# [Java Class Loader](#_How_ClassLoader_Works)

# [Object Oriented Features](#_Object_Oriented_Features:-)

# Inheritance

# Abstraction

# Polymorphism

# [Class, its loading and Object](#_Class_Loading:-)

# [Reflection](#_Reflection:-)

# [Stack Vs Heap](#_Difference_between_Stack)

# [Exceptions](#_Error_vs_Exception)

# [Collections](#_Collections)

# [Spring](#_Spring)

# [Spring Cache](#_Spring_Cache)

# [Micro Services in Spring](#_Micro_Services_with)

# [JPA](#_JPA)

# [Hibernate](#_Hibernate)

# [STRUTS 2](#_STRUTS_2)

# [Database](#_Databases)

# [AJAX](#_AJAX)

# [JQuery](#_JQUERY)

# [SOAP VS REST](#_SOAP_VS_REST)

# [Graph QL](#_SOAP_VS_REST)

# 

# [How ClassLoader Works in Java](http://javarevisited.blogspot.com/2012/12/how-classloader-works-in-java.html)

Java class loaders are used to load classes at runtime. ClassLoader in Java works on three principle:

1. delegation,
2. visibility and
3. Uniqueness.

Delegation principle forward request of class loading to parent class loader and only loads the class, if parent is not able to find or load class. Visibility principle allows child class loader to see all the classes loaded by parent ClassLoader, but parent class loader cannot see classes loaded by child. Uniqueness principle allows to load a class exactly once, which is basically achieved by delegation and ensures that child ClassLoader doesn't reload the class already loaded by parent. Correct understanding of class loader is must to resolve issues like NoClassDefFoundError in Java and java.lang.ClassNotFoundException, which are related to class loading.

In short here is the location from which Bootstrap, Extension and Application ClassLoader load Class

1) Bootstrap ClassLoader - JRE/lib/rt.jar

2) Extension ClassLoader - JRE/lib/ext or any directory denoted by java.ext.dirs

3) Application ClassLoader - CLASSPATH environment variable, -classpath or -cp option, Class-

**Delegation principles**

As discussed on [when a class is loaded and initialized in Java](http://javarevisited.blogspot.sg/2012/07/when-class-loading-initialization-java-example.html), a class is loaded in Java, when its needed. Suppose you have an application specific class called Abc.class, first request of loading this class will come to Application ClassLoader which will delegate to its parent Extension ClassLoader which further delegates to Primordial or Bootstrap class loader. Primordial will look for that class in rt.jar and since that class is not there, request comes to Extension class loader which looks on jre/lib/ext directory and tries to locate this class there, if class is found there than Extension class loader will load that class and Application class loader will never load that class but if its not loaded by extension class-loader than Application class loader loads it from [Classpath in Java](http://java67.blogspot.sg/2012/08/what-is-path-and-classpath-in-java-difference.html). Remember Classpath is used to load class files while [PATH](http://javarevisited.blogspot.ca/2011/10/how-to-set-path-for-java-unix-linux-and.html) is used to locate executable like javac or java command.

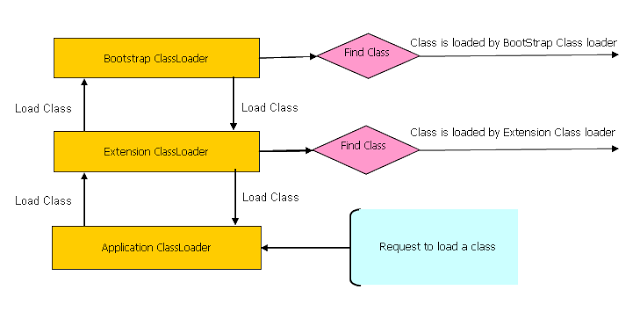
**Visibility Principle**

According to visibility principle, Child ClassLoader can see class loaded by Parent ClassLoader but vice-versa is not true. Which mean if class Abc is loaded by Application class loader than trying to load class ABC explicitly using extension

**Uniqueness Principle**

According to this principle a class loaded by Parent should not be loaded by Child ClassLoader again. Though its completely possible to write class loader which violates Delegation and Uniqueness principles and loads class by itself, its not something which is beneficial. You should follow all class loader principle while writing your own ClassLoader.

Class Loaders are mainly used in Applets.

[](http://1.bp.blogspot.com/-0gOWex7Pb2E/USTOh2K7zpI/AAAAAAAAAjc/_viQADzxrsk/s1600/Java+classloader+working.PNG)

## Object Oriented Features:-

**1.** Inheritance:- Mechanism of creating new class by acquiring the features from the existing class.(Single , multiple , multilevel , hierarchical)

**2.** Polymorphism:- Ability a class to have more than one form.

**3.** Encapsulation**:-**It is a feature by which we combine both data and eligible code that acts upon data into a single block of organization is called encapsulation. It can be achieved by creating a class and creating an object out of it.

Abstraction**:-** Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Method signature: **-**

**main** (String arg []), main () method takes string array as parameters. If command line arguments are supplied they will be placed in the array. JVM is meant for providing execution environment. It is not meant for accepting any value from Java Applications. Hence main should not return any value, hence it is **void**. To give access permissions to JVM, we mark it as **public**. **main**() is marked as **static** in order to allow JVM to call main() without the need of object creation.

Class and Object: **-**

class is a user defined data type. Class is the basis of encapsulation. Class is template or blueprint of proposed object. An instance of a class is **object.**

### Class Loading:-

**Static Class Loading**: Creating objects and instance using new keyword is known as static class loading. The retrieval of class definition and instantiation of the object is done at compile time.  
  
**Dynamic Class Loading**: Loading classes use Class.forName () method. Dynamic class loading is done when the name of the class is not known at compile time.

1. The static class loading is done through the new operator.  
2. Dynamic class loading is achieved through Run time type identification,also called as reflection.  
3. This is done with the help of the following methods getClass(); getName(); getDeclaredFields();   
4. Instance can also be created using forName() method. It loads the class into the current class memory.

## Reflection:-

The process of analyzing all the capabilities of a class at runtime is called reflection. We can know the details of the class variables and methods, constructors at runtime.

**public** **class** ReflectionTest {

**public** **static** **void** main(String args[]) **throws** ClassNotFoundException {

@SuppressWarnings("unchecked")

Class<Person> person = (Class<Person>) Class.*forName*("com.mits.reflection.Person");

//getFields() does not return private field

System.***out***.println("Fields : " + Arrays.*toString*(person.getFields()));

//getDeclaredFields() return both private and non private fields using reflection

System.***out***.println("Declared Fields : " + Arrays.*toString*(person.getDeclaredFields()));

//getDeclaredMethods() return both private and non private methods using reflection

System.***out***.println("Declared methods : " + Arrays.*toString*(person.getDeclaredMethods()));

**try** {

//accessing value of private field using reflection in Java

Person privateRyan = **new** Person("John" , "8989736353");

Field privateField = person.getDeclaredField("phone");

//this call allows private fields to be accessed via reflection

privateField.setAccessible(**true**);

//getting value of private field using reflection

String value = (String) privateField.get(privateRyan);

//print value of private field using reflection

System.***out***.println("private field: " + privateField + " value: " + value);

//accessing private method using reflection

Method privateMethod = person.getDeclaredMethod("call");

//making private method accessible using reflection

privateMethod.setAccessible(**true**);

//calling private method using reflection in java

privateMethod.invoke(privateRyan);

} **catch** (InvocationTargetException ex) {

Logger.*getLogger*(ReflectionTest.**class**.getName()).log(Level.***SEVERE***, **null**, ex);

} **catch** (NoSuchMethodException ex) {

Logger.*getLogger*(ReflectionTest.**class**.getName()).log(Level.***SEVERE***, **null**, ex);

} **catch** (IllegalArgumentException ex) {

Logger.*getLogger*(ReflectionTest.**class**.getName()).log(Level.***SEVERE***, **null**, ex);

} **catch** (IllegalAccessException ex) {

Logger.*getLogger*(ReflectionTest.**class**.getName()).log(Level.***SEVERE***, **null**, ex);

} **catch** (NoSuchFieldException ex) {

Logger.*getLogger*(ReflectionTest.**class**.getName()).log(Level.***SEVERE***, **null**, ex);

} **catch** (SecurityException ex) {

Logger.*getLogger*(ReflectionTest.**class**.getName()).log(Level.***SEVERE***, **null**, ex);

}

}

}

**class** Person{

**public** String name;

**private** String phone;

**public** Person(String name, String phone){

**this**.name = name;

**this**.phone = phone;

}

@SuppressWarnings("unused")

**private** **void** call(){

System.***out***.println("Calling " + **this**.name +" at " + **this**.phone);

}

**public** String getName(){

**return** name;

}

}

## IS-A and HAS-A

* In OO, the concept of **IS-A** is based on class inheritance or interface implementation. IS-A is a way of saying, "this thing is a type of that thing." You express the IS-A relationship in Java through the keywords extends (for *class* inheritance) and implements (for *interface* implementation).
* HAS-A relationships are based on usage, rather than inheritance. In other words, class A HAS-A B if code in class A has a reference to an instance of class B.

public class Animal { }

public class Horse extends Animal {

private Halter myHalter;} Horse has a Halter reference.

### Composition:-

They are absolutely different. Inheritance is an *"is-a"* relationship. Composition is a *"has-a"*. You do composition by having an instance of another class C as a field of your class, instead of extending C. A good example where composition would've been a lot better than inheritance is java.util.Stack, which currently extends java.util.Vector. This is now considered a blunder. A stack *"is-NOT-a"* vector; you should not be allowed to insert and remove elements arbitrarily. It should've been composition instead.

Unfortunately it's too late to rectify this design mistake, since changing the inheritance hierarchy now would break compatibility with existing code. *Had Stack used composition instead of inheritance, it can always be modified to use another data structure without violating the API*.

### Constructor: -

A constructor is a specialized method of class. A constructor name and class name is same, It does not have any return type.

Every time you make a new object, at least one constructor is invoked. Every class has a **constructor**, although if you don't create one explicitly, the compiler will build one for you. A **constructor is different from normal method in the areas – in name, in return type , type of calling, time of calling, frequency of calling, purpose of calling.**

Constructor chaining: -Calling one constructor of a class from within another constructor of same class is known as constructor chaining. ‘**this’** keyword is used for constructor chaining.

Class A{ A(int a, int b){ this();} A(){Sysout(“ Chaining”)};}

THIS-

This acts as an implicit reference for the current object. This is used within constructor of a class to call other constructor of the same class.

Super –

is used within the subclass constructor to call super class constructor. It is used to call superclass constructor, to call overridden methods from overriding methods, from subclass method to refer to subclass variable if its name and subclass variable name is matching.

### Binding-

Associating method call with method definition is called as binding. **Static binding** – if method is called during compilation. Actual object is not used for binding. **Dynamic binding**- It is a binding that happens at run time. Real object is used for binding.

* **Cohesion** refers to the degree in which a class has a single, well-defined role or responsibility.
* **Coupling** refers to the degree to which one class knows about or uses members of another class.
* **Singleton: -** If a class is instantiated only once, ie. Only one object is created.

Class B

{

private static B b;

Static

{b= new B();}

Public static B getInstance()

{

return b;

}

Private B(){ Sysout(“Object of class B is created”);} }

Access using B b1 = B.getInstance();

**Package: -** A package is grouping mechanisms using which related class files of an applications are grouped and made available to other applications or other parts of same application. Java.lang is the default package.

**Over*loaded* methods** let you reuse the same method name in a class, but with different arguments (and optionally, a different return type).

**Overridden methods: -** **when a method is reused in its subclass with the same name.**

Any time you have a class that inherits a method from a superclass, you have the opportunity to override the method (unless, as you learned earlier, the method is marked final). The key benefit of overriding is the ability to define behavior that's specific to a particular subclass type. The following example demonstrates a Horse subclass of Animal overriding the Animal version of the eat () method:

public class Animal {

public void eat() {

System.out.println("Generic Animal Eating Generically");

}

}

class Horse extends Animal {

public void eat() {

System.out.println("Horse eating hay, oats, "

+ "and horse treats");

}}

**Instance Variables**

Instance variables are defined inside the class, but outside of any method, and are only initialized when the class is instantiated. Instance variables are the fields that belong to each unique object.

**Local Variables**

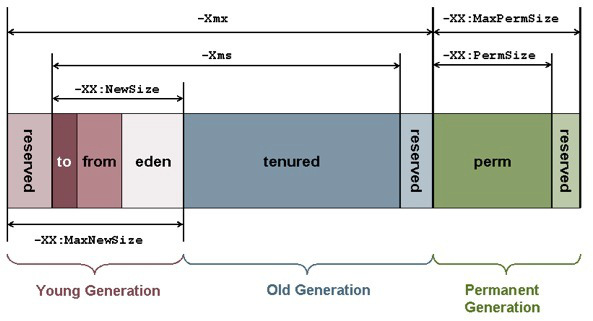
Local variables are variables declared within a method. That means the variable is not just initialized within the method, but also declared within the method. Just as the local variable starts its life inside the method, it's also destroyed when the method has completed. Local variables are always on the stack, not the heap.

■ Instance variables and objects live on the heap.

■ Local variables live on the stack.

### Garbage Collection: -

* When no reference is referring to an object then that object becomes unreachable. Automatic releasing of memory for unreachable object is called garbage collection (a thread in JVM).
* Garbage collection is a mechanism provided by Java Virtual Machine to reclaim heap space from objects which are eligible for Garbage collection.
* You as Java programmer cannot force garbage collection in Java; it will only trigger if JVM thinks it needs a garbage collection based on Java heap size.
* There are methods like System.gc() and Runtime.gc() which is used to send request of Garbage collection to JVM but it’s not guaranteed that garbage collection will happen.



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

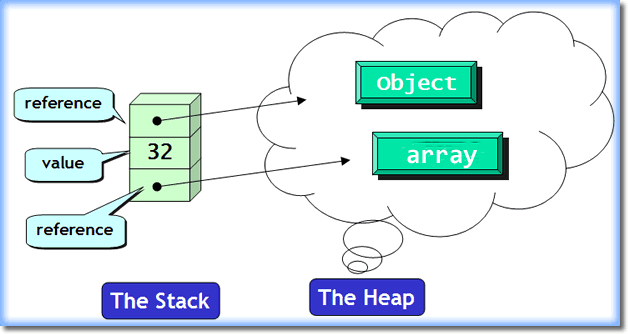
* JVM command line options –Xmx and -Xms is used to setup starting and max size for Java Heap. The ideal ratio of this parameter is either 1:1 or 1:1.5 based on my experience, for example, you can have either both –Xmx and –Xms as 1GB or –Xms 1.2 GB and 1.8 GB.

## **Difference between Stack vs Heap in Java**

Here are few differences between stack and heap memory in Java:

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

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

[](https://1.bp.blogspot.com/-gKWUcwIKWWU/VvPtKUAIFjI/AAAAAAAAFRc/WLCqWfSxlZ4ioocmBuFS3KaRhzs0I13OA/s1600/Difference+between+stack+and+heap+memory+in+Java.gif)

3) If there is no memory left in the stack for storing function call or local variable, JVM will throw java.lang.StackOverFlowError, while if there is no more heap space for creating an object, JVM will throw java.lang.OutOfMemoryError: Java Heap Space

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

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

### Static Variables and Methods

The static modifier is used to create variables and methods that will exist independently of any instances created for the class. All static members exist before you ever make a new instance of a class, and there will be only one copy of a static member regardless of the number of instances of that class. **All instances of a given class share the same value for any given static variable.**

Things you can mark as static:

■ Methods

■ Variables

■ Initialization blocks

**Static:-**Static is a keyword applied to methods, variables and can create blocks.

**Static method: -** also called class methods. Called by using classname.method name.

**Static block: -** As soon as a class is loaded into memory before any object is created whatever task we want to do, that task performing code we write in static block.

First the memory allocated for static variables is initialized then the static block is executed and the main () method is called after the static block execution.

**A *static* initialization block runs *once*, when the class is first loaded. An *instance* initialization block runs once *every time a new instance is created*.**

**Can we override static method in Java**

No, you cannot override static method in Java because [method overriding](http://java67.blogspot.sg/2012/08/what-is-method-overriding-in-java-example-tutorial.html) is based upon dynamic binding at runtime and static methods are bonded using [static binding](http://javarevisited.blogspot.sg/2012/03/what-is-static-and-dynamic-binding-in.html) at compile time. Though you can declare a method with same name and method signature in sub class which does look like you can override static method in Java but in reality that is method hiding. Java won't resolve method call at runtime and depending upon type of Object which is used to call [static method](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html), corresponding method will be called. It means if you use Parent class's type to call static method, original static will be called from patent class, on there other hand if you use Child class's type to call static method, method from child class will be called. In short you cannot override static method in Java. If you use Java IDE like [Eclipse](http://javarevisited.blogspot.sg/2011/02/how-to-setup-remote-debugging-in.html) or Netbeans, they will show warning that static method should be called using class name and not by using object becaues *static method cannot be overridden in Java*. We have confirmed that no, **we cannot override static method**, we can only hide static method in Java. Creating static method with same name and method signature is called Method hiding in Java.

### Abstract Class & Interface:-

* An abstract class can define both abstract and non-abstract methods, an interface can have only abstract methods. An abstract method is a method that's been *declared* (as abstract) but not *implemented*.
* When you create an interface, you're defining a contract for *what* a class can do, without saying anything about *how* the class will do it. An interface is a contract. Another way interfaces differ from abstract classes is that interfaces have very little flexibility in how the methods and variables defined in the interface are declared.
* A method can never, ever, ever be marked as both abstract and final, or both abstract,private.

■ All interface methods are implicitly public and abstract.

■ All variables defined in an interface must be public, static, and final.

■ Interface methods must not be static. Because interface methods are abstract, they cannot be marked final, strictfp, or native. (More on these modifiers later.)

■ An interface can *extend* one or more other interfaces.

■ An interface cannot extend anything but another interface.

■ An interface cannot implement another interface or class.

■ An interface must be declared with the keyword interface.

■ Interface types can be used polymorphically.

## **Why abstract class can have constructor in Java?**

Now if we say we cannot create instance of abstract class then why do Java adds constructor in abstract class. One of the reason which make sense is, when any class extend abstract class, constructor of sub class will invoke constructor of super class either implicitly or explicitly. This [chaining of constructors](http://javarevisited.blogspot.com/2012/12/constructor-chaining-in-java-calling-another-constructor.html) is one of the reason abstract class can have constructors in Java.

**Final:-**

* In other words, a final argument must keep the same value that the parameter had when it was passed into the method.
* final methods cannot be overridden in a subclass.
* A final class cannot be subclassed.
* final variables have the following properties:

final variables cannot be reinitialized once assigned a value.

final reference variables cannot refer to a different object once the object has been assigned to the final variable.

final reference variables must be initialized before the constructor

**Strictfp:-**

* strictfp as a class modifier, but even if you don't declare a class as strictfp, you can still declare an individual method as strictfp.

### Access Modifiers:-

This code compiles just fine, but you can also add modifiers before the class declaration. Modifiers fall into two categories:

■ Access modifiers: public, protected, private.

■ Non-access modifiers (including strictfp, final, and abstract).

Access control in Java is a little tricky because there are four access *controls* (levels of access) but only three access *modifiers*. The fourth access control level (called *default* or *package* access) is what you get when you don't use any of the three access modifiers. In other words, *every* class, method, and instance variable you declare has an access *control*, whether you explicitly type one or not. Although all four access *controls* (which means all three *modifiers*) work for most method and variable declarations, a class can be declared with only public or *default* access;

**When we say code from one class (class A) has access to another class (class B)?**

■ Create an *instance* of class B.

