JAVA

- **Abstraction:** Showing the essential and hiding the non-Essential is known as Abstraction.

- **Encapsulation**: The Wrapping up of data and functions into a single unit is known as Encapsulation.
  Encapsulation is the term given to the process of hiding the implementation details of the object. Once an object is encapsulated, its implementation details are not immediately accessible any more. Instead they are packaged and are only indirectly accessed via the reference of the object.

- **Inheritance**: is the Process by which the *Obj* of one class acquires the properties of *Obj's* another Class.
  A reference variable of a Super Class can be assign to any Sub class derived from the Super class.
  Inheritance is the method of creating the new class based on already existing class, the new class derived is called Sub class which has all the features of existing class and its own, i.e sub class.
  Adv: Reusability of code , accessibility of variables and methods of the Base class by the Derived class.

- **Polymorphism**: The ability to take more that one form, it supports Method Overloading & Method Overriding.
  - **Method overloading**: When a method in a class having the *same method name* with *different arguments* (diff Parameters or Signatures) is said to be Method Overloading. This is *Compile time* Polymorphism.
    Using one identifier to refer to multiple items in the same scope.
  - **Method Overriding**: When a method in a Class having *same method name* with *same arguments* is said to be Method overriding. This is *Run time* Polymorphism.
    Providing a different implementation of a method in a subclass of the class that originally defined the method.
    1. In *Over loading* there is a relationship between the methods available in the same class, where as in ***Over riding*** there is relationship between the Super class method and Sub class method.

2. ***Overloading*** does not block the Inheritance from the Super class , Where as in *Overriding* blocks Inheritance from the Super Class.
3. In ***Overloading*** separate methods share the same name, where as in *Overriding* Sub class method replaces the Super Class.
4. *Overloading* must have different method Signatures , Signatures.

❖ **Dynamic dispatch:** is a mechanism by which a call to Overridden function is resolved at runtime rather than at Compile time , and this is how Java implements Run time Polymorphism.

❖ **Dynamic Binding**: Means the code associated with the given procedure call is not known until the time of call the call at run time. (it is associated with Inheritance & Polymorphism).

❖ **Bite code**: Is a optimized set of instructions designed to be executed by Java-run time system, which is called the Java Virtual machine (JVM), i.e. in its standard form, the JVM is an Interpreter for byte code.
JIT- is a compiler for Byte code, The JIT-Complier is part of the JVM, it complies byte code into executable code in real time, piece-by-piece on demand basis.
Final classes: String, Integer , Color, Math
Abstract class: Generic servlet, Number class

- **variable:** An item of data named by an identifier. Each variable has a type,such as int or Object, and a scope
- **class variable**: A data item associated with a particular class as a whole--not with particular instances of the class. Class variables are defined in class definitions. Also called a static field. See also instance variable.
- **instance variable**: Any item of data that is associated with a particular object. Each instance of a class has its own copy of the instance variables defined in the class. Also called a field. See also class variable.
- **local variable**: A data item known within a block, but inaccessible to code outside the block. For example, any

variable defined within a method is a local variable and can't be used outside the method.

- **class method**: A method that is invoked without reference to a particular object. Class methods affect the class as a whole, not a particular instance of the class. Also called a static method. also instance method.
- **instance method**: Any method that is invoked with respect to an instance of a class. Also called simply a method. See also class method.

❖ **Interface:** Interfaces can be used to implement the Inheritance relationship between the non-related classes that do not belongs to the same hierarchy, i.e. any Class and any where in hierarchy. Using Interface, you can specify what a class must do but not how it does.

- A class can implement more than one Interface.
- An Interface can extend one or more interfaces, by using the keyword *extends*.
- All the data members in the interface are public, static and Final by default.
- An Interface method can have only Public, default and Abstract modifiers.
- An Interface is loaded in memory only when it is needed for the first time.
- A Class, which implements an Interface, needs to provide the implementation of all the methods in that Interface.
- If the Implementation for all the methods declared in the Interface are not provided, the class itself has to declare *abstract*, other wise the Class will not compile.
- If a class Implements two *interface* and both the *Intfs* have identical method *declaration*, it is totally valid.
- If a class implements tow interfaces both have identical method name and argument list, but different return types, the code will not compile.
- An Interface can’t be instantiated. *Intf* Are designed to support dynamic method resolution at run time.
- An interface can not be native, static, synchronize, final, protected or private.
- The Interface *fields* can’t be Private or Protected.
- A Transient variables and Volatile variables can not be members of Interface.

- The extends keyword should not used after the Implements keyword, the Extends must always come before the Implements keyword.
- A top level Interface can not be declared as static or final.
- If an Interface species an exception list for a method, then the class implementing the interface need not declare the method with the exception list.
- If an Interface can't specify an exception list for a method, the class can't throw an exception.
- If an Interface does not specify the exception list for a method, the class can not throw any exception list.

The general form of Interface is

```
Access interface name {
    return-type method-name1(parameter-list);
    type final-varname1=value;
}
```

- **Marker Interfaces:** Serializable, Clonable, Remote, EventListener,

Java.lang is the Package of all classes and is automatically imported into all Java Program

Interfaces: Clonable , Comparable, Runnable

- **Abstract Class:** Abstract classes can be used to implement the inheritance relationship between the classes that belongs same hierarchy.
  - Classes and methods can be declared as abstract.
  - Abstract class can extend only one Class.
  - If a Class is declared as abstract , no instance of that class can be created.
  - If a method is declared as abstract, the sub class gives the implementation of that class.
  - Even if a single method is declared as abstract in a Class , the class itself can be declared as abstract.
  - Abstract class have at least one abstract method and others may be concrete.
  - In abstract Class the keyword abstract must be used for method.
  - Abstract classes have sub classes.
  - Combination of modifiers Final and Abstract is illegal in java.
  - **Abstract Class means:** Which has more than one abstract method which doesn't have method body but at least one of its methods need to be implemented in derived Class.

The general form of abstract class is:
abstract type name (parameter list);
The Number class in the java.lang package represents the abstract concept of numbers. It makes sense to model numbers in a program, but it doesn't make sense to create a generic number object.

❖ Difference Between Interfaces And Abstract class?

- o All the methods declared in the Interface are Abstract, where as abstract class must have atleast one abstract method and others may be concrete.
- o In abstra ct class keyword abstract must be used for method, where as in Interface we need not use the keyword for methods.
- o Abstract class must have Sub class, where as Interface can’t have sub classes.
- o An abstract class can extend only one class, where as an Interface can extend more than one.

❖ What are access specifiers and access modifiers ?

Accesss specifiers	**Access modifiers**	
Public	Public	
Protected	Abstract	
Private	Final	
	Static	
	Volatile	Constant
	Transient	
	Native	

- **Public**: The Variables and methods can be access any where and any package.
- **Protected**: The Variables and methods can be access same Class, same Package & sub class.
- **Private**: The variable and methods can be access in same class only.

Same class	-	Public, Protected, and Private
Same-package & subclass	-	Public, Protected
Same Package & non-sub classes	-	Public, Protected

Different package	& Sub classes	-	Public, Protected
Different package	& non- sub classes	-	Public

- **Identifiers:** are the Variables that are declared under particular Datatype.
- **Literals:** are the values assigned to the Identifiers.

- **Static:** access modifier.
- **Syntax**: **<u>Variable</u>**: Static int b;
- **<u>Method</u>:** static void meth(int x)

  - When a member is declared as Static, it can be accessed before any objects of its class are created and without reference to any object. Eg : main(),it must call before any object exit.
  - Static can be applied to Inner classes, Variables and Methods.
  - Local variables can't be declared as static.
  - A static method can access only static Variables. and they can't refer to **this** or **super** in any way.
  - Static methods can't be abstract.
  - A static method may be called without creating any instance of the class.
  - Only one instance of static variable will exit any amount of class instances.

- **Final:** access modifier
  - All the Variables, methods and classes can be declared as Final.
  - Classes declared as final class can't be sub classed.
  - Method 's declared as final can't be over ridden.
  - If a Variable is declared as final, the value contained in the Variable can't be changed.
  - Static final variable must be assigned in to a value in static initialized block.

- **Transient:** access modifier
  - Transient can be applied only to class level variables.
  - Local variables can't be declared as transient.
  - During serialization, Object's transient variables are not serialized.

- Transient variables may not be final or static. But the complier allows the declaration and no compile time error is generated.

- **Volatile:** access modifier
  - Volatile applies to only variables.
  - Volatile can applied to static variables.
  - Volatile can not be applied to final variables.
  - Transient and volatile can not come together.
  - Volatile is used in multi-processor environments.

- **Native:** access modifier
  - Native applies to only to methods.
  - Native can be applied to static methods also.
  - Native methods can not be abstract.
  - Native methods can throw exceptions.
  - Native method is like an abstract method. The implementation of the abstract class and native method exist some where else, other than the class in which the method is declared.

- **Synchronized:** access modifier
  - Synchronized keyword can be applied to methods or parts of the methods only.
  - Synchronize keyword is used to control the access to critical code in multi-threaded programming.

Declaration of access specifier and access modifiers:

Class	-	Public, Abstract, Final
Inner Class	-	Public, Protected, Private, Final, Static,
Anonymous	-	Public, Protected, Private, Static
Variable	-	Public, Protected, Private, Final, Static, Transient, Volatile
Method	-	Public, Protected, Private, Final, Abstract, Static, Native, Synchronized

Constructor	-	Public, Protected, Private
Free-floating code block	-	Static, Synchronized

- **Package:** A Package is a collection of Classes Interfaces that provides a high-level layer of access protection and name space management.

- **finalize() method:**
  - All the objects have finalize() method, this method is inherited from the Object class.
  - Finalize() is used to release the system resources other than memory(such as file handles& network connec's.
  - Finalize() is used just before an object is destroyed and can be called prior to garbage collection.
  - Finalize() is called only once for an Object. If any exception is thrown in the finalize() the object is still eligible for garbage collection.
  - Finalize() can be called explicitly. And can be overloaded, but only original method will be called by Ga-collect.
  - Finalize() may only be invoked once by the Garbage Collector when the Object is unreachable.
  - The signature finalize(): **protected void finalize() throws Throwable {}**

- **Constructor():**
  - A constructor method is special kind of method that determines how an object is initialized when created.
  - Constructor has the same name as class name.
  - Constructor does not have return type.
  - Constructor cannot be over ridden and can be over loaded.
  - Default constructor is automatically generated by compiler if class does not have once.
  - If explicit constructor is there in the class the default constructor is not generated.
  - If a sub class has a default constructor and super class has explicit constructor the code will not compile.

- **Object:** Object is a Super class for all the classes. The methods in Object class as follows.

Object clone()	final void notify()	Int hashCode()
Boolean equals()	final void notify()	
Void finalize()	String toString()	
Final Class getClass()	final void wait()	

❖ **Class:** The Class class is used to represent the classes and interfaces that are loaded by the JAVA Program.

❖ **Character:** A class whose instances can hold a single character value. This class also defines handy methods that can manipulate or inspect single-character data.
constructors and methods provided by the Character class:

- **Character(char):** The Character class's only constructor, which creates a Character object containing the value provided by the argument. Once a Character object has been created, the value it contains cannot be changed.
- **compareTo(Character):** An instance method that compares the values held by two character objects.
- **equals(Object):** An instance method that compares the value held by the current object with the value held by another.
- **toString():** An instance method that converts the object to a string.
- **charValue():** An instance method that returns the value held by the character object as a primitive char value.
- **isUpperCase(char):** A class method that determines whether a primitive char value is uppercase.

❖ **String:** String is Immutable and String Is a final class. The String class provides for strings whose value will not change.
One accessor method that you can use with both strings and string buffers is the length() method, which returns the number of characters contained in the string or the string buffer.
The methods in String Class:-

toString()	**equals()**	indexOff()	**LowerCase()**
charAt()	compareTo()	lastIndexOff()	**UpperCase()**
getChars()	subString()	**trim()**	

getBytes() **concat()** valueOf()
toCharArray() **replace()**
ValueOf(): converts data from its internal formate into human readable formate.

- ❖ **String Buffer:** Is Mutable, The StringBuffer class provides for strings that will be modified; you use string buffers when you know that the value of the character data will change.
  In addition to length, the StringBuffer class has a method called capacity, which returns the amount of space allocated for the string buffer rather than the amount of space used.
  The methods in StringBuffer Class:-
  length() append() replace() charAt() and setCharAt()
  capacity() insert() substring() getChars()
  ensureCapacity() reverse()
  setLength() delete()

- ❖ **Wraper Classes:** are the classes that allow primitive types to be accessed as Objects.
  These classes are similar to primitive data types but starting with capital letter.

Number	Byte	Boolean
Double	Short	Character
Float	Integer	
	Long	

- ❖ **primitive Datatypes in Java:** According to Java in a Nutshell, 5th ed boolean, byte, char, short, long float, double, int.
  - **Float class**: The Float and Double provides the methods isInfinite() and isNaN().
  - **isInfinite():** returns true if the value being tested is infinetly large or small.
  - **isNaN():** returns true if the value being tested is not a number.

- ❖ **Character class:** defines forDigit() digit().
  - **ForDigit():** returns the digit character associated with the value of num.

  - **digit():** returns the integer value associated with the specified character (which is presumably) according to the specified radix.

- **String Tokenizer:** provide parsing process in which it identifies the delimiters provided by the user, by default delimiters are spaces, tab, new line etc., and separates them from the tokens. Tokens are those which are separated by delimiters.

- **Observable Class:** Objects that subclass the Observable class maintain a list of observers. When an Observable object is updated it invokes the update() method of each of its observers to notify the observers that it has changed state.

- **Observer interface:** is implemented by objects that observe Observable objects.

- **Instanceof():** is used to check to see if an object can be cast into a specified type with out throwing a cast class exception.

- **IsInstanceof()**: determines if the specified Object is assignment-compatible with the object represented by this class. This method is dynamic equivalent of the Java language instanceof operator. The method returns true if the specified Object argument is non-null and can be cast to the reference type represented by this Class object without raising a ClassCastException. It returns false otherwise.

- **Garbage Collection**: When an object is no longer referred to by any variable, java automatically reclaims memory used by that object. This is known as garbage collection. System.gc() method may be used to call it explicitly and does not force the garbage collection but only suggests that the JVM may make an effort to do the Garbage Collection.
  - **this():** can be used to invoke a constructor of the same class.
  - **super():** can be used to invoke a super class constructor.

❖ **Inner class:** classes defined in other classes, including those defined in methods are called inner classes. An inner class can have any accessibility including private.

❖ **Anonymous class**: Anonymous class is a class defined inside a method without a name and is instantiated and declared in the same place and cannot have explicit constructors.

❖ **What is reflection API? How are they implemented?**
Reflection package is used mainlyfor the purpose of getting the class name. by useing the getName method we can get name of the class for particular application. Reflection is a feature of the Java programming language. It allows an executing Java program to examine or "introspect" upon itself, and manipulate internal properties of the program.

❖ **What is heap in Java?**
JAVA is fully Object oriented language. It has two phases first one is Compilation phase and second one is interpratation phase. The Compilation phase convert the java file to class file (byte code is only readable format of JVM) than Intepratation phase interprete the class file line by line and give the proper result.

- **main():** is the method where Java application Begins.
- **String args[]:** receives any command line argument during runtime.
- **System:** is a predefined Class that provides access to the System.
- **Out:** is output stream connected to console.
- **Println:** displays the output.

❖ **Downcasting:** is the casting from a general to a more specific type, i.e casting down the hierarchy. Doing a cast from a base class to more specific Class, the cast does;t convert the Object, just asserts it actually is a more specific extended Object.

❖ **Upcasting:** byte can take Integer values.

Exception

❖ **Exception handling:** Exception can be generated by Java-runtime system or they can be manually generated by code.
Error-Handling becomes a necessary while developing an application to account for exceptional situations that may occur during the program execution, such as

- Run out of memory
- Resource allocation Error
- Inability to find a file
- Problems in Network connectivity.

If the Resource file is not present in the disk, you can use the Exception handling mechanisim to handle such abrupt termination of program.

❖ **Exception class:** is used for the exceptional conditions that are trapped by the program. An exception is an abnormal condition or error that occur during the execution of the program.

❖ **Error:** the error class defines the conditions that do not occur under normal conditions.
Eg: Run out of memory, Stack overflow error.
Java.lang.Object
+….Java.Lang.Throwable
Throwable
+….Java.lang.Error
| +….A whole bunch of errors
| **Exception**
Error
+….Java.Lang.Exception (Unchecked, Checked)
+….Java.Lang.RuntimeException
| +….Various Unchecked Exception
|
+….Various checked Exceptions.

❖ **Two types of exceptions:**

1. **Checked Exceptions**: must be declare in the method declaration or caught in a catch block. Checked exception must be handled at Compile Time. Environmental error that

cannot necessarly be detected by Testing, Eg: disk full, brocken Socket, Database unavailable etc.

2. **Un-checked Exceptions**: Run-time Exceptions and Error, does't have to be declare.(but can be caught).
   Run-time Exceptions: programming errors that should be detectd in Testing, Arithmetic, Null pointer, ArrayIndexOutofBounds, ArrayStore, FilenotFound, NumberFormate, IO, OutofMemory.
   Errors: Virtual mechine error – class not found , out of memory, no such method , illegal access to private field , etc.

