more information on compile-time and post-compile-time weaving, please refer to the AspectJ documentation.

AspectJ load-time weaving (also known as LTW) happens when the target classes are loaded into JVM by a class loader. For a class to be woven, a special class loader is required to enhance the bytecode of the target class. Both AspectJ and Spring provide load-time weavers to add load-time weaving capability to the class loader. You need only simple configurations to enable these load-time weavers.

# Spring MVC: Difference between <context:annotation-config> vs <context:component-scan>

[Follow @HowToDoInJava](https://twitter.com/HowToDoInJava), [Read RSS Feed](http://howtodoinjava.com/feed)

We have already learned few things in [**Spring MVC**](http://howtodoinjava.com/spring-3/) in previous posts. In those tutorials, I did use tags like<context:annotation-config> or <context:component-scan>, but I didn’t explained much in detail about these tags. I am writing this post, specifically to list down the difference between tags <context:annotation-config> and<context:component-scan> so that when you use them in future, you will know, what exactly are you doing.

1) First big difference between both tags is that <context:annotation-config> is **used to activate applied annotations in already registered beans in application context**. Note that it simply does not matter whether bean was registered by which mechanism e.g. using <context:component-scan> or it was defined in application-context.xml file itself.

2) Second difference is driven from first difference itself. It does **register the beans in context + it also scans the annotations inside beans and activate them**. So <context:component-scan&gt; does what<context:annotation-config> does, but additionally it scan the packages and register the beans in application context.

### Example of <context:annotation-config> vs <context:component-scan> uses

I will elaborate both tags in detail with some examples which will make more sense to us. For keeping the example to simple, I am creating just 3 beans, and I will try to configure them in configuration file in various ways, then we will see the difference between various configurations in console where output will get printed.

For reference, below are 3 beans. BeanA has reference to BeanB and BeanC additionally.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54 | package com.howtodoinjava.beans;    import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Component;    @SuppressWarnings("unused")  @Component  public class BeanA {        private BeanB beanB;      private BeanC beanC;        public BeanA(){          System.out.println("Creating bean BeanA");      }        @Autowired      public void setBeanB(BeanB beanB) {          System.out.println("Setting bean reference for BeanB");          this.beanB = beanB;      }        @Autowired      public void setBeanC(BeanC beanC) {          System.out.println("Setting bean reference for BeanC");          this.beanC = beanC;      }  }    //Bean B    package com.howtodoinjava.beans;    import org.springframework.stereotype.Component;    @Component  public class BeanB {      public BeanB(){          System.out.println("Creating bean BeanB");      }  }    //Bean C    package com.howtodoinjava.beans;    import org.springframework.stereotype.Component;    @Component  public class BeanC {      public BeanC(){          System.out.println("Creating bean BeanC");      }  } |

BeanDemo class is used to load and initialize the application context.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | package com.howtodoinjava.test;    import org.springframework.context.ApplicationContext;  import org.springframework.context.support.ClassPathXmlApplicationContext;    public class BeanDemo {      public static void main(String[] args) {          ApplicationContext context = new ClassPathXmlApplicationContext("classpath:beans.xml");      }  } |

Now let’s start writing the configuration file "beans.xml" with variations. I will be omitting the schema declarations in below examples, to keep focus on configuration itself.

#### a) Define only bean tags

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <bean id="beanA" class="com.howtodoinjava.beans.BeanA"></bean>  <bean id="beanB" class="com.howtodoinjava.beans.BeanB"></bean>  <bean id="beanC" class="com.howtodoinjava.beans.BeanC"></bean>    Output:    Creating bean BeanA  Creating bean BeanB  Creating bean BeanC |

In this case, all 3 beans are created and no dependency in injected in BeanA because we didn’t used any property/ref attributes.

#### b) Define bean tags and property ref attributes

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <bean id="beanA" class="com.howtodoinjava.beans.BeanA">      <property name="beanB" ref="beanB"></property>      <property name="beanC" ref="beanC"></property>  </bean>  <bean id="beanB" class="com.howtodoinjava.beans.BeanB"></bean>  <bean id="beanC" class="com.howtodoinjava.beans.BeanC"></bean>    Output:    Creating bean BeanA  Creating bean BeanB  Creating bean BeanC  Setting bean reference for BeanB  Setting bean reference for BeanC |

Now the beans are created and injected as well. No wonder.

#### c) Using only <context:annotation-config />

|  |  |
| --- | --- |
| 1  2  3 | <context:annotation-config />    //No Output |

As I told already, <context:annotation-config /> activate the annotations only on beans which have already been discovered and registered. Here, we have not discovered any bean so nothing happened.