■ *Extend* class B (in other words, become a subclass of class B).

■ *Access* certain methods and variables within class B, depending on the access control of those methods and variables.

**Access specifier** is used for specifying accessibility/visibility mode of the members of the class.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Public** | **Protected** | ***Default*** | **Private** |
| From the same class | Yes | Yes | Yes | Yes |
| From any class in the same package | Yes | Yes | Yes | No |
| From a subclass in the same package | Yes | Yes | Yes | No |
| From a subclass outside the same package | Yes | Yes through inheritance | No | No |
| From any non-subclass class outside the package | Yes | No | No | No |

### Synchronized Methods

The synchronized keyword indicates that a method can be accessed by only one thread at a time. The synchronized modifier can be applied only to methods, not variables, not class, just methods.

### Native

The native modifier indicates that a method is implemented in platform-dependent code, native is a modifier (thus a reserved keyword) and can be applied only to *methods.* Note that a native method's body must be a semicolon (;)

### Transient and Serialization

If you mark an instance variable as **transient**, you're telling the JVM to skip (ignore) this variable when you attempt to serialize the object containing it. Serialization is one of the coolest features of Java; it lets you save (sometimes called "flatten") an object by writing its state (in other words, the value of its instance variables) to a special type of I/O stream. With **serialization** you can save an object to a file, or even ship it over a wire for deserializing at the other end, in another JVM.

### Volatile:-

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.

Volatile variable guarantees that a write will happen before any subsequent read: as stated

**Up-casting** is casting to a supertype, while **downcasting** is casting to a subtype. Supercasting is always allowed, but subcasting involves a type check and can throw a ClassCastException.

**Primitive Types Bits Bytes Minimum Range Maximum Range**

byte 8 1 -27 27-1

short 16 2 -215 215-1

int 32 4 -231 231-1

long 64 8 -263 263-1

float 32 4 n/a n/a

double 64 8 n/a n/a

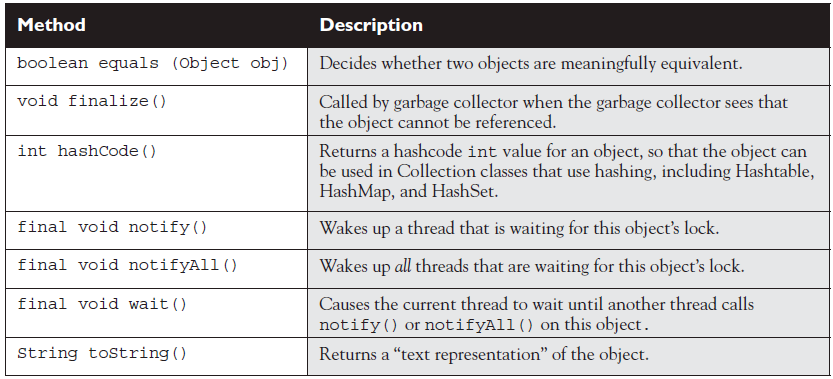
**Arrays:-**

In Java, arrays are objects that store multiple variables of the same type, or variables that are all subclasses of the same type. Arrays can hold either primitives or object references, but the array itself will always be an object on the heap, even if the array is declared to hold primitive elements.

### Marker Interface:

**Marker interface in Java** is interfaces with no field or methods or in simple word **empty interface in java is called marker interface**. Example of market interface is Serializable, Clonnable and Remote interface.

**Object Class**



**Autoboxing and Unboxing**

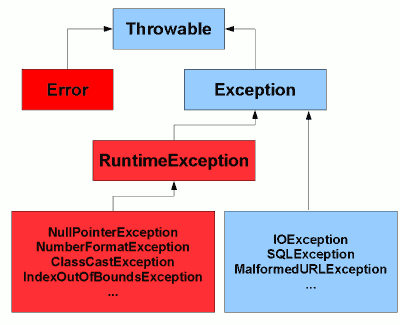
**Wrapper Classes**

* So that a null value is possible
* To include in a Collection
* To treat generically / polymorphically as an Object along with other Objects

### Exception Handling

**Checked exceptions:** A checked exception is an exception that occurs at the compile time, these are also called as compile time exceptions. These exceptions cannot simply be ignored at the time of compilation, the Programmer should take care of (handle) these exceptions.

**Unchecked exceptions:** An Unchecked exception is an exception that occurs at the time of execution, these are also called as Runtime Exceptions, these include programming bugs, such as logic errors or improper use of an API. runtime exceptions are ignored at the time of compilation.



**1) What is Exception in Java?**

Exception is Java’s way to convey both system and programming errors. In Java Exception feature is implemented by using class like Throwable, Exception, RuntimeException and keywords like throw, throws, try, catch and finally. All Exception are derived form Throwable class. Throwable further divides errors in two category one is java.lang.Exception and other is java.lang.Error. java.lang.Error deals with system errors like java.lang.StackOverFlowError or [Java.lang.OutOfMemoryError](http://javarevisited.blogspot.com/2011/09/javalangoutofmemoryerror-permgen-space.html) while Exception is mostly used to deal with programming mistakes, non-availability of requested resource etc.

## **Error vs Exception in Java**

Here is my list of notable difference between Error vs Exception in Java.

1) As I said earlier, Main difference on Error vs Exception is that Error is not meant to catch as even if you catch it you can not recover from it. For example during [OutOfMemoryError](http://javarevisited.blogspot.sg/2012/01/tomcat-javalangoutofmemoryerror-permgen.html), if you catch it you will get it again because GC may not be able to free memory in first place. On the other hand Exception can be caught and handled properly.

2) Error are often fatal in nature and recovery from Error is not possible which is different in case of Exception which may not be fatal in all cases.

**2) What is difference between Checked and Unchecked Exception in Java?**

Main difference between Checked and Unchecked Exception lies in there handling. Checked Exception requires to be handled at compile time using try, catch and finally keywords or else compiler will flag error. This is not a requirement for Unchecked Exceptions. Also all exceptions derived from java.lang.Exception classes are checked exception, exception those which extends RuntimeException, these are known as unchecked exception in Java. You can also check next article for [more differences between Checked and Unchecked Exception](http://javarevisited.blogspot.com/2011/12/checked-vs-unchecked-exception-in-java.html).

**3) What is similarity between NullPointerException and ArrayIndexOutOfBoundException in Java?**

By the way answer of this interview question is both of them are example of unchecked exception and derived form RuntimeException. This question also opens door for difference of array in Java and C programming language, as arrays in C are unbounded and never throw ArrayIndexOutOfBoundException.

**4) What best practices you follow while doing Exception handling in Java?**

Since exception handling is crucial part of project design and good knowledge of this is desirable. There are lot of best practices, which can help to make your code robust and flexible at same time, here are few of them:

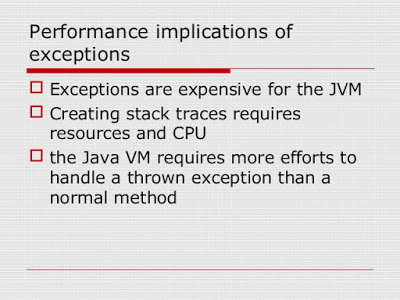
1) Returning boolean instead of returning null to avoid NullPointerException at callers end. Since NPE is most infamous of all Java exceptions, there are lot of techniques and [coding best practices to minimize NullPointerException](http://javarevisited.blogspot.com/2013/05/ava-tips-and-best-practices-to-avoid-nullpointerexception-program-application.html). You can check that link for some specific examples.

2) Non empty catch blocks. Empty catch blocks are considered as one of the bad practices in Exception handling because they just ate Exception without any clue, at bare minimum print stack trace but you should do alternative operation which make sense or defined by requirements.

3) Prefer Unchecked exception over checked until you have a very good reason of not to do so. it improves readability of code by removing boiler plate exception handling code

4) Never let your database Exception flowing till client error. since most of application deal with database and [SQLException](http://javarevisited.blogspot.com/2012/01/javasqlsqlexception-invalid-column.html) is a checked Exception in Java you should consider handling any database related errors in DAO layer of your application and only returning alternative value or something meaningful RuntimeException which client can understand and take action.

5) calling close() methods for connections, statements, and streams on finally block in Java.

[](http://2.bp.blogspot.com/-Xya2SK0Iva0/VXhOeWE7svI/AAAAAAAAC-8/FzhFhLejgQY/s1600/Java+Exception+Performance+impact.jpg)

**5) Why do you think Checked Exception exists in Java, since we can also convey error using RuntimeException ?**

One of the reason I see is that its a design decision, which is influenced by experience in programming language prior to Java e.g. C++. Most of checked exceptions are in java.io package, which make sense because if you request any system resource and its not available, than a robust program must be able to handle that situation gracefully. By declaring IOException as checked Exception, Java ensures that your provide that gracefully exception handling. Another possible reason could be to ensuring that system resources like file descriptors, which are limited in numbers, should be released as soon as you are done with that using catch or finally block.

**6) What is difference between throw and throws keyword in Java?**

throw and throws keyword may look quite similar, especially if you are new to Java programming and haven't seen much of it. Though they are similar in terms that both are used in Exception handling, they are different on how and where they are used in code. throws keyword is used in method signature to declare which checked exception method can throw, you can also declare unchecked exception, but that is not mandatory by compiler. This signifies lot of things like method is not going to handle Exception instead its throwing it, if method throws checked Exception then caller should provide compile time exception handling etc. On the other hand throw keyword is actually used to throw any Exception. Syntactically you can throw any Throwable (i.e. Throwable or any class derived from Throwable) , throw keyword transfers control of execution to caller so it can be used in place of return keyword. Most common example of using throw in place of return is throwing UnSupportedOperationException from an empty method as shown below :

private static void show() {

throw new UnsupportedOperationException("Not yet implemented");

}

**7) What is Exception chaining in Java?**

Exception chaining is a popular exception handling concept in Java, where another exception is thrown in response of an exception and creating a chain of Exceptions. This technique mostly used to wrap a checked exception into an unchecked or RuntimeException. By the way if you are throwing new exception due to another exception then always include original exception so that handler code can access root cause by using methods like getCause() and initCause().

**8) Have you written your own custom Exception in Java? How do you do that?**

Of course most of us has written custom or business Exceptions like AccountNotFoundExcepiton. This can be used for sophisticated and precise exception handling with tweak involved in whether you would choose a checked or unchecked exception. By creating a specific exception for specific case, you also gives lot of options to caller to deal with them elegantly. I always prefer to have a precise exception than a general exception. Though creating lots of specific exceptions quickly increase number of classes in your project, maintaining a practical balance between specific and general exceptions are key to success.

**9) What changes has been introduced in JDK7 related to Exception handling in Java ?**

A relatively new and recent Exception interview question in Java. JDK7 has introduced two major feature which is related to Error and Exception handling, one is ability to handle [multiple exception in one catch block](http://javarevisited.blogspot.com/2011/07/jdk7-multi-cache-block-example-tutorial.html), popularly known as multi cache block and other is [ARM blocks in Java 7](http://javarevisited.blogspot.sg/2011/09/arm-automatic-resource-management-in.html) for automatic resource management, also known as try with resource. Both of these feature can certainly help to reduce boiler plate code required for handling checked exceptions in Java and significantly improves readability of code. Knowledge of this feature, not only helps to write better error and exception code in Java, but also helps to do well during interviews. I also recommend reading Java 7 Recipes book to get more insight on useful features introduced in Java 7, including these two.

**10) Have you faced OutOfMemoryError in Java? How did you solved that?**

This Java Error interview questions is mostly asked on senior level Java interviews and here interviewer is interested on your approach to tackle dangerous OutOfMemoryError. Admit it we always face this error no matter which kind of project you are working so if you say no it doesn't go very well with interviewer. I suggest even if you are not familiar or not faced it in reality but have 3 to 4 years of experience in Java, be prepare for it. At the same time, this is also a chance to impress interviewer by showing your advanced technical knowledge related to finding memory leaks, profiling and debugging. I have noticed that these skills almost always creates a positive impression. You can also see my post on [how to fix java.lang.OutOfMemoryError](http://javarevisited.blogspot.com/2011/09/javalangoutofmemoryerror-permgen-space.html) for more detail on this topic.

**11) Does code form finally executes if method returns before finally block or JVM exits ?**

This Java exception interview question can also be asked in code format, where given a code with System.exit() in try block and something in finally block. It’s worth knowing that, finally block in Java executes even when return keyword is used in try block. Only time they don’t execute is when you call JVM to exit by executing System.exit(0)from try block in Java.

**12) What is difference in final, finalize and finally keyword in Java?**

final and finally are keyword, while finalize is method. final keyword is very useful for creating an [Immutable class in Java](http://javarevisited.blogspot.com/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html) By making a class final, we prevent it from being extended, similarly by making a method final, we prevent it from being overridden. On the other hand, finalize() method is called by garbage collector, before that object is collected, but this is not guaranteed by Java specification. finally keyword is the only one which is related to error and exception handling and you should always have finally block in production code for closing connection and resources. See [here](http://javarevisited.blogspot.com/2012/11/difference-between-final-finally-and-finalize-java.html) for more detailed answer of this question.

**IOStreams**

### Serialization:-

Object Serialization in Java is a process used to convert Object into a binary format which can be persisted into disk or sent over network to any other running [Java virtual machine](http://javarevisited.blogspot.sg/2011/11/hotspot-jvm-options-java-examples.html); the reverse process of creating object from binary stream is called deserialization in Java. Java provides Serialization API for serializing and deserializing object which includes java.io.Serializable, java.io.Externalizable, ObjectInputStream and ObjectOutputStream etc. Java programmers are free to use default Serialization mechanism which Java uses based upon structure of class but they are also free to use there own custom binary format, which is often advised as Serialization best practice, Because serialized binary format becomes part of Class's exported API and it can potentially break [Encapsulation in Java](http://javarevisited.blogspot.sg/2012/03/what-is-encapsulation-in-java-and-oops.html) provided by private and [package-private fields](http://javarevisited.blogspot.sg/2012/05/how-to-access-private-field-and-method.html).

How to make a Java class Serializable?

Making a class Serializable in Java is very easy, Your Java class just needs to implements java.io.Serializable interface and JVM will take care of serializing object in default format. Decision to making a [Class](http://javarevisited.blogspot.sg/2011/10/class-in-java-programming-general.html) Serializable should be taken concisely because though near term cost of making a Class Serializable is low, long term cost is substantial and it can potentially limit your ability to further modify and change its implementation because like any public API, serialized form of an object becomes part of public API and when you change structure of your class by implementing addition interface, adding or removing any field can potentially break default serialization, this can be minimized by using a custom binary format but still requires lot of effort to ensure backward compatibility. One example of How Serialization can put constraints on your ability to change class is SerialVersionUID. If you don't explicitly declare SerialVersionUID then JVM generates its based upon structure of class which depends upon interfaces a class implements and several other factors which is subject to change. Suppose you implement another interface than [JVM](http://javarevisited.blogspot.sg/2011/12/jre-jvm-jdk-jit-in-java-programming.html) will generate a different SerialVersionUID for new version of class files and when you try to load old object object serialized by old version of your program you will get InvalidClassException.

### 1) What is the difference between Serializable and Externalizable interface in Java?

Externalizable provides us writeExternal() and readExternal() method which gives us flexibility to control java serialization mechanism instead of relying on Java's default serialization. Correct implementation of Externalizable interface can [improve performance of application](http://javarevisited.blogspot.sg/2012/01/improve-performance-java-database.html) drastically.

### 2) How many methods Serializable has? If no method then what is the purpose of Serializable interface?

Serializable interface exists in java.io package and forms core of java serialization mechanism. It doesn't have any method and also called [Marker Interface in Java](http://javarevisited.blogspot.sg/2012/01/what-is-marker-interfaces-in-java-and.html). When your class implements java.io.Serializable interface it becomes Serializable in Java and gives compiler an indication that use Java Serialization mechanism to serialize this object.

### 3) What is serialVersionUID? What would happen if you don't define this?

SerialVersionUID is an ID which is stamped on object when it get serialized usually hashcode of object, you can use tool serialver to see serialVersionUID of a serialized object . SerialVersionUID is used for version control of object. you can specify serialVersionUID in your [class file](http://javarevisited.blogspot.sg/2012/05/10-points-about-class-file-in-java.html) also. Consequence of not specifying serialVersionUID is that when you add or modify any field in class then already serialized class will not be able to recover because serialVersionUID generated for new class and for old serialized object will be different. Java serialization process relies on correct serialVersionUID for recovering state of serialized object and throws java.io.InvalidClassException in case of serialVersionUID mismatch, to learn more about serialversionuid see this [article](http://javarevisited.blogspot.sg/2014/05/why-use-serialversionuid-inside-serializable-class-in-java.html).

### 4) While serializing you want some of the members not to serialize? How do you achieve it?

This is sometime also asked as what is the use of [transient variable](http://javarevisited.blogspot.sg/2011/09/transient-keyword-variable-in-java.html), does transient and [static variable](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html) gets serialized or not etc. so if you don't want any field to be part of object's state then declare it either static or transient based on your need and it will not be included during Java serialization process.

### Q 5) What will happen if one of the members in the class doesn't implement Serializable interface?

If you try to serialize an object of a class which implements Serializable, but the object includes a reference to an non- Serializable class then a ‘NotSerializableException’ will be thrown at runtime and this is why I always put a *SerializableAlert* (comment section in my code) , one of the [code comment best practices](http://javarevisited.blogspot.sg/2011/08/code-comments-java-best-practices.html), to instruct developer to remember this fact while adding a new field in a Serializable class.

### 6) If a class is Serializable but its super class in not, what will be the state of the instance variables inherited from super class after deserialization?

Java serialization process only continues in object hierarchy till the class is Serializable i.e. implements Serializable [interface in Java](http://javarevisited.blogspot.sg/2012/04/10-points-on-interface-in-java-with.html) and values of the instance variables inherited from super class will be initialized by calling constructor of Non-Serializable Super class during deserialization process. Once the [constructor chaining](http://javarevisited.blogspot.sg/2012/01/what-is-constructor-overloading-in-java.html) will started it wouldn't be possible to stop that , hence even if classes higher in hierarchy implements Serializable interface , there constructor will be executed. As you see from the statement this Serialization interview question looks very tricky and tough but if you are familiar with key concepts its not that difficult.

### Question 7) Can you Customize Serialization process or can you override default Serialization process in Java?

The answer is yes you can. We all know that for serializing an object ObjectOutputStream.writeObject (saveThisobject) is invoked and for reading object ObjectInputStream.readObject() is invoked but there is one more thing which Java Virtual Machine provides you is to define these two method in your class. If you define these two methods in your class then JVM will invoke these two methods instead of applying default serialization mechanism. You can customize behavior of object serialization and deserialization here by doing any kind of pre or post processing task. Important point to note is making these methods private to avoid being inherited, [overridden or overloaded](http://javarevisited.blogspot.sg/2011/12/method-overloading-vs-method-overriding.html). Since only Java Virtual Machine can call private method integrity of your class will remain and Java Serialization will work as normal. In my opinion this is one of the best question one can ask in any Java Serialization interview, a good follow-up question is why should you provide custom serialized form for your object?

### Question 8) Suppose super class of a new class implement Serializable interface, how can you avoid new class to being serialized?

If Super Class of a Class already implements Serializable interface in Java then its already Serializable in Java, since you can not unimplemented an interface its not really possible to make it Non Serializable class but yes there is a way to avoid serialization of new class. To avoid Java serialization you need to implement writeObject() and readObject() method in your Class and need to throw NotSerializableException from those method. This is another benefit of customizing java serialization process as described in above [Serialization interview question](http://javarevisited.blogspot.sg/2011/04/top-10-java-serialization-interview.html) and normally it asked as follow-up question as interview progresses.