❖ **Java Exception handling can be managed by five keywords:**

- **Try:** The try block governs the statements that are enclosed within it and defines the scope of exception handler associated with it. Try block follows catch or finally or both.
- **Catch**: This is a default exception handler. since the exception class is the base class for all the exception class, this handler id capable of catching any type of exception.
  The catch statement takes an *Object* of exception class as a *parameter*, if an exception is thrown the statement in the catch block is executed. The catch block is restricted to the statements in the proceeding try block only.

```
Try {
   // statements that may cause exception
} catch(Exception obj) {
}
```

- **Finally:** when an exception is raised, the statement in the try block is ignored, some times it is necessary to process certain statements irrespective of wheather an exception is raised or not, the finally block is used for this purpose.
- **Throw:** The throw class is used to call exception explicitly. You may want to throw an exception when the user enters a wrong login ID and pass word, you can use throw statement to do so.
  The throw statement takes an single argument, which is an Object of exception class.

❖ **Throw<throwable Instance>:** If the Object does not belong to a valid exception class the compiler gives error.

- **Throws:** The throws statement species the list of exception that has thrown by a method.

If a method is capable of raising an exception that is does not handle, it must specify the exception has to be handle by the *calling* method, this is done by using the throw statement.

[<access specifier>] [<access modifier>] <return type> <method name> <arg-list> [<exception-list>]

Eg:

```
public void accept password() throws illegalException {
    System.out.println("Intruder");
    Throw new illegalAccesException;
}
```

Multithreaded Programming

- A multithreaded program contains two or more parts that can run concurrently, Each part a program is called thread and each part that defines a separate path of excution.
  Thus multithreading is a specified from of multitasking.

- **There are two distinct types of multitasking.**
  Process: A Process is, in essence, a program that is executing.
  Process-based: is heavy weight- allows you run two or more programs concurrently.
  Example: you can use JAVA compiler at the same time you are using text editor.
  Here a program is a small unit of code that can be dispatched by scheduler.
  Thread-based: is Light weight- A Program can perform two or more tasks simultaneously.
  Creating a thread:
  Example: A text editor can formate at the same time you can print, as long as these two tasks are being perform separate treads.
  Thread: can be defined as single sequential flow of control with in a program.
  Single Thread: Application can perform only one task at a time.
  Multithreaded: A process having more than one thread is said to be multithreaded.
  The multiple threads in the process run at the same time, perform different task and interact with each other.

- **Daemon Thread:** Is a low priority thread which runs immedeatly on the back ground doing the Garbage Collection operation for the Java Run time System.
  SetDaemon( ) – is used to create DaemonThread.

- **Creating a Thread:**
  1. By implementing the Runnable Interface.
  2. By extending the thread Class.

❖ **Thread Class:** Java.lang.Threadclass is used to construct and access the individual threads in a multithreaded application.
Syntax: Public Class <class name> extends Thread {}
The Thread class define several methods.

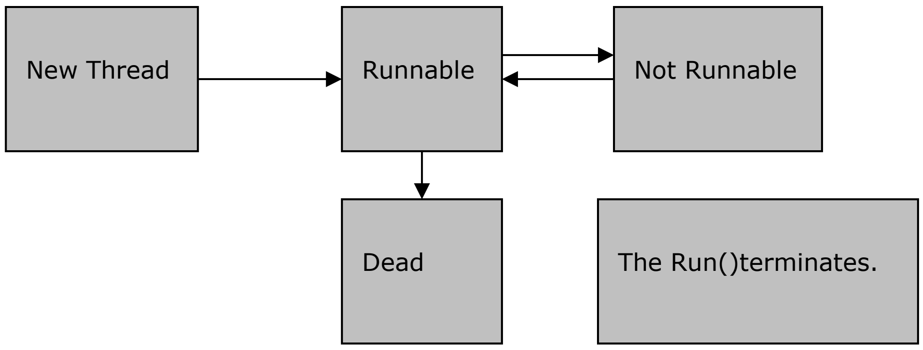
- Getname() – obtain a thread name.
- Getname() - obtain thread priority.
- Start() - start a thread by calling a Run().
- Run() - Entry point for the thread.
- Sleep() - suspend a thread for a period of time.
- IsAlive() - Determine if a thread is still running.
- Join() - wait for a thread to terminate.

❖ **Runable Interface:** The Runnable interface consist of a Single method Run(), which is executed when the thread is activated.
When a program need ti inherit from another class besides the thread Class, you need to implement the Runnable interface.
Syntax: public void <Class-name> extends <SuperClass-name> implements Runnable
Example:

```
public Class myapplet extends Japplet implements Runnable {
    // Implement the Class
}
```

* Runnable interface is the most advantageous method to create threads because we need not extend thread Class here.

❖ **Life Cycle of Thread:**



New Thread: When an instance of a thread class is created, a thread enters the new thread state.

```
Thread newThread = new Thread(this);
```

You have to invoke the Start() to start the thread. ie, newThread.Start();
Runnable: when the Start() of the thread is invoked the thread enters into the Runnable State.
Not Runnable: A thread is said to be not runnable state if it
- Is Slleping
- Is Waiting
- Is being blocked by another thread.

sleep(long t); where t= no: of milliseconds for which the thread is inactive.
The sleep() is a static method because it operates on the current thread.
Dead: A thread can either die natuarally or be killed.
- A thread dies a natural death when the loop in the Run() is complete.
- Assigning null to the thread Object kills the thread.
- If th loop in the Run() has a hundread iterations , the life of the thread is a hundread iterators of the loop.

IsAlive(): of the thread class is used to determine wheather a thread has been started or stopped. If isAlive() returns true the thread is still running otherwise running completed.
Thread Priorities: are used by the thread scheduler to decide when each thread should ne allowed to run.To set a thread priority, use te setpriority( ), which is a member of a thread. final void setpriority(int level) - here level specifies the new priority seting for the calling thread.

The value level must be with in the range:-
MIN_PRIORITY = 1
NORM_PRIORITY = 5
MAX_PRIORITY = 10
You can obtain the current priority setting by calling getpriority() of thread.
final int getpriority()

- **Synchronization:** Two ro more threads trying to access the same method at the same point of time leads to synchronization. If that method is declared as Synchronized , only one thread can access it at a time. Another thread can access that method only if the first thread’s task is completed.

- **Synchronized statement:** Synchronized statements are similar to *Synchronized method*.
  A Synchronized statements can only be executed after a thread has acquired a lock for the object or Class reffered in the Synchronized statements.
  The general form is:
  ```
  Synchronized(object) {
     // statements to be Synchronized
  }
  ```

- **Inter Thread Communication:** To Avoid pooling, Java includes an elegant interprocess communication mechanisim.
  Wait() - tells the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor & call notify().
  notify() - wake up the first thread that called wait() on the same Object.
  notifyall() – wake up all the threads that called wait() on the same Object.
  The highest priority thread aill run fast.

- **Serialization:** The process of writing the state of Object to a byte stream to transfer over the network is known as Serialization.

- **Deserialization:** and restored these Objects by deserialization.

- **Externalizable:** is an interface that extends Serializable interface and sends data into strems in compressed format.
  It has two methods:
  WriteExternal(Objectoutput out)
  ReadExternal(objectInput in)

- **I/O Package:**
  Java.io.*;
  There are two classifications.
  1. **ByteStream:** Console Input
     Read() - one character
     Readline() – one String

BufferReader br = new BufferReader(new InputStreamReader(System.in));

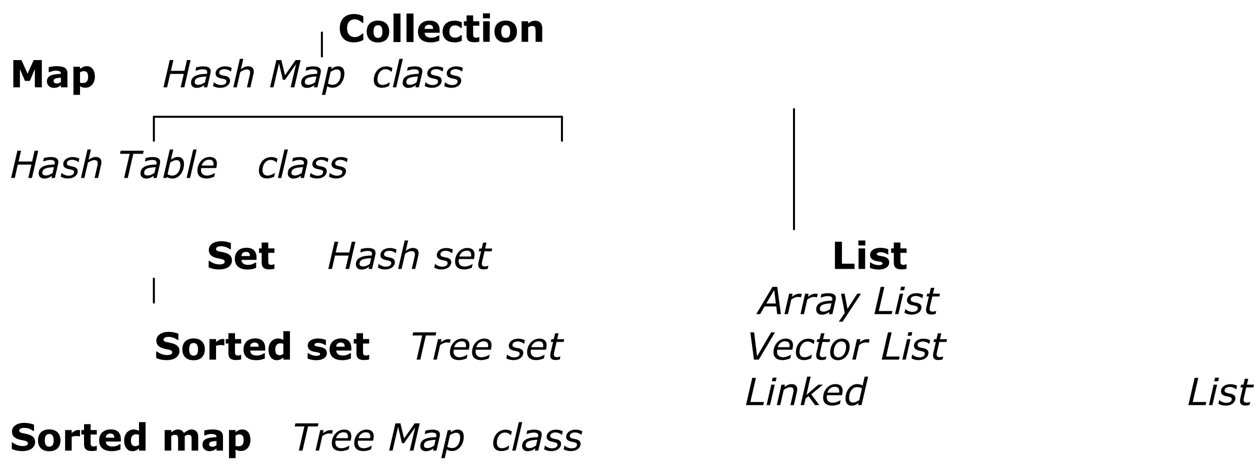
2. **CharacterStream:** File
   FileInputStream - Store the contents to the File.
   FileOutStream - Get the contents from File.
   PrintWrite pw = new printwriter(System.out.true);
   Pw.println("");

Example:

```
Class myadd {
    public static void main(String args[]) {
        BufferReader br = new BufferReader(new
        InputStreamReader(System.in));
        System.out.println("Enter A no : ");
        int a = Integer.parseInt(br.Read());
        System.out.println("Enter B no : ");
        int b = Integer.parseInt(br.Read());
        System.out.println("The Addition is  : " (a+b));
     }
}
```

Collections

❖ **Collections:** A collection allows a group of objects to be treated as a single unit. collection define a set of core Interfaces as follows.



❖ **Collection Interface:**

- The CI is the root of collection hierarchy and is used for common functionality across all collections. There is no direct implementation of Collection Interface.

❖ **Set Interface:** extends *Collection* Interface. The Class Hash set implements Set Interface.

- Is used to represent the group of *unique* elements.
- Set stores elements in an unordered way but does *not* contain duplicate elements.

❖ **Sorted set:** extends *Set* Interface. The class *Tree Set* implements Sorted set Interface.

- It provides the extra functionality of keeping the elements sorted.
- It represents the collection consisting of *Unique*, sorted elements in *ascending order*.

❖ **List:** extends Collection Interface. The classes *Array List, Vector List & Linked List* implements List Interface.

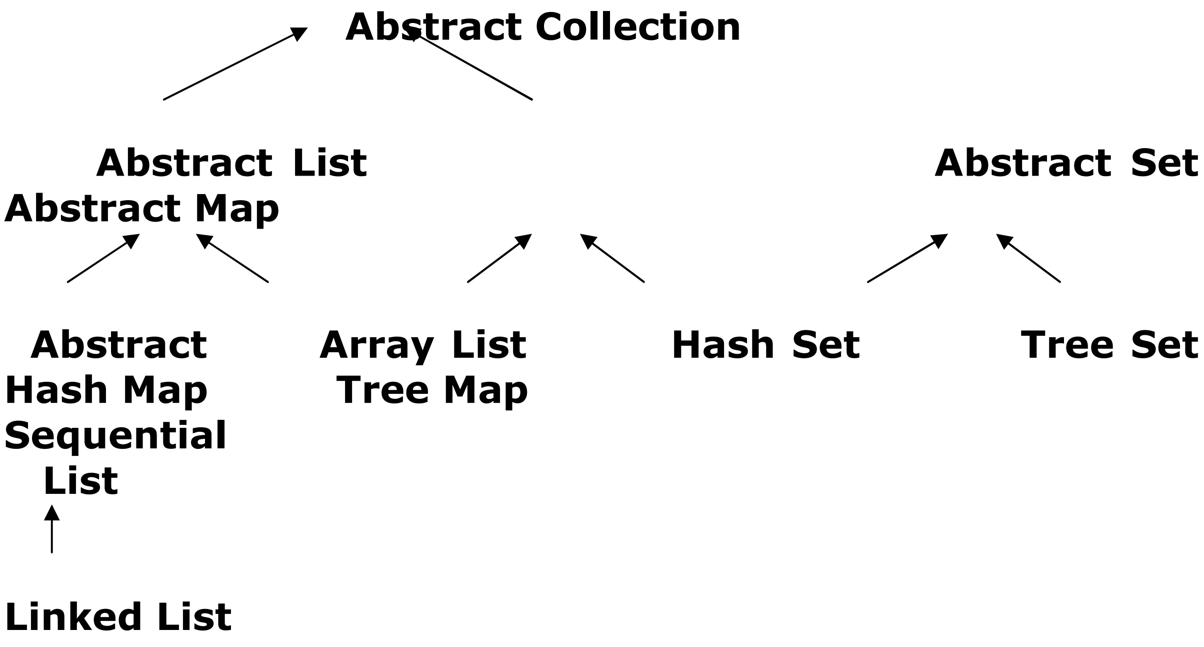
- Represents the sequence of numbers in a fixed order.
- But may contain duplicate elements.
- Elements can be inserted or retrieved by their position in the List using Zero based index.

- List stores elements in an ordered way.

❖ **Map Interface:** basic Interface. The classes*Hash Map & Hash Table* implements Map interface.
- Used to represent the mapping of unique keys to values.
- By using the key value we can retrive the values. Two basic operations are get() & put().

❖ **Sorted Map:** extends *Map* Interface. The Class *Tree Map* implements Sorted Map Interface.
- Maintain the values of key order.
- The entries are maintained in ascending order.

❖ **Collection classes:**



List	**Map**
Abstract List	**Dictonary**
Vector	**HashTable**
Stack	**Properities**

❖ **HashSet:** Implements Set Interface. HashSet hs=new HashSet();
- The elements are not stored in sorted order.
  hs.add(“m”);

- **TreeSet:** Implements Sorted set Interface.
  TreeSet ts=new TreeSet();
  - The elements are stored in sorted ascending order.
    ts.add("H");
  - Access and retrieval times are quit fast, when storing a large amount of data.

- **Vector:** Implements List Interface.
  - Vector implements *dynamic* array. Vector v = new vector();
  - Vector is a *growable* object.
    V1.addElement(new Integer(1));
  - Vector is *Synchronized*, it can't allow special *characters* and *null values*.
  - All vector starts with intial capacity, after it is reached next time if we want to store object in vector, the vector automatically allocates space for that Object plus extra room for additional Objects.

- **ArrayList:** Implements List Interface.
  - Array can dynamically increase or decrease size. ArrayList a1=new ArrayList();
  - Array List are ment for Random ascessing.
    A1.add("a");
  - Array List are created with intial size, when the size is increased, the collection is automatically enlarged. When an Objects are removed, the array may be shrunk.

- **Linked List:** Implements List Interface.
  - Inserting or removing elements in the middle of the array. LinkedList l1=new LinkedList();
  - Linked list are meant for *Sequential accessing*.
    L1.add("R");
  - Stores Objects in a separate link.

- **Map Classes:** Abstract Map; Hash Map; Tree Map

- **Hash Map:** Implements Map Interface. Hashmap(), Hashmap(Map m), Hashmap(int capacity)
  - The Elements may not in Order.

- Hash Map is *not synchronized* and permits null values
- Hash Map is *not serialized*. Hashmap hm = new HashMap();
- Hash Map supports *Iterators*.
  hm.put("Hari",new Double(11.9));

❖ **Hash Table:** Implements Map Interface.
- Hash Table is *synchronized* and does *not* permit *null* values.
- Hash Table is *Serialized*. Hashtable ht = new Hashtable( );
- Stores key/value pairs in Hash Table.
  ht.put("Prasadi",new Double(74.6));

A Hash Table stores information by using a mechanism called hashing. In hashing the informational content of a key is used to determine a unique value, called its Hash Code. The Hash Code is then used as the index at which the data associated with the key is stored. The Transformation of the key into its Hash Code is performed automatically- we never see the Hash Code. Also the code can't directly index into h c.

❖ **Tree Map:** Implements Sorted Set Interface. TreeMap tm=new TreeMap();
- The elements are stored in *sorted ascending order*.
  tm.put("Prasad",new Double(74.6));
- Using *key value* we can retrieve the data.
- Provides an efficient means of storing *key/value pairs* in sorted order and allows *rapid retrivals*.

❖ **Iterator:** Each of collection class provided an iterator().
By using this iterator Object, we can access each element in the collection – one at a time.
We can remove();
Hashnext() – go next; if it returns false – end of list.

Iterarator **Enumerator**
Iterator itr = a1.iterator(); Enumerator vEnum = v.element();
While(itr.hashNext()) System.out.println("Elements in Vector :");
{ while(vEnum.hasMoreElements())

```
    Object element = itr.next();
    System.out.println(vEnum.nextElement() + " ");
    System.out.println(element + " ");
}
```

Collections

❖ **Introduction:**

- Does your class need a way to easily search through thousands of items quickly?
- Does it need an ordered sequence of elements and the ability to rapidly insert and remove elements in the middle of the sequence?
- Does it need an array like structure with random-access ability that can grow at runtime?

```
        List                      Map
          |                        |
    Abstract List              Dictonary
          |                        |
        Vector                 HashTable
          |                        |
        Stack                  Properities
```

❖ **The Enumeration Interface:**

- enumerate (obtain one at a time) the elements in a collection of objects.

specifies two methods:

1. **boolean hasMoreElements():** Returns true when there are still more elements to extract, and false when all of the elements have been enumerated.
2. **Object nextElement():** Returns the next object in the enumeration as a generic Object reference.

