**1.What are the principle concepts of OOPS?**

There are four principle concepts upon which object oriented design and programming rest. They are:

* Abstraction
* Polymorphism
* Inheritance
* Encapsulation (i.e. easily remembered as A-PIE).

**2.What is Abstraction?**

Abstraction is process of **hiding the implementation details** and showing only the functionality.  
Abstraction in java is achieved by using interface and abstract class. Interface give 100% abstraction and abstract class give 0-100% abstraction.

 A class that is declared as **abstract** is known as abstract class.

A method that is declare as abstract and **does not have implementation** is known as abstract method.  
If you define abstract method than class must be abstract.

**3.What is Encapsulation?**

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as as single unit. In encapsulation the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class, therefore it is also known as data hiding.

To achieve encapsulation in Java

* Declare the variables of a class as private.
* Provide public setter and getter methods to modify and view the variables values.

**4. What is Inheritance?**

* Inheritance is the process by which objects of one class acquire the properties of objects of another class.
* A class that is inherited is called a superclass.
* The class that does the inheriting is called a subclass.
* Inheritance is done by using the keyword extends.
* The two most common reasons to use inheritance are:
  + To promote code reuse
  + To use polymorphism

**5. What is Polymorphism?**

* Polymorphism is briefly described as "one interface, many implementations." Polymorphism is a characteristic of being able to assign a different meaning or usage to something in different contexts - specifically, to allow an entity such as a variable, a function, or an object to have more than one form.

**What is the difference between abstraction and encapsulation?**

* **Abstraction** focuses on the outside view of an object (i.e. the interface) **Encapsulation** (information hiding) prevents clients from seeing it’s inside view, where the behavior of the abstraction is implemented.
* **Abstraction** solves the problem in the design side while **Encapsulation** is the Implementation.
* **Encapsulation** is the deliverables of Abstraction. Encapsulation barely talks about grouping up your abstraction to suit the developer needs.

**7.How does Java implement polymorphism?**

(Inheritance, Overloading and Overriding are used to achieve Polymorphism in java).  
Polymorphism manifests itself in Java in the form of multiple methods having the same name.

* In some cases, multiple methods have the same name, but different formal argument lists (overloaded methods).
* In other cases, multiple methods have the same name, same return type, and same formal argument list (overridden methods).

**8.Explain the different forms of Polymorphism.**

There are two types of polymorphism one is **Compile time polymorphism** and the other is run time polymorphism. Compile time polymorphism is method overloading. **Runtime time polymorphism** is done using inheritance and interface.  
**Note**: *From a practical programming viewpoint, polymorphism manifests itself in three distinct forms in Java:*

* *Method overloading*
* *Method overriding through inheritance*
* *Method overriding through the Java interface*

**9.What is runtime polymorphism or dynamic method dispatch?**

In Java, runtime polymorphism or dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime rather than at compile-time. In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

**10.What is Dynamic Binding?**

Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic binding (also known as late binding) means that the code associated with a given procedure call is not known until the time of the call at run-time. It is associated with polymorphism and inheritance.

**11.What is method overloading?**

Method Overloading means to have two or more methods with same name in the same class with different arguments. The benefit of method overloading is that it allows you to implement methods that support the same semantic operation but differ by argument number or type.  
**Note**:

* *Overloaded methods MUST change the argument list*
* *Overloaded methods CAN change the return type*
* *Overloaded methods CAN change the access modifier*
* *Overloaded methods CAN declare new or broader checked exceptions*
* *A method can be overloaded in the same class or in a subclass*

**12.What is method overriding?**

Method overriding occurs when sub class declares a method that has the same type arguments as a method declared by one of its superclass. The key benefit of overriding is the ability to define behavior that’s specific to a particular subclass type.  
**Note**:

* *The overriding method cannot have a more restrictive access modifier than the method being overridden (Ex: You can’t override a method marked public and make it protected).*
* *You cannot override a method marked final*
* *You cannot override a method marked static*

A class can't be declared as protected. Only methods can be declared as protected.

**Local variables** are those which are declared within a block of code like methods. Local variables should be initialized before accessing them.

**What is the access scope of a protected method?-** A protected method can be accessed by the classes within the same package or by the subclasses of the class in any package.

You should declare your class as **final**. But you can't define your class as final, if it is an abstract class. A class declared as final can't be extended by any other class. java.lang.String, java.lang.Math are final classes

**final** is a modifier which can be applied to a class or a method or a variable. final class can't be inherited, final method can't be overridden and final variable can't be changed.   
  
Finally is an exception handling code section which gets executed whether an exception is raised or not by the try block code segment.   
  
finalize() is a method of Object class which will be executed by the JVM just before garbage collecting object to give a final chance for resource releasing activity.

We cannot declare top level class as static, but only inner class can be declared static.

Static variables are class level variables where all objects of the class refer to the same variable. If one object changes the value then the change gets reflected in all the objects.

Can a abstract class be declared final? Not possible. An abstract class without being inherited is of no use and hence will result in compile time error.

Abstract classes can't be instantiated.

Public and abstract are the only applicable modifiers for method declaration in an interface.

Yes an Interface can inherit another Interface, for that matter an Interface can extend more than one Interface.

Can a class be defined inside an Interface –Yes

Can an Interface be defined inside a class- Yes

A public class may be accessed outside of its package. A non-public class may not be accessed outside of its package.

A field variable is a variable that is declared as a member of a class. A local variable is a variable that is declared local to a method.

Only public and abstract modifiers are allowed for methods in interfaces.

An interface is a description of a set of methods that conforming implementing classes must have.  
Note: You can’t mark an interface as final. Interface variables must be static. An Interface cannot extend anything but another interfaces.

Interfaces may have member variables, but these are implicitly public, static, and final- in other words, interfaces can declare only constants, not instance variables that are available to all implementations and may be used as key references for method arguments for example.

Only public and abstract modifiers are allowed for methods in interfaces

**Marker interfaces** are those which do not declare any required methods, but signify their compatibility with certain operations. The *java.io.Serializable* interface and *Cloneable* are typical marker interfaces. These do not contain any methods, but classes must implement this interface in order to be serialized and de-serialized.

Class methods **are methods which are declared as static**. The method can be called without creating an  instance of the class. Instance methods are **not declared as static**.

List interface--The List interface provides support for ordered collections of objects. Lists may contain duplicate elements.

* **ArrayList** : Resizable-array implementation of the List interface. The best all-around implementation of the List interface.
* **Vector** : Synchronized resizable-array implementation of the List interface with additional "legacy methods."
* **LinkedList** : Doubly-linked list implementation of the List interface. May provide better performance than the ArrayList implementation if elements are frequently inserted or deleted within the list. Useful for queues and double-ended queues (deques).

**What are the advantages of ArrayList over arrays ?** It can grow dynamically It provides more powerful insertion and search mechanisms than arrays.

**How do you decide when to use ArrayList and When to use LinkedList?**

If you need to support random access, without inserting or removing elements from any place other than the end, then ArrayList offers the optimal collection. If, however, you need to frequently add and remove elements from the middle of the list and only access the list elements sequentially, then LinkedList offers the better implementation.