### Question 9) Which methods are used during Serialization and DeSerialization process in Java?

This is very common interview question in Serialization basically interviewer is trying to know; Whether you are familiar with usage of readObject(), writeObject(), readExternal() and writeExternal() or not. Java Serialization is done by java.io.ObjectOutputStream class. That class is a filter stream which is wrapped around a lower-level byte stream to handle the serialization mechanism. To store any object via serialization mechanism we call ObjectOutputStream.writeObject(saveThisobject) and to deserialize that object we call ObjectInputStream.readObject() method. Call to writeObject() method trigger serialization process in java. one important thing to note about readObject() method is that it is used to read bytes from the persistence and to create object from those bytes and its return an [Object](http://javarevisited.blogspot.sg/2012/03/10-object-oriented-design-principles.html) which needs to be type cast to correct type.

### Question 10) Suppose you have a class which you serialized it and stored in persistence and later modified that class to add a new field. What will happen if you deserialize the object already serialized?

It depends on whether class has its own serialVersionUID or not. As we know from above question that if we don't provide serialVersionUID in our code java compiler will generate it and normally it’s [equal to hashCode of object](http://javarevisited.blogspot.sg/2011/02/how-to-write-equals-method-in-java.html). by adding any new field there is chance that new serialVersionUID generated for that class version is not the same of already serialized object and in this case Java Serialization API will [throw](http://javarevisited.blogspot.sg/2012/02/difference-between-throw-and-throws-in.html) java.io.InvalidClassException and this is the reason its recommended to have your own serialVersionUID in code and make sure to keep it same always for a single class.

### 11) What are the compatible changes and incompatible changes in Java Serialization Mechanism?

The real challenge lies with change in class structure by adding any field, method or removing any field or method is that with already serialized object. As per Java Serialization specification adding any field or method comes under compatible change and changing class hierarchy or UN-implementing Serializable interfaces some under non compatible changes. For complete list of compatible and non compatible changes I would advise reading Java serialization specification.

### 12) Can we transfer a Serialized object vie network?

Yes you can transfer a Serialized object via network because Java serialized object remains in form of bytes which can be transmitter via network. You can also store serialized object in Disk or database as Blob.

### 13) Which kind of variables is not serialized during Java Serialization?

This question asked sometime differently but the purpose is same whether Java developer knows specifics about [static and transient variable](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html) or not. Since static variables belong to the class and not to an object they are not the part of the state of object so they are not saved during Java Serialization process. As Java Serialization only persist state of object and not object itself. Transient variables are also not included in java serialization process and are not the part of the object’s serialized state. After this question sometime interviewer ask a follow-up if you don't store values of these variables then what would be value of these variable once you deserialize and recreate those object.

### MultiThreading

## **Difference between start and run in Java Thread**

So what is difference between start and run method? Main difference is that when program calls start() method a **new Thread** is created and code inside run() method is executed in new Thread while if you call run() method directly **no new Thread is created** and code inside run() will execute on **current Thread**.

**What are differences between wait and sleep method in java?**

Only major difference is wait release the lock or monitor while sleep doesn't release any lock or monitor while waiting. Wait is used for inter-thread communication while sleep is used to introduce pause on execution.

**What is the difference between Runnable and Callable in Java?**  
Both Runnable and Callable represent task which is intended to be executed in a separate thread. Runnable is there from JDK 1.0 while Callable was added on JDK 1.5. Main difference between these two is that Callable's call() method can return value and throw Exception, which was not possible with Runnable's run() method. Callable return Future object, which can hold the result of computation.

**Race condition** are cause of some subtle programming bugs when Java programs are exposed to concurrent execution environment. As the name suggests, a race condition occurs due to race between multiple threads, if a thread which is supposed to execute first lost the race and executed second, behaviour of code changes, which surface as non-deterministic bugs.

**How to stop a thread in Java?**   
I always said that Java provides rich APIs for everything but ironically Java doesn't provide a sure shot way of stopping thread. There was some control methods in JDK 1.0 e.g. stop(), suspend() and resume() which was deprecated in later releases due to potential deadlock threats, from then Java API designers has not made any effort to provide a consistent, thread-safe and elegant way to stop threads.

**What happens when an Exception occurs in a thread?**  
In simple words, If not caught thread will die, if an uncaught exception handler is registered then it will get a call back.

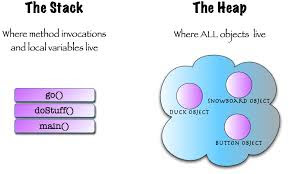
**How do you share data between two thread in Java?**   
You can share data between threads by using shared object, or concurrent data structure like BlockingQueue. See this tutorial to learn  [inter-thread communication in Java](http://javarevisited.blogspot.sg/2013/12/inter-thread-communication-in-java-wait-notify-example.html). It implements Producer consumer pattern using wait and notify methods, which involves sharing objects between two threads.

**16) What is the difference between notify and notifyAll in Java?**   
since multiple threads can wait on single monitor lock, Java API designer provides method to inform only one of them or all of them, once waiting condition changes, but they provide half implementation. There notify() method doesn't provide any way to choose a particular thread, that's why its only useful when you know that there is only one thread is waiting. On the other hand, notifyAll() sends notification to all threads and allows them to compete for locks, which ensures that at-least one thread will proceed further. See my [blog post](http://javarevisited.blogspot.com/2012/10/difference-between-notify-and-notifyall-java-example.html) on similar topic for a more detailed answer and code example.  
**17) Why wait, notify and notifyAll are not inside thread class?**   
This is a design related question, which checks what candidate thinks about existing system or does he ever thought of something which is so common but looks in-appropriate at first. In order to answer this question, you have to give some reasons why it make sense for these three method to be in Object class, and why not on Thread class. One reason which is obvious is that Java provides lock at object level not at thread level. Every object has lock, which is acquired by thread. Now if thread needs to wait for certain lock it make sense to call wait() on that object rather than on that thread. Had wait() method declared on Thread class, it was not clear that for which lock thread was waiting. In short, since wait, notify and notifyAll operate at lock level, it make sense to defined it on object class because lock belongs to object. You can also see this [article](http://javarevisited.blogspot.sg/2012/02/why-wait-notify-and-notifyall-is.html) for more elaborate answer of this question.  
  
**18) What is ThreadLocal variable in Java?**   
ThreadLocal variables are special kind of variable available to Java programmer. Just like instance variable is per instance, ThreadLocal variable is per thread. It's a nice way to achieve thread-safety of expensive-to-create objects, for example you can make SimpleDateFormat thread-safe using ThreadLocal. Since that class is expensive, its not good to use it in local scope, which requires separate instance on each invocation. By providing each thread their own copy, you shoot two birds with one arrow. First, you reduce number of instance of expensive object by reusing fixed number of instances, and Second, you achieve thread-safety without paying cost of synchronization or immutability. Another good example of thread local variable is ThreadLocalRandom class, which reduces number of instances of expensive-to-create Random object in multi-threading environment. See this [answer](http://javarevisited.blogspot.sg/2012/05/how-to-use-threadlocal-in-java-benefits.html) to learn more about thread local variables in Java.  
  
**19) What is FutureTask in Java?**   
FutureTask represents a cancellable asynchronous computation in concurrent Java application. This class provides a base implementation of Future, with methods to start and cancel a computation, query to see if the computation is complete, and retrieve the result of the computation. The result can only be retrieved when the computation has completed; the get methods will block if the computation has not yet completed. A FutureTask object can be used to wrap a Callable or Runnable object. Since FutureTask also implements Runnable, it can be submitted to an Executor for execution.  
  
**20) What is the difference between the interrupted() and isInterrupted() method in Java?**   
Main difference between interrupted() and isInterrupted() is that former clears the interrupt status while later does not. The interrupt mechanism in Java multi-threading is implemented using an internal flag known as the interrupt status. Interrupting a thread by calling Thread.interrupt() sets this flag. When interrupted thread checks for an interrupt by invoking the [static method](http://java67.blogspot.com/2012/11/what-is-static-class-variable-method.html) Thread.interrupted(), interrupt status is cleared. The non-static isInterrupted() method, which is used by one thread to query the interrupt status of another, does not change the interrupt status flag. By convention, any method that exits by throwing an InterruptedException clears interrupt status when it does so. However, it's always possible that interrupt status will immediately be set again, by another thread invoking interrupt

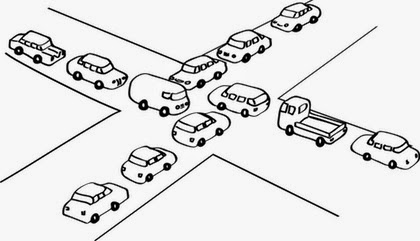
**21) Why wait and notify method are called from synchronized block?**   
Main reason for calling wait and notify method from either synchronized block or method is that it made mandatory by Java API. If you don't call them from synchronized context, your code will throw IllegalMonitorStateException. A more subtle reason is to avoid the race condition between wait and notify calls. To learn more about this, check my similarly titled post [here](http://javarevisited.blogspot.com/2011/05/wait-notify-and-notifyall-in-java.html).

**22) Why should you check condition for waiting in a loop?**   
Its possible for a waiting thread to receive false alerts and spurious wake up calls, if it doesn't check the waiting condition in loop, it will simply exit even if condition is not met. As such, when a waiting thread wakes up, it cannot assume that the state it was waiting for is still valid. It may have been valid in the past, but the state may have been changed after the notify() method was called and before the waiting thread woke up. That's why it always better to call wait() method from loop, you can even create template for calling wait and notify in Eclipse. To learn more about this question, I would recommend you to read Effective Java items on thread and synchronization.  
**23) What is the difference between synchronized and concurrent collection in Java?**   
Though both synchronized and concurrent collection provides thread-safe collection suitable for multi-threaded and concurrent access, later is more scalable than former. Before Java 1.5, Java programmers only had synchronized collection which becomes source of contention if multiple thread access them concurrently, which hampers scalability of system. Java 5 introduced concurrent collections like ConcurrentHashMap, which not only provides thread-safety but also improves scalability by using modern techniques like lock stripping and partitioning internal table. See this [answer](http://javarevisited.blogspot.com/2010/10/what-is-difference-between-synchronized.html) for more differences between synchronized and concurrent collection in Java.

**24) What is the difference between Stack and Heap in Java?**   
Stack is a memory area which is closely associated with threads. To answer this question, both stack and heap are specific memories in Java application. Each thread has their own stack, which is used to store local variables, method parameters and call stack. Variable stored in one Thread's stack is not visible to other. On another hand, the heap is a common memory area which is shared by all threads. Objects whether local or at any level is created inside heap. To improve performance thread tends to cache values from heap into their stack, which can create problems if that variable is modified by more than one thread, this is where volatile variables come into the picture. volatile suggest threads read the value of variable always from main memory. See this [article](http://javarevisited.blogspot.com/2013/01/difference-between-stack-and-heap-java.html) for learning more about stack and heap in Java to answer this question in greater detail.

[](https://3.bp.blogspot.com/-vJvHCwr7ozY/VuBB4nlNpkI/AAAAAAAAFCk/8mqWs5unUK4/s1600/Heap+vs+Stack+in+Java.jpg)

**25) What is thread pool? Why should you thread pool in Java?**(answer)  
Creating thread is expensive in terms of time and resource. If you create thread at time of request processing it will slow down your response time, also there is only a limited number of threads a process can create. To avoid both of these issues, a pool of thread is created when application starts-up and threads are reused for request processing. This pool of thread is known as "thread pool" and threads are known as worker thread. From JDK 1.5 release, Java API provides Executor framework, which allows you to create different types of thread pools e.g. single thread pool, which process one task at a time, fixed thread pool (a pool of fixed number of threads) or cached thread pool (an expandable thread pool suitable for applications with many short lived tasks). See this [article](http://javarevisited.blogspot.com/2013/07/how-to-create-thread-pools-in-java-executors-framework-example-tutorial.html) to learn more about thread pools in Java to prepare detailed answer of this question.  
**26) Write code to solve Producer Consumer problem in Java?**   
Most of the threading problem you solved in the real world are of the category of Producer consumer pattern, where one thread is producing task and another thread is consuming that. You must know how to do inter thread communication to solve this problem. At the lowest level, you can use wait and notify to solve this problem, and at a high level, you can leverage Semaphore or BlockingQueue to implement Producer consumer pattern, as shown in this [tutorial](http://javarevisited.blogspot.sg/2012/02/producer-consumer-design-pattern-with.html).  
**27) How do you avoid deadlock in Java? Write Code?**

[](http://4.bp.blogspot.com/-m2IldPcxiJI/U6-Zwvkdd1I/AAAAAAAABns/-zHIHjzM3nM/s1600/deadlock+in+Java.jpg)

Deadlock is a condition in which two threads wait for each other to take action which allows them to move further. It's a serious issue because when it happen your program hangs and doesn't do the task it is intended for. In order for deadlock to happen, following four conditions must be true:

* **Mutual Exclusion :** At least one resource must be held in a non-shareable mode. Only one process can use the resource at any given instant of time.
* **Hold and Wait:** A process is currently holding, at least, one resource and requesting additional resources which are being held by other processes.
* **No Pre-emption:** The operating system must not de-allocate resources once they have been allocated; they must be released by the holding process voluntarily.
* **Circular Wait:** A process must be waiting for a resource which is being held by another process, which in turn is waiting for the first process to release the resource.

The easiest way to avoid deadlock is to prevent *Circular wai*t, and this can be done by acquiring locks in a particular order and releasing them in reverse order so that a thread can only proceed to acquire a lock if it held the other one. Check this [tutorial](http://javarevisited.blogspot.com/2010/10/what-is-deadlock-in-java-how-to-fix-it.html) for the actual code example and detailed discussion on techniques for avoiding deadlock in Java.  
**28) What is the difference between livelock and deadlock in Java?**(answer)  
This question is extension of previous interview question. A livelock is similar to a deadlock, except that the states of the threads or processes involved in the livelock constantly change with regard to one another, without any one progressing further. Livelock is a special case of resource starvation. A real-world example of livelock occurs when two people meet in a narrow corridor, and each tries to be polite by moving aside to let the other pass, but they end up swaying from side to side without making any progress because they both repeatedly move the same way at the same time. In short, the main difference between livelock and deadlock is that in former state of process change but no progress is made.  
**29) How do you check if a Thread holds a lock or not?**(answer)  
I didn't even know that you can check if a Thread already holds lock before this question hits me in a telephonic round of Java interview. There is a method called holdsLock() on java.lang.Thread, it returns true if and only if the current thread holds the monitor lock on the specified object. You can also check this article for the more detailed [answer](http://javarevisited.blogspot.com/2010/10/how-to-check-if-thread-has-lock-on.html).  
**30) How do you take thread dump in Java?**([answer](http://javarevisited.blogspot.com/2011/07/java-multi-threading-interview.html))  
There are multiple ways to take thread dump of Java process depending upon operating system. When you take thread dump, JVM dumps state of all threads in log files or standard error console. In windows you can use Ctrl + Break key combination to take thread dump, on Linux you can use kill -3 command for same. You can also use a tool called jstack for taking thread dump, it operate on process id, which can be found using another tool called jps.  
**31) Which JVM parameter is used to control stack size of a thread?**(answer)  
This is the simple one, -Xss parameter is used to control stack size of Thread in Java. You can see this [list of JVM options](http://javarevisited.blogspot.com/2011/11/hotspot-jvm-options-java-examples.html) to learn more about this parameter.  
**32) What is the difference between synchronized and ReentrantLock in Java?**(answer)  
There were days when the only way to provide mutual exclusion in Java was via synchronized keyword, but it has several shortcomings e.g. you can not extend lock beyond a method or block boundary, you can not give up trying for a lock etc. Java 5 solves this problem by providing more sophisticated control via Lock interface. ReentrantLock is a common implementation of Lock interface and provides re-entrant mutual exclusion Lock with the same basic behavior and semantics as the implicit monitor lock accessed using synchronized methods and statements, but with extended capabilities. See [this article](http://javarevisited.blogspot.com/2013/03/reentrantlock-example-in-java-synchronized-difference-vs-lock.html) learn about those capabilities and some more differences between synchronized vs ReentrantLock in Java.  
**33) There are three threads T1, T2, and T3? How do you ensure sequence T1, T2, T3 in Java?**(answer)  
Sequencing in multi-threading can be achieved by different means but you can simply use the join() method of thread class to start a thread when another one has finished its execution. To ensure three threads execute you need to start the last one first e.g. T3 and then call join methods in reverse order e.g. T3 calls T2. join and T2 calls T1.join, these ways T1 will finish first and T3 will finish last. To learn more about join method, see this [tutorial](http://javarevisited.blogspot.sg/2013/02/how-to-join-multiple-threads-in-java-example-tutorial.html).  
**34) What does yield method of Thread class do?**(answer)  
Yield method is one way to request current thread to relinquish CPU so that other thread can get a chance to execute. Yield is a static method and only guarantees that current thread will relinquish the CPU but doesn't say anything about which other thread will get CPU. Its possible for the same thread to get CPU back and start its execution again. See this [article](http://java67.blogspot.sg/2012/08/difference-between-yield-and-wait.html) to learn more about yield method and to answer this question better.  
**35) What is the concurrency level of ConcurrentHashMap in Java?**(answer)  
ConcurrentHashMap achieves it's scalability and thread-safety by partitioning actual map into a number of sections. This partitioning is achieved using concurrency level. Its optional parameter of ConcurrentHashMap constructor and it's default value is 16. The table is internally partitioned to try to permit the indicated number of concurrent updates without contention. To learn more about concurrency level and internal resizing, see my post [How ConcurrentHashMap works in Java](http://javarevisited.blogspot.com/2013/02/concurrenthashmap-in-java-example-tutorial-working.html).  
**36) What is Semaphore in Java?** (answer)  
Semaphore in Java is a new kind of synchronizer. It's a counting semaphore. Conceptually, a semaphore maintains a set of permits. Each acquire() blocks if necessary until a permit is available, and then takes it. Each release() adds a permit, potentially releasing a blocking acquirer. However, no actual permit objects are used; the Semaphore just keeps a count of the number available and acts accordingly. Semaphore is used to protect an expensive resource which is available in fixed number e.g. database connection in the pool. See this [article](http://javarevisited.blogspot.com/2012/05/counting-semaphore-example-in-java-5.html) to learn more about counting Semaphore in Java.  
**37) What happens if you submit a task when the queue of the thread pool is already filled?**(answer)  
This is another tricky question on my list. Many programmers will think that it will block until a task is cleared but its true. ThreadPoolExecutor's submit() method throws RejectedExecutionException if the task cannot be scheduled for execution.  
**38) What is the difference between the submit() and execute() method thread pool in Java?**(answer)  
Both methods are ways to submit a task to thread pools but there is a slight difference between them. execute(Runnable command) is defined in Executor interface and executes given task in future, but more importantly, it does not return anything. Its return type is void. On other hand submit() is an overloaded method, it can take either Runnable or Callable task and can return Future object which can hold the pending result of computation. This method is defined on ExecutorService interface, which extends Executor interface, and every other thread pool class e.g. ThreadPoolExecutor or ScheduledThreadPoolExecutor gets these methods. To learn more about thread pools you can check this [article](http://javarevisited.blogspot.sg/2013/07/how-to-create-thread-pools-in-java-executors-framework-example-tutorial.html).  
**39) What is blocking method in Java?**(answer)  
A blocking method is a method which blocks until the task is done, for example, accept() method of ServerSocket blocks until a client is connected. here blocking means control will not return to the caller until the task is finished. On the other hand, there is an asynchronous or non-blocking method which returns even before the task is finished. To learn more about blocking method see this [answer](http://javarevisited.blogspot.sg/2012/02/what-is-blocking-methods-in-java-and.html).  
**40) Is Swing thread-safe? What do you mean by Swing thread-safe?**(answer)  
You can simply this question as No, Swing is not thread-safe, but you have to explain what you mean by that even if the interviewer doesn't ask about it. When we say swing is not thread-safe we usually refer its component, which can not be modified in multiple threads. All update to GUI components has to be done on AWT thread, and Swing provides synchronous and asynchronous callback methods to schedule such updates. You can also read my article to learn more about [swing and thread-safety](http://javarevisited.blogspot.com/2013/08/why-swing-is-not-thread-safe-in-java-Swingworker-Event-thread.html) to better answer this question. Even next two questions are also related to this concept.  
**41) What is the difference between invokeAndWait and invokeLater in Java?**(answer)  
These are two methods Swing API provides Java developers for updating GUI components from threads other than Event dispatcher thread. InvokeAndWait() synchronously update GUI component, for example, a progress bar, once progress is made, the bar should also be updated to reflect that change. If progress is tracked in a different thread, it has to call invokeAndWait() to schedule an update of that component by Event dispatcher thread. On another hand, invokeLater() is an asynchronous call to update components. You can also refer this [answer](http://javarevisited.blogspot.com/2011/09/invokeandwait-invokelater-swing-example.html) for more points.  
**42) Which method of Swing API are thread-safe in Java?**(answer)  
This question is again related to swing and thread-safety though components are not thread-safe there is a certain method which can be safely called from multiple threads. I know about repaint(), and revalidate() being thread-safe but there are other methods on different swing components e.g. setText() method of JTextComponent, insert() and append() method of JTextArea class.  
**43) How to create an Immutable object in Java?**(answer)  
This question might not look related to multi-threading and concurrency, but it is. Immutability helps to simplify already complex concurrent code in Java. Since immutable object can be shared without any synchronization its very dear to Java developers. Core value object, which is meant to be shared among thread should be immutable for performance and simplicity. Unfortunately there is no @Immutable annotation in Java, which can make your object immutable, hard work must be done by Java developers. You need to keep basics like initializing state in constructor, no setter methods, no leaking of reference, keeping separate copy of mutable object to create Immutable object. For step by step guide see my post, [how to make an object Immutable in Java](http://javarevisited.blogspot.com/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html). This will give you enough material to answer this question with confidence.  
**44) What is ReadWriteLock in Java?**(answer)  
In general, read write lock is the result of lock stripping technique to improve the performance of concurrent applications. In Java, ReadWriteLock is an interface which was added in Java 5 release. A ReadWriteLock maintains a pair of associated locks, one for read-only operations and one for writing. The read lock may be held simultaneously by multiple reader threads, so long as there are no writers. The write lock is exclusive. If you want you can implement this interface with your own set of rules, otherwise you can use ReentrantReadWriteLock, which comes along with JDK and supports a maximum of 65535 recursive write locks and 65535 read locks.  
**45) What is busy spin in multi-threading?**([answer](http://java67.blogspot.com/2015/09/60-java-interview-questions-for-quick.html))  
Busy spin is a technique which concurrent programmers employ to make a thread wait on certain condition. Unlike traditional methods e.g. wait(), sleep() or yield() which all involves relinquishing CPU control, this method does not relinquish CPU, instead it the just runs empty loop. Why would someone do that? to preserve CPU caches. In a multi-core system, it's possible for a paused thread to resume on a different core, which means rebuilding cache again. To avoid cost of rebuilding cache, programmer prefer to wait for much smaller time doing busy spin. You can also see this [answer](http://java67.blogspot.com/2012/08/5-thread-interview-questions-answers-in.html) to learn more about this question.  
**46) What is the difference between the volatile and atomic variable in Java?**(answer)  
This is an interesting question for Java programmer, at first, volatile and atomic variable look very similar, but they are different. Volatile variable provides you happens-before guarantee that a write will happen before any subsequent write, it doesn't guarantee atomicity. For example count++ operation will not become atomic just by declaring count variable as volatile. On the other hand AtomicInteger class provides atomic method to perform such compound operation atomically e.g. getAndIncrement() is atomic replacement of increment operator. It can be used to atomically increment current value by one. Similarly you have atomic version for other data type and reference variable as well.