❖ **Vector:**

- Vector implements *dynamic* array. Vector v = new vector();
- Vector is a *growable* object.
  V1.addElement(new Integer(1));
- Vector is *Synchronized*, it can't allow special *characters* and *null* values.
- Vector is a variable-length array of object references.
- Vectors are created with an initial size.
- When this size is exceeded, the vector is automatically enlarged.
- When objects are removed, the vector may be shrunk.

Constructors:

- **Vector():** Default constructor with initial size 10.
- **Vector(int size):** Vector whose initial capacity is specified by size.
- **Vector(int size, int incr):** Vector whose initialize capacity is specified by size and whose increment is specified by incr.

Methods:

- **final void addElement(Object element):** The object specified by element is added to the vector.
- **final Object elementAt(int index):** Returns the element at the location specified by index.
- **final boolean removeElement(Object element):** Removes element from the vector
- **final boolean isEmpty():** Returns true if the vector is empty, false otherwise.
- **final int size():** Returns the number of elements currently in the vector.
- **final boolean contains(Object element):** Returns true if element is contained by the vector and false if it is not.

❖ **Stack:**

- Stack is a subclass of Vector that implements a standard last-in, first-out stack

Constructors:

- Stack() Creates an empty stack.

Methods:

- **Object push(Object item):** Pushes an item onto the top of this stack.
- **Object pop():** Removes the object at the top of this stack and returns that object as the value of this function. An EmptyStackException is thrown if it is called on empty stack.
- **boolean empty():** Tests if this stack is empty.
- **Object peek():** Looks at the object at the top of this stack without removing it from the stack.
- **int search(Object o):** Determine if an object exists on the stack and returns the number of pops that would be required to bring it to the top of the stack.

❖ **HashTable:**

- Hash Table is *synchronized* and does *not* permit *null* values.

- Hash Table is *Serialized*.

```
Hashtable ht = new Hashtable( );
```

- Stores key/value pairs in Hash Table.
  ```
  ht.put("Prasadi",new Double(74.6));
  ```
- Hashtable is a concrete implementation of a Dictionary.
- Dictionary is an abstract class that represents a key/value storage repository.
- A Hashtable instance can be used store arbitrary objects which are indexed by any other arbitrary object.
- A Hashtable stores information using a mechanism called hashing.
- When using a Hashtable, you specify an object that is used as a key and the value (data) that you want linked to that key.

Constructors:
- **Hashtable()**
- **Hashtable(int size)**

Methods:
- **Object put(Object key,Object value):** Inserts a key and a value into the hashtable.
- **Object get(Object key):** Returns the object that contains the value associated with key.
- **boolean contains(Object value):** Returns true if the given value is available in the hashtable. If not, returns false.
- **boolean containsKey(Object key):** Returns true if the given key is available in the hashtable. If not, returns false.
- **Enumeration elements():** Returns an enumeration of the values contained in the hashtable.
- **int size():** Returns the number of entries in the hashtable.

❖ Properties
- Properties is a subclass of Hashtable
- Used to maintain lists of values in which the key is a String and the value is also a String

Constructors:
- **Properties()**
- **Properties(Properties propDefault):** Creates an object that uses propDefault for its default value.

Methods:

- **String getProperty(String key):** Returns the value associated with key.
- **Strng getProperty(String key, String defaultProperty):** Returns the value associated with key. defaultProperty is returned if key is neither in the list nor in the default property list.
- **Enumeration propertyNames():** Returns an enumeration of the keys. This includes those keys found in the default property list.

❖ The Interfaces in Collections Framework

$$\begin{tikzcd}
\textbf{Collection} \arrow[d] \arrow[dr] & & \textbf{Map} \arrow[d] & \textbf{Iterator} \arrow[d] \\
\textbf{Set} \arrow[d, dash] & \textbf{List} & \textbf{SortedMap} & \textbf{ListIterator} \\
\textbf{SortedSet} & & &
\end{tikzcd}$$

❖ Collection:

- A collection allows a group of objects to be treated as a single unit.
- The Java collections library forms a framework for collection classes.
- The CI is the root of collection hierarchy and is used for common functionality across all collections.
- There is no direct implementation of Collection Interface.

Two fundamental interfaces for containers:

- Collection

boolean add(Object element): Inserts element into a collection

Set Interface: extends *Collection* Interface. The Class Hash set implements Set Interface.

- Is used to represent the group of *unique* elements.
- Set stores elements in an unordered way but does *not* contain duplicate elements.
- identical to Collection interface, but doesn’t accept duplicates.

Sorted set: extends *Set* Interface. The class *Tree Set* implements Sorted set Interface.

- It provides the extra functionality of keeping the elements sorted.
- It represents the collection consisting of *Unique*, sorted elements in *ascending order*.
- expose the comparison object for sorting.

❖ **List Interface:**

- ordered collection – Elements are added into a particular position.
- Represents the sequence of numbers in a fixed order.
- But may contain duplicate elements.
- Elements can be inserted or retrieved by their position in the List using Zero based index.
- List stores elements in an ordered way.

Map Interface: Basic Interface.The classes *Hash Map & HashTable* implements Map interface.

- Used to represent the mapping of unique keys to values.
- By using the key value we can retrive the values.
- Two basic operations are get() & put().

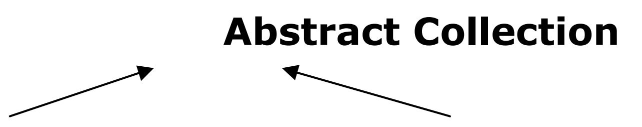
boolean put(Object key, Object value) : Inserts given value into map with key
Object get(Object key): Reads value for the given key.

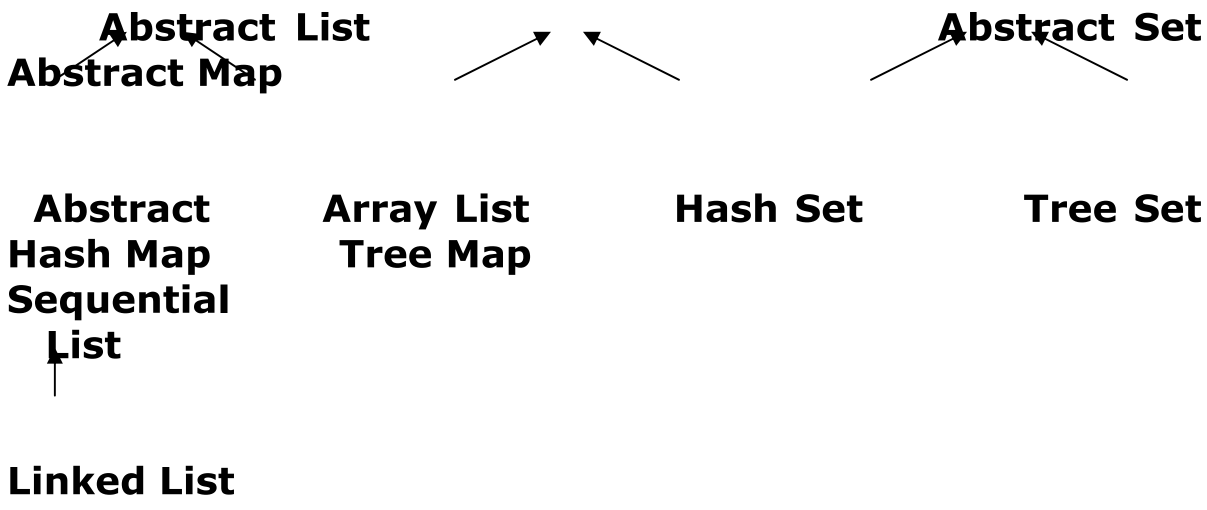
Tree Map Class: Implements Sorted Set Interface.

- The elements are stored in *sorted ascending order*.
- Using *key value* we can retrieve the data.
- Provides an efficient means of storing *key/value pairs* in sorted order and allows *rapid retrivals*.

```
TreeMap tm=new TreeMap();
tm.put( “Kishore”, new Double(74.6));
```

❖ **The Classes in Collections Framework**





ArrayList

- Similar to Vector: it encapsulates a dynamically reallocated Object[] array
- Why use an ArrayList instead of a Vector?
- All methods of the Vector class are synchronized, It is safe to access a Vector object from two threads.
- ArrayList methods are not synchronized, use ArrayList in case of no synchronization
- Use *get* and *set* methods instead of *elementAt* and *setElementAt* methods of vector

HashSet

- Implements a set based on a hashtable
- The default constructor constructs a hashtable with 101 buckets and a load factor of 0.75
  HashSet(int initialCapacity)
  HashSet(int initialCapacity,float loadFactor)
  loadFactor is a measure of how full the hashtable is allowed to get before its capacity is automatically increased
  - Use Hashset if you don’t care about the ordering of the elements in the collection.

TreeSet

- Similar to hash set, with one added improvement
- A tree set is a *sorted collection*
- Insert elements into the collection in any order, when it is iterated, the values are automatically presented in sorted order

Maps: Two implementations for maps:
HashMap

- hashes the keys.
- The Elements may not in Order.
- Hash Map is *not synchronized* and permits null values.
- Hash Map is *not serialized*.
- Hash Map supports *Iterators*.

TreeMap

- uses a total ordering on the keys to organize them in a search tree.
- The hash or comparison function is applied *only to the keys.*
- The values associated with the keys are not hashed or compared.

❖ **How are memory leaks possible in Java?**
If any object variable is still pointing to some object which is of no use, then JVM will not garbage collect that object and object will remain in memory creating memory leak

❖ **What are the differences between EJB and Java beans?**
the main difference is Ejb componenets are distributed which means develop once and run anywhere. java beans are not distributed. which means the beans cannot be shared .

❖ **What would happen if you say this = null?**
this will give a compilation error as follows cannot assign value to final variable this

❖ **Will there be a performance penalty if you make a method synchronized? If so, can you make any design changes to improve the performance?**
Yes, the performance will be down if we use synchronization.

one can minimise the penalty by including garbage collection algorithm, which reduces the cost of collecting large numbers of short - lived objects. and also by using Improved thread synchronization for invoking the synchronized methods.the invoking will be faster.

❖ **How would you implement a thread pool?**

public class ThreadPool extends java.lang.Object implements ThreadPoolInt

This class is an generic implementation of a thread pool, which takes the following input

a) Size of the pool to be constructed
b) Name of the class which implements Runnable (which has a visible default constructor) and constructs a thread pool with active threads that are waiting for activation. once the threads have finished processing they come back and wait once again in the pool.

This thread pool engine can be locked i.e. if some internal operation is performed on the pool then it is preferable that the thread engine be locked. Locking ensures that no new threads are issued by the engine. However, the currently executing threads are allowed to continue till they come back to the passivePool

❖ **How does serialization work?**

Its like FIFO method (first in first out).

❖ **How does garbage collection work?**

There are several basic strategies for garbage collection: reference counting, mark-sweep, mark-compact, and copying. In addition, some algorithms can do their job incrementally (the entire heap need not be collected at once, resulting in shorter collection pauses), and some can run while the user program runs (concurrent collectors). Others must perform an entire collection at once while the user program is suspended (so-called stop-the-world collectors). Finally, there are hybrid collectors, such as the generational collector employed by the 1.2 and later JDKs, which use different collection algorithms on different areas of the heap.

❖ **How would you pass a java integer by reference to another function?**

Passing by reference is impossible in JAVA but Java support the object reference so.

Object is the only way to pass the integer by refrence.

❖ **What is the sweep and paint algorithm?**

The painting algorithm takes as input a source image and a list of brush sizes. sweep algo is that it computes the arrangement of n lines in the plane... a correct algorithm,

- ❖ **Can a method be static and synchronized?**
  no a static mettod can't be synchronized.

- ❖ **Do multiple inheritance in Java?**
  Its not possible directly. That means this feature is not provided by Java, but it can be achieved with the help of Interface. By implementing more than one interface.

- ❖ **What is data encapsulation? What does it buy you?**
  The most common example I can think of is a javabean. Encapsulation may be used by creating 'get' and 'set' methods in a class which are used to access the fields of the object. Typically the fields are made private while the get and set methods are public.

  dEncapsulation can be used to validate the data that is to be stored, to do calculations on data that is stored in a field or fields, or for use in introspection (often the case when using javabeans in Struts, for instance).

- ❖ **What is reflection API? How are they implemented?**
  Reflection package is used mainlyfor the purpose of getting the class name. by using the getName method we can get name of the class for particular application.

  Reflection is a feature of the Java programming language. It allows an executing Java program to examine or "introspect" upon itself, and manipulate internal properties of the program.

- ❖ **What are the primitive types in Java?**
  According to Java in a Nutshell, 5th edition.
  boolean, byte, char, short, long float, double, int.

- ❖ **Is there a separate stack for each thread in Java?**
  No

- ❖ **What is heap in Java?**

JAVA is fully Object oriented language. It has two phases first one is Compilation phase and second one is interpratation phase. The Compilation phase convert the java file to class file (byte code is only readable format of JVM) than Intepratation phase interorate the class file line by line and give the proper result.

- **In Java, how are objects / values passed around?**
  In Java Object are passed by reference and Primitive data is always pass by value
  Do primitive types have a class representation
  Primitive data type has a wrapper class to present.
  Like for int - Integer , for byte Byte, for long Long etc...

- **How all can you free memory?**
  With the help of finalize() method.

  If a programmer really wants to explicitly request a garbage collection at some point, System.gc() or Runtime.gc() can be invoked, which will fire off a garbage collection at that time.

- **Does java do reference counting?**
  It is more likely that the JVMs you encounter in the real world will use a tracing algorithm in their garbage-collected heaps.

- **What does a static inner class mean? How is it different from any other static member?**
  A static inner class behaves like any "outer'' class. It may contain methods and fields.

  It is not necessarily the case that an instance of the outer class exists even when we have created an instance of the inner class. Similarly, instantiating the outer class does not create any instances of the inner class.

  The methods of a static inner class may access all the members (fields or methods) of the inner class but they can access only static members (fields or methods) of the outer class. Thus, f can access the field x, but it cannot access the field y.

- **How do you declare constant values in java?**
  Using Final keyword we can declare the constant values How all can you instantiate final members Final member can be instantiate only at the time of declaration. null

- **How is serialization implemented in Java?**
  A particular class has to implement an Interface java.io.Serializable for implementing serialization. When you have an object passed to a method and when the object is reassigned to a different one, then is the original reference lost No Reference is not lost. Java always passes the object by reference, now two references is pointing to the same object.

- **What are the different kinds of exceptions? How do you catch a Runtime exception?**
  There are 2 types of exceptions.
  1. Checked exception
  2. Unchecked exception.
  1. **Checked exception** is catched at the compile time while unchecked exception is checked at run time. Environmental error that cannot necessarily be detected by testing;
     e.g. disk full, broken socket, database unavailable, etc.
  2. **Unchecked exception**
     Errors: Virtual machine error: class not found, out of memory, no such method, illegal access to private field, etc.
     Runtime Exceptions: Programming errors that should be detected in testing: index out of bounds, null pointer, illegal argument, etc.
     Checked exceptions must be handled at compile time. Runtime exceptions do not need to be. Errors often cannot be.

- **What are the differences between JIT and HotSpot?**
  The Hotspot VM is a collection of techniques, the most significant of which is called "adaptive optimization.

  The original JVMs interpreted bytecodes one at a time. Second-generation JVMs added a JIT compiler, which compiles each method to native code upon first execution, then executes the native code. Thereafter, whenever the method is called, the native code is executed. The adaptive optimization technique used by Hotspot is a hybrid approach, one that combines bytecode interpretation and run-time compilation to native code.

Hotspot, unlike a regular JIT compiling VM, doesn't do "premature optimization"

- **What is a memory footprint? How can you specify the lower and upper limits of the RAM used by the JVM? What happens when the JVM needs more memory?**
  when JVM needs more memory then it does the garbage collection, and sweeps all the memory which is not being used.

- **What are the disadvantages of reference counting in garbage collection?**
  An advantage of this scheme is that it can run in small chunks of time closely interwoven with the execution of the program. This characteristic makes it particularly suitable for real-time environments where the program can't be interrupted for very long. A disadvantage of reference counting is that it does not detect cycles. A cycle is two or more objects that refer to one another, for example, a parent object that has a reference to its child object, which has a reference back to its parent. These objects will never have a reference count of zero even though they may be unreachable by the roots of the executing program. Another disadvantage is the overhead of incrementing and decrementing the reference count each time. Because of these disadvantages, reference counting currently is out of favor.

- **Is it advisable to depend on finalize for all cleanups?**
  The purpose of finalization is to give an opportunity to an unreachable object to perform any clean up before the object is garbage collected, and it is advisable.

- **can we declare multiple main() methods in multiple classes. ie can we have each main method in its class in our program?**
  YES.