**Set interface-**Sets do not allow duplicate elements, Contains no methods other than those inherited from Collection, It adds the restriction that duplicate elements are prohibited ,Two Set objects are equal if they contain the same elements

* HashSet
* TreeSet
* LinkedHashSet
* EnumSet
* sortedSet
* **HashSet-**A HashSet is an unsorted, unordered Set. It uses the hashcode of the object being inserted (so the more efficient your hashcode() implementation the better access performance you’ll get).
* **HashSet is based on hashMap and can contain null element.**
* **Use this class when you want a collection with no duplicates and you don’t care about order when you iterate through it.**
* **TreeSet**
* TreeSet is a Set implementation that keeps the elements in sorted order. The elements are sorted according to the natural order of elements or by the comparator provided at creation time.
* **EnumSet ?**
* An EnumSet is a specialized set for use with enum types, all of the elements in the EnumSet type that is specified, explicitly or implicitly, when the set is created.

**Map Interface**

* A map is an object that stores associations between keys and values (key/value pairs).
* Given a key, you can find its value. Both keys and values are objects.
* The keys must be unique, but the values may be duplicated.
* Some maps can accept a null key and null values, others cannot.

1. HashMap
2. HashTable
3. TreeMap
4. EnumMap

**TreeMap**

TreeMap actually implements the SortedMap interface which extends the Map interface. In a TreeMap the data will be sorted in ascending order of keys according to the natural order for the key's class, or by the comparator provided at creation time. TreeMap is based on the Red-Black tree data structure.

**How do you decide when to use HashMap and when to use TreeMap ?**

For inserting, deleting, and locating elements in a Map, the HashMap offers the best alternative. If, however, you need to traverse the keys in a sorted order, then TreeMap is your better alternative. Depending upon the size of your collection, it may be faster to add elements to a HashMap, then convert the map to a TreeMap for sorted key traversal.

**Difference between HashMap and Hashtable ?**

|  |  |
| --- | --- |
| **HashMap** | **Hashtable** |
| HashMap lets you have null values as well as one null key. | HashTable  does not allows null values as key and value. |
| The iterator in the HashMap is fail-safe (If you change the map while iterating, you’ll know).  U will get java.util.ConcurrentModificationException | The enumerator for the Hashtable is not fail-safe. |
| HashMap is unsynchronized. | Hashtable is synchronized. |

**Note**: **Only one NULL is allowed as a key in HashMap. HashMap does not allow multiple keys to be NULL. Nevertheless, it can have multiple NULL values.**

**Comparable interface ?**

The Comparable interface is used to sort collections and arrays of objects using the Collections.sort() and java.utils.Arrays.sort() methods respectively. The objects of the class implementing the Comparable interface can be ordered.

The Comparable interface in the generic form is written as follows:

interface Comparable<T>

where T is the name of the type parameter.  
  
All classes implementing the Comparable interface must implement the compareTo() method that has the return type as an integer. The signature of the compareTo() method is as follows:

int i = object1.compareTo(object2)

If object1 < object2: The value of i returned will be negative.

If object1 > object2: The value of i returned will be positive.

If object1 = object2: The value of i returned will be zero.

|  |  |
| --- | --- |
| **Q:** | **Q13: How HashMap works in Java?**  **A:** A HashMap in Java stores key-value pairs. The HashMap requires a hash function and uses hashCode and equals methods, in order to put and retrieve elements to and from the collection respectively. When the put method is invoked, the HashMap calculates the hash value of the key and stores the pair in the appropriate index inside the collection. If the key exists, its value is updated with the new value. Some important characteristics of a HashMap are its capacity, its load factor and the threshold resizing.  **Q14: What is the importance of hashCode() and equals() methods?**  **A:** A HashMap in Java uses the hashCode and equals methods to determine the index of the key-value pair. These methods are also used when we request the value of a specific key. If these methods are not implemented correctly, two different keys might produce the same hash value and thus, will be considered as equal by the collection. Furthermore, these methods are also used to detect duplicates. Thus, the implementation of both methods is crucial to the accuracy and correctness of the HashMap.  **What is HashMap and Map?** |
| **A:** | Map is Interface and Hashmap is class that implements that.   * What is the key requirement for an object to be used as a key in hash based collection e.g. HashMap or Hashtable or ConcurrentHashMap?   The object should implement both equals() and hashcode() methods.   * What are immutable classes in Java? How to make immutable class in Java?   Immutable classes are classes, whose objects once created cannot be modified once created. Any modification to these immutable objects will result in another immutable object. Example classes are String and StringBuffer classes in Java.  To make an immutable class the class should be made final. All the fields in the class should also be made final and utmost care should be taken so as to ascertain that object reference must not leak during construction phase of the object.   * What happens internally when you invoke get(Key key) method in hash based collections like HashMap and Hashtable?   The following steps happen:   * + - The Key.hashcode() method gets invoked to get the bucket location in the array backed Hash collections.     - In the backing array the keys and values are stored in an internal class called as Entry. If there is only one Entry at the bucket location, the value is returned. If due to some reasons two keys have same hashcode (this happens), the bucket location will have two entries and it forms a sort of a linked list data structure. In that case, it traverses through the list comparing keys in each entry using keys.equals() until it return true. Once true is returned, the value is returned. * Why should an object which is to be used as key in collection classes need to be immutable?   This is required because the hashcode() method when being called should always return the same value.   * Have you heard of ConcurrentHashMap, if so what is it?   This is an alternative to Hashtable in Java and as the name suggests, it is synchronized, thread safe and highly usable in multi-threaded programming.   * How to make HashMap synchronized in nature?   You can use Collections.synchronizedMap(HashMap). This will return a collection very much similar to Hashtable.   * What are the difference/similarities between HashSet and TreeSet?   Some of the difference are:-   * + HashSet is faster than TreeSet with regards to performance.   + HashSet doesn’t preserve ordering but TreeSet does.   + HashSet allows null objects but TreeSet doesn’t.   + HashSet is backed by HashMap and TreeSet is backed by TreeMap.   Some of the similarities are:-   * + Both are thread-safe and synchronized in nature.   + Both implement Set interface |

## [What does synchronized means in Hashtable context?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "synchronized-Hashtable)

## Synchronized means only one thread can modify a hash table at one point of time. Any thread before performing an update on a hashtable will have to acquire a lock on the object while others will wait for lock to be released.

## [How can we make Hashmap synchronized?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "Hashmap-synchronized)

HashMap can be synchronized by Map m = Collections.synchronizedMap(hashMap);

## [What is the Difference between Enumeration and Iterator interface?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "Difference-between-Enumeration-Iterator)

Enumeration and Iterator are the interface available in java.util package. The functionality of Enumeration interface is duplicated by the Iterator interface. New implementations should consider using Iterator in preference to Enumeration. Iterators differ from enumerations in following ways:

1. Enumeration contains 2 methods namely hasMoreElements() & nextElement() whereas Iterator contains three methods namely hasNext(), next(),remove().
2. Iterator adds an optional remove operation, and has shorter method names. Using remove() we can delete the objects but Enumeration interface does not support this feature.
3. Enumeration interface is used by legacy classes. Vector.elements() & Hashtable.elements() method returns Enumeration. Iterator is returned by all Java Collections Framework classes. java.util.Collection.iterator() method returns an instance of Iterator.