**47) What happens if a thread throws an Exception inside synchronized block?**(answer)  
This is one more tricky question for average Java programmer, if he can bring the fact about whether lock is released or not is a key indicator of his understanding. To answer this question, no matter how you exist synchronized block, either normally by finishing execution or abruptly by throwing exception, thread releases the lock it acquired while entering that synchronized block. This is actually one of the reasons I like synchronized block over lock interface, which requires explicit attention to release lock, generally this is achieved by releasing the lock in a [finally block](http://javarevisited.blogspot.com/2012/11/difference-between-final-finally-and-finalize-java.html).  
  
**48) What is double checked locking of Singleton?**(answer)  
This is one of the very popular question on Java interviews, and despite its popularity, chances of candidate answering this question satisfactory is only 50%. Half of the time, they failed to write code for double checked locking and half of the time they failed how it was broken and fixed on Java 1.5. This is actually an old way of creating thread-safe singleton, which tries to optimize performance by only locking when Singleton instance is created first time, but because of complexity and the fact it was broken for JDK 1.4,  I personally don't like it. Anyway, even if you not prefer this approach its good to know from interview point of view. Since this question deserve a detailed answer, I have answered in a separate post, you can read my post [how double checked locking on Singleton works](http://javarevisited.blogspot.sg/2014/05/double-checked-locking-on-singleton-in-java.html) to learn more about it.  
  
**49) How to create thread-safe Singleton in Java?**(answer)  
This question is actually follow-up of the previous question. If you say you don't like double checked locking then Interviewer is bound to ask about alternative ways of creating thread-safe Singleton class. There are actually man, you can take advantage of class loading and static variable initialization feature of JVM to create instance of Singleton, or you can leverage powerful enumeration type in Java to create Singleton. I actually preferred that way, you can also read this [article](http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html) to learn more about it and see some sample code.  
  
**50) List down 3 multi-threading best practice you follow?**([answer](http://javarevisited.blogspot.com/2015/05/top-10-java-multithreading-and.html))  
This is my favorite question because I believe that you must follow certain best practices while writing concurrent code which helps in performance, debugging and maintenance. Following are three best practices, I think an average Java programmer should follow:

* **Always give meaningful name to your thread**This goes a long way to find a bug or trace an execution in concurrent code. OrderProcessor, QuoteProcessor or TradeProcessor is much better than Thread-1. Thread-2 and Thread-3. The name should say about task done by that thread. All major framework and even JDK follow this best practice.
* **Avoid locking or Reduce scope of Synchronization**  
  Locking is costly and context switching is even costlier. Try to avoid synchronization and locking as much as possible and at a bare minimum, you should reduce critical section. That's why I prefer synchronized block over synchronized method because it gives you absolute control on the scope of locking.
* **Prefer Synchronizers over wait and notify**  
  Synchronizers like CountDownLatch, Semaphore, CyclicBarrier or Exchanger simplifies coding. It's very difficult to implement complex control flow right using wait and notify. Secondly, these classes are written and maintained by best in business and there is good chance that they are optimized or replaced by better performance code in subsequent JDK releases. By using higher level synchronization utilities, you automatically get all these benefits.
* **Prefer Concurrent Collection over Synchronized Collection**  
  This is another simple best practice which is easy to follow but reap good benefits. Concurrent collection are more scalable than their synchronized counterpart, that's why its better to use them while writing concurrent code. So next time if you need map, think about ConcurrentHashMap before thinking Hashtable. See my article [Concurrent Collections in Java](http://javarevisited.blogspot.com/2013/02/concurrent-collections-from-jdk-56-java-example-tutorial.html), to learn more about modern collection classes and how to make best use of them.

**51) How do you force to start a Thread in Java?**(answer)  
This question is like how do you force garbage collection in Java, there is no way though you can make a request using System.gc() but it's not guaranteed. On Java multi-threading there is absolute no way to force start a thread, this is controlled by thread scheduler and Java exposes no API to control thread schedule. This is still a random bit in Java.  
  
**52) What is the fork-join framework in Java?**(answer)  
The fork join framework, introduced in JDK 7 is a powerful tool available to Java developer to take advantage of multiple processors of modern day servers. It is designed for work that can be broken into smaller pieces recursively. The goal is to use all the available processing power to enhance the performance of your application. One significant advantage of The fork/join framework is that it uses a work-stealing algorithm. Worker threads that run out of things to do can steal tasks from other threads that are still busy. See this [article](http://javarevisited.blogspot.com/2011/09/fork-join-task-java7-tutorial.html) for the much more detailed answer to this question.  
  
**53) What is the difference between calling wait() and sleep() method in Java multi-threading?**(answer)  
Though both wait and sleep introduce some form of pause in Java application, they are the tool for different needs. Wait method is used for inter thread communication, it relinquishes lock if waiting for a condition is true and wait for notification when due to an action of another thread waiting condition becomes false. On the other hand sleep() method is just to relinquish CPU or stop execution of current thread for specified time duration. Calling sleep method doesn't release the lock held by current thread. You can also take look at this [article](http://javarevisited.blogspot.com/2011/12/difference-between-wait-sleep-yield.html) to answer this question with more details.

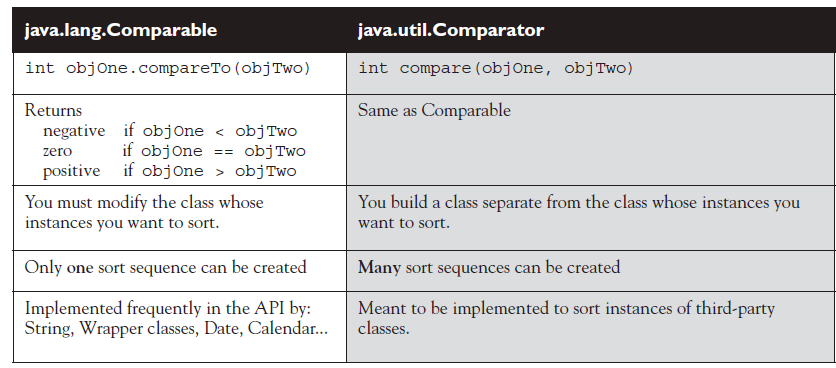
### Collections

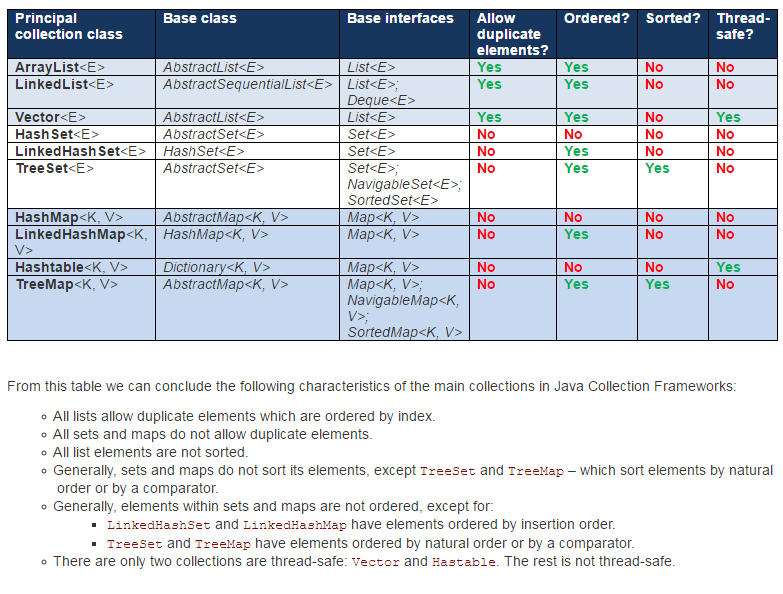
Collections are like containers that groups multiple items in a single unit.

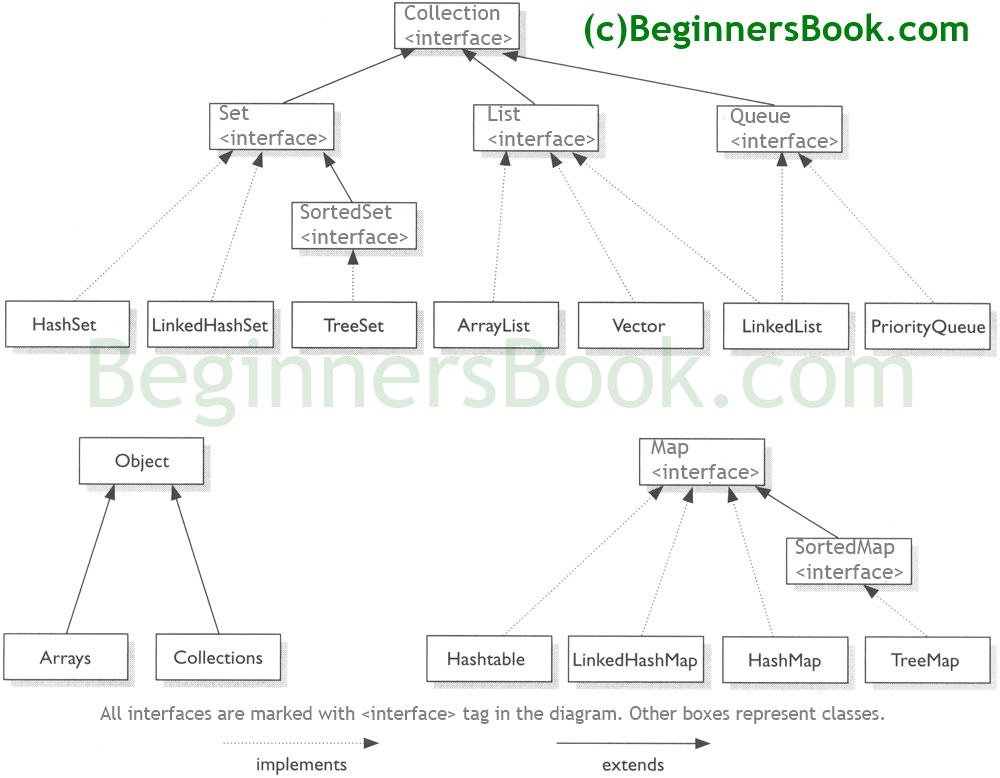
Java Collections Framework interfaces provides the abstract data type to represent collection. java.util.Collection is the root interface of Collections Framework. It is on the top of Collections framework hierarchy. It contains some important methods such as size(), iterator(), add(), remove(), clear() that every Collection class must implement.

**Concurrent vs Synchronized: -** For your needs, use ConcurrentHashMap. It allows concurrent modification of the Map from several threads without the need to block them. Collections.synchronizedMap(map) creates a blocking Map which will degrade performance, albeit ensure consistency (if used properly).

Use the second option if you need to ensure data consistency, and each thread needs to have an up-to-date view of the map. Use the first if performance is critical, and each thread only inserts data to the map, with reads happening less frequently.







### Tree Map

* is sorted
* null values are not allowed
* Only homogeneous data allowed
* uniqueness is maintained
* if we try to insert same value already present it will override the previous value

### Linked Hash Map

* LinkedHasphMap preserves the insertion order
* null values are allowed
* uniqueness is maintained
* if we try to insert same value already present it will override the previous value.

Hash Table

* Reverse sorting order 4,3,2,1
* null values are not allowed
* uniqueness is maintained
* if we try to insert same value already present it will override the previous value

Hash Map

* HasphMap is ordered
* null values are allowed
* uniqueness is maintained
* if we try to insert same value already present it will override the previous value.

Tree Set

* TreeSet is sorted in particular order
* only homogeneous data allowed
* null values are not allowed
* Duplicates are not allowed

## **List vs Set in Java**

1) Fundamental difference between List and Set in Java is **allowing duplicate elements**. List in Java allows duplicates while **Set doesn't allow any duplicate**. If you insert duplicate in Set it will replace the older value. Any implementation of Set in Java will only contains unique elements.

2) Another significant difference between List and Set in Java is order. **List is an Ordered Collection** while Set is an unordered Collection. List maintains **insertion order of elements**, means any element which is inserted before will go on lower index than any element which is inserted after. Set in Java doesn't maintain any order. Though Set provide another alternative called SortedSet which can store Set elements in specific Sorting order defined by [Comparable and Comparator](http://javarevisited.blogspot.com/2011/06/comparator-and-comparable-in-java.html) methods of Objects stored in Set.

3) Set uses [equals() method](http://javarevisited.blogspot.com/2011/02/how-to-write-equals-method-in-java.html) to check uniqueness of elements stored in Set, while SortedSet uses [compareTo() method](http://javarevisited.blogspot.com/2011/11/how-to-override-compareto-method-in.html) to implement natural sorting order of elements. In order for an element to behave properly in Set and SortedSet, [equals and compareTo must be consistent](http://javarevisited.blogspot.com/2011/11/how-to-override-compareto-method-in.html) to each other.

4) Popular implementation of List interface in Java includes ArrayList, Vector and LinkedList. While popular implementation of Set interface includes HashSet, TreeSet and LinkedHashSet.

## **When to use List and Set in Java**

Another good follow-up question is "**when do you use List and Set in Java**" , which can also be answered based on properties of List and Set we have learn here.These *difference between Set and List* also teaches us when to use Set and when to prefer List. its pretty clear that if you need to maintain insertion order or object and you collection can contain duplicates than List is a way to go. On the other hand if your requirement is to maintain unique collection without any duplicates than Set is the way to go.

Important point to note is that both List and Set are derived from Collection Interface. In short **main difference between List and Set in Java is that List is an ordered collection** which allows duplicates while Set is an unordered collection which doesn't allow duplicates.

## **Comparator vs Comparable in Java**

1) Comparator in Java is defined in java.util package while Comparable interface in Java is defined in java.lang package, which very much says that Comparator should be used as an utility to sort objects which Comparable should be provided by default.

2) Comparator interface in Java has method public int compare (Object o1, Object o2) which returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second. While Comparable interface has method public int compareTo(Object o) which returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

3) If you see then logical difference between these two is *Comparator in Java* compare two objects provided to him, while Comparable interface compares "this" reference with the object specified. I have shared lot of tips on [how to override compareTo() method](http://javarevisited.blogspot.sg/2011/11/how-to-override-compareto-method-in.html) and avoid some common mistakes programmer makes while implementing Comparable interface.

4) Comparable in Java is used to implement **natural ordering of object**. In Java API String, Date and wrapper classes implements Comparable interface.Its always good practice to override compareTo() for value objects.

5) If any class implement Comparable interface in Java then collection of that object either [List](http://javarevisited.blogspot.sg/2012/04/difference-between-list-and-set-in-java.html) or Array can be sorted automatically by using  Collections.sort() or Arrays.sort() method and object will be sorted based on there natural order defined by CompareTo method.

6)Objects which implement *Comparable in Java*  can be used as keys in a SortedMap like [TreeMap](http://javarevisited.blogspot.sg/2011/12/treemap-java-tutorial-example-program.html) or elements in a SortedSet  for example TreeSet, without specifying any Comparator.