JDBC

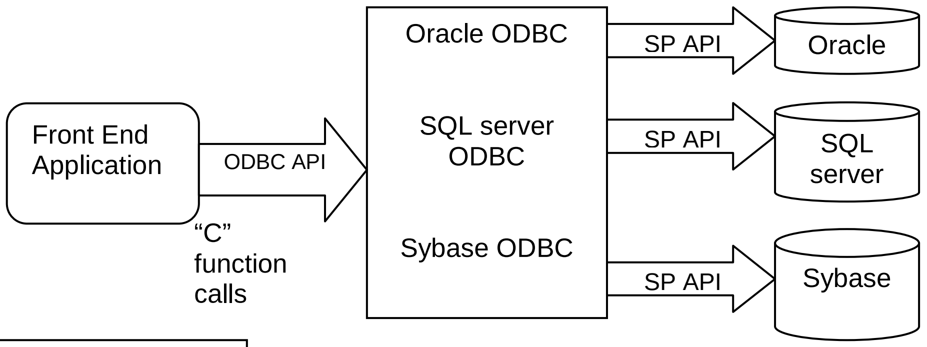
❖ **How to Interact with DB?**

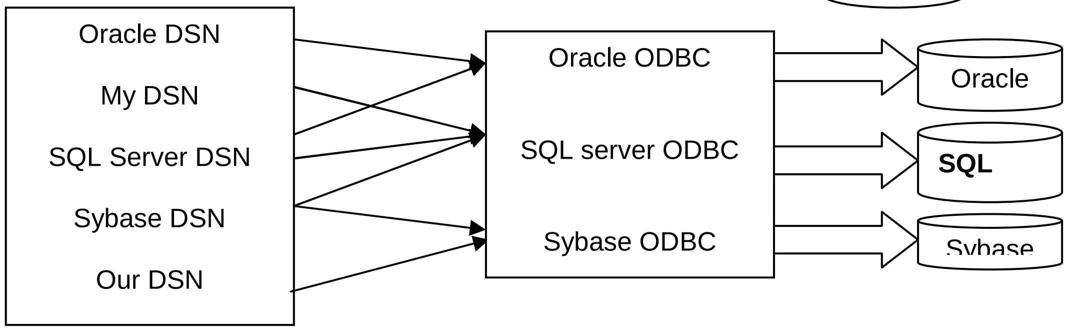
Generally every DB vendor provides a User Interface through which we can easily execute SQL query's and get the result (For example Oracle Query Manager for Oracle, and TOAD (www.quest.com) tool common to all the databases). And these tools will help DB developers to create database. But as a programmer we want to interact with the DB dynamically to execute some SQL queries from our application (Any application like C, C++, JAVA etc), and for this requirement DB vendors provide some Native Libraries (Vendor Specific) using this we can interact with the DB i.e. If you want to execute some queries on Oracle DB, oracle vendor provides an OCI (Oracle Call Interface) Libraries to perform the same.

❖ **About ODBC: What is ODBC?**

ODBC (Open Database Connectivity) is an ISV (Independent software vendor product) composes of native API to connect to different databases through via a single API called ODBC.
Open Database Connectivity (ODBC) is an SQL oriented application programming interface developed by in collaboration with IBM and some other database vendors.
ODBC comes with Microsoft products and with all databases on Windows OS.

❖ **ODBC Architecture**





- ❖ **Advantages**
  - Single API (Protocol) is used to interact with any DB
  - Switching from one DB to another is easy
  - Doesn't require any modifications in the Application when you want to shift from one DB to other.

- ❖ **What for JDBC?**

  As we have studied about ODBC and is advantages and came to know that it provides a common API to interact with any DB which has an ODBC Service Provider's Implementation written in Native API that can be used in your applications.
  If an application wants to interact with the DB then the options which have been explained up to now in this book are:
  1. Using Native Libraries given by the DB vendor
  2. Using ODBC API

  And we have listed there Advantages and Disadvantages.
  But if the application is a JAVA application then the above given options are not recommended to be used due to the following **reasons**
  1. Native Libraries given by DB vendor
     a. Application becomes vendor dependent and
     b. The application has to use JNI to interact with Native Lib which may cause serious problem for Platform Independency in our applications.
  2. And the second option given was using ODBC API which can solve the 1.a problem but again this ODBC API is also a Native API, so we have to use JNI in our Java applications which lead to the 1.b described problem.

And the answer for these problems is JDBC (**J**ava **D**ata **B**ase **C**onnectivity) which provides a common Java API to interact with any DB.

❖ What is JDBC?

As explained above JDBC standards for **J**ava **D**ata **B**ase **C**onnectivity. It is a specification given by Sun Microsystems and standards followed by X/Open SAG (SQL Access Group) CLI (Call Level Interface) to interact with the DB.
Java programing language methods. The JDBC API provides database-independent connectivity between the JAVA Applications and a wide range of tabular data bases. JDBC technology allows an application component provider to:

- Perform connection and authentication to a database server
- Manage transactions
- Moves SQL statements to a database engine for preprocessing and execution
- Executes stored procedures
- Inspects and modifies the results from SELECT statements

❖ JDBC API

JDBC API is divided into two parts
1. JDBC Core API
2. JDBC Extension or Optional API

❖ JDBC Core API (java.sql package)

This part of API deals with the following futures
1. Establish a connection to a DB
2. Getting DB Details
3. Getting Driver Details
4. maintaining Local Transaction
5. executing query's
6. getting result's (ResultSet)
7. preparing pre-compiled SQL query's and executing
8. executing procedures & functions

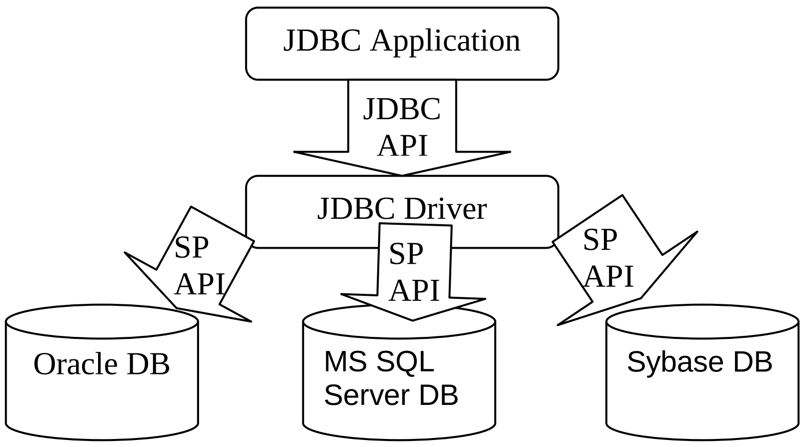
❖ JDBC Ext OR Optional API (javax.sql package)

This part of API deals with the following futures

1. Resource Objects with Distributed Transaction Management support
2. Connection Pooling.

These two parts of Specification are the part of J2SE and are inherited into J2EE i.e. this specification API can be used with all the component's given under J2SE and J2EE.

❖ JDBC Architecture:



In the above show archetecture diagram the JDBC Driver forms an abstraction layer between the JAVA Application and DB, and is implemented by 3rd party vendors or a DB Vendor. But whoever may be the vendor and what ever may be the DB we need not to worry will just us JDCB API to give instructions to JDBC Driver and then it's the responsibility of JDBC Driver Provider to convert the JDBC Call to the DB Specific Call.

And this 3rd party vendor or DB vendor implemented Drivers are classified into 4-Types namely

❖ Types Of Drivers:

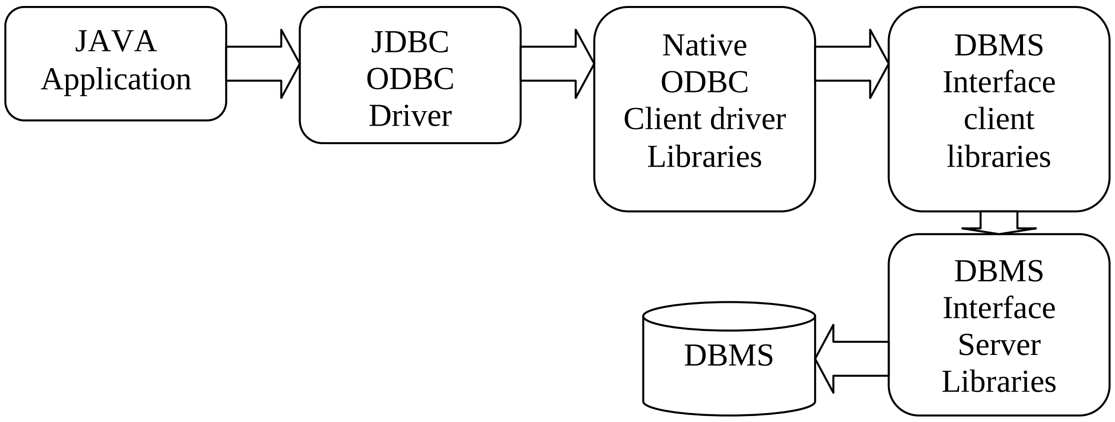
1. Type-1 (JDBC ODBC-Bridge Driver) **JDBC-ODBC Bridge Driver**
2. Type-2 (Java-Native API Driver) **Native API Partly JAVA Driver (Thick Driver)**

3. Type-3 (Java Net Protocol Driver) **Intermediate DataBase Access Server**
4. Type-4 (Java Native Protocol driver) **Pure JAVA Driver (Thin driver)**

Type-1 : <u>JDBC-ODBC Bridge Driver</u> :

Since ODBC is written in C-language using pointers, so JAVA does't support pointers, a java program can't communate directly with the DataBase. The JDBC-ODBC bridge drivertransulates JDBC API calls to ODBC API calls.

Architecture


JAVA Application
JDBC ODBC Driver
Native ODBC Client driver Libraries
DBMS Interface client libraries
DBMS Interface Server Libraries
DBMS

This type of Driver is designed to convert the JDBC request call to ODBC call and ODBC response call to JDBC call.

The JDBC uses this interface in order to communicate with the database, so neither the database nor the middle tier need to be Java compliant. However ODBC binary code must be installed on each client machine that uses this driver. This bridge driver uses a configured data source.

Advantages

- Simple to use because ODBC drivers comes with DB installation/Microsoft front/back office product installation
- JDBC ODBC Drivers comes with JDK software

Disadvantages

- More number of layers between the application and DB. And more number of API conversions leads to the downfall of the performance.
- Slower than type-2 driver

Where to use?

This type of drivers are generaly used at the development time to test your application's.
Because of the disadvantages listed above it is not used at production time. But if we are not available with any other type of driver implementations for a DB then we are forced to use this type of driver (for example **Microsoft Access**).

Examples of this type of drivers
JdbcOdbcDriver from sun
Sun's JdbcOdbcDriver is one of type-1 drivers and comes along with sun j2sdk (JDK).

Setting environment to use this driver

1. **Software**
   ODBC libraries has to be installed.
2. **classpath**
   No additional classpath settings are required apart from the runtime jar (c:\j2sdk1.4\jre\lib\rt.jar) which is defaultly configured.
3. **Path**
   No additional path configuration is required.

How to use this driver

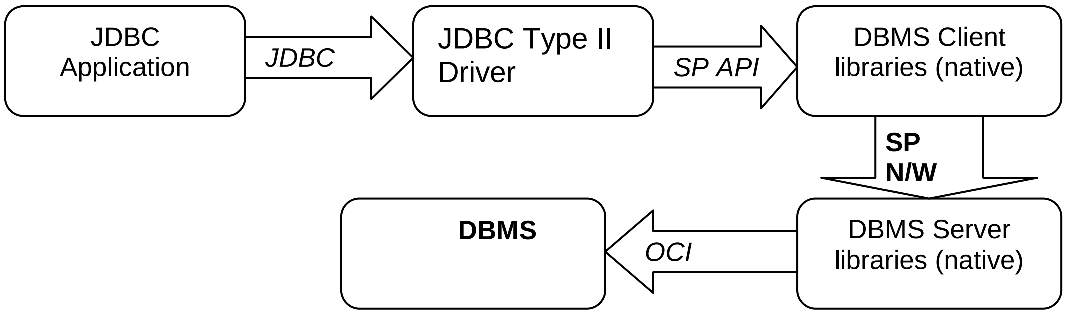
1. **Driver class name** → sun.jdbc.odbc.JdbcOdbcDriver
2. **Driver URL** → dbc:odbc:<DSN>
   here <DSN> (**D**ata **S**ource **N**ame) is an ODBC datasource name which is used by ODBC driver to locate one of the ODBC Service Provider implementation API which can in-turn connect to DB.
   Steps to create <DSN>
   1. run '**Data Sources (ODBC)'** from Control Panal\Administrative Tools\
      (for Windows 2000 server/2000 professional/XP)
      run **'ODBC Data Sources'** from Control Panel\

2. click on **Add** button available on the above displayed screen.
this opens a new window titled '**Create New Data Source**'

which displays all the available DB's lable DB's ODBC drivers currently installed on your system.
3. Select the suitable driver and click on **Finish**
4. Give the required info to the driver (like username, service id etc)

Type-2 : *Native API Partly JAVA Driver (Thick Driver)* :
JDBC Database calls are translated into Vendor-specific API calls. The database will process the request and send the results back through API to JDBC Driver – this will translate the results to the JDBC standard and return them to the Java application.
The Vendor specific language API must be installed on every client that runs the JAVA application.

Architecture


JDBC Application
JDBC
JDBC Type II Driver
SP API
DBMS Client libraries (native)
SP N/W
DBMS Server libraries (native)
OCI
DBMS

This driver converts the JDBC call given by the Java application to a DB specific native call (i.e. to C or C++) using JNI (**J**ava **N**ative **I**nterface).
Advantages : Faster than the other types of drivers due to native library participation in socket programing.
Disadvantage : DB spcifiic native client library has to be installed in the client machine.

- Preferablly work in local network environment because network service name must be configured in client system

Where to use?
This type of drivers are suitable to be used in server side applications.

Not recommended to use with the applications using two tire model (i.e. client and database layer's) because in this type of model client used to interact with DB using the driver and in such a situation the client system sould have the DB native library.

Examples of this type of drivers

1. OCI 8 (**O**racle **C**all **I**nterface) for Oracle implemented by Oracle Corporation.

Setting environment to use this driver

- **Software:** Oracle client software has to be installed in client machine
- **classpath** → %ORACLE_HOME%\ora81\jdbc\lib\classes111.zip
- **path** → %ORACLE_HOME%\ora81\bin

How to use this driver

- **Driver class name** → oracle.jdbc.driver.OracleDriver
- **Driver URL** → jdbc:oracle:oci8:@TNSName

Note: TNS Names of Oracle is available in Oracle installed folder %ORACLE_HOME%\Ora81\network\admin\tnsnames.ora

2. Weblogic Jdriver for Oracle implemented by BEA Weblogic:

Setting environment to use this driver

- Oracle client software has to be installed in client machine
- weblogicoic dll's has to be set in the path
- **classpath** → d:\bea\weblogic700\server\lib\weblogic.jar
- **path** → %ORACLE_HOME%\ora81\bin; d:\bea\weblogic700\server\bin\<subfolder><sub folder> is
  - **oci817_8** if you are using Oracle 8.1.x
  - **oci901_8** for Oracle 9.0.x
  - **oci920_8** for Oracle 9.2.x

How to use this driver

- **Driver class name** → weblogic.jdbc.oci.Driver
- **Driver URL** → jdbc:weblogic:oracle:HostName

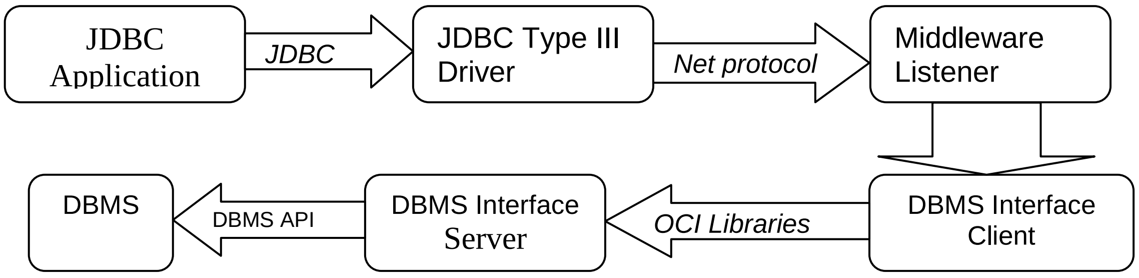
Type-3 Intermediate DataBase Access Server :

Type-3 Driver uses an Intermediate(middleware) database driver that has the ability to connect multiple JAVA clients to multiple database servers.

Client connect to the Databse server via an Intermediate server component (such as listener) that acts as a gateway for multple database servers.

Bea weblogic includes Type-3 Driver.

Architecture :



This type of drivers responsibility is to convert JDBC call to Net protocol (Middleware listener dependent) format and redirect the client request to Middleware Listener and middleware listener inturn uses type-1, type-2 or type-4 driver to interact with DB.

Advantages:

- It allows the flexibility on the architecture of the application.
- In absence of DB vendor supplied driver we can use this driver
- Suitable for Applet clients to connect DB, because it uses Java libraries for communication between client and server.

Disadvantages:

- From client to server communication this driver uses Java libraries, but from server to DB connectivity this driver uses native libraries, hence number of API conversion and layer of interactions increases to perform operations that leads to performance deficit.
- Third party vendor dependent and this driver may not provide suitable driver for all DBs

Where to use?