## [What is the difference between Sorting performance of Arrays.sort() vs Collections.sort() ? Which one is faster? Which one to use and when?](http://www.fromdev.com/2008/05/java-collections-questions.html" \l "15)

Many developers are concerned about the performance difference between java.util.Array.sort() java.util.Collections.sort() methods. Both methods have same algorithm the only difference is type of input to them. Collections.sort() has a input as List so it does a translation of List to array and vice versa which is an additional step while sorting. So this should be used when you are trying to sort a list. Arrays.sort is for arrays so the sorting is done directly on the array. So clearly it should be used when you have a array available with you and you want to sort it.

**If an Employee class is present and its objects are added in an arrayList. Now I want the list to be sorted on the basis of the employeeID of Employee class. What are the steps?**

Ans) 1) Implement Comparable interface for the Employee class and override the compareTo(Object obj) method in which compare the employeeID

2) Now call Collections.sort( ArrayList) method and pass list as an argument.

*Now consider that Employee class is a jar file.*

1) Since Comparable interface cannot be implemented, create Comparator and override the compare(Object obj, Object obj1) method .

2) Call Collections.sort() on the list and pass comparator as an argument.

**Exceptions**

java.lang.Object ,java.lang.Throwable ,java.lang.Exception ,java.lang.Error

* **Checked exceptions**: A checked exception is some subclass of Exception (or Exception itself), excluding class RuntimeException and its subclasses. Each method must either handle all checked exceptions by supplying a catch clause or list each unhandled checked exception as a thrown exception.

Checked exceptions are checked at compile-time. It means if a method is throwing a checked exception then it should handle the exception using [try-catch block](http://beginnersbook.com/2013/04/try-catch-in-java/) or it should declare the exception using [throws keyword](http://beginnersbook.com/2013/04/difference-between-throw-and-throws-in-java/), otherwise the program will give a compilation error. It is named as **checked exception** because these exceptions are **checked** at Compile time.

* SQLException
* IOException
* DataAccessException
* ClassNotFoundException
* InvocationTargetException
* **Unchecked exceptions:** All Exceptions that extend the RuntimeException class are unchecked exceptions. Class Error and its subclasses also are unchecked.

Unchecked exceptions are not checked at compile time. It means if your program is throwing an unchecked exception and even if you didn’t handle/declare that exception, the program won’t give a compilation error. Most of the times these exception occurs due to the bad data provided by user during the user-program interaction. It is up to the programmer to judge the conditions in advance, that can cause such exceptions and handle them appropriately. All Unchecked exceptions are direct sub classes of **RuntimeException** class.

* NullPointerException
* ArrayIndexOutOfBoundsException
* ArithmeticException
* IllegalArgumentException

**Immutable classes** are Java classes whose objects cannot be modified once created. Any modification in Immutable objects result in new object. For example is [String is immutable in Java](http://javarevisited.blogspot.sg/2010/10/why-string-is-immutable-in-java.html)

**What is the difference between throw and throws?**   
  
Throw is used to explicitly raise a exception within the program, the statement would be throw new Exception(); throws clause is used to indicate the exceptions that are not handled by the method. It must specify this behavior so the callers of the method can guard against the exceptions.  
  
Throws is specified in the method signature. If multiple exceptions are not handled, then they are separated by a comma. the statement would be as follows: public void doSomething() throws IOException,MyException{}

**Differentiate between Checked Exceptions and Unchecked Exceptions?**   
  
Checked Exceptions are those exceptions which should be explicitly handled by the calling method. Unhandled checked exceptions results in compilation error.   
  
Unchecked Exceptions are those which occur at runtime and need not be explicitly handled. RuntimeException and it's subclasses, Error and it's subclasses fall under unchecked exceptions.

**What are the constraints imposed by overriding on exception handling?**  
An overriding method in a subclass May only throw exceptions declared in the parent class or children of the exceptions declared in the parent class.

**Where does Exception stand in the Java tree hierarchy?**

java.lang.Object ,java.lang.Throwable ,java.lang.Exception ,java.lang.Error

**Why Errors are Not Checked?** **Why Runtime Exceptions are Not Checked?**

The *runtime exception* classes (RuntimeException and its subclasses) are exempted from compile-time checking

**Errors** are also unchecked exception & the programmer is not required to do anything with these. In fact it is a bad idea to use a try-catch clause for Errors. Most often, recovery from an Error is not possible & the program should be allowed to terminate. Examples include OutOfMemoryError, StackOverflowError

**Thread**

Threaded programming is normally used when a program is required to do more than one task at the same time up control over thread execution indefinitely so that other threads can run.

**Why are there separate wait and sleep methods?**

The static Thread.sleep(long) method maintains control of thread execution but delays the next action until the sleep time expires. The wait method gives

**What are the two ways of creating thread?**

1. **Extend the Thread class** and override the run() method in your class. Create an instance of the subclass and invoke the start() method on it, which will create a new thread of execution. e.g.

public class NewThread extends Thread{

public void run(){   
// the code that has to be executed in a separate new thread goes here  
}   
public static void main(String [] args){   
NewThread c = new NewThread();   
c.start();   
}

}

1. **Implements the Runnable interface.**The class will have to implement the run() method in the Runnable interface. Create an instance of this class. Pass the reference of this instance to the Thread constructor a new thread of execution will be created. e.g. class

public class NewThread implements Runnable{

public void run(){   
// the code that has to be executed in a separate new thread goes here  
}   
public static void main(String [] args){   
NewThread c = new NewThread();   
Thread t = new Thread(c);  
t.start();  
}

}

**What are the different states of a thread's lifecycle?**

1) **New** – When a thread is instantiated it is in New state until the start() method is called on the thread instance. In this state the thread is not considered to be alive.   
2) **Runnable** – The thread enters into this state after the start method is called in the thread instance. The thread may enter into the Runnable state from Running state. In this state the thread is considered to be alive.   
3) **Running** – When the thread scheduler picks up the thread from the Runnable thread’s pool, the thread starts running and the thread is said to be in Running state.  
 4) **Waiting/Blocked/Sleeping** – In these states the thread is said to be alive but not runnable. The thread switches to this state because of reasons like wait method called or sleep method has been called on the running thread or thread might be waiting for some i/o resource so blocked.