[**Why String is Immutable or Final in Java**](http://javarevisited.blogspot.sg/2010/10/why-string-is-immutable-in-java.html)

The string is Immutable in Java because String objects are cached in String pool. Since cached String literals are shared between multiple clients there is always a risk, where one client's action would affect all another client. For example, if one client changes the value of String "Test" to "TEST", all other clients will also see that value as explained in the first example. Since caching of String objects was important from performance reason this risk was avoided by making String class Immutable. At the same time, [*String was made final*](http://java67.blogspot.com/2014/01/why-string-class-has-made-immutable-or-final-java.html) so that no one can compromise invariant of String class e.g. Immutability, Caching, hashcode calculation etc by extending and overriding behaviors. Another reason of *why String class is immutable* could die due to HashMap.

Since Strings are very popular as HashMap key, it's important for them to be immutable so that they can retrieve the value object which was stored in HashMap. Since [HashMap works in the principle of hashing](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html), which requires same has value to function properly. Mutable String would produce two different hashcodes at the time of insertion and retrieval if contents of String was modified after insertion, potentially losing the value object in the map.

StringBuilder is not thread safe. In other words, its methods are not synchronized.

## How HashMap Internally Works in Java

**Have you used HashMap before**or**What is HashMap? Why do you use it**

HashMap accept null while Hashtable doesn't, [HashMap is not synchronized](http://javarevisited.blogspot.com/2010/10/difference-between-hashmap-and.html), HashMap is fast and so on along with basics like its stores key and value pairs etc.

**Do you Know how HashMap works in Java** or **How does get () method of HashMap works in Java**

**HashMap works on the principle of hashing**, we have put(key, value) and get(key) method for storing and retrieving Objects from HashMap. When we pass Key and Value object  to put() method on Java HashMap, HashMap implementation calls [hashCode method](http://javarevisited.blogspot.sg/2011/10/override-hashcode-in-java-example.html) on Key object and applies returned hashcode into its own hashing function to find a bucket location for storing Entry object, important point to mention is that HashMap in Java stores both key and value object as Map.Entry in a bucket which is essential to understand the retrieving logic.

**What will happen if two different objects have the same hashcode?**

Now from here onwards real confusion starts, sometime candidate will say that since hashcode is equal, both objects are equal and HashMap  will throw exception or not store them again etc, Then you might want to remind them about [equals() and hashCode() contract](http://javarevisited.blogspot.sg/2011/02/how-to-write-equals-method-in-java.html) that two unequal objects in Java can have same hashcode. Some will give up at this point and few will move ahead and say "Since hashcode is same, bucket location would be same and collision will occur in HashMap Since HashMap uses LinkedList to store object, this entry (object of Map.Entry comprise key and value )  will be stored in [LinkedList](http://javarevisited.blogspot.sg/2012/02/difference-between-linkedlist-vs.html). Great this answer make sense though there are many collision resolution methods available  like linear probing and chaining, this is simplest and HashMap in Java does follow this.

**How will you retrieve Value object  if two Keys will have the same hashcode?**

Interviewee will say we will call get() method and then HashMap uses Key Object's hashcode to find out bucket location and retrieves Value object but then you need to remind him that there are two Value objects are stored in same bucket , so they will say about [traversal in LinkedList](http://javarevisited.blogspot.sg/2010/10/how-do-you-find-length-of-singly-linked.html) until we find the value object , then you ask *how do you identify value object because you don't  have value object to compare* ,Until they know that HashMap  stores both Key and Value in LinkedList node or as Map.Entry they won't be able to resolve this issue and will try and fail.

But those bunch of people who remember this key information will say that after finding bucket location, we will **call keys.equals() method** to identify a correct node in LinkedList and return associated value object for that key in Java HashMap. Perfect this is the correct answer.

In many cases interviewee fails at this stage because they get confused between[hashCode()](http://javarevisited.blogspot.sg/2011/10/override-hashcode-in-java-example.html) and equals(**)** or keys and values object in Java HashMap  which is pretty obvious because they are dealing with the hashcode() in all previous questions and equals() come in picture only in case of retrieving value object from HashMap in Java. Some good developer point out here that using immutable, [final object](http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html) with proper equals() and hashcode() implementation would act as perfect Java HashMap  keys and **improve the performance of Java HashMap  by reducing collision**. Immutability *also allows caching their hashcode of different keys* which makes overall retrieval process very fast and suggest that [String](http://javarevisited.blogspot.sg/2011/07/string-vs-stringbuffer-vs-stringbuilder.html) and various wrapper classes e.g. Integer very good keys in Java HashMap.

Now if you clear this entire Java HashMap interview,  You will be surprised by this very interesting question "**What happens On HashMap in Java if the size of the HashMap  exceeds a given threshold defined by load factor ?"**. Until you know how HashMap  works exactly you won't be able to answer this question. If the size of the Map exceeds a given threshold defined by load-factor e.g. if the load factor is .75 it will act to re-size the map once it filled 75%. Similar to other collection classes like [ArrayList](http://javarevisited.blogspot.sg/2011/05/example-of-arraylist-in-java-tutorial.html),  Java HashMap re-size itself by creating a new bucket array of size twice of the previous size of HashMap and then start putting every old element into that new bucket array. This process is called rehashing because it also applies the hash function to find new bucket location.

If you manage to answer this question on HashMap in Java you will be greeted by **"do you see any problem with resizing of HashMap  in Java"** , you might not be able to pick the context and then he will try to give you hint about multiple thread accessing the Java HashMap and potentially looking for **race condition on HashMap  in Java**.

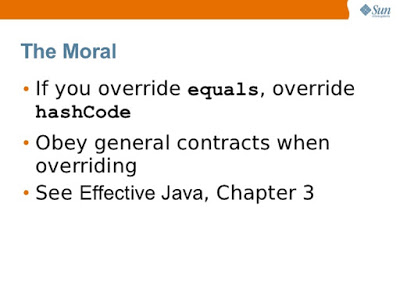
So the answer is Yes there is potential [race condition](http://javarevisited.blogspot.sg/2012/02/what-is-race-condition-in.html) exists while resizing HashMap in Java, if two [thread](http://javarevisited.blogspot.sg/2011/02/how-to-implement-thread-in-java.html) at the same time found that now HashMap needs resizing and they both try to resizing. on the process of resizing of HashMap in Java, the element in the bucket which is stored in linked list get reversed in order during their migration to new bucket because Java HashMap  doesn't append the new element at tail instead it append new element at the head *to avoid tail traversing*. If race condition happens then you will end up with an infinite loop. Though this point, you can potentially argue that what the hell makes you think to use HashMap  in multi-threaded environment to interviewer :)

## **Equals and HashCode Interview questions in Java**

Here is my list of 10 interesting questions on both of these methods.  I have seen, programmer struggles to write equals() and hashCode() by hands, for a rich class, which contains different data types e.g. int, float, date etc. Reading those items and trying examples will give you enough confidence to face any question on equals and hashCode methods. I also suggest to read [Effective Java Items on equals() and hashCode()](http://www.amazon.com/dp/0321356683/?tag=javamysqlanta-20) to fill your gaps in knowledge of this two critical methods.

**When you are writing equals() method, which other method or methods you need to override?**

hashcode is the right answer. Since equals and hashCode have their contract, so overriding one and not other will break the contract between them. By the way this question can lead to an interesting discussion, if Interviewer likes to go on deep e.g. he may ask what are those contracts, what happens if those contracts break etc. I like to give an example How equals and hashcode are used in hash based collections e.g. [Hashtable](http://javarevisited.blogspot.sg/2012/01/java-hashtable-example-tutorial-code.html), that leaves positive impression more often. You can also mention about [compareTo()](http://javarevisited.blogspot.com/2011/11/how-to-override-compareto-method-in.html) here to score some additional point, this method should also need to be consistent with equals, which is another interesting question on our list.

[](http://2.bp.blogspot.com/-ZJLVjuc0AhI/VXhRI2tYhII/AAAAAAAAC_Q/5e4m76mGeLs/s1600/Equals+and+HashCode+In+Java.jpg)

**Can two objects which are not equal have the same hashCode?** YES, two objects, which are not equal to equals() method can still return same hashCode. By the way, this is one of the confusing bit of equals and hashcode contract.

**How does get() method of HashMap works, if two keys have the same hashCode?** This is the follow-up of previous interview questions on equals and hashcode, in fact, sometimes this leads to a discussion of the earlier point. When two key return same hashcode, they end up in the same bucket. Now, in order to find the correct value, you used keys.equals() method to compare with key stored in each Entry of linked list there. Remember to point out keys.equals() method, because that's what interviewer is looking for. You can also see here for full [list of interview question on Java HashMap](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html).

**Where have you written equals() and hashCode in your project?**

This is to see if the developer has even written these methods or not. Of course, almost all of Java programmer are exposed to this, you can point out value objects, Hibernate entities from your domain, where you have overridden equals and hashCode. Always gives examples from your domain and from your project, rather than a trivial example from a test program, because if Interviewer is asking this question, it means he is interested in examples from your domain.

**Suppose your Class has an Id field, should you include in equals()? Why?**

This question is asked to one of my readers as [Hibernate Interview question](http://javarevisited.blogspot.com/2013/05/10-hibernate-interview-questions-answers-java-j2ee-senior.html), well including id is not a good idea in equals() method because this method should check equality based upon content and business rules. Also including id, which is mostly a database identifier and not available to transient object until they are saved into the database.

**What happens if equals() is not consistent with compareTo() method?**

This is an interesting questions, which asked along with equals() and hashCode() contract. Some java.util.Set implementation e.g. SortedSet or it's concrete implementation TreeSet uses compareTo() method for comparing objects. If compareTo() is not consistent means doesn't return zero, if equals() method returns true, it may break Set contract, which is not to avoid any duplicates.

**What happens if you compare an object to null using equals()?**

When a null object is passed as an argument to equals() method, it should return false, it must not throw NullPointerException, but if you call equals method on reference, which is null it will throw NullPointerException. That’s why it’s better to use == operator for comparing null e.g. if(object != null) object.equals(anohterObject). By the way, if you comparing String literal with another String object then you better call equals() method on the String literal rather than known object to avoid NPE, one of those simple [tricks to avoid NullPointerException in Java](http://javarevisited.blogspot.com/2013/05/ava-tips-and-best-practices-to-avoid-nullpointerexception-program-application.html).

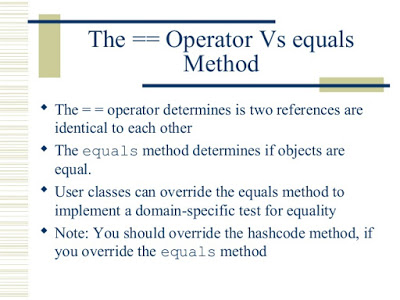
**What is the difference in using instanceof and getClass() method for checking type inside equals?**

This question was asked multiple times, sometimes by looking at your equals() and hashCode implementation. Well, key difference comes from the point that instanceof operator returns true, even if compared with subclass e.g. Subclass instanceof Superclass is true, but with getClass() it's false. By using getClass() you ensure that your equals() implementation doesn't return true if compared with subclass object. While if you use instanceof operator, you end up breaking symmetry rule for equals which says that if a.equals(b) is true than b.equals(a) should also be true. Just replace a and b with an instance of Superclass and Subclass, and you will end up breaking symmetry rule for equals() method.

**How do you avoid NullPointerException, while comparing two Strings in Java?**

Since when compared to null, equals return false and doesn't throw NullPointerException, you can use this property to avoid NPE while using comparing String. Suppose you have a known String "abc" and you are comparing with an unknown String variable str, then you should call equals as "abc".equals(str), this will not throw [Exception in thread Main: java.lang.NullPointerException](http://javarevisited.blogspot.com/2012/06/common-cause-of-javalangnullpointerexce.html), even if str is null. On the other hand, if you call str.equals("abc"), it will throw NPE. So be careful with this. By the way this is one of the Java coding best practices, which Java developer should follow, while using equals() method.

**What is the difference between "==" and equals() method in Java?** See [here](http://javarevisited.blogspot.com/2012/07/auto-boxing-and-unboxing-in-java-be.html) for a detailed discussion on how it affect equality checking of String and Integer in the autoboxing world.

[](https://1.bp.blogspot.com/-BRU6TtqXFR4/VvQBMmKvl7I/AAAAAAAAFSs/5_VVWHwBGQc2JwXdSFP39W9trtTEwmTtQ/s1600/Difference+between+equals+and+==+operator+in+Java.jpg)

### ENUM:-

## [What is Enum in Java](http://javarevisited.blogspot.com/2011/08/enum-in-java-example-tutorial.html)

Now back to primary questions **“What is Enum in java”** simple *answer Enum is a keyword in java* and on more detail term Java Enum is a type like class and interface and can be used to define a set of Enum constants. Enum constants are [implicitly static and final](http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html) and you can not change their value once created. Enum in Java provides type-safety and can be used inside switch statement like int variables.   
Since enum is a keyword you can not use as a variable name and since its only introduced in JDK 1.5 all your previous code which has an enum as a variable name will not work and needs to be refactored.  
Benefits of using Enums in Java

1) **Enum is type-safe** you can not assign anything else other than predefined Enum constants to an Enum variable. It is a compiler error to assign something else, unlike the public static final variables used in [Enum int pattern](http://javarevisited.blogspot.com/2015/10/133-java-interview-questions-answers-from-last-5-years.html) and [Enum String pattern](http://java67.blogspot.com/2015/03/top-40-core-java-interview-questions-answers-telephonic-round.html).  
3) The best feature of Enum is **you can use Enum in Java inside Switch statement** like int or char primitive data type.   
4) Adding new constants on [Enum in Java](http://java67.blogspot.com/2014/04/what-java-developer-should-know-about-Enumeration-type-in-Java.html) is easy and you can add new constants without breaking the existing code.  
Important points about Enum in Java

1) **Enums in Java are type-safe** and has their own namespace. It means your enum will have a type for example "Currency" in below example and you can not assign any value other than specified in Enum Constants.  
2**) Enum in Java are reference types** like [class](http://javarevisited.blogspot.sg/2011/10/class-in-java-programming-general.html) or [interface](http://javarevisited.blogspot.sg/2012/04/10-points-on-interface-in-java-with.html)and you can define constructor, methods and variables inside java Enum which makes it more powerful than Enum in C and C++ as shown in next example of Java Enum type.  
3) You can **specify values of enum constants at the creation time** as shown in below example:  
**public** **enum** Currency {**PENNY**(*1*), **NICKLE**(*5*), **DIME**(*10*), **QUARTER**(*25*)};

But for this to work you need to define a member variable and a constructor because PENNY (1) is actually [calling a constructor](http://javarevisited.blogspot.sg/2012/01/what-is-constructor-overloading-in-java.html) which accepts int value, see below example.  
**public** **enum** Currency {

**PENNY**(1), **NICKLE**(5), **DIME**(10), **QUARTER**(25);

**private** **int** value;

**private** **Currency**(**int** *value*){this**.**value **=** value;}};

**The constructor of enum in java** must be [**private**](http://javarevisited.blogspot.sg/2012/03/private-in-java-why-should-you-always.html) any other access modifier will result in compilation error. Now to get the value associated with each coin you can define a public getValue() method inside Java enum like any normal Java class. Also, the semicolon in the first line is optional.  
4) Enum constants are implicitly [static](http://javarevisited.blogspot.sg/2012/03/mixing-static-and-non-static.html) and [final](http://javarevisited.blogspot.sg/2010/10/why-string-is-immutable-in-java.html) and can not be changed once created. For example, below code of java enum will result in compilation error:

**Currency.PENNY** **=** **Currency.DIME**;

The final field EnumExamples.Currency.PENNY cannot be reassigned.  
5) **Enum in java can be used as an argument on switch statement** and with "case:" like int or char primitive type. This feature of java enum makes them very useful for switch operations. Let’s see an example of how to use java enum inside switch statement:

**Currency** usCoin **=** **Currency.DIME**;  
 **switch** (usCoin) {

**case** **PENNY:**

**System.**out**.**println("Penny coin");

**break**;

**case** **NICKLE:**

**System.**out**.**println("Nickle coin");

**break**;

**case** **DIME:**

**System.**out**.**println("Dime coin");

**break**;

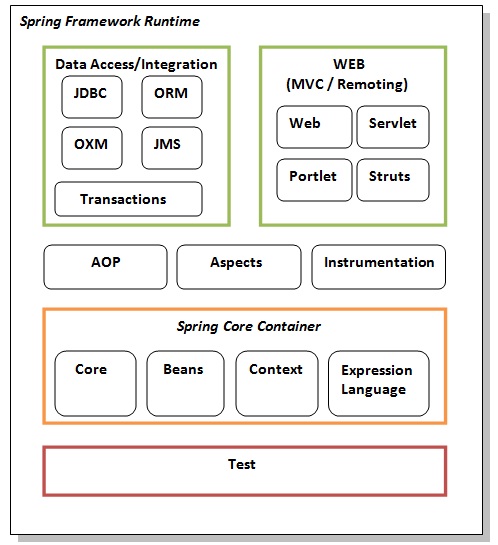
**case** **QUARTER:**

**System.**out**.**println("Quarter coin");

}

# Spring

The Spring framework comprises of many modules such as core, beans, context, expression language, AOP, Aspects, Instrumentation, JDBC, ORM, OXM, JMS, Transaction, Web, Servlet, Struts etc. These modules are grouped into Test, Core Container, AOP, Aspects, Instrumentation, Data Access / Integration, Web (MVC / Remoting) as displayed in the following diagram.



Spring is a light weight software and applications are light weight applications:-

1. Spring containers are light weight containers because they can be activated anywhere by just creating objects from certain predefined classes.
2. Most of the spring applications can be executed without the heavy weight application server and webserver software.
3. Resources of spring application can be developed without using spring API.
4. While executing spring applications much memory and CPU resources are not needed.

Spring is loosely coupled because:-

1. We can use either individual modules or multiple modules together.
2. We can integrate Spring with other Java Technologies in application development like Servlets, JSP, Struts and Hibernate.
3. It enables to develop large scale applications by enabling middleware services like security, transaction management.
4. Spring provide AOP module and could also integrate with spring applications.

[](https://www.java4s.com/wp-content/uploads/2013/07/Spring-MVC-execution-flow.png)

## Spring MVC 3.2 Execution Flow

Step **1**: First request will be received by DispatcherServlet  
Step **2**: DispatcherServlet will take the help of HandlerMapping and get to know the Controller class name associated with the given request  
Step **3**: So request transfer to the Controller, and then controller will process the request by executing appropriate methods and returns ModeAndView object (contains Model data and View name) back to the DispatcherServlet  
Step 4: Now DispatcherServlet send the model object to the ViewResolver to get the actual view page  
Step **5**: Finally DispatcherServlet will pass the Model object to the View page to display the result

**POJO/POJI:-**  If a Java class/Interface that is acting as a resource of certain technology based s/w application is not extending , implementing predefined classes/interfaces of that technology specific API.

### Dependency Look up vs Dependency Injection:-

* Dependency Look - Resource explicitly searches and gathers dependent value from others. The way we collect only the required resources. But resources should spend some time to gather them before using it.
* **DI: -** The assigning of values to the resources automatically and dynamically by the server. It will be used directly without wasting time to allocate the required resources. It may also inject unnecessary values.

**Whenever beanfactory container is activated, it does not create any spring bean class object.** It creates Spring bean class objects and completes dependency injection only when a getBean() method is called.

**Spring Bean: -** The concrete java class that is configured in spring configuration file or thru Annotation.

**Container** is an S/w Application or a Java Class that can take care of the whole life of a resource. Spring Containers will do lifecycle management and Dependency injection of a spring bean.