- Suitable for Applets when connecting to databases

Examples of this type of drivers:

1. IDS Server (**Intersolv**) driver available for most of the Databases

Setting environment to use this driver

- **Software:** IDS software required to be downloaded from the following URL

[http://www.idssoftware.com/idsserver.html -> Export Evaluation]

- **classpath** → C:\IDSServer\classes\jdk14drv.jar
- **path** →

How to use this driver

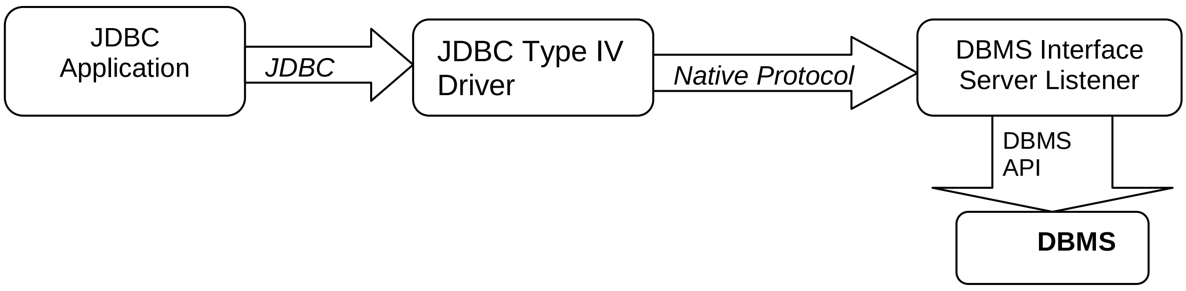
- **Driver class name** → ids.sql.IDSDriver
- **Driver URL** → jdbc:ids://localhost:12/conn?dsn='IDSExamples'

Note: DSN Name must be created in ServerDSN

Type-4 <u>Pure JAVA Driver (Thin driver)</u> :

Type-4 Driver translates JDBC-API calls to direct network calls using vendor specific networking protocols by making direct server connections with the database.

Architecture



This type of driver converts the JDBC call to a DB defined native protocol.

Advantage

- Type-4 driver are simple to deploy since there is No client native libraries required to be installed in client machine
- Comes with most of the Databases

Disadvantages:

- Slower in execution compared with other JDBC Driver due to Java libraries are used in socket communication with the DB

Where to use?

- This type of drivers are sutable to be used with server side applications, client side application and Java Applets also.

Examples of this type of drivers

1) Thin driver for Oracle implemented by Oracle Corporation
Setting environment to use this driver

- **classpath** → %ORACLE_HOME%\ora81\jdbc\lib\classes111.zip

How to use this driver

- **Driver class name** → oracle.jdbc.driver.OracleDriver
- **Driver URL** → jdbc:oracle:thin:@HostName:<port no>:<SID>

<port no> → 1521
<SID> -> ORCL

2) MySQL Jconnector for MySQL database
Setting environment to use this driver

- **classpath** → C:\mysql\mysql-connector-java-3.0.8-stable\mysql-connector-java-3.0.8-stable-bin.jar

How to use this driver

- **Driver class name** → com.mysql.jdbc.Driver
- **Driver URL** → jdbc:mysql:///test

Chapter 3 [JDBC Core API]

In this chapter we are going to discuss about 3 versions of JDBC: JDBC 1.0, 2.0 and 3.0

Q) How JDBC API is common to all the Databases and also to all drivers?

A) Fine! The answer is JDBC API uses Factory Method and Abstract Factory Design pattern implementations to make API common to all the Databases and Drivers. In fact most of the classes available in JDBC API are interfaces, where Driver vendors must provide implementation for the above said interfaces.

Q) Then how JDBC developer can remember or find out the syntaxes of vendor specific classes?

A) No! developer need not have to find out the syntaxes of vendor specific implementations why because DriverManager is one named class available in JDBC API into which if you register Driver class name, URL, user and password, DriverManager class in-turn brings us one Connection object.
Q) Why most of the classes given in JDBC API are interfaces?
A) Why abstract class and abstract methods are?
Abstract class forces all sub classes to implement common methods whichever are required implementations. Only abstract method and class can do this job. That's' why most part of the JDBC API is a formation of interfaces.

JDBC API comes in 2 packages
java.sql.*
javax.sql.*
First of all I want to discuss briefly about all the list of interfaces and classes available in java.sql. package
Interfaces index
Driver
Every JDBC Driver vendor must one sub class of this class for initial establishment of Connections. DriverManager class need to be first registered with this class before accepting URL and other information for getting DB connection.

Method index

- Connection connect(String url, Properties info)
  This method takes URL argument and user name & password info as Properties object
- boolean acceptURL(String url)
  This method returns boolean value true if the given URL is correct, false if any wrong in URL
- boolean jdbcComplaint()
  JDBC compliance requires full support for the JDBC API and full support for SQL 92 Entry Level. It is expected that JDBC compliant drivers will be available for all the major commercial databases.

<u>Connection</u>
Connection is class in-turn holds the TCP/IP connection with DB. Functions available in this class are used to manage connection live-ness as long as JDBC application wants to

connect with DB. The period for how long the connection exists is called as Session. This class also provides functions to execute various SQL statements on the DB. For instance the operations for DB are mainly divided into 3 types

- DDL (create, alter, and drop)
- DML (insert, select, update and delete)
- DCL (commit, rollback) and also
- call function_name (or) call procedure_name

Method Index

- Statement createStatement()
- PreparedStatement prepareStatement(String preSqlOperation)
- CallableStatement prepareCall(String callToProc())

<u>Statement</u>

Statement class is the super class in its hierarchy. Provides basic functions to execute query (select) and non-related (create, alter, drop, insert, update, delete) query operations.

Method Index

- int executeUpdate(String sql)

This function accepts non-query based SQL operations; the return value int tells that how many number of rows effected/updated by the given SQL operation.

- ResultSet executeQuery(String sql)

This function accepts SQL statement SELECT and returns java buffer object which contains temporary instance of SQL structure maintaining all the records retrieved from the DB. This object exists as long as DB connection exist.

- boolean execute()

This function accepts all SQL operations including SELECT statement also.

<u>PreparedStatement</u>

PreparedStatement class is sub classing from Statement class. While connection class prepareStatement function is creating one new instance this class, function takes one String argument that contains basic syntax of SQL operation represented with "?" for IN parameter representation. In the further stages of the JDBC program, programmer uses setXXX(int index, datatype identifier) to pass values into IN parameter and requests exdcute()/ exuecteUpdate() call.

Method Index

- setInt(int index, int value) – similar functions are provided for all other primitive parameters
- setString(int index, String value)
- setObject(int index, Object value)
- setBinaryStream(int index, InputStream is, int length)

CallableStatement

ResultSet ResultSetMetaData
DatabaseMetaData

BLOB	CLOB	REF
SavePoint	Struct	
SQLInput	SQLOutput	SQLData

Class diagram required here

```
// TypeI DriverTest,java
package com.digitalbook.j2ee.jdbc;
import java.sql.*;
public class TypeIDriverTest
{
    Connection con;
    Statement stmt;
    ResultSet rs;
   public TypeIDriverTest ()
   {
      try {
       // Load driver class into default ClassLoader
       Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
          // Obtain a connection with the loaded driver
       con                              =DriverManager.getConnection
("jdbc:odbc:digitalbook","scott","tiger");
          URL                          String                          -
```
("<protocol>:<subprotocol>:<subname>", " ", " "); }
```
       // create a statement
       st=con.createStatement();
       //execute SQL query
       rs =st.executeQuery ("select ename,sal from emp");
       System.out.println ("Name                    Salary");
       System.out.println ("--------------------------------");
       while(rs.next())
       {
       System.out.println                          (rs.getString(1)+"
"+rs.getString(2));
```

```
      }
      rs.close ();
      stmt.close ();
      con.close ();
      }
      catch(Exception e)
      {
      e.printStackTrace ();
      }
        }
      public static void main (String args[])
      {
            TypeIDriverTest demo=new TypeIDriverTest ();
      }
 }
```

// TypeIIDriverTest,java

```
package com.digitalbook.j2ee.jdbc;
import java.sql.*;
public class TypeIIDriverTest
{
   Connection con;
   Statement stmt;
   ResultSet rs;
  public TypeIIDriverTest ()
  {
     try {
      // Load driver class into default ClassLoader
      Class.forName ("oracle.jdbc.driver.OracleDriver");
        // Obtain a connection with the loaded driver
con                              =DriverManager.getConnection
("jdbc:oracle:oci8:@digital","scott","tiger");
       // create a statement
      st=con.createStatement();
      //execute SQL query
      rs =st.executeQuery ("select ename,sal from emp");
      System.out.println ("Name                      Salary");
      System.out.println ("--------------------------------");
      while(rs.next())
      {
```

```
      System.out.println                              (rs.getString(1)+"
"+rs.getString(2));
      }
      rs.close ();
      stmt.close ();
      con.close ();
      }
      catch(Exception e)
      {
      e.printStackTrace ();
      }
        }
      public static void main (String args[])
      {
            TypeIIDriverTest demo=new TypeIIDriverTest ();
      }
 }
```

Chapter 9 : **[javax.sql package]**

This package supplements the java.sql package and is included as a part of JDK 1.4 version. This package mainly provides following features:

1. DataSource interface was introduced in substitution to DriverManager class for getting connection objects.
2. Connection Pooling
3. Distributed TX management
4. RowSets

Applications can directly use DataSource and RowSet API but connection pooling and Distributed TX management APIs are used internally by the middle-tier infrastructure.

DataSource

DataSource is an interface. Driver vendor will provide implementation for this interface (That means in case JDBC Driver Type II driver Oracle vendor for Oracle DB, Intersolv in case of IDSServer). This object is used to obtain connections into any type of JDBC program. Though DriverManager class is ideal for getting DB connection object, this class provides some extra features over DriverManager class:

- Applications will obtain DB connection objects through via this factory class

- DataSource object will be registered into JNDI, hence any application connected in the network can obtain this object by requesting JNDI API, DataSource class is having one method called getConnection() geives one Connection object
- Application do not need to hard code a driver class
- Changes can be made to a data source properties, which means that it is not necessary to make changes in application code when something about the data source or driver changes
- Connection pooling and Distributed transactions are available through only the connection obtained from this object. Connection obtained through DriverManager class do not have this capability

DataSource interface is implemented by driver vendor. There are 3 types of implementations available:

1. **Basic Implementation-** Produces a standard connection object.
2. **Connection Pooling Implementation-** Produces a connection object that automatically participates in connection pooling. This implementation works with a middle-tier connection pooling manager.
3. **Distributed transaction implementation-** Produces a connection object that may be used for distributed transactions and almost always participates in connection pooling. This implementation works with a middle-tier transaction manager and almost always with a connection pool manager.

A driver that is accessed via a DataSource object does not register itself with the DriverManager. Rather, a DataSource object is retrieved though a lookup operation and then used to create a Connection object. With a basic implementation, the connection obtained through a DataSource object is identical to a connection obtained through the DriverManager facility.

Method Index

- Connection getConnection() – This function returns Connection object on demand of this method.
- Connection getConnection(String user, String pass) – This function returns Connection object on demand of this method by passing username and password.

Sub classes of this interface are
Type III Driver – IDSServer – Intersolv – ids.jdbc.IDSDataSource

Type III Driver – WebLogic – BEA – weblogic.jdbc.jta.DataSource – XA Support

Connection Pooling

Connections made via a DataSource object that is implemented to work with a middle tier connection pool manager will participate in connection pooling. This can improve the performance dramatically because creating a new connection is very expensive.

Connection Pool provides following features:

- Substantial improvement in the performance of DB application can be accomplished by pre-caching the DB connection objects
- CPM supplied DB connections are remote enable
- CPM supplied DB connections are cluster aware
- CPM supplied DB connections supports DTM (distributed TXs)
- CPM supplied DB connections are not actual DB Connection objects, in turn they are remote object, hence even though client closes DB connection using con.close() the actual connection may not be closed instead RMI connection between client to CPM are closed
- CPM supplied DB connection objects are serializable, hence client from any where in the network can access DB connections

The classes and interfaces used for connection pooling are:

1. ConnectionPoolDataSource
2. PooledConnection
3. ConnectionEvent
4. ConnectionEventListener

Connection Pool Manager resided on middle tier system uses these classes and interfaces behind the scenes. When the ConnectionPooledDataSource object is called on to create PooledConnection object, the connection pool manager will register as a ConnectionEventListener object with the new PooledConnection object. When the connection is closed or there is an error, the connection pool manager (being listener) gets a notification that includes a ConnectionEvent object.

Distributed Transactions

As with pooled connections, connections made via data source object that is implemented to work with the middle tier infrastructure may participate in distributed transactions. This gives an application the ability to involve data sources on multiple servers in a single transaction.
The classes and interfaces used for distributed transactions are:

- XADataSource
- XAConnection

These interfaces are used by transaction manager; an application does not use them directly.

The XAConnection interface is derived from the PooledConnection interface, so what applies to a pooled connection also applies to a connection that is part of distributed transaction. A transaction manager in the middle tier handles everything transparently. The only change in application code is that an application cannot do anything that would interfere with the transaction manager's handling of the transaction. Specifically application cannot call the methods Connection.commit or Connection.rollback and it cannot set the connection to be in auto-commit mode.

An application does not need to do anything special to participate in a distributed transaction. It simply creates connections to the data sources it wants to use via the DataSource.getConnection method, just as it normally does. The transaction manager manages the transaction behind the scenes. The XADataSource interface creates XAConnection objects, and each XAConnection object creates an XAResource object that the transaction manager uses to manage the connection.

<u>Rowsets</u>

The RowSet interface works with various other classes and interfaces behind the scenes. These can be grouped into three categories.

1. **<u>Event Notification</u>**

- **RowSetListener**
  A RowSet object is a JavaBeans^{TM} component because it has properties and participates in the JavaBeans event notification mechanism. The RowSetListener interface is implemented by a component that wants to be notified about events that occur to a particular RowSet object. Such a component registers itself as a

listener with a rowset via the RowSet.addRowSetListener method.

- When the RowSet object changes one of its rows, changes all of it rows, or moves its cursor, it also notifies each listener that is registered with it. The listener reacts by carrying out its implementation of the notification method called on it.
- **RowSetEvent**
  As part of its internal notification process, a RowSet object creates an instance of RowSetEvent and passes it to the listener. The listener can use this RowSetEvent object to find out which rowset had the event.

2. Metadata

RowSetMetaData

This interface, derived from the ResultSetMetaData interface, provides information about the columns in a RowSet object. An application can use RowSetMetaData methods to find out how many columns the rowset contains and what kind of data each column can contain.

The RowSetMetaData interface provides methods for setting the information about columns, but an application would not normally use these methods. When an application calls the RowSet method execute, the RowSet object will contain a new set of rows, and its RowSetMetaData object will have been internally updated to contain information about the new columns.

3. The Reader/Writer Facility

A RowSet object that implements the RowSetInternal interface can call on the RowSetReader object associated with it to populate itself with data. It can also call on the RowSetWriter object associated with it to write any changes to its rows back to the data source from which it originally got the rows. A rowset that remains connected to its data source does not need to use a reader and writer because it can simply operate on the data source directly.

RowSetInternal

By implementing the RowSetInternal interface, a RowSet object gets access to its internal state and is able to call on its reader and writer. A rowset keeps track of the values in its current rows and of the values that immediately preceded the current ones, referred to as the *original* values. A rowset also keeps track of

(1) the parameters that have been set for its command and (2) the connection that was passed to it, if any. A rowset uses the RowSetInternal methods behind the scenes to get access to this information. An application does not normally invoke these methods directly.

RowSetReader

A disconnected RowSet object that has implemented the RowSetInternal interface can call on its reader (the RowSetReader object associated with it) to populate it with data. When an application calls the RowSet.execute method, that method calls on the rowset's reader to do much of the work. Implementations can vary widely, but generally a reader makes a connection to the data source, reads data from the data source and populates the rowset with it, and closes the connection. A reader may also update the RowSetMetaData object for its rowset. The rowset's internal state is also updated, either by the reader or directly by the method RowSet.execute.

RowSetWriter

A disconnected RowSet object that has implemented the RowSetInternal interface can call on its writer (the RowSetWriter object associated with it) to write changes back to the underlying data source. Implementations may vary widely, but generally, a writer will do the following:

- Make a connection to the data source
- Check to see whether there is a conflict, that is, whether a value that has been changed in the rowset has also been changed in the data source
- Write the new values to the data source if there is no conflict
- Close the connection

The RowSet interface may be implemented in any number of ways, and anyone may write an implementation. Developers are encouraged to use their imaginations in coming up with new ways to use rowsets.