5) **Dead** – When the thread finishes its execution i.e. the run() method execution completes, it is said to be in dead state. A dead state cannot be started again. If a start() method is invoked on a dead thread a runtime exception will occur.+

**What is use of synchronized keyword?**

synchronized keyword can be applied to static/non-static methods or a block of code. Only one thread at a time can access synchronized methods and if there are multiple threads trying to access the same method then other threads have to wait for the execution of method by one thread. Synchronized keyword provides a lock on the object and thus prevents race condition. E.g.

public void synchronized method(){}    
public void synchronized staticmethod(){}  
public void myMethod(){

            synchronized (this){             // synchronized keyword on block of  code  
            }

}

**What is the difference when the synchronized keyword is applied to a static method or to a non static method?**

When a synch non static method is called a lock is obtained on the object. When a synch static method is called a lock is obtained on the class and not on the object. The lock on the object and the lock on the class donâ€™t interfere with each other. It means, a thread accessing a synch non static method, then the other thread can access the synch static method at the same time but can’t access the synch non static method.

**What is the difference between yield() and sleep()?**

**yield()** allows the current thread to release its lock from the object and scheduler gives the lock of the object to the other thread with same priority.

**sleep()** allows the thread to go to sleep state for x milliseconds. When a thread goes into sleep state it doesn’t release the lock.

**What is the difference between wait() and sleep()?**

**wait()** is a method of Object class**. sleep()** is a method of Object class.

**sleep()** allows the thread to go to sleep state for x milliseconds. When a thread goes into sleep state it doesn’t release the lock. **wait()** allows thread to release the lock and goes to suspended state. The thread is only active when a notify() or notifAll() method is called for the same object.

**What is difference between notify() and notfiyAll()?**

**notify( )** wakes up the first thread that called **wait( )** on the same object. **notifyAll( )** wakes up all the threads that called **wait( )** on the same object. The highest priority thread will run first.

**What happens if a start method is not invoked and the run method is directly invoked?**

If a thread has been instantiated but not started its is said to be in new state. Unless until a start() method is invoked on the instance of the thread, it will not said to be alive. If you do not call a start() method on the newly created thread instance thread is not considered to be alive. If the start() method is not invoked and the run() method is directly called on the Thread instance, the code inside the run() method will not run in a separate new thread but it will start running in the existing thread.

**When jvm starts up, which thread will be started up first?**

Ans) When jvm starts up the thread executing main method is started.

**What are the daemon threads?**

Ans) Daemon thread are service provider threads run in the background,these not used to run the application code generally.When all user threads(non-daemon threads) complete their execution the jvm exit the application whatever may be the state of the daemon threads. Jvm does not wait for the daemon threads to complete their execution if all user threads have completed their execution.

To create Daemon thread set the daemon value of Thread using setDaemon(boolean value) method. By default all the threads created by user are user thread. To check whether a thread is a Daemon thread or a user thread use isDaemon() method.

Example of the Daemon thread is the Garbage Collector run by jvm to reclaim the unused memory by the application. The Garbage collector code runs in a Daemon thread which terminates as all the user threads are done with their execution.

**How many locks does an object have?**

Ans) Each object has only one lock.

**Can a class have both Synchronized and non-synchronized methods?**

Ans) Yes a class can have both synchronized and non-synchronized methods

## [Why do we need run() & start() method both. Can we achieve it with only run method?](http://www.fromdev.com/2008/05/java-threading-questions.html" \l "why-do-we-need-run-----start---method-both--can-we-achieve-it-with-only-run-method-" \o "Why do we need run() & start() method both. Can we achieve it with only run method?)

We need run() & start() method both because JVM needs to create a separate thread which can not be differentiated from a normal method call. So this job is done by start method native implementation which has to be explicitly called. Another advantage of having these two methods is we can have any object run as a thread if it implements Runnable interface. This is to avoid Java’s multiple inheritance problems which will make it difficult to inherit another class with Thread.

## [What happens when I make a static method as synchronized?](http://www.fromdev.com/2008/05/java-threading-questions.html" \l "what-happens-when-i-make-a-static-method-as-synchronized-" \o "What happens when I make a static method as synchronized?)

Synchronized static methods have a lock on the class "Class", so when a thread enters a synchronized static method, the class itself gets locked by the thread monitor and no other thread can enter any static synchronized methods on that class. This is unlike instance methods, as multiple threads can access "same synchronized instance methods" at same time for different instances.

## [What is a deadlock?](http://www.fromdev.com/2008/05/java-threading-questions.html" \l "what-is-a-deadlock-" \o "What is a deadlock?)

Deadlock is a situation where two or more threads are blocked forever, waiting for each other. This may occur when two threads, each having a lock on one resource, attempt to acquire a lock on the other's resource. Each thread would wait indefinitely for the other to release the lock, unless one of the user processes is terminated. In terms of Java API, thread deadlock can occur in following conditions:

* When two threads call Thread.join() on each other.
* When two threads use nested synchronized blocks to lock two objects and the blocks lock the same objects in different order.

**[What is immutable object?](http://www.fromdev.com/2008/05/java-threading-questions.html" \l "what-is-immutable-object--how-does-it-help-on-writing-concurrent-application-" \o "What is immutable object? How does it help on writing concurrent application?)**

An object is considered immutable if its state cannot change after it is constructed

**Java 5 features**

* **Enhanced For loop**

Basic syntax

– for(Type **item**: **collectionOrArrayOfTypes**) {

doSomethingWith(**item**);

}

for(String entry: entries) {

System.out.println(entry);

}

* **Generics**

System allows a type or method to operate on objects of various types while providing compile-time type safety. It adds compile-time type safety to the Collections Framework and eliminates the drudgery of casting. Array doesn't support Generics

Declare the data structure with the type(s) in angle brackets immediately after class name

– **ArrayList<String> myStrings = new ArrayList<String>();**

**What is Bounded and Unbounded wildcards in Generics ?**  
  
 Two kinds of Bounded wildcards <? extends T> which impose an upper bound by ensuring that type must be sub class of T and <? super T> where its imposing lower bound by ensuring Type must be super class of T. This Generic Type must be instantiated with Type within bound otherwise it will result in compilation error. On the other hand <?> represent and unbounded type because <?> can be replace with any Type

Main difference between raw type and parametrized type List<Object> is that, [compiler](http://javarevisited.blogspot.sg/2011/12/jre-jvm-jdk-jit-in-java-programming.html) will not check type-safety of raw type at compile time but it will do that for parametrized type and by using Object as Type it inform compiler that it can hold any Type of Object e.g. String or Integer. This Java Generics interview question is based on correct understanding of raw type in Generics

* **Autoboxing**

Autoboxing means Java compiler will automatically convert primitive to object wherever necessary. for example if you are writing a method which accept an int argument and you are passing an Integer object, Unlike previous Java version,  Java compiler will not [throw](http://javarevisited.blogspot.sg/2012/02/difference-between-throw-and-throws-in.html) any compile time error, Instead it will convert Integer to int by using unboxing in Java. On the other hand if method expect an Double Object as argument and you pass double primitive then Java will convert primitive to Double object using Autoboxing in Java

**Autoboxing example**

ArrayList<Integer> intList = new ArrayList<Integer>();  
intList.add(1); //autoboxing - primitive to object  
int number = intList.get(0); // unboxing