## **Spring Core Model:-**

This Core Module is the base module of a spring frame work application.  
This Core Module will provide the following information to the spring programmers. It gives a light weight container called Bean Factory. We can understand spring bean lifecycle management and DI.  
  1. The way of creating spring Beans.  
    2. The way of Configuring Spring Beans.  
    3. The way of injecting the dependencies.  
    4. The way of obtaining a spring container object.  
    5. The way of accessing the spring beans from container.

Spring Core Module Application contains the following resources:-

1. Spring Interface- Its like a POJI and an optional resource but recommended to write.
2. Spring Bean Class- A Concrete class configured in spring config file , implements spring interface methods , constructors , lifecycle methods which supports DI.
3. Spring Cfg File.
4. Client Application.(Java with main method)

Spring support 3 types of DI:-

Setter Injection(setXX)

1. BeanFatoryAware
2. ApplicationContextAware
3. BeanNameAware

, Constructor Injection , Interface Injection.

* spring will look for 0 param constructor to create the spring bean class object, even though we use only parameterized constructor/setter to initialize a bean.
* Bean property can be taken as static /non static , but the setter method used for setter injection must be considered as non static.

## **Differentiate between BeanFactory and ApplicationContext in spring.**

Spring gives 2 built in container and light weight containers:-

1. **Bean Factory( Part of Core Module)**

* Bean instantiation/wiring
* To activate spring bean factory , create an Object of java class that implements org.springframework.beans.factory.BeanFactory Interface.
* XMLBeanFatory factory = new XMLBeanFactory(“beanpointer in springcfg”);
* Whenever Bean Factory container is activated the entries of the spring cfg file will be verified by using SAXXMLParser.

1. **Application Context (Part of Spring JEE/Context Module)**

3 ways to create Application context FileSystemXMLApplicationContext , ClassPathXMLApplicationContext, XMLWebApplicationContext.

* Bean instantiation/wiring
* To Activate App Context container in our apps, create object for java class that implements org.springframework.context.ApplicationContext Inteface.
* SytemXMLApplicationContext ctx = new FileSytemXMLApplicationContext(“beanpointer in springcfg”);
* Pre installation of Singleton Beans.
* Ability to work with multiple Spring cfg files in a single instance of container activation.
* Ability to read values from prop files.
* I18n
* Support for Event handling. We need Event Listeners to perform this.

So if you need any of the points presented on the Application Context side, you should use ApplicationContext.

Init- container execute this method after bean instantiation and DI

Destroy- container execute the method when it is about to destroy our spring bean class object. Used to nullify the bean properties and release non java resources with the spring beans.

InitalizingBean and DisposableBean were 2 interfaces to call init and destroy methods.

FactoryBean(I)- getObject(), getObjectType(), isSingleton()

AutoWiring:-Performing Dependency Injection on bean properties is called wiring.

Explicit:- using tags in cfg file. And value in bean tag

We need to enable autowire attribute on bean tag , the container automatically detects and injects values to the bean properties.

byname

byType

constructor

audodetect – perform either constructor or by type. If 0 param constructor exists it perform bytype else constructor injection.

@component:- makes java class a Spring bean and auto detectable from classpath for DI.

@Service:-which contain business logic.

@Repository- DAO objects for communication to db.

@Value – to inject a value

@Resource:- to inject value to reference type bean properties.

@Autowired :- to enable autowiring on bean properties.

**Spring Bean Scope:-**

|  |  |
| --- | --- |
| **Scope** | **Description** |
| singleton | This scopes the bean definition to a single instance per Spring IoC container (default). |
| prototype | This scopes a single bean definition to have any number of object instances based on number of calls for factory.getBean(). |
| request | This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext. |
| session | This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext. |
| global-session | This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext |

**BeanPostProcessor(I):-**

**JDBC Connection Pool:-** It is a factory that contains set of readily available JDBC Connections. All the connection objects in the pool represents connectivity with the same db s/w.DriverManagerDataSource class gives JDBC Datasource object pointing to jdbc connection pool.

## **Spring DAO (JDBC) Module:-**

This Module will provide an abstraction layer on top of JDBC technology.  
        While working with JDBC technology directly as programmers, we are writing the Boiler-plate code (Repeated code) like Loading the driver, opening a connection, creating a statement and closing the objects etc…  
         Apart from Boiler-plate code, we also need to handle the Exceptions of JDBC explicitly because JDBC Exceptions are checked exceptions.  
          In order to avoid the Boiler-plate code and to also avoid exception handling Burdon, spring frame work has provided this DAO module.

* Persistence logic gives the inputs to the application and to view the results of the application. Integration logic controls and monitors all the operations of the application.
* The Java class that separated the persistence logic from other logics of the application and to make the persistence logic as flexible to modify the reusable logics is called DAO.
* Spring JDBC select query gives results in List, Map, data Structures and we can send this data structures over the network.
* Spring JDBC internally uses Plain JDBC even though plain jdbc throws checked Exception called SQL Exception. The spring jdbc converts that one to unchecked Exception called DataAccessException.

## **Spring ORM Module:-**

This ORM Module is an abstraction layer on top of the ORM tools and which makes a spring programmer to transfer a data between application and database in the form of objects.  
            ORM tools are very good tools for communicating with databases but having some drawbacks, these drawbacks are eliminated in spring ORM module.

## **Spring JEE Module:-**

              This Module is for implementing the middleware services required for Business logic. This spring JEE module is an abstraction layer on top of

**RMI-** The Java Remote Method Invocation (RMI) system allows an object running in one Java virtual machine to invoke methods on an object running in another Java virtual machine. RMI provides for remote communication between programs written in the Java programming language.

* Locate Remote Objects
* Communicate with remote Objects.
* Load class definitions for objects that are passed around.

Java mail – MIMEMessageHelper/ SimpleMailMessage

JMS/ Spring JMS - JmsTemplate class handles the creation and releasing of resources when sending or synchronously receiving messages.

HTTPInvoker

There is a difference between AOP and JEE modules  
             AOP is just for applying the services (or) injecting the services but not for implementing the services, whereas JEE is a module for implementing the services.  
              For real time Business logic development with middleware services, we use spring core, spring AOP, and spring JEE modules.  
  
Spring Web Module:-  
  
This module is either for integrating with web applications created in other frame works (or) for creating complete MVC to based web application in spring frame work.  
                Spring web module is also called Spring web MVC module and this module is for creating web applications of java with MVC to architecture.

#### Spring Web Services:-

## **Spring AOP Module:-**

Aspect Oriented Programming is a new way to developing real time applications with required services  
             The AOP module will provide us the way of separating the services from Business logic and also it provides us the way of injecting the services for business logic at runtime.  
             In spring terminology, the services are also called “Cross-Cutting Concerns” (Middle ware services).

## 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). AOP provides the way to dynamically add the cross-cutting concern before, after or around the actual logic using simple pluggable configurations. 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 your EmployeeManager 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 EmployeeManager interface). 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.

<beans xmlns="<http://www.springframework.org/schema/beans>"

    xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"

    xmlns:aop="<http://www.springframework.org/schema/aop>"

    xsi:schemaLocation="<http://www.springframework.org/schema/beans>

<http://www.springframework.org/schema/beans/spring-beans-3.0.xsd>

<http://www.springframework.org/schema/aop>

<http://www.springframework.org/schema/aop/spring-aop-3.0.xsd> ">

    <!-- We don't need to this; This is required only in annotation based AOP support -->

    <!-- <aop:aspectj-autoproxy /> -->

    <aop:config>

        <!-- Spring AOP Pointcut definitions -->

        <aop:pointcut id="loggingOperation"

            expression="execution(\*com.howtodoinjava.demo.aop.EmployeeManager.\*(..))" />

        <aop:pointcut id="transactionOperation"

            expression="execution(\* com.howtodoinjava.demo.aop.EmployeeManager.getEmployeeById(..))" />

       <!-- Spring AOP aspect 1 -->

        <aop:aspect id="loggingAspect" ref="loggingAspectBean">

            <!-- Spring AOP advises -->

            <aop:before pointcut-ref="loggingOperation" method="logBefore" />

            <aop:after pointcut-ref="loggingOperation" method="logAfter" />

            </aop:aspect>

         <!-- Spring AOP aspect 2 -->

        <aop:aspect id="transactionAspect" ref="transactionAspectBean">

            <aop:before pointcut-ref="transactionOperation" method="getEmployeeById" />

        </aop:aspect>

     </aop:config>

     <!-- Spring AOP aspect instances -->

    <bean id="loggingAspectBean" class="com.howtodoinjava.demo.aop.EmployeeCRUDLoggingAspect" />

    <bean id="transactionAspectBean" class="com.howtodoinjava.demo.aop.EmployeeCRUDTransactionAspect" />

   <!-- Target Object -->

    <bean id="employeeManager" class="com.howtodoinjava.demo.aop.EmployeeManagerImpl" />

</beans>

## **Spring Model View Controller (MVC)**

### **What is Spring MVC framework?**

Spring comes with a [full-featured MVC framework for building web applications](http://examples.javacodegeeks.com/enterprise-java/spring/mvc/spring-mvc-hello-world-example/). Although Spring can easily be integrated with other MVC frameworks, such as Struts, Spring’s MVC framework uses IoC to provide a clean separation of controller logic from business objects. It also allows to declaratively bind request parameters to business objects.

Dependency Injection :- The underlying server or container pushes the values to resources automatically and dynamically.

Performing DI on bean properties is called wiring.-Using tags

Explicit and Auto wiring.- based on autowire attribute or annotations.- byname,bytype ,constructor , autodetect.

### **What is Dependency Injection in Spring?**

[Dependency Injection](http://www.javacodegeeks.com/2014/02/dependency-injection-options-for-java.html), This concept says that you do not create your objects but describe how they should be created. You don’t directly connect your components and services together in code but describe which services are needed by which components in a configuration file. A container (the IOC container) is then responsible for hooking it all up.

## What is bean wiring?

- Combining together beans within the Spring container is known as bean wiring or wiring. When wiring beans, you should tell the container what beans are needed and how the container should use dependency injection to tie them together.

### **What are the different types of IoC (dependency injection)?**

* **Constructor-based dependency injection:** Constructor-based DI is accomplished when the container invokes a class constructor with a number of arguments, each representing a dependency on other class.
* **Setter-based dependency injection:** Setter-based DI is accomplished by the container calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean.

### **Which DI would you suggest Constructor-based or setter-based DI?**

You can use both Constructor-based and Setter-based Dependency Injection. The best solution is using constructor arguments for mandatory dependencies and setters for optional dependencies.

The Singleton design pattern addresses all of these concerns. With the Singleton design pattern you can:

* Ensure that only one instance of a class is created
* Provide a global point of access to the object
* Allow multiple instances in the future without affecting a singleton class's clients

public class ClassicSingleton {

private static ClassicSingleton instance = null;

protected ClassicSingleton() {

// Exists only to defeat instantiation.

}

public static ClassicSingleton getInstance() {

if(instance == null) {

instance = new ClassicSingleton();

}

return instance;

}

}

### **What is the role of DispatcherServlet in Spring MVC?**

The Spring Web MVC framework is designed around a DispatcherServlet that handles all the HTTP requests and responses. DispatcherServlet is the servlet by which Spring handles HTTP request and redirects the request to required resources. To work with Spring MVC, we need to define it in our web.xml .

<servlet>

<servlet-name>dispatcher</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

### **WebApplicationContext**

The WebApplicationContext is an extension of the plain ApplicationContext that has some extra features necessary for web applications. It differs from a normal ApplicationContext in that it is capable of resolving themes, and that it knows which servlet it is associated with.

### **What is Controller in Spring MVC framework?**

Controllers provide access to the application behavior that you typically define through a service interface. Controllers interpret user input and transform it into a model that is represented to the user by the view. Spring implements a controller in a very abstract way, which enables you to create a wide variety of controllers.

### **What is the role of DispatcherServlet in Spring MVC?**

DispatcherServlet is the servlet by which Spring handles HTTP request and redirects the request to required resources. To work with Spring MVC, we need to define it in our web.xml .

<servlet>

<servlet-name>dispatcher</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

### **What is minimum web.xml configuration to run Spring MVC?**

To run the Spring MVC, we need to define DispatcherServlet, contextConfigLocation and ContextLoaderListener in web.xml. Find the sample web.xml.

<servlet>

<servlet-name>dispatcher</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>dispatcher</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/dispatcher-servlet.xml</param-value>

</context-param>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

### **How to handle views in Spring MVC using XML?**

Ans: To handle views in Spring MVC, we need to configure **InternalResourceViewResolver** bean in spring XML where we need to define prefix and suffix of our views name. Find the sample declaration.

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/pages/"/>

<property name="suffix" value=".jsp"/>

</bean>

### **How to start Spring MVC using spring boot?**

Ans: Spring provides **spring-boot-starter-web** using which we can resolve all Spring MVC required JAR. In our project, we can include it using maven as

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

<version>1.2.2.RELEASE</version>

</dependency>

If we want to use gradle, we use as

dependencies {

compile 'org.springframework.boot:spring-boot-starter-web:1.2.2.RELEASE'

}

### **How to create Controller class in Spring MVC?**

Ans: To create a Controller in Spring MVC, create a class and annotate it with **@Controller** and @RequestMapping. @Controller declares this to be controller and **@RequestMapping** defines the path mapping of controller. Find the sample controller.

@Controller

@RequestMapping("/page")

public class PersonController {

@Autowired

private IPersonService personService;

@RequestMapping("/login")

public String hello(@RequestParam(value="userId", required=false) String userId,

@RequestParam(value="location", required=false) String location,

Model model) {

model.addAttribute("msg", "Hello "+personService.getPersonName() );

model.addAttribute("userId", userId);

model.addAttribute("location", location);

return "result";

}

}

We need to create method annotated with @RequestMapping. Using this mapping request, URL finds the method to execute. **@RequestParam** annotation is used to get request parameter. **Model** class is used to respond values. Using **Model. addAttribute(key,value)** , we send the values to respond back. The URL will become **/page/login** to execute hello() method in the above controller code snippet. The return value of the method is view name. In the above case, result.jsp will be executed.

### **How to access values from Model in JSP.**

Ans: Use JSTL, to retrieve values from Model as

${userId}

${location}

### **How to configure DispatcherServlet without web.xml in Spring MVC?**

Ans: Create a class implementing **WebApplicationInitializer** interface. We need to define **onStartup()** method. Here we can register annotation based application configuration class, servlet and mappings, listener etc. Find the sample WebApplicationInitializer.

public class WebAppInitializer implements WebApplicationInitializer {

public void onStartup(ServletContext servletContext) throws ServletException {

AnnotationConfigWebApplicationContext ctx = new AnnotationConfigWebApplicationContext();

ctx.register(AppConfig.class);

ctx.setServletContext(servletContext);

Dynamic dynamic = servletContext.addServlet("dispatcher", new DispatcherServlet(ctx));

dynamic.addMapping("/");

dynamic.setLoadOnStartup(1);

}

}

In ServletContext instance, we add servlet. Using javax.servlet.ServletRegistration.Dynamic class we define mappings for the servlet.

### **How to define Spring MVC view in @Configuration class without spring XML.**

We need to create a bean in @Configuration class for **UrlBasedViewResolver**. This class has different methods like setPrefix, setSuffix and setViewClass. Find the sample bean definition for UrlBasedViewResolver.

@Bean

public UrlBasedViewResolver setupViewResolver() {

UrlBasedViewResolver resolver = new UrlBasedViewResolver();

resolver.setPrefix("/views/");

resolver.setSuffix(".jsp");

resolver.setViewClass(JstlView.class);

return resolver;

}

### **How to handle Multipart to upload file in Spring MVC?**

Ans: First we need to define a bean **MultipartConfigElement** in our configuration class. Here we can define maximum file size and other configuration related to file upload. We do it as

@Bean

public MultipartConfigElement multipartConfigElement() {

MultipartConfigFactory factory = new MultipartConfigFactory();

factory.setMaxFileSize("128KB");

factory.setMaxRequestSize("128KB");

return factory.createMultipartConfig();

}

In WebApplicationInitializer implementation we need to register **MultipartConfigElement** as

Dynamic dynamic = servletContext.addServlet("dispatcher", new DispatcherServlet(ctx));

dynamic.setMultipartConfig(ctx.getBean(MultipartConfigElement.class));

Now in controller class, the methods must be defined with the<> MultipartFile argument as

@RequestMapping(value="/singleSave", method=RequestMethod.POST )

public @ResponseBody String singleSave(@RequestParam("file") MultipartFile file,

@RequestParam("desc") String desc ){

}

### **What is the role of @EnableWebMvc in Spring MVC.**

Ans: @EnableWebMvc annotation is applied on configuration class with @Configuration annotation. Using @EnableWebMvc, spring enables the MVC related configuration.

### @Controller annotation

The @Controller annotation indicates that a particular class serves the role of a controller. Spring does not require you to extend any controller base class or reference the Servlet API.

### @RequestMapping annotation

@RequestMapping annotation is used to map a URL to either an entire class or a particular handler method.

## **@RequestParam(value="userId", required=false**

## **How to Call Stored procedure in Spring Framework?**

## **Ans:-Using CallableStatement:**

This is similar to plain JDBC where we create CallableStatement from Connection object and use it for calling stored procedure.  
**The Code:**

final String procedureCall = "{call PROC\_TEST(?, ?, ?)}";  
Connection connection = null;  
try {

//Get Connection instance from dataSource  
connection = jdbcTemplate.getDataSource().getConnection();  
CallableStatement callableSt = connection.prepareCall(procedureCall);  
callableSt.setString(1, "Lalit");  
callableSt.setString(2, " Chaudhari");  
callableSt.registerOutParameter(3, Types.VARCHAR);  
  
//Call Stored Procedure  
callableSt.executeUpdate();  
System.out.println(callableSt.getString(3));

}catch (SQLException e) {

e.printStackTrace();

} finally {

if(connection != null)  
try {  
connection.close();  
} catch (SQLException e) {  
e.printStackTrace();  
}  
}

In above code, as we are directly using the connection object, connection should be closed before exiting method.

## **What is JdbcTemplate in Spring? And how to use it?**

## **JdbcTemplate Class**

The JdbcTemplate class executes SQL queries, update statements and stored procedure calls, performs iteration over ResultSets and extraction of returned parameter values. It also catches JDBC exceptions and translates them to the generic, more informative, exception hierarchy defined in the org.springframework.dao package.

Instances of the *JdbcTemplate* class are *threadsafe* once configured. So you can configure a single instance of a *JdbcTemplate* and then safely inject this shared reference into multiple DAOs.

package com.tutorialspoint;

import java.util.List;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.tutorialspoint.StudentJDBCTemplate;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context =

new ClassPathXmlApplicationContext("Beans.xml");

StudentJDBCTemplate studentJDBCTemplate =

(StudentJDBCTemplate)context.getBean("studentJDBCTemplate");

System.out.println("------Records Creation--------" );

studentJDBCTemplate.create("Zara", 11);

studentJDBCTemplate.create("Nuha", 2);

studentJDBCTemplate.create("Ayan", 15);

System.out.println("------Listing Multiple Records--------" );

List<Student> students = studentJDBCTemplate.listStudents();

for (Student record : students) {

System.out.print("ID : " + record.getId() );

System.out.print(", Name : " + record.getName() );

System.out.println(", Age : " + record.getAge());

}

System.out.println("----Updating Record with ID = 2 -----" );

studentJDBCTemplate.update(2, 20);

System.out.println("----Listing Record with ID = 2 -----" );

Student student = studentJDBCTemplate.getStudent(2);

System.out.print("ID : " + student.getId() );

System.out.print(", Name : " + student.getName() );

System.out.println(", Age : " + student.getAge());

}

}

Following is the configuration file **Beans.xml**:

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd ">

<!-- Initialization for data source -->

<bean id="dataSource"

class="org.springframework.jdbc.datasource.DriverManagerDataSource">

<property name="driverClassName" value="com.mysql.jdbc.Driver"/>

<property name="url" value="jdbc:mysql://localhost:3306/TEST"/>

<property name="username" value="root"/>

<property name="password" value="password"/>

</bean>

<!-- Definition for studentJDBCTemplate bean -->

<bean id="studentJDBCTemplate"

class="com.tutorialspoint.StudentJDBCTemplate">

<property name="dataSource" ref="dataSource" />

</bean>

</beans>

Difference between FileSystemResource and ClassPathResource?