Type III Driver – WebLogic – BEA – weblogic.jdbc.common.internal.ConnectionPool

Type III Driver – WebLogic – BEA – weblogic.jdbc.connector.internal.ConnectionPool

Type II & IV driver – Oracle DB - Oracle –

- **JDBC:**

There are three types of statements in JDBC
Create statement : Is used to execute single SQL statements.
Prepared statement: Is used for executing parameterized quaries. Is used to run pre-compiled SEQL Statement.
Callable statement: Is used to execute stored procedures.
Stored Procedures: Is a group of SQL statements that perform a logical unit and performs a particular task.
Are used to encapsulate a set operations or queries t execute on data.
execute()   – returns Boolean value
executeupdate( ) – returns resultset Object
executeupdate( ) – returns integer value

Loading the Driver:

```
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  Conn=DriverManager.getConnection("jdbc:odbc:dsn",
"username", "password");
```

(ORACLE Driver)

```
  Class.forName("Oracle.jdbc.driver.OracleDriver");

Conn=DriverManager.getConnection("jdbc:oracle:thin:@192.168.1.
105:1521:dbn", "username", "password");
```

Data base connection:

```
Public static void main(String args[]);
Connection con;
Statement st;
Resultset rs;
try {                      // Getting all rows from Table
      Clas.forName("sun.jdbc.odbc.jdbcodbc");
  Conn=DriverManager.getConnction("jdbc.odbc.dsn", "username" ,
"password");
st = con.createstatement( );
rs = st.executestatement("SELECT * FROM mytable");
   while(rs.next());
      {
       String s= rs.getString(1);    or rs.setString("COL_A");
       int i = rs. getInt(2);
       Float f = rs.getfloat(3);
    Process(s,i,f);
```

```
     }
    catch(SQLException e)
     { }
//Getting particular rows from Table
st = con.createstatement( );
rs = st.executequery(“SELECT * FROM mytable WHERE COL A =
“Prasad”);
      while(rs.next( ));
       {
        String s = rs.getString(1);
         Int i = rs.getint(2);
        Float f = rs.getfloat(3);
      Process(s,i,f);
       }
         Catch(SQLException e);   {  }
//updating a row from table.
try {
st =  con.createstatement( );
int numupdated = st.executeupdate(“UPDATE mytable SET COL_A
= “prasad”  WHERE  COL_B=”746”);
rs = st.executeupdate();
conn.close();  }
catch(SQLExceptione); {  }
// Receiving rows from table
try  {
      st = con.preparedstatement( );
   rs   =   st.executestatement(“SELECT   *   FROM   mytable   SET
COL_A=?’);
  int colunm=1;
  rs.setString(colunm,”hari”);
rs = st.executeQuery( );
//update rwo from table
 st = con.createstatement( );
int numupdated = st.executeupdate(“UPDATE mytable SET COL_A
=? WHERE COL_B=?”);
int column=1;
rs.setString(colunm,”Prasad”);
int column=2;
rs.setString(column,”746”);
int numupdated = st.executeupdate( );
}   catch(SqlException e);  {  }
```

```
//callable statement
try {
    cst = con.preparecall("{call add1(??,??)}");
  cst.setint(1,a);
  cst.setint(2,b);
  cst.registerOurPrameter(1,Types.INTEGER);
  cst.executeQuery( );
 System.out.println("rs.getString( )");  }
```

❖ **Connection Pool with webLogic server:**

You can connect the database in your app using:

```
Class.forName("weblogic.jdbc.oci.Driver").newInstance();
Java.sql.Connection conn = Driver.connect("jdbc:weblogic:Oracle:dbn", "username", "password");
```

(**Or**)

```
java.util.Properties prop = new java.util.Properties();
prop.put("user", "Krishna");
prop.put("password","Kishore");
java.sql.Driver d = (java.sql.Driver)Class.forName("weblogic.jdbc.oci.Driver").new Instance();
java.sql.Connection conn = d.connect("jdbc:weblogic:Oracle:dbn", prop);
public static void main(String args[]) throws Exception {
   java.sql.Connection con=null;
   java.sql.satement st =null;
   try {
      context ctx=null;
      Hashtable ht = new Hashtable();
      ht.put(Context.INTIAL_CONTEXT_FACTORY,"weblogic:jndi:WLInitialContextFACTORY");
      ht.put(Context_PROVIDER_URL,"t3://Localhost:7001");
      //get a context from JNDI lookup
      ctx = newIntialContext():
      java.sql.Datasourse ds =(java.sql.DataSource)ctx.lookup("OraclegbJNDI");
      con =ds.getConnection();
      System.out.Println("Making Connection……");
      st = conn.createstatement();
   } finally {
        try {
            if(stmt !=null)
            stmt.close( );
            if(stmt !=null)
            con.close( );
        }
    }
}
```

❖ **What is a transaction?**

transaction is collection of logical operation that perform a task Transaction should ACID properties.

A for **Automicity**
C for **Consistency**
I for **Isolation**
D for **Durability**

A transaction can be termed as any operation such as storing, retrieving, updating or deleting records in the table that hits the database.

❖ **What is the purpose of setAutoCommit()?**

It is set as ConnectionObject.setAutoComit(); after any updates through the program cannot be effected to the database.We have commit the transctions .For this puprpose we can set AutoCommit flag to Connection Object.

What are the three statements in JDBC & differences between them which is used to run simple sql statements like select and update

1. PrepareStatment is used to run Pre compiled sql.
2. CallableStatement is used to execute the stored procedures.

❖ **What is stored procedure. How do you create stored procedure ?**

Stored procedures is a group of SQL statements that performs a logical unit and performs a particular task.

Stored procedures are used to encapsulate a set of operations or queries to execute on data.

Stored Procedure is a stored program in database, PL/SQL program is a Stored Procedure. Stored Procedures can be called from java by CallableStatement.

A precompiled collection of SQL statements stored under a name and processed as a unit.

Stored procedures can:

1. Accept input parameters and return multiple values in the form of output parameters to the calling procedure or batch.
2. Contain programming statements that perform operations in the database, including calling other procedures.
3. Return a status value to a calling procedure or batch to indicate success or failure (and the reason for failure).

❖ **What are batch updates?**

Batch Update facility allows multiple update operations to be submitted to a database for processing at once. Using batch updates will improve the performance.

❖ **What is the difference between Resultset and Rowset?**

A RowSet is a disconnected, serializable version of a JDBC ResultSet.

The RowSet is different than other JDBC interfaces in that you can write a RowSet to be vendor neutral. A third party could write a RowSet implementation that could be used with any JDBC-compliant database. The standard implementation supplied by Sun uses a ResultSet to read the rows from a database and then stores those rows as Row objects in a Vector inside the RowSet. In fact, a RowSet implementation could be written to get its data from any source. The only requirement is that the RowSet acts as if it was a ResultSet. Of course, there is no reason that a vendor couldn't write a RowSet implementation that is vendor specific.

The standard implementations have been designed to provide a fairly good range of functionality.

The implementations provided are:

CachedRowSetImpl - This is the implementation of the RowSet that is closest to the definition of RowSet functionality that we discussed earlier. There are two ways to load this RowSet. The execute () method will load the RowSet using a Connection object. The populate() method will load the RowSet from a previously loaded ResultSet.

WebRowSetImpl - This is very similar to the CachedRowSetImpl (it is a child class) but it also includes methods for converting the rows into an XML document and loading the RowSet with an XML document. The XML document can come from any Stream or Reader/Writer object. This could be especially useful for Web Services.
JdbcRowSetImpl - This is a different style of implementation that is probably less useful in normal circumstances. The purpose of this RowSet is to make a ResultSet look like a JavaBean. It is not serializable and it must maintain a connection to the database.

The remaining two implementations are used with the first three implementations:
FilteredRowSetImpl - This is used to filter data from an existing RowSet. The filter will skip records that don't match the criteria specified in the filter when a next() is used on the RowSet.
JoinRowSetImpl - This is used to simulate a SQL join command between two or more RowSet objects.

❖ **What are the steps for connecting to the database using JDBC?**
Using DriverManager:
- Load the driver class using class.forName(driverclass) and class.forName() loads the driver class and passes the control to DriverManager class
- DriverManager.getConnection() creates the connection to the databse

Using DataSource:
1. DataSource is used instead of DriverManager in Distributed Environment with the help of JNDI.
2. Use JNDI to lookup the DataSource from Naming service server.
3. DataSource.getConnection method will return Connection object to the database.

❖ **What is Connection Pooling?**
Connection pooling is a cache of data base connections that is maintained in memory , so that the connections may be reuse.

Connection pooling is a place where a set of connections are kept and are used by the different programers with out creating conncections to the database(it means there is a ready made connection available for the programmers where he can use). After using the connection he can send back that connection to the connection pool. Number of connections in connection pool may vary.

❖ **How do you implement Connection Pooling?**
Connection Pooling can be implemented by the following way.
- A javax.sql.ConnectionPoolDataSource interface that serves as a resource manager connection factory for pooled java.sql.Connection objects.
- Each database vendors provide the implementation for that interface.

For example, the oracle vendors implementation is as follows:
- oracle.jdbc.pool.oracleConnectionPoolDataSource Class.
- A javax.sql.PooledConnection interface encapsulates the physical connection for the database.
- Again, the vendor provides the implementation.

❖ **What Class.forName() method will do?**
Class.forName() is used to load the Driver class which is used to connect the application with Database. Here Driver class is a Java class provided by Database vendor.

❖ **What is the difference between JDBC 1.0 and JDBC 2.0?**
The JDBC 2.0 API includes many new features in the java.sql package as well as the new Standard Extension package, javax.sql. This new JDBC API moves Java applications into the world of heavy-duty database computing. New features in the java.sql package include support for SQL3 data types, scrollable result sets, programmatic updates, and batch updates. The new JDBC Standard Extension API, an integral part of Enterprise JavaBeans (EJB) technology, allows you to write distributed transactions that use connection pooling, and it also makes it possible to connect to virtually any tabular data source, including files and spread sheets.

The JDBC 2.0 API includes many new features like

1. **Scrollable result sets**
2. **Batch updates**
3. **Connection Pooling**
4. **Distributed transactions**
5. **set autocomit ()**

❖ **What is JDBC?**

JDBC is a layer of abstraction that allows users to choose between databases. It allows you to change to a different database engine and to write to a single API. JDBC allows you to write database applications in Java without having to concern yourself with the underlying details of a particular database.

❖ **What are the two major components of JDBC?**

One implementation interface for database manufacturers, the other implementation interface for application and applet writers.

❖ **What is JDBC Driver interface?**

The JDBC Driver interface provides vendor-specific implementations of the abstract classes provided by the JDBC API. Each vendors driver must provide implementations of the java.sql.Connection, Statement, PreparedStatement, CallableStatement, ResultSet and Driver.

❖ **What are the common tasks of JDBC?**

Create an instance of a JDBC driver or load JDBC drivers through jdbc.drivers

- Register a driver
- Specify a database
- Open a database connection
- Submit a query
- Receive results

❖ **What packages are used by JDBC?**

There are 8 packages:

- java.sql.Driver,
- Connection,
- Statement,

- PreparedStatement,
- CallableStatement,
- ResultSet,
- ResultSetMetaData,
- DatabaseMetaData.

❖ **What are the flow statements of JDBC?**

A URL string -->getConnection-->DriverManager-->Driver-->Connection-->Statement-->executeQuery-->ResultSet.

1. Register the Driver
2. load the Driver
3. get the connection
4. create the statement
5. Execute the query
6. fetch the results with ResultSet

❖ **What are the steps involved in establishing a connection?**

This involves two steps:

- loading the driver.
- making the connection.

❖ **How can you load the drivers?**

Loading the driver or drivers you want to use is very simple and involves just one line of code. If, for example, you want to use the JDBC-ODBC Bridge driver, the following code will load it:

Example:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Your driver documentation will give you the class name to use. For instance, if the class name is jdbc.DriverXYZ , you would load the driver with the following line of code:

Example:

```
Class.forName("jdbc.DriverXYZ");
```

❖ **What Class.forName will do while loading drivers?**

It is used to create an instance of a driver and **register it with the DriverManager**. When you have loaded a driver, it is available for making a connection with a DBMS.

❖ **How can you make the connection?**
In establishing a connection is to have the appropriate driver connect to the DBMS. The following line of code illustrates the general idea:
Example:
```
String url = "jdbc:odbc:Fred";
Connection con = DriverManager.getConnection(url,
"Fernanda", "J8");
```

❖ **How can you create JDBC statements?**
A Statement object is what sends your SQL statement to the DBMS. You simply create a Statement object and then execute it, supplying the appropriate execute method with the SQL statement you want to send. For a SELECT statement, the method to use is executeQuery. For statements that create or modify tables, the method to use is executeUpdate.
Example:
It takes an instance of an active connection to create a Statement object. In the following example, we use our Connection object con to create the Statement object stmt:
```
Statement stmt = con.createStatement();
```

❖ **How can you retrieve data from the ResultSet?**
First JDBC returns results in a ResultSet object, so we need to declare an instance of the class ResultSet to hold our results. The following code demonstrates declaring the ResultSet object rs.
Example:
```
ResultSet rs = stmt.executeQuery("SELECT COF_NAME, PRICE
FROM COFFEES");
```
Second:
```
String s = rs.getString("COF_NAME");
```
The method getString is invoked on the ResultSet object rs , so getString will retrieve (get) the value stored in the column COF_NAME in the current row of rs

❖ **What are the different types of Statements?**
1) **Create Statement:** For Simple statement used for static query.
2) **Prepared Statement:** For a runtime / dynamic query. Where String is a dynamic query you want to execute

3) **Callable Statement (Use prepareCall):** //For Stored procedure Callable statement, where sql is stored procedure.

```
try {
    Connection conn = DriverManager.getConnection("URL",'USER"."PWD");
    Statement stmt = conn.createStatement();
    PreparedStatement pstmt = conn.prepareStatement(String sql);
    CallableStatement cstmt = conn.prepareCall(String sql);
} catch (SQLException ee) {
    ee.printStackTrace();
}
```

Don't forget all the above statements will throw the SQLException, so we need to use try catch for the same to handle the exception.

- **How can you use PreparedStatement?**

This special type of statement is derived from the more general class, Statement. If you want to execute a Statement object many times, it will normally reduce execution time to use a PreparedStatement object instead. The advantage to this is that in most cases, this SQL statement will be sent to the DBMS right away, where it will be compiled. As a result, the PreparedStatement object contains not just an SQL statement, but an SQL statement that has been precompiled. This means that when the PreparedStatement is executed, the DBMS can just run the PreparedStatement 's SQL statement without having to compile it first.

Example:

```
PreparedStatement updateSales = con.prepareStatement("UPDATE COFFEES SET SALES = ? WHERE COF_NAME LIKE ?");
```

- **How to call a Stored Procedure from JDBC?**

The first step is to create a CallableStatement object. As with Statement an and PreparedStatement objects, this is done with an open Connection object. A CallableStatement object contains a call to a stored procedure;

Example:

```
CallableStatement cs = con.prepareCall("{call SHOW_SUPPLIERS}");
ResultSet rs = cs.executeQuery();
```

❖ **How to Retrieve Warnings?**

SQLWarning objects are a subclass of SQLException that deal with database access warnings. Warnings do not stop the execution of an application, as exceptions do; they simply alert the user that something did not happen as planned. A warning can be reported on a Connection object, a Statement object (including PreparedStatement and CallableStatement objects), or a ResultSet object. Each of these classes has a getWarnings method, which you must invoke in order to see the first warning reported on the calling object

Example:

```
SQLWarning warning = stmt.getWarnings();
if (warning != null) {
  while (warning != null) {
    System.out.println("Message: " + warning.getMessage());
    System.out.println("SQLState: " + warning.getSQLState());
    System.out.print("Vendor error code: ");
    System.out.println(warning.getErrorCode());
    warning = warning.getNextWarning();
  }
}
```

❖ **How to Make Updates to Updatable Result Sets?**

Another new feature in the JDBC 2.0 API is the ability to update rows in a result set using methods in the Java programming language rather than having to send an SQL command. But before you can take advantage of this capability, you need to create a ResultSet object that is updatable. In order to do this, you supply the ResultSet constant CONCUR_UPDATABLE to the createStatement method.