* **Varargs**

Provides flexibility for formatting, math, and other special-purposes methods

* **Static import**

In order to access static members, it is necessary to qualify references with the class they came from. For example, one must say:

double r = Math.cos(Math.PI \* theta);

import static java.lang.Math.PI;

double r = cos(PI \* theta);

**Java 7 features**

* Diamond Operator

Old----

Map<String, List<Trade>> trades = new TreeMap<String, List<Trade>> ();

New Diamond operator <> ---

Map<String, List<Trade>> trades = new TreeMap **<>** ();

* **Multi-catch**

Catch (ExceptionOne | ExceptionTwo | ExceptionThree e)

* **try-with-resources statement**
* **Using strings in switch statements**

**What is lazy loading?**

Lazy initialization has two objectives:

* delay an expensive operation until it's absolutely necessary
* store the result of that expensive operation, such that you won't need to repeat it again

lazy loading means not creating an object until the first time it is accessed. Lazy loading typically looks like this:  
public class Example {  
private Vector data = null;  
public Vector getData() {  
if (this.data == null) {  
this.data = new Vector();  
// Load data into vector …  
}  
return this.data;  
}  
}  
This technique is most useful when you have large hierarchies of objects (such as a product catalog). You can lazy-load subordinate objects as you navigate down the hierarchy, and thereby only create objects when you need them.

**Initialization on Demand Holder** ([design pattern](http://en.wikipedia.org/wiki/Design_pattern_(computer_science))) idiom is a [lazy-loaded](http://en.wikipedia.org/wiki/Lazy_initialization) [singleton](http://en.wikipedia.org/wiki/Singleton_pattern). In all versions of Java, the idiom enables a safe, highly concurrent lazy initialization with good performance.[[1]](http://en.wikipedia.org/wiki/Initialization-on-demand_holder_idiom#cite_note-1)

public class Something {

private Something() {}

private static class LazyHolder {

private static final Something INSTANCE = new Something();

}

public static Something getInstance() {

return LazyHolder.INSTANCE;

}

}

Singleton pattern Example connection Factory

public class ConnectionFactory {  
        String driverName = "com.mysql.jdbc.Driver";  
        String conUrl = "jdbc:mysql://192.168.10.13:3306/onlinexamination";  
        String dbUser = "root";  
        String dbPwd = "root";  
  
        private static ConnectionFactory connectionFactory = null;  
  
        private ConnectionFactory() {  
                try {  
                        Class.forName(driverName);  
                } catch (ClassNotFoundException e) {  
                        e.printStackTrace();  
                }  
        }  
  
        public Connection getConnection()  {  
                Connection conn = null;  
                try {  
                        conn = DriverManager.getConnection(conUrl, dbUser, dbPwd);  
  
                } catch (Exception e) {  
                        e.toString();  
                }  
                return conn;  
        }  
  
        public static ConnectionFactory getInstance() {  
                if (connectionFactory == null) {  
                        connectionFactory = new ConnectionFactory();  
                }  
                return connectionFactory;  
        }  
}

### Lazy Initialization

Lazy initialization method to implement Singleton pattern creates the instance in the global access method.

package com.journaldev.singleton;

public class LazyInitializedSingleton {

private static LazyInitializedSingleton instance;

private LazyInitializedSingleton(){}

public static LazyInitializedSingleton getInstance(){

if(instance == null){

instance = new LazyInitializedSingleton();

}

return instance;

}

}

this implementation works fine incase of single threaded environment but when it comes to multithreaded systems, it can cause issues if multiple threads are inside the if loop at the same time. It will destroy the singleton pattern and both threads will get the different instances of singleton class.

### Thread Safe Singleton

The easier way to create a thread-safe singleton class is to make the global access method [synchronized](http://www.journaldev.com/1061/java-synchronization-and-thread-safety-tutorial-with-examples), so that only one thread can execute this method at a time

package com.journaldev.singleton;

public class ThreadSafeSingleton {

private static ThreadSafeSingleton instance;

private ThreadSafeSingleton(){}

public static synchronized ThreadSafeSingleton getInstance(){

if(instance == null){

instance = new ThreadSafeSingleton();

} return instance; }

}

* **Creational Patterns - Factory Pattern**

Factory of Of classes. if we have a super class and n sub-classes, and based on data provided, we have to return the object of one of the sub-classes, we use a factory pattern.

E.g. Person – Gender –Male – Female – Mr/Mrs

The Factory patterns can be used in some cases:  
1. When a class does not know which class of objects it must create.  
2. A class specifies its sub-classes to specify which objects to create.  
3. when you have to create an object of any one of sub-classes depending on the data provided.

* **Creational Patterns - Abstract Factory Pattern**

This pattern is one level of abstraction higher than factory pattern. the abstract factory returns the factory of classes. Like Factory pattern returned one of the several sub-classes, this returns such factory which later will return one of the sub-classes.

E.g. Suppose we need to get the specification of various parts of a computer based on which work the computer will be used for.  
The different parts of computer are, say Monitor, RAM and Processor. The different types of computers are PC, Workstation and Server.

public abstract class Computer {

public abstract Parts getRAM();

public abstract Parts getProcessor();

}

public class PC extends Computer {

public Parts getRAM() {  
return new Parts("512 MB");  
}

}

public class Workstation extends Computer {

}

ComputerType type = new ComputerType();

Computer computer = type.getComputer("Server");  
System.out.println("Monitor: "+computer.getMonitor().getSpecification());  
System.out.println("RAM: "+computer.getRAM().getSpecification());  
System.out.println("Processor: "+computer.getProcessor().getSpecification());

public Computer getComputer(String computerType) {

if (computerType.equals("PC"))  
comp = new PC();  
else if(computerType.equals("Workstation"))  
comp = new Workstation();  
else if(computerType.equals("Server"))  
comp = new Server();

return comp;

}

**use Abstract Factory Pattern**  
One of the main advantages of Abstract Factory Pattern is that it isolates the concrete classes that are generated. The names of actual implementing classes are not needed to be known at the client side. Because of the isolation, you can change the implementation from one factory to another.

Override hashcode method

**Question: What will happen if you put return statement or System.exit () on try or catch block? Will finally block execute?**

The finally block will always execute. This question challenge that concept by putting a return statement in the try or catch block or calling System.exit from try or catch block. Answer of this tricky question in Java is that finally block will execute even if you put a return statement in the try block or catch block but finally block won't run if you call System.exit form try or catch.

**Question: Can you override a private or static method in Java?**

[you can not override a private or static method in Java](http://java67.blogspot.sg/2012/08/can-we-override-static-method-in-java.html), if you create a similar method with same return type and same method arguments in child class then it will hide the super class method, this is known as method hiding. Similarly, you cannot override a private method in sub class because it's not accessible there, what you do is create another private method with the same name in the child class.