#### d) Using <context:annotation-config /> with bean declarations

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <context:annotation-config />  <bean id="beanA" class="com.howtodoinjava.beans.BeanA"></bean>  <bean id="beanB" class="com.howtodoinjava.beans.BeanB"></bean>  <bean id="beanC" class="com.howtodoinjava.beans.BeanC"></bean>    Output:    Creating bean BeanA  Creating bean BeanB  Setting bean reference for BeanB  Creating bean BeanC  Setting bean reference for BeanC |

In above configuration, we have discovered the beans using <bean> tags. Now when we use<context:annotation-config />, it simply activates @Autowired annotation and bean injection inside BeanA happens.

#### e) Using only <context:component-scan />

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <context:component-scan base-package="com.howtodoinjava.beans" />    Output:    Creating bean BeanA  Creating bean BeanB  Setting bean reference for BeanB  Creating bean BeanC  Setting bean reference for BeanC |

Above configuration does both things as I mentioned earlier in start of post. It does the bean discovery (searches for@Component annotation in base package) and then activates the additional annotations (e.g. Autowired).

#### f) Using both <context:component-scan /> and <context:annotation-config />

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <context:annotation-config />  <context:component-scan base-package="com.howtodoinjava.beans" />  <bean id="beanA" class="com.howtodoinjava.beans.BeanA"></bean>  <bean id="beanB" class="com.howtodoinjava.beans.BeanB"></bean>  <bean id="beanC" class="com.howtodoinjava.beans.BeanC"></bean>    Output:    Creating bean BeanA  Creating bean BeanB  Setting bean reference for BeanB  Creating bean BeanC  Setting bean reference for BeanC |

# How to use Spring @Component, @Repository, @Service and @Controller Annotations?

In [**spring autowiring concepts**](http://howtodoinjava.com/2013/05/08/spring-beans-autowiring-concepts/), we learned about @Autowired annotation that it handles only wiring. You still have to define the beans themselves so the container is aware of them and can inject them for you. But with @Component,@Repository, @Service and @Controller annotations in place and after enabling automatic component scanning, spring will automatically import the beans into the container so you don’t have to define them explicitly with XML. These annotations are called **Stereotype annotations** as well.

Before jumping to example use of these annotations, let’s learn quick facts about these annotations which will help you in making a better decision about when to use which annotation.

## @Component, @Repository, @Service and @Controller annotations

1) The @Component annotation marks a java class as a bean so the component-scanning mechanism of spring can pick it up and pull it into the application context. To use this annotation, apply it over class as below:

|  |  |
| --- | --- |
| 1  2  3  4 | @Component  public class EmployeeDAOImpl implements EmployeeDAO {      ...  } |

2) Although above use of @Component is good enough but you can use more suitable annotation that provides additional benefits specifically for DAOs i.e. @Repository annotation. The @Repository annotation is a specialization of the @Component annotation with similar use and functionality. In addition to importing the DAOs into the DI container, **it also makes the unchecked exceptions (thrown from DAO methods) eligible for translation** into SpringDataAccessException.

3) The @Service annotation is also a specialization of the component annotation. It doesn’t currently provide any additional behavior over the @Component annotation, but it’s a good idea to use @Service over @Component in service-layer classes because **it specifies intent better**. Additionally, tool support and additional behavior might rely on it in the future.

4) @Controller annotation marks a class as a Spring Web MVC controller. It too is a @Component specialization, so beans marked with it are automatically imported into the DI container. When you add the @Controller annotation to a class, you can use another annotation i.e. @RequestMapping; to map URLs to instance methods of a class.

In real life, you will face very rare situations where you will need to use @Component annotation. Most of the time, you will using @Repository, @Service and @Controller annotations. @Component should be used when your class does not fall into either of three categories i.e. controller, manager and dao.

**If you want to define name of the bean with which they will be registered in DI container, you can pass the name in annotation itself e.g. @Service (“employeeManager”).**

## How to enable component scanning

Above four annotation will be scanned and configured only when they are scanned by DI container of spring framework. To enable this scanning, you will need to use “**context:component-scan**” tag in yourapplicationContext.xml file. e.g.

|  |  |
| --- | --- |
| 1  2  3 | <context:component-scan base-package="com.howtodoinjava.demo.service" />  <context:component-scan base-package="com.howtodoinjava.demo.dao" />  <context:component-scan base-package="com.howtodoinjava.demo.controller" /> |

The **context:component-scan** element requires a base-package attribute, which, as its name suggests, specifies a starting point for a recursive component search. You may not want to give your top package for scanning to spring, so you should declare three **component-scan** elements, each with a **base-package** attribute pointing to a different package.

When component-scan is declared, you no longer need to declare **context:annotation-config**, because autowiring is implicitly enabled when component scanning is enabled.

As I already said that you use @Repository, @Service and @Controller annotations over DAO, manager and controller classes. But in real life, at DAO and manager layer we often have separate classes and interfaces. Interface for defining the contract, and classes for defining the implementations of contracts. Where to use these annotations? Let’s find out.

Always use these annotations over concrete classes; not over interfaces.

Once you have these stereotype annotations on beans, you can directly use bean references defined inside concrete classes. Note the references are of type interfaces. Spring DI container is smart enough to inject the correct instance in this case.