Example:

```
Connection con = DriverManager.getConnection("jdbc:mySubprotocol:mySubName");
```

```
Statement stmt =
con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
ResultSet uprs = ("SELECT COF_NAME, PRICE FROM
COFFEES");
```

Servlets

- **Web Components**
    - Servlets
    - Java Server Pages (JSP)
    - Tags and Tag Libraries

- **What’s a Servlet?**
    - Java’s answer to CGI programming
    - Program runs on Web server and builds pages on the fly

- **When would you use servlets?**
  – Data changes frequently e.g. weather-reports
  – Page uses information from databases e.g. on-line stores
  – Page is based on user-submitted data e.g search engines
- **Servlet Class Hierarchy**
    - javax.servlet.Servlet
      – Defines methods that all servlets must implement
    - init()
    - service()
    - destroy()
    - javax.servlet.GenericServlet
      – Defines a generic, protocol-independent servlet
    - javax.servlet.http.HttpServlet
      – To write an HTTP servlet for use on the Web
    - doGet()
    - doPost()
    - javax.servlet.ServletConfig
      – A servlet configuration object
      – Passes information to a servlet during initialization
    - Servlet.getServletConfig()
    - javax.servlet.ServletContext
      – To communicate with the servlet container
      – Contained within the ServletConfig object
    - ServletConfig.getServletContext()
    - javax.servlet.ServletRequest
      – Provides client request information to a servlet
    - javax.servlet.ServletResponse
      – Sending a response to the client

- **Basic Servlet Structure**

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Hello World extends HttpServlet {
    // Handle get request
    public void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException {
        // request – access incoming HTTP headers and HTML
        form data
        // response - specify the HTTP response line and headers
        // (e.g. specifying the content type, setting cookies).
        PrintWriter out = response.getWriter(); //out - send
        content to browser
        out.println("Hello World");
    }
}
```

❖ **Servlet Life Cycle**

- Loading and Instantiation
- Initialization
- Request Handling
- End of Service

❖ **Session Tracking**

- Typical scenario – shopping cart in online store
- Necessary because HTTP is a "stateless" protocol
- Session Tracking API allows you to
  – look up session object associated with current request
  – create a new session object when necessary
  – look up information associated with a session
  – store information in a session
  – discard completed or abandoned sessions

❖ **Session Tracking API - I**

- Looking up a session object
  – HttpSession session = request.getSession(true);
  – Pass true to create a new session if one does not exist
- Associating information with session
  –
  session.setAttribute("user",request.getParameter("name"))

– Session attributes can be of any type
- Looking up session information
  – String name = (String) session.getAttribute("user")

❖ **Session Tracking API - II**
- **getId:** – the unique identifier generated for the session
- **isNew:** – true if the client (browser) has never seen the session
- **getCreationTime:** – time in milliseconds since session was made
- **getLastAccessedTime:** – time in milliseconds since the session was last sent from client
- **getMaxInactiveInterval:** –# of seconds session should go without access before being invalidated. – negative value indicates that session should never timeout

Javax.Servlet Interface	**Classes**
Servlet	Genericservlet
ServletRequest	ServletInputStream
ServletResponce	ServletOutputStream
ServletConfig	ServletException
ServletContext	UnavailableException
SingleThreadModel	-

Javax.Servlet.Http	**Classes**
HttpServletRequest	Cookie
HttpServletResponse	HttpServlet
HttpSession	HttpSessionBindingEvent
HttpSessionContext	HttpUtils
HttpSessionBindingListener	-

Exceptions
ServletException
UnavailableException

Servlets

❖ **What is the servlet?**
Servlets are modules that extend request/response-oriented servers, such as Java-enabled web servers. For example, a servlet may be responsible for taking data in an HTML order-entry form and applying the business logic used to update a company's order database.
- Servlets are used to enhance and extend the functionality of Webserver.
- Servlets handles Java and HTML separately.

❖ **What are the uses of Servlets?**
A servlet can handle multiple requests concurrently, and can synchronize requests. This allows servlets to support systems such as on-line conferencing. Servlets can forward requests to other servers and servlets. Thus servlets can be used to balance load among several servers that mirror the same content, and to partition a single logical service over several servers, according to task.

❖ **What are th characters of Servlet?**
As Servlet are written in java, they can make use of extensive power of the JAVA API,such as networking and URL access,multithreading,databaseconnectivity,RMI object serialization.
Efficient: The initilazation code for a servlet is executed only once, when the servlet is executed for the first time.
Robest: provide all the powerfull features of JAVA, such as Exception handling and garbage collection.
Portable: This enables easy portability across Web Servers.
Persistance: Increase the performance of the system by executing features data access.

❖ **What is the difference between JSP and SERVLETS**
Servlets: servlet tieup files to independitently handle the static presentation logic and dynamic business logic , due to this a changes made to any file requires recompilation of the servlet.
- The servlet is Pre-Compile.

JSP: Facilities segregation of work profiles to Web-Developer and Web-Designer, Automatically incorporates changes made to any file (PL & BL) , no need to recompile. Web-Developer write the code for Bussiness logic whereas Web-Designer designs the layout for the WebPage by HTML & JSP.
- The JSP is Post-Compile.

❖ **What are the advantages using servlets than using CGI?**
Servlets provide a way to generate dynamic documents that is both easier to write and faster to run. It is efficient, convenient, powerful, portable, secure and inexpensive. Servlets also address the problem of doing server-side programming with platform-specific APIs. They are developed with Java Servlet API, a standard Java extension.

❖ **What is the difference between servlets and applets?**
Servlets are to servers. Applets are to browsers. Unlike applets, however, servlets have no graphical user interface.

❖ **What is the difference between GenericServlet and HttpServlet?**
GenericServlet is for servlets that might not use HTTP, like for instance FTP service.As of only Http is implemented completely in **HttpServlet**. The GenericServlet has a service() method that gets called when a client request is made. This means that it gets called by both incoming requests and the HTTP requests are given to the servlet as they are.
GenericServlet belongs to javax.servlet package
GenericServlet is an abstract class which extends Object and implements Servlet, ServletConfig and java.io.Serializable interfaces.
The direct subclass to GenericServlet is HttpServlet.It is a protocol-independent servlet

❖ **What are the differences between GET and POST service methods?**
Get Method: Uses Query String to send additional information to the server.
- Query String is displayed on the client Browser.

Query String: The additional sequence of characters that are appended to the URL ia called Query String. The length of the Query string is limited to 255 characters.
- The amount of information you can send back using a GET is restricted as URLs can only be 1024 characters.

Post Method: The Post Method sends the Data as packets through a separate socket connection. The complete transaction is invisible to the client. The post method is slower compared to the Get method because Data is sent to the server as separate packates.
- You can send much more information to the server this way - and it's not restricted to textual data either. It is possible to send files and even binary data such as serialized Java objects!

❖ **What is the servlet life cycle?**
In Servlet life cycles are,
init(), services(), destory().
Init(): Is called by the Servlet container after the servlet has ben Instantiated.
- Contains all information code for servlet and is invoked when the servlet is first loaded.
- The init() does not require any argument , returns a void and throws Servlet Exception.
- If init() executed at the time of servlet class loading.And init() executed only for first user.
- You can Override this method to write initialization code that needs to run only once, such as loading a driver , initializing values and soon, Inother case you can leave normally blank.
Public void init(ServletConfig Config) throws ServletException

Service(): is called by the Servlet container after the init method to allow the servlet to respond to a request.
- Receives the request from the client and identifies the type of request and deligates them to doGet() or doPost() for processing.
Public void service(ServletRequest request,ServletResponce response) throws ServletException, IOException

Destroy(): The Servlet Container calls the destroy() before removing a Servlet Instance from Sevice.

- Excutes only once when the Servlet is removed from Server.
Public void destroy()
If services() are both for get and post methods.
- So if u want to use post method in html page,we use doPost() or services() in servlet class.
- if want to use get methods in html page,we can use doGet() or services() in servlet calss.
- Finally destory() is used to free the object.

- **What is the difference between ServletContext and ServletConfig?**

Both are interfaces.
Servlet Config(): The servlet engine implements the ServletConfig interface in order to pass configuration information to a servlet. The server passes an object that implements the ServletConfig interface to the servlet's init() method.

A ServletConfig object passes configuration information from the server to a servlet. ServletConfig also includes ServletContext object.
getParameter(), getServletContext(), getServletConfig(), GetServletName()

Servlet Context(): The ServletContext interface provides information to servlets regarding the environment in which they are running. It also provides standard way for servlets to write events to a log file.

ServletContext defines methods that allow a servlet to interact with the host server. This includes reading server-specific attributes, finding information about particular files located on the server, and writing to the server log files. I f there are several virtual servers running, each one may return a different ServletContext.
getMIMEType(), getResourse(), getContext(),getServerInfo(),getServletContetName()

- **Can I invoke a JSP error page from a servlet?**

Yes, you can invoke the JSP error page and pass the exception object to it from within a servlet. The trick is to create a request dispatcher for the JSP error page, and pass the exception object as a javax.servlet.jsp.jspException request attribute. However, note that you can do this from only within controller servlets.

- **If your servlet opens an OutputStream or PrintWriter, the JSP engine will throw the following translation error:**
  java.lang.IllegalStateException: Cannot forward as OutputStream or Writer has already been obtained

- **Can I just abort processing a JSP?**
  Yes.Because your JSP is just a servlet method,you can just put (whereeever necessary) a <**% return; %**>

- **What is a better approach for enabling thread-safe servlets and JSPs? SingleThreadModel Interface or Synchronization?**
  Although the SingleThreadModel technique is easy to use, and works well for low volume sites, it does not scale well. If you anticipate your users to increase in the future, you may be better off implementing explicit synchronization for your shared data. The key however, is to effectively minimize the amount of code that is synchronzied so that you take maximum advantage of multithreading.

  Also, note that SingleThreadModel is pretty resource intensive from the server's perspective. The most serious issue however is when the number of concurrent requests exhaust the servlet instance pool. In that case, all the unserviced requests are queued until something becomes free - which results in poor performance. Since the usage is non-deterministic, it may not help much even if you did add more memory and increased the size of the instance pool.

- **If you want a servlet to take the same action for both GET and POST request, what should you do?**
  Simply have doGet call doPost, or vice versa.

- **Which code line must be set before any of the lines that use the PrintWriter?**

setContentType() method must be set before transmitting the actual document.

- ❖ **How HTTP Servlet handles client requests?**
  An HTTP Servlet handles client requests through its service method. The service method supports standard HTTP client requests by dispatching each request to a method designed to handle that request.

- ❖ **What is the Servlet Interface?**
  The central abstraction in the Servlet API is the Servlet interface. All servlets implement this interface, either directly or, more commonly, by extending a class that implements it such as HttpServlet.
  Servlets-->Generic Servlet-->HttpServlet-->MyServlet.
  The Servlet interface declares, but does not implement, methods that manage the servlet and its communications with clients. Servlet writers provide some or all of these methods when developing a servlet.

- ❖ **When a servlet accepts a call from a client, it receives two objects. What are they?**
  ServeltRequest: which encapsulates the communication from the client to the server.
  ServletResponse: which encapsulates the communication from the servlet back to the Client.
  ServletRequest and ServletResponse are interfaces defined by the javax.servlet package.

- ❖ **What information that the ServletRequest interface allows the servlet access to?**
  Information such as the names of the parameters passed in by the client, the protocol (scheme) being used by the client, and the names of the remote host that made the request and the server that received it. The input stream, ServletInputStream.Servlets use the input stream to get data from clients that use application protocols such as the HTTP POST and PUT methods.

- ❖ **What information that the ServletResponse interface gives the servlet methods for replying to the client?**

It Allows the servlet to set the content length and MIME type of the reply. Provides an output stream, ServletOutputStream and a Writer through which the servlet can send the reply data.

- ❖ **Difference between single thread and multi thread model servlet**

A servlet that implements SingleThreadModel means that for every request, a single servlet instance is created. This is not a very scalable solution as most web servers handle multitudes of requests. A multi-threaded servlet means that one servlet is capable of handling many requests which is the way most servlets should be implemented.

a. A single thread model for servlets is generally used to protect sensitive data (bank account operations).
b. Single thread model means instance of the servlet gets created for each request recieved. Its not thread safe whereas in multi threaded only single instance of the servlet exists for what ever # of requests recieved. Its thread safe and is taken care by the servlet container.
c. A servlet that implements SingleThreadModel means that for every request, a single servlet instance is created. This is not a very scalable solution as most web servers handle multitudes of requests. A multi-threaded servlet means that one servlet is capable of handling many requests which is the way most servlets should be implemented.

A single thread model for servlets is generally used to protect sensitive data (bank account operations).

- ❖ **What is servlet context and what it takes actually as parameters?**

Servlet context is an object which is created as soon as the Servlet gets initialized.Servlet context object is contained in Servlet Config. With the context object u can get access to specific resource (like file) in the server and pass it as a URL to be displayed as a next screen with the help of RequestDispatcher

Example:

```
ServletContext app = getServletContext();
RequestDispatcher disp;
```

```
if(b==true)
    disp = app.getRequestDispatcher
    ("jsp/login/updatepassword.jsp");
else
    disp = app.getRequestDispatcher
    ("jsp/login/error.jsp");
```

this code will take user to the screen depending upon the value of b. in ServletContext u can also get or set some variables which u would like to retreive in next screen.

Example:

```
context.setAttribute("supportAddress", "temp@temp.com");
```

Better yet, you could use the web.xml context-param element to designate the address, then read it with the getInitParameter method of ServletContext.

- **Can we call destroy() method on servlets from service method?**

  destroy() is a servlet life-cycle method called by servlet container to kill the instance of the servlet. "Yes". You can call destroy() from within the service(). It will do whatever logic you have in destroy() (cleanup, remove attributes, etc.) but it won't "unload" the servlet instance itself. That can only be done by the container

- **What is the use of ServletConfig and ServletContext..?**

  An interface that describes the configuration parameters for a servlet. This is passed to the servlet when the web server calls its init() method. Note that the servlet should save the reference to the ServletConfig object, and define a getServletConfig() method to return it when asked. This interface defines how to get the initialization parameters for the servlet and the context under which the servlet is running.

  An interface that describes how a servlet can get information about the server in which it is running. It can be retrieved via the getServletContext() method of the ServletConfig object.

- **What is difference between forward() and sendRedirect()..? Which one is faster then other and which works on server?**

  Forward(): javax.Servlet.RequestDispatcher interface.

- RequestDispatcher.forward() works on the Server.
- The forward() works inside the WebContainer.
- The forward() restricts you to redirect only to a resource in the same web-Application.
- After executing the forward(), the control will return back to the same method from where the forward method was called.
- the forward() will redirect in the application server itself, it does'n come back to the client.
- The forward() is faster than Sendredirect() .

To use the forward() of the requestDispatcher interface, the first thing to do is to obtain RequestDispatcher Object. The Servlet technology provides in three ways.

2. By using the getRequestDispatcher() of the javax.Servlet.ServletContext interface , passing a String containing the path of the other resources, path is relative to the root of the ServletContext.
   ```
   RequestDispatcher
   rd=request.getRequestDispatcher("secondServlet");
   Rd.forward(request, response);
   ```
3. getRequestDispatcher() of the javax.Servlet.Request interface , the path is relative to current HtpRequest.
   ```
   RequestDispatcher
   rd=getServletContext().getRequestDispatcher("servlet/seco
   ndServlet");
   Rd.forward(request, response);
   ```
4. By using the getNameDispatcher() of the javax.Servlet.ServletContext interface.
   ```
   RequestDispatcher
   rd=getServletContext().getNameDispatcher("secondServlet"
   );
   Rd.forward(request, response);
   ```

Sendredirect(): javax.Servlet.Http.HttpServletResponce interface
- RequestDispatcher.SendRedirect() works on the browser.
- The SendRedirect() allows you to redirect trip to the Client.
- The SendRedirect() allows you to redirect to any URL.
- After executing the SendRedirect() the control will not return back to same method.
- The Client receives the Http response code 302 indicating that temporarly the client is being redirected to the specified

location , if the specified location is relative , this method converts it into an absolute URL before redirecting.
- The SendRedirect() will come to the Client and go back,.. ie URL appending will happen.
Response. SendRedirect(“absolute path”);
Absolutepath – other than application, relative path - same application.

When you invoke a forward request, the request is sent to another resource on the server, without the client being informed that a different resource is going to process the request. This process occurs completely with in the web container. When a sendRedirtect method is invoked, it causes the web container to return to the browser indicating that a new URL should be requested. Because the browser issues a completely new request any object that are stored as request attributes before the redirect occurs will be lost. This extra round trip a redirect is slower than forward.

- **do we have a constructor in servlet ? can we explictly provide a constructor in servlet programme as in java program ?**

  We can have a constructor in servlet .
  Session: A session is a group of activities that are performed by a user while accesing a particular website.
  Session Tracking: The process of keeping track of settings across session is called session tracking.
  Hidden Form Fields: Used to keep track of users by placing hidden fields in the form.
  - The values that have been entered in these fields are sent to the server when the user submits the Form.
  URL-rewriting: this is a technique by which the URL is modified to include the session ID of a particular user and is sent back to the Client.
  - The session Id is used by the client for subsequent transactions with the server.
  Cookies: Cookies are small text files that are used by a webserver to keep track the Users.
  A cookie is created by the server and send back to the client , the value is in the form of Key-value pairs. Aclient can accept

20 cookies per host and the size of each cookie can be maximum of 4 bytes each.
HttpSession: Every user who logs on to the website is autometacally associated with an HttpSession Object.
- The Servlet can use this Object to store information about the users Session.
- HttpSession Object enables the user to maintain two types of Data.
- ie State and Application.

❖ **How to communicate between two servlets?**
Two ways:
a. Forward or redirect from one Servlet to another.
b. Load the Servlet from ServletContext and access methods.

❖ **How to get one Servlet's Context Information in another Servlet?**
Access or load the Servlet from the Servlet Context and access the Context Information

❖ **The following code snippet demonstrates the invocation of a JSP error page from within a controller servlet:**

```
protected void sendErrorRedirect(HttpServletRequest request,
HttpServletResponse response, String errorPageURL,
Throwable e) throws ServletException, IOException {
    request.setAttribute ("javax.servlet.jsp.jspException", e);
    getServletConfig().getServletContext().
    getRequestDispatcher(errorPageURL).forward(request,
    response);
}
public void doPost(HttpServletRequest request,
HttpServletResponse response) {
    try {
        // do something
    } catch (Exception ex) {
        try {
            sendErrorRedirect(request,response,"/jsp/MyErrorPage.
            jsp", ex);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

JSP (JavaServer Pages)

❖ Why JSP Technology?

- Servlets are good at running logic
  - – Not so good at producing large amounts of output
  - – **out.write()** is ugly
- JSP pages are great at producing lots of textual output
  - – Not so good at lots of logic
  - – <**% %**> is ugly

❖ How does it Work?