**Question: What do the expression 1.0 / 0.0 will return? will it throw Exception? any compile time error?**  
Answer: This is another tricky question from Double class. Though Java developer knows about the double primitive type and Double class, while doing floating point arithmetic they don't pay enough attention to Double.INFINITY, NaN, and -0.0 and other rules that govern the arithmetic calculations involving them. The simple answer to this question is that it will not throw ArithmeticExcpetion and return Double.INFINITY. Also, note that the comparison x == Double.NaN always evaluates to false, even if x itself is a NaN. To test if x is a NaN, one should use the method call Double.isNaN(x) to check if given number is NaN or not. If you know SQL, this is very close to NULL there.

**Q17: What is the difference between fail-fast and fail-safe Iterators?**

**A:** Fail-fast Iterators throws ConcurrentModificationException when one Thread is iterating over collection object and other thread structurally modify Collection either by adding, removing or modifying objects on underlying collection. They are called fail-fast because they try to immediately throw Exception when they encounter failure. On the other hand fail-safe Iterators works on copy of collection instead of original collection.

**What will happen if we put a key object in a HashMap which is already there?**

How HashMap works in Java. HashMap .if you put the same key again then it will replace the old mapping because HashMap doesn't allow duplicate keys. The Same key will result in the same hashcode and will end up at the same position in the bucket. Each bucket contains a linked list of Map.Entry object, which contains both Key and Value. Now Java will take Key object form each entry and compare with this new key using equals() method, if that return true then value object in that entry will be replaced by new value.

What will happen if an exception is thrown from the finally block?  
**Answer:** The program will exit if the exception is not catched in the finally block.

 How does decorator design pattern works in I/O classes?  
Answer:  The various classes like BufferedReader , BufferedWriter workk on the underlying stream classes. Thus Buffered\* class will provide a Buffer for Reader/Writer classes.

 If I give you an assignment to design Shopping cart web application, how will you define the architecture of this application. You are free to choose any framework, tool or server?  
Answer:  Usually I will choose a MVC framework which will make me use other design patterns like Front Controller, Business Delegate, Service Locater, DAO, DTO, Loose Coupling etc. Struts 2 is very easy to configure and comes with other plugins like Tiles, Velocity and Validator etc. The architecture of Struts becomes the architecture of my application with various actions and corresponding JSP pages in place.

9) how does volatile variable works in Java ?

Each thread has its own stack, and so its own copy of variables it can access. When the thread is created, it copies the value of all accessible variables in its own memory. The volatile keyword is used to say to the jvm "Warning, this variable may be modified in an other Thread". Without this keyword the JVM is free to make some optimizations, like never refreshing those local copies in some threads. The volatile force the thread to update the original variable for each variable. The volatile keyword could be used on every kind of variable, either primitive or objects! Maybe the subject of another article, more detailed...

10) How to detect memory leak in Java.

5) When a singleton is not singleton in Java ?

Three places Serialization, RMI and ClassLoader as different ClassLoader can load multiple classes

**What is difference between StringBuffer and StringBuilder in Java**

Stringbuffer methods are synchronized while StringBuilder is non synchronized.

**String** is immutable, if you try to alter their values, another object gets created, whereas **StringBuffer and** StringBuilder are mutable so they can change their values. Thread-Safety **Difference**: The **difference** between **StringBuffer and** StringBuilder is that **StringBuffer** is thread-safe

<load-on-startup>1</load-on-startup>  
value >= 0 means that the servlet is loaded when the web-app is deployed or when the server starts. value < 0 : servlet is loaded whenever the container feels like.

org.apache.struts2.dispatcher.FilterDispatcher

org.springframework.web.context.ContextLoaderListener

PL/SQL

### What is the basic structure of PL/SQL?

PL/SQL uses BLOCK structure as its basic structure. Each PL/SQL program consists of SQL and PL/SQL statement which form a PL/SQL block.

PL/SQL block contains 3 sections.

1. The Declaration Section (optional)
2. The Execution Section (mandatory)
3. The Exception handling Section (optional)

### What is the difference between FUNCTION, PROCEDURE AND PACKAGE in PL/SQL?

**Function**: The main purpose of a PL/SQL function is generally to compute and return a single value. A function has a return type in its specification and must return a value specified in that type.

**Procedure**: A procedure does not have a return type and should not return any value but it can have a return statement that simply stops its execution and returns to the caller. A procedure is used to return multiple values otherwise it is generally similar to a function.

**Package**: A package is schema object which groups logically related PL/SQL types , items and subprograms. You can also say that it is a group of functions, procedure, variables and record type statement. It provides modularity, due to this facility it aids application development. It is used to hide information from unauthorized users.

### 1) How many types of triggers exist in PL/SQL?

There are 12 types of triggers in PL/SQL that contains the combination of BEFORE, AFTER, ROW, TABLE, INSERT, UPDATE, DELETE and ALL keywords.

* BEFORE ALL ROW INSERT
* AFTER ALL ROW INSERT
* BEFORE INSERT
* AFTER INSERT etc.

### 15) What are the cursor attributes used in PL/SQL?

**%ISOPEN**: it checks whether the cursor is open or not.

**%ROWCOUNT**: returns the number of rows affected by DML operations: INSERT,DELETE,UPDATE,SELECT.

**%FOUND**: it checks whether cursor has fetched any row. If yes - TRUE.

**%NOTFOUND**: it checks whether cursor has fetched any row. If no - TRUE.

### 12) What is stored Procedure?

A stored procedure is a sequence of statement or a named PL/SQL block which performs one or more specific functions. It is similar to a procedure in other programming languages. It is stored in the database and can be repeatedly executed. It is stored as schema object. It can be nested, invoked and parameterized.

### 13) How to execute a stored procedure?

There are two way to execute a stored procedure.

From the SQL prompt, write EXECUTE or EXEC followed by procedure\_name.

1. EXECUTE or [EXEC] procedure\_name;

Simply use the procedure name

1. procedure\_name;

### 7) What is cursor and why it is required?

A cursor is a **temporary work area** created in a system memory when an SQL statement is executed.

A cursor contains information on a select statement and the row of data accessed by it. This temporary work area stores the data retrieved from the database and manipulate this data. A cursor can hold more than one row, but can process only one row at a time. Cursor are required to process rows individually for queries.

### 18) How many types of cursors are available in PL/SQL?

There are two types of cursors in PL/SQL.

1. Implicit cursor, and
2. explicit cursor

* java memory how object stored

In Java, when we only declare a variable of a class type, only a reference is created (memory is not allocated for the object). To allocate memory to an object, we must use new(). So the object is always allocated memory on heap

* hashmap internal working how it's do hashing what stored in bucket if duplicate what happened
* executer framework/pattern

Spring Annotation

* Spring MVC life cycle viewresolver

The Spring web MVC framework provides model-view-controller architecture and ready components that can be used to develop flexible and loosely coupled web applications. The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.

* The **Model** encapsulates the application data and in general they will consist of POJO.
* The **View** is responsible for rendering the model data and in general it generates HTML output that the client's browser can interpret.
* The **Controller** is responsible for processing user requests and building appropriate model and passes it to the view for rendering.