* FileSystemResource can locate given spring bean configuration file from the specified path of the file system. Here we can pass either absolute path or relative path.

FileSystemResourceres = new FileSystemResource("src/com/example/cfgs/applicationContext.xml");

* ClassPathResource can locate given spring bean configuration file from directories or jar files that are added to classpath/Build path.

ClassPathResource res = new ClassPathResource("applicationContext.xml");

(But add com.example.cfgs package to build path)

## Spring OAuth:-

OAuth will enable a 3rd party application to obtain limited access to HTTP Service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and HTTP Service or by allowing 3rd party application to obtain access on its own behalf.

1. Resource Owner- An entity capable of granting access to protected resources. When the resource owner is a person , it is referred to as an end user.
2. Resource Server- The server hosting the protected resources, capable of accepting and responding to a protected resource requests using access tokens.
3. **client:**  
   An application making protected resource requests on behalf of the resource owner and with its authorization. It could be a mobile app asking your permission to access your Facebook feeds, a REST client trying to access REST API, a web site [Stackoverflow e.g.] providing an alternative login option using Facebook account.
4. **authorization server:**  
   The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

# Spring Cache

## **Caching Annotations**

* @Cacheable : Triggers cache population
* @CachePut : Updates the cache, without interfering with the method execution
* @CacheEvict : Triggers cache eviction[removing items from cache]
* @Caching : Regroups multiple cache operations to be applied on a method
* @CacheConfig : Shares some common cache-related settings at class-level
* @EnableCaching : Configuration level annotation, enables Caching

### @Cacheable

Used for Cache-population. @Cacheable annotation indicates that the result of invoking a method (or all methods in a class) can be cached.

### @CachePut

Used for Cache-update operation. Method annotated with @CachePut are **always gets executed** and there result gets stored in the cache, eventually overriding any entry with same key in cache. @CachePut, like @Cacheable, supports several attributes, having similar functionality as described above.

#### @CacheEvict

Used for Cache-removal /cache-cleanup operation. @CacheEvict annotation indicates that a method (or all methods on a class) triggers a cache evict operation, removing specific [or all] items from cache. Various attributes provides complete control to enforce the required behavior for cache-eviction.

# @Caching annotation comes handy when you want to specify multiple annotations of the same type, such as @CacheEvict or @CachePut on same method.

# @EnableCaching annotation, usually applied on a @Configuration class, triggers a post processor that inspects every Spring bean for the presence of caching annotations [@Cacheable, @CacheEvict, @CachePut] on public methods. If such an annotation is found, a proxy is automatically created to intercept the method call and handle the caching behavior accordingly.

# @CacheConfig is a class-level annotation which can be used to specify the common caching related settings directly on class level, thus freeing user from duplicating them on each method level. You can of course override the setting specified on class level, on individual method. Common configuration setting that can be specified at class level are cache names, custom KeyGenerator, the custom CacheManager & custom CacheResolve.

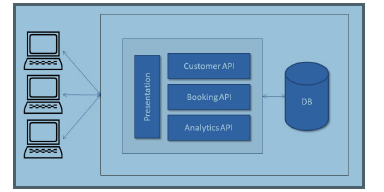
# ****@Transactional****, Spring dynamically creates a proxy that implements the same interface(s) as the class you're annotating. And when clients make calls into your object, the calls are intercepted and the behaviors injected via the proxy mechanism.

# Micro Services with Spring

# Traditional Monolithic Design:- In traditional monolithic design, everything is bundled in the same archive such as Presentation code, Application Logic and Business Logic code, and DAO and related code that interacts with the database files or another source.

# 

**Monolithic Design with Services: -** After SOA, applications started being developed based on services, where each component provides the services to other components or external entities. The following diagram depicts the monolithic application with different services; here services are being used with a Presentation component. All services, the Presentation component, or any other components are bundled together:



# Limitations of Monolithic Application:-

# One Dimensional scalability.

# Release Rollback in case of failure.

# Problems in adopting new technologies.

# Ease of development –could be done better.

# Deployment with micro services build pipeline.

* **Cgroups** = limits how much you can use;
* **namespaces** = limits what you can see (and therefore use)

# Micro Services based Design: - Micro Services are implementation or a service oriented Architecture which provides flexibility to deploy our applications independently which are light weight

# Spring Cloud Sleuth:- Spring Cloud Sleuth implements a distributed tracing solution for Spring Cloud, borrowing heavily from [Dapper](http://research.google.com/pubs/pub36356.html), [Zipkin](https://github.com/openzipkin/zipkin) and HTrace. For most users Sleuth should be invisible, and all your interactions with external systems should be instrumented automatically. You can capture data simply in logs, or by sending it to a remote collector service.

# Client App

# Server App

# Cloud Foundry:- It is a Platform –as – a Service open source service provided by pivotal. We can tail logs based on

# traceid- for tracing a particular request and

# spanid – specific to a microservice

# <https://console.run.pivotal.io/organizations/5a497a93-62bd-4660-a704-bbe7f1a333cb/domains>

# 

# Spring Cloud Eureka:-

# Client App

# Server App

# 

# 

# Hystix and Zuul:-

# Spring Cloud Config server:-

# Kafka vs Rest Template:-

# When we want to multiple publishers and wanted to partition the load, we use Kafka.

# Graph QL:-

# Cloud Native vs Traditional Applcation:-

# Approach to design and build, run the apps

# CI, Container Engines and Orchestrators.

# Improved Speed, Scalability and management cost.

# Devops + CI + Cloud Infra.

# Why Cloud Native:-

# Modularity

# Observability

# Deployability

# Testability

# Disposability

# Replaceability

# 

# 

# 

# 

# Load Balancing:-

# Load Balencers do the routing.

# Distribute incoming traffic to the network by efficiently distributing across multiple servers.

# Reliability and high availability is maintained by redirecting requests only to the servers which are available.

# Ease of use in adding and removing servers in the n/w as per the demand.

# Load Balencing Algorithms – Round Robin, Least Connections, IP Hashing – based on the req sent to the server based on client IP Address.

# Sticky Session – browser will keep the session id in cache and same id is given for every request.

# JHipster:-

# Hystix: - It’s like a catch block, which does some fallback by using circuit breaker design pattern.

# Hystrix.stream key as input in dashboard application to see the status of a particular request.

# Actuator:-

# MicroMeter:-Monitoring applications based on endpoints. It’s like a bridging the gap b/w microservices to support different ways of collecting the information

# Prometheus:-Targets

# Docker

# JPA

* JPA is the EE standard specification for ORM in Java EE. Hibernate is also an implementation of this specification, in that you can use the standard JPA APIs and configure your application to use Hibernate as the provider of the spec under the covers.
* JPA Provides api to manage relational data in java based applications, CRUD Operations.
* JPA is a specification and Hibernate is one of the implementations.
* App server provide the implementation of JPA Specification.
* EntityManagerFactory emf = Persistence.createEntityManagerFactory(persistenceunitName);

EntityManager em = emf.createEntityManager();

em.getTransaction().begin();

em.persist(EntityObj);

em.getTranasaction().commit();em.close();

* ORM is the approach of taking object oriented data and mapping to a relational datastore (e.g. a RDBMS)
* Hibernate is an implementation of an ORM framework(Mapping(XML Configs and annotations) which allows mappings of java objects into tables)

|  |  |
| --- | --- |
| **EntityManagerFactory** | This is a factory class of EntityManager. It creates and manages multiple EntityManager instances. |
| **EntityManager** | It is an Interface, it manages the persistence operations on objects. It works like factory for Query instance. |
| **Entity** | Entities are the persistence objects, stores as records in the database. |
| **EntityTransaction** | It has one-to-one relationship with EntityManager. For each EntityManager, operations are maintained by EntityTransaction class. |
| **Persistence** | This class contain static methods to obtain EntityManagerFactory instance. |
| **Query** | This interface is implemented by each JPA vendor to obtain relational objects that meet the criteria. |

Themeroller

<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/persistence

http://java.sun.com/xml/ns/persistence/persistence\_2\_0.xsd">

<persistence-unit name="Eclipselink\_JPA" transaction-type="RESOURCE\_LOCAL">

<class>com.tutorialspoint.eclipselink.entity.Employee</class>

<properties>

<property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/jpadb"/>

<property name="javax.persistence.jdbc.user" value="root"/>

<property name="javax.persistence.jdbc.password" value="root"/>

<property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>

<property name="eclipselink.logging.level" value="FINE"/>

<property name="eclipselink.ddl-generation" value="create-tables"/>

</properties>

</persistence-unit>

</persistence>

public class CreateEmployee {

public static void main( String[ ] args ) {

EntityManagerFactory emfactory = Persistence.createEntityManagerFactory( "Eclipselink\_JPA" );

EntityManager entitymanager = emfactory.createEntityManager( );

entitymanager.getTransaction( ).begin( );

Employee employee = new Employee( );

employee.setEid( 1201 );

employee.setEname( "Gopal" );

employee.setSalary( 40000 );

employee.setDeg( "Technical Manager" );

entitymanager.persist( employee );

entitymanager.getTransaction( ).commit( );

entitymanager.close( );

emfactory.close( );

}

}

# Hibernate

It provides a powerful query language (HQL) that is similar to SQL. However, HQL is fully object-oriented and understands concepts like inheritance, polymorphism and association.

 Hibernate is an open source project from Red Hat Community and used worldwide. This makes it a better choice than others because learning curve is small and there are tons of online documentations and help is easily available in forums.

 Hibernate is easy to integrate with other Java EE frameworks, it’s so popular that Spring Framework provides built-in support for integrating hibernate with Spring applications.

 Hibernate supports lazy initialization using proxy objects and perform actual database queries only when it’s required.

 Hibernate cache helps us in getting better performance.

 For database vendor specific feature, hibernate is suitable because we can also execute native sql queries.

Some of the important advantages of Hibernate framework over JDBC are:

1. Hibernate removes a lot of boiler-plate code that comes with JDBC API, the code looks more cleaner and readable.
2. Hibernate supports inheritance, associations and collections. These features are not present with JDBC API.
3. Hibernate implicitly provides transaction management, in fact most of the queries can’t be executed outside transaction. In JDBC API, we need to write code for transaction management using commit and rollback. Read more at [JDBC Transaction Management](http://www.journaldev.com/2483/jdbc-transaction-management-and-savepoint-example-tutorial).
4. JDBC API throws SQLException that is a checked exception, so we need to write a lot of try-catch block code. Most of the times it’s redundant in every JDBC call and used for transaction management. Hibernate wraps JDBC exceptions and throw JDBCException or HibernateException un-checked exception, so we don’t need to write code to handle it. Hibernate built-in transaction management removes the usage of try-catch blocks.
5. Hibernate Query Language (HQL) is more object oriented and close to java programming language. For JDBC, we need to write native sql queries.
6. Hibernate supports caching that is better for performance, JDBC queries are not cached hence performance is low.
7. Hibernate provide option through which we can create database tables too, for JDBC tables must exist in the database.
8. Hibernate configuration helps us in using JDBC like connection as well as JNDI DataSource for connection pool. This is very important feature in enterprise application and completely missing in JDBC API.
9. Hibernate supports JPA annotations, so code is independent of implementation and easily replaceable with other ORM tools. JDBC code is very tightly coupled with the application.

###  Name some important interfaces of Hibernate framework?

Some of the important interfaces of Hibernate framework are:

1. **SessionFactory (org.hibernate.SessionFactory)**: SessionFactory is an immutable thread-safe cache of compiled mappings for a single database. We need to initialize SessionFactory once and then we can cache and reuse it. SessionFactory instance is used to get the Session objects for database operations.
2. **Session (org.hibernate.Session)**: Session is a single-threaded, short-lived object representing a conversation between the application and the persistent store. It wraps JDBC java.sql.Connection and works as a factory for org.hibernate.Transaction. We should open session only when it’s required and close it as soon as we are done using it. Session object is the interface between java application code and hibernate framework and provide methods for CRUD operations.
3. **Transaction (org.hibernate.Transaction)**: Transaction is a single-threaded, short-lived object used by the application to specify atomic units of work. It abstracts the application from the underlying JDBC or JTA transaction. A org.hibernate.Session might span multiple org.hibernate.Transaction in some cases.

###  What is hibernate configuration file?

Hibernate configuration file contains database specific configurations and used to initialize SessionFactory. We provide database credentials or JNDI resource information in the hibernate configuration xml file. Some other important parts of hibernate configuration file is Dialect information, so that hibernate knows the database type and mapping file or class details.

###  What is hibernate mapping file?

Hibernate mapping file is used to define the entity bean fields and database table column mappings. We know that JPA annotations can be used for mapping but sometimes XML mapping file comes handy when we are using third party classes and we can’t use annotations.

###  Name some important annotations used for Hibernate mapping?

Hibernate supports JPA annotations and it has some other annotations in org.hibernate.annotations package. Some of the important JPA and hibernate annotations used are:

1. **javax.persistence.Entity**: Used with model classes to specify that they are entity beans.
2. **javax.persistence.Table**: Used with entity beans to define the corresponding table name in database.
3. **javax.persistence.Access**: Used to define the access type, either field or property. Default value is field and if you want hibernate to use getter/setter methods then you need to set it to property.
4. **javax.persistence.Id**: Used to define the primary key in the entity bean.
5. **javax.persistence.EmbeddedId**: Used to define composite primary key in the entity bean.
6. **javax.persistence.Column**: Used to define the column name in database table.
7. **javax.persistence.GeneratedValue**: Used to define the strategy to be used for generation of primary key. Used in conjunction with javax.persistence.GenerationType enum.
8. **javax.persistence.OneToOne**: Used to define the one-to-one mapping between two entity beans. We have other similar annotations as OneToMany, ManyToOne and ManyToMany
9. **org.hibernate.annotations.Cascade**: Used to define the cascading between two entity beans, used with mappings. It works in conjunction with org.hibernate.annotations.CascadeType
10. **javax.persistence.PrimaryKeyJoinColumn**: Used to define the property for foreign key. Used with org.hibernate.annotations.GenericGenerator and org.hibernate.annotations.Parameter

Here are two classes showing usage of these annotations.

|  |  |
| --- | --- |
| Employee.java | |
| 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 | package com.journaldev.hibernate.model;  import javax.persistence.Access;  import javax.persistence.AccessType;  import javax.persistence.Column;  import javax.persistence.Entity;  import javax.persistence.GeneratedValue;  import javax.persistence.GenerationType;  import javax.persistence.Id;  import javax.persistence.OneToOne;  import javax.persistence.Table;    import org.hibernate.annotations.Cascade;    @Entity  @Table(name = "EMPLOYEE")  @Access(value=AccessType.FIELD)  public class Employee {        @Id      @GeneratedValue(strategy = GenerationType.IDENTITY)      @Column(name = "emp\_id")      private long id;        @Column(name = "emp\_name")      private String name;        @OneToOne(mappedBy = "employee")      @Cascade(value = org.hibernate.annotations.CascadeType.ALL)      private Address address;        //getter setter methods  } |
| Address.java | |
| 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 | package com.journaldev.hibernate.model;    import javax.persistence.Access;  import javax.persistence.AccessType;  import javax.persistence.Column;  import javax.persistence.Entity;  import javax.persistence.GeneratedValue;  import javax.persistence.Id;  import javax.persistence.OneToOne;  import javax.persistence.PrimaryKeyJoinColumn;  import javax.persistence.Table;    import org.hibernate.annotations.GenericGenerator;  import org.hibernate.annotations.Parameter;    @Entity  @Table(name = "ADDRESS")  @Access(value=AccessType.FIELD)  public class Address {        @Id      @Column(name = "emp\_id", unique = true, nullable = false)      @GeneratedValue(generator = "gen")      @GenericGenerator(name = "gen", strategy = "foreign", parameters = { @Parameter(name = "property", value = "employee") })      private long id;        @Column(name = "address\_line1")      private String addressLine1;        @OneToOne      @PrimaryKeyJoinColumn      private Employee employee;        //getter setter methods |

<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE hibernate-configuration SYSTEM

"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name="hibernate.dialect">

org.hibernate.dialect.MySQLDialect

</property>

<property name="hibernate.connection.driver\_class">

com.mysql.jdbc.Driver

</property>

<!-- Assume students is the database name -->

<property name="hibernate.connection.url">

jdbc:mysql://localhost/test

</property>

<property name="hibernate.connection.username">

root

</property>

<property name="hibernate.connection.password">

cohondob

</property>

</session-factory>

</hibernate-configuration>

**How will you integrate Hibernate with spring framework?**

- To integrate hibernate with spring framework, the hibernate configuration will go in spring configuration file.  
- The configuration file will look like as below :

<beans>  
<bean id="propertyConfigurer"  
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer"  
p:location="/WEB-INF/jdbc.properties"></bean>   
<!—jdbc.properties database related properties -?  
  
<bean id="dataSource"  
class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close"  
p:driverClassName="${jdbc.driverClassName}"  
p:url="${jdbc.databaseurl}" p:username="${jdbc.username}"  
p:password="${jdbc.password}"></bean>  
  
<bean id="sessionFactory"  
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">  
<property name="dataSource" ref="dataSource"></property>  
<property name="configLocation">  
<value>classpath:hibernate.cfg.xml</value>  
</property>  
<property name="configurationClass">  
<value>org.hibernate.cfg.AnnotationConfiguration</value>  
</property>  
<property name="hibernateProperties">  
<props>  
<prop key="hibernate.dialect">${jdbc.dialect}</prop>  
<prop key="hibernate.show\_sql">true</prop>  
</props>  
</property>  
</bean>  
  
<bean id="employeeDAO" class="com.test.dao.EmployeeDaoImpl"></bean>  
<bean id="employeeManager" class="com.test.service.EmployeeManagerImpl"></bean>  
  
<tx:annotation-driven />  
  
<bean id="transactionManager"  
class="org.springframework.orm.hibernate3.HibernateTransactionManager">  
<property name="sessionFactory" ref="sessionFactory"></property>  
</bean>  
</beans>

first level caching happens on a per session basis whereas second level caching can be shared across multiple sessions.

**First-level cache** always Associates with the **Session object**. Hibernate uses this cache by default. Here, it processes one transaction after another one, means wont process one transaction many times. Mainly it reduces the number of SQL queries it needs to generate within a given transaction. That is instead of updating after every modification done in the transaction, it updates the transaction only at the end of the transaction.

1.2) Second-level cache

**Second-level cache** always associates with the **Session Factory object**. While running the transactions, in between it loads the objects at the Session Factory level, so that those objects will be available to the entire application, not bound to single user. Since the objects are already loaded in the cache, whenever an object is returned by the query, at that time no need to go for a database transaction. In this way the second level cache works. Here we can use query level cache also.

Hibernate Framework comes with a powerful object-oriented query language – Hibernate Query Language (HQL). It’s very similar to SQL except that we use Objects instead of table names, that makes it more close to object oriented programming.