## The DispatcherServlet

The Spring Web model-view-controller (MVC) framework is designed around a *DispatcherServlet* that handles all the HTTP requests and responses. The request processing workflow of the Spring Web MVC *DispatcherServlet* is illustrated in the following diagram:



Following is the sequence of events corresponding to an incoming HTTP request to *DispatcherServlet*:

* After receiving an HTTP request, *DispatcherServlet* consults the *HandlerMapping* to call the appropriate *Controller*.
* The *Controller* takes the request and calls the appropriate service methods based on used GET or POST method. The service method will set model data based on defined business logic and returns view name to the *DispatcherServlet*.
* The *DispatcherServlet* will take help from *ViewResolver* to pickup the defined view for the request.
* Once view is finalized, The *DispatcherServlet* passes the model data to the view which is finally rendered on the browser.

All the above mentioned components ie. HandlerMapping, Controller and ViewResolver are parts of *WebApplicationContext* which is an extension of the plain *ApplicationContext* with some extra features necessary for web applications.

WEB-INF

<servlet>

<servlet-name>HelloWeb</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

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

</servlet>

<servlet-mapping>

<servlet-name>HelloWeb</servlet-name>

<url-pattern>\*.jsp</url-pattern>

</servlet-mapping>

frontcontroller

Front-Controller: Is a initial level of contract point for handling a request. The front controller provides a centralized entry point for that controls and manages web request handling. By centralizing decision point and controls.  
  
The front controller also reduce your java code and business logic by promoting code reuse ability across the requests. The front controller coordinates with dispatcher components. Dispatcher are responsible to view management only, the one who bring the view components to the user is called dispatcher.

In Spring MVC org.springframework.web.servlet.DispatcherServlet is a front controller who handles all the user request and process the request as per there mapping.

hashmap memory ? String as key in hashmap ? ..... A a = new A() as key ? hashmap collusion

String good idea immutable object hence key not able to change .

A a = new A() as key prob during Garbage collection reference to the object not able to garbage collected.

database locking unlocking

Table Locks (TM)  ,Table Locks (TM) , Locks in DML Operations

Read write locks

transaction code in your project

Global transactions enable you to work with multiple transactional resources, typically relational databases and message queues.

Local transactions are resource-specific, such as a transaction associated with a JDBC connection. Local transactions may be easier to use, but have significant disadvantages: they cannot work across multiple transactional resources.

taggetinvokationException

what is transaction ,transaction type

A transaction is a series of actions that are treated as a single unit of work. These actions should either complete in its entirety or not at all.

PlatformTransactionManager is a general interface for all Spring transaction managers

1. In the first example, we manage the transaction programmatically by embedding transaction management code in the business method like starting a new transaction, commit or rollback of transaction. The main advantage of this method is that we get a precise control on when to commit or rollback the transaction.
2. In the second example, we will avoid the boilerplate transaction management code, by delegating the execution to TransactionTemplate class. All we have to do is encapsulate our business method in form of a callback method and pass it to the TransactionTemplate.execute. TransactionTemplate provides a template method around which a transactional boundary is started, the callback method is called and then transaction is committed. In case of any exception (checked or unchecked) or error during the execution of the business method, rollback of the transaction happens.

exception handling code in struts

JMS message type of what are messages

diff between JMS and websevices which to prefer

how garbage collection works diagram not concept

spring transaction TransactionDefinition.ISOLATION\_DEFAULT

if we have three methods in DAO and in BO level i would like to handle transaction at one go.

Maven -- i already have repository and if i have another repository to point out how you will do

replace current .m2 folder contents with new one

you can take back up of current one

jst rename current one and copy new repository to that location and name it as .m2

sessionfactory more than one

Yes if we have more than one datasource we can configure sessionfactory for each datasource

@Qualifier is used to identify which sessionfactory needs to be used

@Autowired

@Qualifier(value="sessionFactory")

private SessionFactory sessionFactory;

@Autowired

@Qualifier(value="secondDBSessionFactory")

private SessionFactory secondDBSessionFactory;

more than one session

yes

bean life cycle

core class of hibernate

what is diamond problem why not more than 2 extends

cocurrency packages executer framwork

can we have handler mapping in annotation

spring AOP

Spring ORM

can we initialize abstract class

constructor in abstract class ?

Onsite

12 years Hands on experience in

* Core Java
* Framework used Struts , spring , Hibernate
* UI skills JavaScript , JQUERY , HTML5, AngularJS 1.4
* DB oracle sql , PL-SQL
* SOAP – RestFul webservices
* Web logic Application Server 12C , JBOSS
* build tools Maven , Sonar , Jenkins code coverage ,Tortoise SVN for version control

The PDTS is an AT&T application, it is a Customer Care tool created for management to standardize, track and report on coaching/training interactions on a daily or frequent basis**.** It allows managers with direct reports to record coaching discussions which then become a part of a permanent record for the employee. PDTS Provides reporting on development and coaching methods to management. Provides detailed reports on duration of time Managers have spent coaching their Non-Management Employees. PDTS establishes consistent development standards and tools for managers.

Technology used Java,J2ee ,Struts framework , Oracle database PL –SQL, Unix

* Having 12 years of IT experience
* 8 years in tech Mahindra last 3 years in ATT project.

all experience in design, code development and Implementation of web Applications using Java and J2EE Technologies.

* Hands on experience on design, development and Enhancements, unit testing, defect fixing. Interaction with onshore team
* Worked extensively in the Development of Web applications using core java and server side technologies like Servlets, jsp , struts, Spring. SOAP Rest webservices ,JMS
* Hands-on work experience on database SQL and PL-SQL

Agile based project my contribution as a team member is actively involved in Impact Analysis, Coding and testing phases.

* Design, code development and Enhancements, unit testing, production defect fixing. Interaction with onshore team.
* Involved in the team meetings to discuss the issues and problems
* Involved in the creation of Application Detail Design Document.

Recently I have work on

Junit implementation MockStrutsTestCase sonar

Jboss migration

Improving code quality

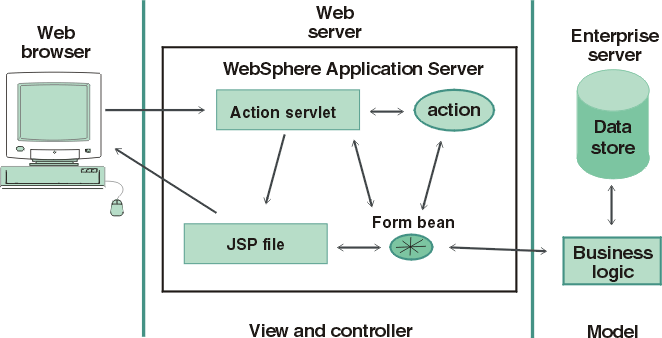
**Sonar** is a web based code quality analysis **tool** for Maven based **Java** projects. It covers a wide area of code quality check points which include: Architecture & Design, Complexity, Duplications, Coding Rules, Potential Bugs, Unit Test

**Jenkins** is an open-source continuous integration software tool written in the Java programming language for testing and reporting on isolated changes in a larger code base in real time. The software enables developers to find and solve defects in a code base rapidly and to automate testing of their builds.

Current Project Architecture

MVC Struts based

Struts framework: Model 2 architecture

-+

This diagram shows the structure of an application that has been designed by using model-view-controller principles.

Struts contribution to MVC components

* **Model:** application state .. Java bean
* **View:** presentation of data (JSP, HTML)
* **Controller:** routing of the application flow

Store procedure

A stored procedure is a group of SQL statements that has been created and stored in the database. A stored procedure will accept input parameters so that a single procedure can be used over the network by several clients using different input data. A stored procedures will reduce network traffic and increase the performance. If we modify a stored procedure all the clients will get the updated stored procedure.

Sample of creating a stored procedure

CREATE PROCEDURE test\_display

AS

SELECT FirstName, LastName

FROM tb\_test;

EXEC test\_display;

 A stored procedure allows modular programming.

You can create the procedure once, store it in the database, and call it any number of times in your program.

 A stored procedure allows faster execution.

If the operation requires a large amount of SQL code that is performed repetitively, stored procedures can be faster. They are parsed and optimized when they are first executed, and a compiled version of the stored procedure remains in a memory cache for later use. This means the stored procedure does not need to be reparsed and reoptimized with each use, resulting in much faster execution times.

 A stored procedure can reduce network traffic.

 Stored procedures provide better security to your data

Users can be granted permission to execute a stored procedure even if they do not have permission to execute the procedure's statements directly.

In SQL Server we have different types of stored procedures:

* System stored procedures
* User-defined stored procedures
* Extended stored Procedures

Create Procedure Employee details

As

Begin

Select \* from Employee

End

Execute Employee details

**Cursor**

A **cursor** is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the **active set**.

* Implicit cursors
* Explicit cursors

Explicit cursors are programmer-defined cursors for gaining more control over the **context area**. An explicit cursor should be defined in the declaration section of the PL/SQL Block.

EJB session bean

Session bean encapsulates business logic only, it can be invoked by local, remote and webservice client.

It can be used for calculations, database access etc.

The life cycle of session bean is maintained by the application server (EJB Container).

**Types of Session Bean**

There are 3 types of session bean.

**1) Stateless Session Bean**: It doesn't maintain state of a client between multiple method calls.

**2) Stateful Session Bean**: It maintains state of a client across multiple requests.

**3) Singleton Session Bean**: One instance per application, it is shared between clients and supports concurrent access.

#### 1) Create stateless bean component

To create the stateless bean component, you need to create a remote interface and a bean class.

File: AdderImplRemote.java

1. package com.javatpoint;
2. import javax.ejb.Remote;
4. **@Remote**
5. public interface AdderImplRemote {
6. int add(int a,int b);
7. }

File: AdderImpl.java

1. package com.javatpoint;
2. import javax.ejb.Stateless;
4. **@Stateless(mappedName="st1")**
5. public class AdderImpl implements AdderImplRemote {
6. public int add(int a,int b){
7. return a+b;
8. }
9. }

#### 2) Create stateless bean client

The stateless bean client may be local, remote or webservice client. Here, we are going to create remote client. It is console based application. Here, we are not using dependency injection. The dependency injection can be used with web based client only.

File: AdderImpl.java

1. package com.javatpoint;
2. import javax.naming.Context;
3. import javax.naming.InitialContext;
5. public class Test {
6. public static void main(String[] args)throws Exception {
7. Context context=new InitialContext();
8. AdderImplRemote remote=(AdderImplRemote)context.lookup("**st1**");
9. System.out.println(remote.add(32,32));
10. }
11. }

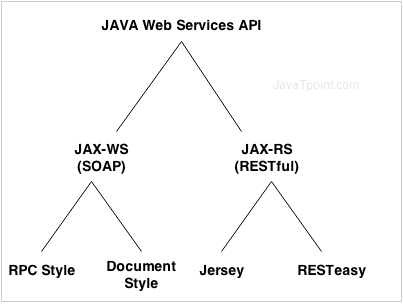
#### 2) Create stateless bean client

The stateless bean client may be local, remote or webservice client. Here, we are going to create remote client. It is console based application. Here, we are not using dependency injection. The dependency injection can be used with web based client only.

File: AdderImpl.java

1. package com.javatpoint;
2. import javax.naming.Context;
3. import javax.naming.InitialContext;
5. public class Test {
6. public static void main(String[] args)throws Exception {
7. Context context=new InitialContext();
8. AdderImplRemote remote=(AdderImplRemote)context.lookup("st1");
9. System.out.println(remote.add(32,32));
10. }
11. }

Web services.



SOAP web services

SOAP is XML based protocol. It is platform independent and language independent. By using SOAP, you will be able to interact with other programming language applications.

1. **JAX-WS**: for SOAP web services. The are two ways to write JAX-WS application code: by RPC style and Document style.

## RPC Style

* + RPC style web services use method name and parameters to generate XML structure.
  + The generated WSDL is **difficult to be validated** against schema.

**WSDL file:**

In WSDL file, it doesn't specify the types details.

<types/>

## Document Style

1) Document style web services **can be validated against predefined schema**.

2) In document style, SOAP message is **sent as a single document**.

**WSDL file:**

In WSDL file, it specifies types details having namespace and schemaLocation.

1. <types>
2. <xsd:schema>
3. <xsd:import namespace="http://javatpoint.com/" schemaLocation="http://localhost:7779/ws/hello?xsd=1"/>
4. </xsd:schema>
5. </types>

//Service Endpoint Interface

@WebService

@SOAPBinding(style = Style.RPC)

public interface HelloWorld{

@WebMethod String getHelloWorldAsString(String name);

2)

REST is an architectural style not a protocol.

**Permits different data format**: RESTful web service permits different data format such as Plain Text, HTML, XML and JSON.

**JAX-RS**: for RESTful web services. There are mainly 2 implementation currently in use for creating JAX-RS application: Jersey and RESTeasy.

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;

import javax.ws.rs.core.MediaType;

@Path("/hello")

public class Hello {

// This method is called if HTML and XML is not requested

@GET

@Produces(MediaType.TEXT\_PLAIN)

public String sayPlainTextHello() {

return "Hello Jersey Plain";

}

   <servlet-name>Jersey REST Service</servlet-name>

     <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>

HTML5

New tag implemented in HTML5

|  |  |
| --- | --- |
| <audio> | Defines an audio file. |
| <canvas> | This is used for rendering dynamic bitmap graphics on the fly, such as graphs or games. |
| <command> | Represents a command the user can invoke. |
| <datalist> | Together with the a new list attribute for input can be used to make comboboxes |
| <details> | Represents additional information or controls which the user can obtain on demand |
| <embed> | Defines external interactive content or plugin. |
| <figure> | Represents a piece of self-contained flow content, typically referenced as a single unit from the main flow of the document. |
| <footer> | Represents a footer for a section and can contain information about the author, copyright information, et cetera. |
| <header> | Represents a group of introductory or navigational aids. |

Jquery

Spring

Angular 1.4

Angular 2 features