###  what are best practices to follow with Hibernate framework?

Some of the best practices to follow in Hibernate are:

* Always check the primary key field access, if it’s generated at the database layer then you should not have a setter for this.
* By default hibernate set the field values directly, without using setters. So if you want hibernate to use setters, then make sure proper access is defined as @Access(value=AccessType.PROPERTY).
* If access type is property, make sure annotations are used with getter methods and not setter methods. Avoid mixing of using annotations on both filed and getter methods.
* Use native sql query only when it can’t be done using HQL, such as using database specific feature.
* If you have to sort the collection, use ordered list rather than sorting it using Collection API.
* Use named queries wisely, keep it at a single place for easy debugging. Use them for commonly used queries only. For entity specific query, you can keep them in the entity bean itself.
* For web applications, always try to use JNDI DataSource rather than configuring to create connection in hibernate.
* Avoid Many-to-Many relationships, it can be easily implemented using bidirectional One-to-Many and Many-to-One relationships.
* For collections, try to use Lists, maps and sets. Avoid array because you don’t get benefit of lazy loading.
* Do not treat exceptions as recoverable, roll back the Transaction and close the Session. If you do not do this, Hibernate cannot guarantee that in-memory state accurately represents the persistent state.
* Prefer DAO pattern for exposing the different methods that can be used with entity bean
* Prefer lazy fetching for associations.

# 

# STRUTS 2

**## Struts 2 Architecture Flow**

1. User sends a request for the action

2. Container maps the request in the WebController(web.xml) and gets the class name of controller.

3. Container then invokes the controller [StrutsPrepareAndExecuteFilter / Filter Dispatcher (version <=2.1)]

4. Controller gets the information for the action from the ActionMapper

5. Controller invokes the ActionProxy

6. ActionProxy gets the information of action and interceptor stack from the configuration manager which gets the information from the struts.xml file.

7. ActionProxy forwards the request to the ActionInvocation

8. ActionInvocation invokes each interceptors and action

9. A result is generated

10. The result is sent back to the ActionInvocation

11. A HttpServletResponse is generated

12. Response is sent to the user

**## Steps performed in the development of struts Application with Annotations using Maven**

**- web.xml**

- <listener>

<listener-class>org.apache.struts2.tiles.StrutsTilesListener</listener-class>

</listener>

> Listener used to automatically inject ServletContext init parameters so that they don't need to be configured explicitly for tiles integration.

- org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter

> Handles both the preparation and execution phases of the Struts dispatching process.

- Now add the filter mapping and welcome file list.

**-Components**

**1. Action :-** Create an action class which will contain complete business logic and control the interaction between the user, the model, and the view. Actions are the core of the Struts2 framework, as they are for any MVC (Model View Controller) framework. Each URL is mapped to a specific action, which provides the processing logic necessary to service the request from the user.

**2. Interceptors:-** Create interceptors if required, or use existing interceptors. This is part of Controller.

3. **View:-**Create a JSPs to interact with the user to take input and to present the final messages.

**4. Configuration Files :-** Create configuration files to couple the Action, View and Controllers. These files are struts.xml, web.xml, struts.properties.

Here are some of the great features that may force you to consider Struts2:

* **POJO forms and POJO actions** - Struts2 has done away with the Action Forms that were an integral part of the Struts framework. With Struts2, you can use any POJO to receive the form input. Similarly, you can now see any POJO as an Action class.
* **Tag support** - Struts2 has improved the form tags and the new tags allow the developers to write less code.
* **AJAX support** - Struts2 has recognised the takeover by Web2.0 technologies, and has integrated AJAX support into the product by creating AJAX tags, that function very similar to the standard Struts2 tags.
* **Easy Integration** - Integration with other frameworks like spring, Tiles and SiteMesh is now easier with a variety of integration available with Struts2.
* **Template Support** - Support for generating views using templates.
* **Plugin Support** - The core Struts2 behaviour can be enhanced and augmented by the use of plugins. A number of plugins are available for Struts2.
* **Profiling** - Struts2 offers integrated profiling to debug and profile the application. In addition to this, Struts also offers integrated debugging with the help of built in debugging tools.
* **Easy to modify tags** - Tag markups in Struts2 can be tweaked using Freemarker templates. This does not require JSP or java knowledge. Basic HTML, XML and CSS knowledge is enough to modify the tags.
* **Promote less configuration** - Struts2 promotes less configuration with the help of using default values for various settings. You don't have to configure something unless it deviates from the default settings set by Struts2.
* **View Technologies:** - Struts2 has a great support for multiple view options (JSP, Freemarker, Velocity and XSLT)

The above are just the top ten features of Struts 2 that makes it an entreprise ready framework.

## Struts 2 disadvantages

Though Struts 2 comes with a list of great features but I would not forget to mention few negative points about Struts 2 and would need lots of improvments:

* **Bigger learning curve** - To use MVC with Struts, you have to be comfortable with the standard JSP, Servlet APIs and a large & elaborate framework.
* **Poor documentation** - Compared to the standard servlet and JSP APIs, Struts has fewer online resources, and many first-time users find the online Apache documentation confusing and poorly organized.
* **Less transparent** - With Struts applications, there is a lot more going on behind the scenes than with normal Java-based Web applications which makes it difficult to understand the framework.

The **Data Access Object** is basically an object or an interface that provides access to an underlying database or any other persistence storage. DAO contains the implementation of Data calls say insert, update, delete but DTO's are mere accessors ie, getters and setters used to fetch the data.

Struts 1.x Modules(Controller ,Form beans ,Action Forwards ,Config file(struts.xml- Action Servlet) ,Web.xml ,Action Class +POJO , I18n , Aware Interfaces.]

If a class implements the interface com.opensymphony.xwork2.ModelDriven then it needs to return an object from the getModel() method.

**Struts 2 Aware interfaces** are used to put information into the request, response, context or session object. The action class must implement these interfaces to store information so that it can be retrieved from other action class.

Application Aware :- implements ApplicationAware

Implements SEssionAware setSession(), #session.key

Implements ServletRequestAware/ResponseAware # request, setRequest(req)

Implements ParamAware execute() , setParameter(map m)

Interceptors

Interceptors are conceptually the same as servlet filters or the JDKs Proxy class. Interceptors allow for crosscutting functionality to be implemented separately from the action as well as the framework. You can achieve the following using interceptors:

* Providing preprocessing logic before the action is called.
* Providing postprocessing logic after the action is called.
* Catching exceptions so that alternate processing can be performed.

Init() , intercept , destroy()

Token Interceptor

Timer

Execute and Wait

Logger

# Databases

### 

### DDL

**Data Definition Language** (DDL) statements are used to define the database structure or schema. Some examples:

* CREATE - to create objects in the database
* ALTER - alters the structure of the database
* DROP - delete objects from the database
* TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed
* COMMENT - add comments to the data dictionary
* RENAME - rename an object

### DML

**Data Manipulation Language** (DML) statements are used for managing data within schema objects. Some examples:

* SELECT - retrieve data from the a database
* INSERT - insert data into a table
* UPDATE - updates existing data within a table
* DELETE - deletes all records from a table, the space for the records remain
* MERGE - UPSERT operation (insert or update)
* CALL - call a PL/SQL or Java subprogram
* EXPLAIN PLAN - explain access path to data
* LOCK TABLE - control concurrency

### DCL

**Data Control Language** (DCL) statements. Some examples:

* GRANT - gives user's access privileges to database
* REVOKE - withdraw access privileges given with the GRANT command

### TCL

**Transaction Control** (TCL) statements are used to manage the changes made by DML statements. It allows statements to be grouped together into logical transactions.

* COMMIT - save work done
* SAVEPOINT - identify a point in a transaction to which you can later roll back
* ROLLBACK - restore database to original since the last COMMIT
* SET TRANSACTION - Change transaction options like isolation level and what rollback segment to use

Select max(salary) from employee where salary < (select max(salary) from employees;

Select top 1 salary from (Select distinct top 2 salary from employees order by salary desc)

Select \* from employees order by salary desc go

With result as

{

Select salary ,dense\_rank()/rownumber() over (order by salary desc) as denserank from employees

}

Select top 1 salary from result where result.denserank =3.

Recursive CTE , anchor and recursive member.

With employeeCTE as

{

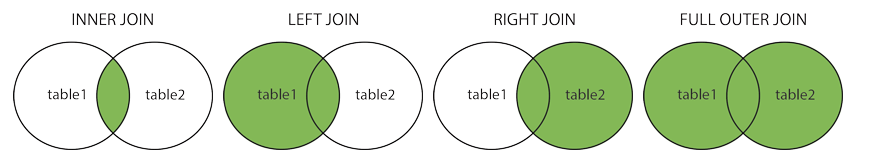
Select \* ,rownumber() over (partition by ID order by ID) as rownumber from employees

}

delete from employeeCTE where rowNumber>1;

select \* ,datediff(month,hiredate,getdate()) as Diff from employees order by hiredate desc;

isnumeric(value) =1

select top 1 departmentname from employees join department on emp.did = dep.deid group by deptname order by count(\*) desc; 

* **(INNER) JOIN**: Returns records that have matching values in both tables
* **LEFT (OUTER) JOIN**: Return all records from the left table, and the matched records from the right table
* **RIGHT (OUTER) JOIN**: Return all records from the right table, and the matched records from the left table
* **FULL (OUTER) JOIN**: Return all records when there is a match in either left or right table

If you want to select all the rows from the **LEFT table**(In our example **Candidate Table**) that have a non null foreign key value(CompanyId in**Candidate Table**is the foreign key) then we use **INNER JOIN**. A query involving an **INNER JOIN**for the **Candidate and Company**Table is shown below.

If you want to select all the rows from the **LEFT table**( In our example **Candidate** Table ) including the rows that have a null foreign key value ( CompanyId in **Candidate**Table is the foreign key ) then we use **LEFT OUTER JOIN**. A query involving a **LEFT OUTER JOIN**for the **Candidate and Company**Table is shown below.

If you want to select all the rows from the **LEFT Table**( In our example **Candidate**Table) that have non null foreign key values plus all the rows from the **RIGHT table**( In our example **Company**Table) including the rows that are not referenced in the **LEFT Table**, then we use **RIGHT OUTER JOIN**. A query involving a **RIGHT OUTER JOIN**for the **Candidate**and **Company Table**is shown below.  
  
**SELECT Cand.CandidateId, Cand.FullName, Cand.CompanyId, Comp.CompanyId, Comp.CompanyName**  
**FROM Candidate Cand**  
**RIGHT OUTER JOIN Company Comp**  
**ON Cand.CompanyId = Comp.CompanyId**

If you want to select all the rows from the **LEFT Table**( In our example **Candidate**Table ) plus all the rows from the **RIGHT table**( In our example**Company**Table ) , then we use **FULL OUTER JOIN**. A query involving a **FULL OUTER JOIN**for the **Candidate and Company**Table is shown below.  
  
**SELECT Cand.CandidateId, Cand.FullName, Cand.CompanyId, Comp.CompanyId, Comp.CompanyName**  
**FROM Candidate Cand**  
**FULL OUTER JOIN Company Comp**  
**ON Cand.CompanyId = Comp.CompanyId**

**SELECT E1.[NAME],E2.[NAME] AS [MANAGER NAME]**  
**FROM EMPLOYEE E1**  
**INNER JOIN EMPLOYEE E2**  
**ON E2.EMPLOYEEID =E1.MANAGERID**

## Query Optimization:-

* Rule based , Cost based

Set autotrace on;

Top 20 or rownum

* Explain plan for

Select \* from product where product\_id=’121’;

* Select \* from table(DBMS\_XPLAN.DISPLAY);- based on CPU cost and index , time taken.
* ORDER BY handles the order the records will be returned.
* GROUP BY finds similar results and clumps them together.
* DISTINCT prevents the return of duplicate data.

Top of Form

# JAVASCRIPT:-

In JavaScript, undefined means a variable has been declared but has not yet been assigned a value, such as:

var TestVar;

alert(TestVar); //shows undefined

alert(typeof TestVar); //shows undefined

null is an assignment value. It can be assigned to a variable as a representation of no value:

var TestVar = null;

alert(TestVar); //shows null

alert(typeof TestVar); //shows object

From the preceding examples, it is clear that undefined and null are two distinct types: undefined is a type itself (undefined) while null is an object.

null === undefined // false

null == undefined // true

null === null // true

and

null = 'value' // ReferenceError

undefined = 'value' // 'value'

Passing in an object, however, passes it in by reference. In this case, any property of that object is accessible within the function.

String, Number, Boolean, Array, Object.

### **AJAX**

AJAX = **A**synchronous **J**avaScript **A**nd **X**ML.

AJAX is not a programming language.

AJAX just uses a combination of:

* A browser built-in XMLHttpRequest object (to request data from a web server)
* JavaScript and HTML DOM (to display or use the data).
* Cannot access Cross Domains.
* Steps for AJAX Call
* An event occurs in a web page (the page is loaded, a button is clicked)
* An XMLHttpRequest object is created by JavaScript
* The XMLHttpRequest object sends a request to a web server
* The server processes the request
* The server sends a response back to the web page
* The response is read by JavaScript
* Proper action (like page update) is performed by JavaScript.

|  |  |
| --- | --- |
| **Method** | **Description** |
| new XMLHttpRequest() | Creates a new XMLHttpRequest object |
| abort() | Cancels the current request |
| getAllResponseHeaders() | Returns header information |
| getResponseHeader() | Returns specific header information |
| open(*method,url,async,user,psw*) | Specifies the request  *method*: the request type GET or POST *url*: the file location *async*: true (asynchronous) or false (synchronous) *user*: optional user name *psw*: optional password |
| send() | Sends the request to the server Used for GET requests |
| send(*string*) | Sends the request to the server. Used for POST requests |
| setRequestHeader() | Adds a label/value pair to the header to be sent |

|  |  |
| --- | --- |
| **Property** | **Description** |
| onreadystatechange | Defines a function to be called when the readyState property changes |
| readyState | Holds the status of the XMLHttpRequest. 0: request not initialized  1: server connection established 2: request received  3: processing request  4: request finished and response is ready |
| responseText | Returns the response data as a string |
| responseXML | Returns the response data as XML data |
| status | Returns the status-number of a request 200: "OK" 403: "Forbidden" 404: "Not Found" For a complete list go to the [Http Messages Reference](https://www.w3schools.com/tags/ref_httpmessages.asp)  1xx: Information  2xx:Successful  3xx:redirection  4xx:Client Error  5xx:Server Error |
| statusText | Returns the status-text (e.g. "OK" or "Not Found") |

function loadDoc() {  
    var xhttp = new XMLHttpRequest();  
    xhttp.onreadystatechange = function() {  
        if (this.readyState == 4 && this.status == 200) {  
            document.getElementById("demo").innerHTML =  
            this.responseText;  
       }  
    };  
    xhttp.open("GET", "ajax\_info.txt", true);  
    xhttp.send();   
}

## The PUT Method

**PUT is used to send data to a server to create/update a resource.**

The difference between POST and PUT is that PUT requests are idempotent. That is, calling the same PUT request multiple times will always produce the same result. In contrast, calling a POST request repeatedly make have side effects of creating the same resource multiple times.

### JQUERY

* jQuery is a lightweight, "write less, do more", JavaScript library.
* The purpose of jQuery is to make it much easier to use JavaScript on your website.
* jQuery takes a lot of common tasks that require many lines of JavaScript code to accomplish, and wraps them into methods that you can call with a single line of code.

The jQuery library contains the following features:

* HTML/DOM manipulation
* CSS manipulation
* HTML event methods
* Effects and animations
* AJAX
* Utilities

The jQuery team knows all about cross-browser issues, and they have written this knowledge into the jQuery library. jQuery will run exactly the same in all major browsers.

CDN (Content Delivery Network)- It will load from the nearest available server.

$(this).hide() - hides the current element.

$("p").hide() - hides all <p> elements.

$(".test").hide() - hides all elements with class="test".

$("#test").hide() - hides the element with id="test".

$(function(){ **OR** $(document).ready(function(){  
  
   *// jQuery methods go here...*  
  
});

|  |  |  |
| --- | --- | --- |
| **Syntax** | **Description** | **Example** |
| $("\*") | Selects all elements |  |

|  |  |  |
| --- | --- | --- |
| $(this) | Selects the current HTML element |  |
| $("p.intro") | Selects all <p> elements with class="intro" |  |
| $("p:first") | Selects the first <p> element |  |
| $("ul li:first") | Selects the first <li> element of the first <ul> |  |
| $("ul li:first-child") | Selects the first <li> element of every <ul> |  |
| $("[href]") | Selects all elements with an href attribute |  |
| $("a[target='\_blank']") | Selects all <a> elements with a target attribute value equal to "\_blank" |  |
| $("a[target!='\_blank']") | Selects all <a> elements with a target attribute value NOT equal to "\_blank" |  |
| $(":button") | Selects all <button> elements and <input> elements of type="button" |  |
| $("tr:even") | Selects all even <tr> elements |  |
| $("tr:odd") | Selects all odd <tr> elements |  |

* text() - Sets or returns the text content of selected elements
* html() - Sets or returns the content of selected elements (including HTML markup)
* val() - Sets or returns the value of form fields
* append() - Inserts content at the end of the selected elements
* prepend() - Inserts content at the beginning of the selected elements
* after() - Inserts content after the selected elements
* before() - Inserts content before the selected elements
* $("p").remove(".test, .demo");- removes all the p elements with class name as test and demo.

**LOAD**

* The jQuery load() method is a simple, but powerful AJAX method. The load() method loads data from a server and puts the returned data into the selected element.
* Load injects the data directly into the DOM. If you don't need this behavior, use get.

$("button").click(function(){  
    $("#div1").load("demo\_test.txt", function(responseTxt, statusTxt, xhr){  
        if(statusTxt == "success")  
            alert("External content loaded successfully!");  
        if(statusTxt == "error")  
            alert("Error: " + xhr.status + ": " + xhr.statusText);  
    });  
});

**GET**

$("button").click(function(){  
    $.get("demo\_test.asp", function(data, status){  
        alert("Data: " + data + "\nStatus: " + status);  
    });  
});

**POST**

$("button").click(function(){  
    $.post("demo\_test\_post.asp",  
    {  
        name: "Donald Duck",  
        city: "Duckburg"  
    },  
    function(data, status){  
        alert("Data: " + data + "\nStatus: " + status);  
    });  
});

## The jQuery noConflict() Method

The noConflict() method releases the hold on the $ shortcut identifier, so that other scripts can use it.

You can of course still use jQuery, simply by writing the full name instead of the shortcut:

**JQuery Mobile**

It is a user interface framework, built on jQuery Core and used for developing responsive websites or applications that are accessible on mobile, tablet, and desktop devices. It uses features of both jQuery and jQueryUI to provide API features for mobile web applications. This tutorial will teach you the basics of jQuery Mobile framework. We will also discuss some detailed concepts related to jQuery Mobile.

**TIAA Code**

* **Spring-cfg.xml:-**

### SOAP VS REST

* SOAP –Simple Object Access Protocol – It provides communication b/w applications over HTTP. It will provide a way to communicate b/w applications running on different OS with different technologies and programming languages.
* <SOAP-ENV:Envelope xmlns:SOAP-ENV = "http://www.w3.org/2001/12/soap-envelope"

SOAP-ENV:encodingStyle = "http://www.w3.org/2001/12/soap-encoding">

<SOAP-ENV:Header>

...

...

</SOAP-ENV:Header>

<SOAP-ENV:Body>

...

...

<SOAP-ENV:Fault>

...

...

</SOAP-ENV:Fault>

...

</SOAP-ENV:Body>

</SOAP\_ENV:Envelope>