- *"JSP page"*
  - – Mixture of text, Script and directives
  - – Text could be text/ html, text/ xml or text/ plain
- *"JSP engine"*
  - – 'Compiles' page to servlet
  - – Executes servlet's **service()** method
- Sends text back to caller
- Page is
  - – Compiled once
  - – Executed many times

❖ Anatomy of a JSP

```
<%@ page language="java" contentType="text/html" %>
<html>
   <body bgcolor="white">
      <jsp:useBean                                    id="greeting"
      class="com.pramati.jsp.beans.GreetingBean">
      <jsp:setProperty name="greeting" property="*"/>
      </jsp:userBean>
         The following information was saved:
         User Name:
      <jsp:getProperty                              name="greeting"
      property="userName"/>
      Welcome!
   </body>
</html>
```

❖ JSP Elements

- Directive Elements
  - – Information about the page

– Remains same between requests
– E.g., scripting language used

- **Action Elements:**
  – Take action based on info required at request-time
- Standard
- Custom (Tags and Tag Libraries)
- Scripting Elements
  – Add pieces of code to generate output based on conditions

❖ Directives

- Global information used by the "JSP engine"
- Of form **<%@ directive attr_ list %>**
- Or **<jsp: directive. directive attr_ list />**
  – Directive could be
  - Page
  - Include
  - Taglib
    – E. g.,

```
<%@ page info=" written by DevelopMentor" %>
<jsp: directive. page import=" java. sql.*" />
<%@ include file ="\ somefile. txt" %>
<%@ taglib    uri = tags prefix=" foo" %>
```

❖ Actions Within a JSP Page

- Specifies an action to be carried out by the "JSP engine"
- Standard or custom
  – Standard must be implemented by all engines
  – Custom defined in tag libraries
- Standard actions 'scoped' by 'jsp' namespace
- Have name and attributes

```
<jsp: useBean id=" clock" class=" java.util.Date" />
<ul> The current date at the server is:
    <li>    Date:    <jsp:    getProperty    name="clock"
    property="date" />
    <li>    Month:    <jsp:    getProperty    name="clock"
    property="month" />
</ul>
```

❖ Standard JSP Actions:

- jsp:useBean
- jsp:getProperty

- jsp:setProperty
- jsp:include
- jsp:forward
- jsp:param
- jsp:plugin

Scriptlets

- Of form <% /* code goes here*/ %>
  – Gets copied into _ jspService method of generated servlet
- Any valid Java code can go here

  Code: **Output**

```
<% int j; %>                            <value> 0</ value>
<% for (j = 0; j < 3; j++) {%>          <value> 1</ value>
<value>                                 <value> 2</ value>
<% out. write(""+ j); %>
</ value><% } %>
```

Declarations (<%! … %>)

- Used to declare class scope variables or methods
  <%! int j = 0; %>
- Gets declared at class- level scope in the generated servlet
- public class SomeJSP extends HttpServlet implements HttpJspPage {

```
    …
    int j = 0;
    void _jspService(…) {}
}
```

❖ **JSP to Servlet Translation**

```
<%@ page import="javax.ejb.*,javax.naming.*,java.rmi.*
,java.util.*" %>
<HTML>
    <HEAD><TITLE>Hello.jsp</TITLE></HEAD>
    <BODY>
        <%      String checking = null;
            String name = null;
            checking = request.getParameter("catch");
            if (checking != null) {
                name = request.getParameter("name");%>
                <b> Hello <%=name%>
            <% }
        %>
        <FORM METHOD='POST' action="Hello.jsp">
            <table   width="500"   cellspacing="0"   cellpadding="3"
            border="0">
                <caption>Enter your name</caption>
                <tr>
                    <td><b>Name</b></td>
                    <td><INPUT      size="20"       maxlength="20"
                    TYPE="text" NAME="name"></td>
                </tr>
            </table>
            <INPUT          TYPE='SUBMIT'          NAME='Submit'
            VALUE='Submit'>
            <INPUT TYPE='hidden' NAME='catch' VALUE='yes'>
        </FORM>
    </BODY>
</HTML>
```

❖ **Generated Servlet…**

```
public     void     _jspService(HttpServletRequest     request,
HttpServletResponse    response)    throws    ServletException
,IOException {
    out.write("<HTML><HEAD><TITLE>Hello.jsp</TITLE></H
EAD><BODY>" );
    String checking = null;
    String name = null;
    checking = request.getParameter("catch");
    if (checking != null) {
        name = request.getParameter("name");
        out.write("\r\n\t\t<b> Hello " );
        out.print(name);
        out.write("\r\n\t\t" );
    }
    out.write("\r\n\t\t<FORM METHOD='POST' action="
        +"\"Hello.jsp\">\r\n\t\t\t<table            width=\"500\"
        cell"……………………………..
    }
}
```

❖ **Tags & Tag Libraries**

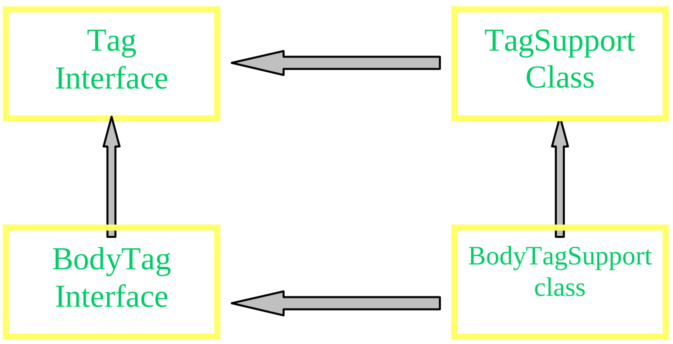
What Is a Tag Library?

- JSP technology has a set of pre- defined tags

    – <jsp: useBean .../>
- These are HTML like but…
- … have limited functionality
- Can define new tags
    – Look like HTML
    – Can be used by page authors
    – "Java code" is executed when tag is encountered
    – Allow us to keep Java code off the page
- Better separation of content and logic

May Have Tags To…
- Process an SQL command
- Parse XML and output HTML
- Automatically call into an "EJB component" (EJB ™ technology- based component)
- Get called on every request to initialize script variables
- Iterate over a ResultSet and display the output in an HTML table

Primary Tag Classes (javax.servlet.jsp.tagext.Tag)



Example:

```
<%@ taglib uri="/WEB-INF/mylib.tld" prefix="test" %>
<html>
    <body bgcolor="white">
        <test:hello name="Robert" />
    </body>
</html>
```

```
public class HelloTag extends TagSupport {
    private String name = "World";
    public void setName(String name) { this.name =
    name; }
    public int doEndTag() {
    pageContext.getOut().println("Hello " + name); }
}
```

mylib.tld

```
<taglib>......
    <tag>
        <name>hello</name>
        <tagclass>com.pramati.HelloTag</tagclass>
        <bodycontent>empty</bodycontent>
        <attribute><name>name</name></attribute>
    </tag>
</taglib>
```

How Tag Handler methods are invoked:

```
<prefix:tagName
    attr1="value1" ------------ setAttr1("value1")
    attr2="value2" ------------ setAttr2("value2")
>------------ doStartTag()
This tags's body
</ prefix:tagName> ------------ doEndTag()
```

- Implementation of JSP page will use the tag handler for each 'action' on page.

❖ Summary

- The JSP specification is a powerful system for creating structured web content.
- JSP technology allows non- programmers to develop dynamic web pages.
- JSP technology allows collaboration between programmers and page designers when building web applications.
- JSP technology uses the Java programming language as the script language.
- The generated servlet can be managed by directives.
- JSP components can be used as the view in the MVC architecture.
- Authors using JSP technology are not necessarily programmers using Java technology.
- Want to keep "Java code" off a "JSP Page".

- Custom actions (tag libraries) allow the use of elements as a replacement for Java code.

❖ **What is JSP- JavaServer Pages?**
JavaServer Pages. A server-side technology, JavaServer pages are an extension to the Java servlet technology that was developed by Sun. JSPs have dynamic scripting capability that works in tandem with HTML code, separating the page logic from the static elements -- the actual design and display of the page. Embedded in the HTML page, the Java source code and its extensions help make the HTML more functional, being used in dynamic database queries, for example. JSPs are not restricted to any specific platform or server.

Jsp contains both static and dynamic resources at run time.Jsp extends web server functionalities.

❖ **What are advantages of JSP?**
Whenever there is a change in the code, we dont have to recompile the jsp. it automatically does the compilation. by using custom tags and tag libraries the length of the java code is reduced.

❖ **What is the difference between include directive & jsp:include action?**
include directive(): if the file includes static text if the file is rarely changed (the JSP engine may not recompile the JSP if this type of included file is modified). If you have a common code snippet that you can reuse across multiple pages (e.g. headers and footers).

jsp:include: for content that changes at runtime .to select which content to render at runtime (because the page and src attributes can take runtime expressions) for files that change often JSP:includenull.

❖ **What are Custom tags. Why do you need Custom tags. How do you create Custom tag?**
5. Custom tags are those which are user defined.

6. Inorder to separate the presentation logic in a separate class rather than keeping in jsp page we can use custom tags.
7. **Step 1:** Build a class that implements the javax.servlet.jsp.tagext.Tag interface as follows. Compile it and place it under the web-inf/classes directory (in the appropriate package structure).

```
package examples;
import java.io.*; //// THIS PROGRAM IS EVERY TIME I MEAN WHEN  U REFRESH THAT PARTICULAR  CURRENT DATE THIS CUSTOM TAG WILL DISPLAY
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
public class ShowDateTag implements Tag {
    private PageContext pageContext;
    private Tag parent;
    public int doStartTag() throws JspException {
        return SKIP_BODY;
    }
    public int doEndTag() throws JspException {
        try {
            pageContext.getOut().write("" + new
            java.util.Date());
        } catch (IOException ioe) {
            throw new JspException(ioe.getMessage());
        }
        return EVAL_PAGE;
    }
    public void release() {
    }
    public void setPageContext(PageContext page) {
        this.pageContext = page;
    }
    public void setParent(Tag tag) {
        this.parent = tag;
    }
    public Tag getParent() {
        return this.parent;
    }
}
```

Step 2: Now we need to describe the tag, so create a file called taglib.tld and place it under the web-inf directory."http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd"> 1.0 1.1 myTag http://www.mycompany.com/taglib My own tag library showDate examples.ShowDateTag Show the current date

Step 3: Now we need to tell the web application where to find the custom tags, and how they will be referenced from JSP pages. Edit the web.xml file under the web-inf directory and insert the following XML fragement.http://www.mycompany.com/taglib /WEB-INF/taglib.tld

Step 4: And finally, create a JSP page that uses the custom tag.Now restart the server and call up the JSP page! You should notice that every time the page is requested, the current date is displayed in the browser. Whilst this doesn't explain what all the various parts of the tag are for (e.g. the tag description, page context, etc) it should get you going. If you use the tutorial (above) and this example, you should be able to grasp what's going on! There are some methods in context object with the help of which u can get the server (or servlet container) information.

Apart from all this with the help of ServletContext u can implement ServletContextListener and then use the get-InitParametermethod to read context initialization parameters as the basis of data that will be made available to all servlets and JSP pages.

- **What are the implicit objects in JSP & differences between them**

There are nine implicit objects in JSP.

8. **request:** The request object represents httprequest that are trigged by service( ) invocation.
   javax.servlet
9. **response:** The response object represents the servers response to request.
   javax.servlet

10. **pageContext:** The page context specifies the single entry point to many of the page attributes and is the convient place to put shared data.
javax.servlet.jsp.pagecontext

11. **session:** the session object represents the session created by the current user.
javax.Servlet.http.HttpSession
12. **application:** the application object represents servlet context , obtained from servlet configaration.
javax.Servlet.ServletContext
13. **out:** the out object represents to write the out put stream.
javax.Servlet.jsp.jspWriter
14. **Config:** the config object represents the servlet config interface from this page,and has scope attribute.
javax.Servlet.ServletConfig
15. **page:** The object is th eInstance of page implementation servlet class that are processing the current request.
java.lang.Object
16. **exception:** These are used for different purposes and actually u no need to create these objects in JSP. JSP container will create these objects automatically.
java.lang.Throwable

You can directly use these objects.

Example:
If i want to put my username in the session in JSP.
JSP Page: In the about page, i am using session object. But this session object is not declared in JSP file, because, this is implicit object and it will be created by the jsp container.
If u see the java file for this jsp page in the work folder of apache tomcat, u will find these objects are created.

❖ **What is jsp:usebean. What are the scope attributes & difference between these attributes?**
page, request, session, application

❖ **What is difference between scriptlet and expression?**

With **expressions** in JSP, the results of evaluating the expression are converted to a string and directly included within the output page. Typically expressions are used to display simple values of variables or return values by invoking a bean's getter methods. JSP expressions begin within tags and do not include semicolons:
But **scriptlet** can contain any number of language statements, variable or method declarations, or expressions that are valid in the page scripting language. Within scriptlet tags, you can declare variables or methods to use later in the file, write expressions valid in the page scripting language, use any of the JSP mplicit objects or any object declared with a.

- **What is Declaration**
  Declaration is used in JSP to declare methods and variables.To add a declaration, you must use the sequences to enclose your declarations.

- **How do you connect to the database from JSP**
  To be precise to connect jdbc from jsp is not good idea ofcourse if ur working on dummy projects connecting to msaccess u can very well use the same connection objects amd methods in ur scriplets and define ur connection object in init() method.

  But if its real time u can use DAO design patterns which is widely used. for ex u write all ur connection object and and sql quires in a defiened method later use transfer object [TO]which is all ur fields have get/set methods and call it in business object[BO] so DAO is accessd with precaution as it is the crucial. Finally u define java bean which is a class holding get/set method implementing serialization thus the bean is called in the jsp. So never connect to jdbc directly from client side since it can be hacked by any one to get ur password or credit card info.

- **How do you call stored procedures from JSP**
  By using callable statement we can call stored procedures and functions from the database.

- **How do you restrict page errors display in the JSP page** set isErrorPage=false

- **How do you pass control from one JSP page to another**
  we can forward control to aother jsp using jsp action tags forward or incllude

- **How do I have the JSP-generated servlet subclass my own custom servlet class, instead of the default?**
  One should be very careful when having JSP pages extend custom servlet classes as opposed to the default one generated by the JSP engine. In doing so, you may lose out on any advanced optimization that may be provided by the JSPengine. In any case, your new superclass has to fulfill the contract with the JSPngine by: Implementing the HttpJspPage interface, if the protocol used is HTTP, or implementing JspPage otherwise Ensuring that all the methods in the Servlet interface are declared final Additionally, your servlet superclass also needs to do the following:
  The service() method has to invoke the _jspService() method
  The init() method has to invoke the jspInit() method
  The destroy() method has to invoke jspDestroy()
  If any of the above conditions are not satisfied, the JSP engine may throw a translation error. Once the superclass has been developed, you can have your JSP extend it as follows:
  <%@ page extends="packageName.ServletName" %>

- **How does a servlet communicate with a JSP page?**
  The following code snippet shows how a servlet instantiates a bean and initializes it with FORM data posted by a browser. The bean is then placed into the request, and the call is then forwarded to the JSP page, Bean1.jsp, by means of a request dispatcher for downstream processing.

```
public void doPost (HttpServletRequest request,
HttpServletResponse response) {
    try {
        govi.FormBean f = new govi.FormBean();
        String id = request.getParameter("id");
        f.setName(request.getParameter("name"));
        f.setAddr(request.getParameter("addr"));
        f.setAge(request.getParameter("age"));
        //use the id to compute
        //additional bean properties like info
        //maybe perform a db query, etc.
        // . . .
        f.setPersonalizationInfo(info);
        request.setAttribute("fBean",f);
        getServletConfig().getServletContext().getRequest
        Dispatcher      ("/jsp/Bean1.jsp").forward(request,
        response);
    } catch (Exception ex) {
        . . .
    }
}
```

The JSP page Bean1.jsp can then process fBean, after first extracting it from the default request scope via the useBean action.

```
jsp:useBean id="fBean" class="govi.FormBean"
scope="request"/
jsp:getProperty name="fBean" property="name" /
jsp:getProperty name="fBean" property="addr" /
jsp:getProperty name="fBean" property="age" /
jsp:getProperty name="fBean" property="personalizationInfo"
/
```

❖ **Is there a way I can set the inactivity lease period on a per-session basis?**

Typically, a default inactivity lease period for all sessions is set within your JSPengine admin screen or associated properties file. However, if your JSP engine supports the Servlet 2.1 API, you can manage the inactivity lease period on a per-session basis. This is done by invoking the HttpSession.setMaxInactiveInterval() method, right after the session has been created.

Example:

```
<% session.setMaxInactiveInterval(300); %>
```

would reset the inactivity period for this session to 5 minutes. The inactivity interval is set in seconds.

❖ **How can I set a cookie and delete a cookie from within a JSP page?**

A cookie, mycookie, can be deleted using the following scriptlet:

```
<%
    //creating a cookie
    Cookie mycookie = new Cookie("aName","aValue");
    response.addCookie(mycookie);
    //delete a cookie
    Cookie killMyCookie = new Cookie("mycookie", null);
    killMyCookie.setMaxAge(0);
    killMyCookie.setPath("/");
    response.addCookie(killMyCookie);
%>
```

❖ **How can I declare methods within my JSP page?**

You can declare methods for use within your JSP page as declarations. The methods can then be invoked within any other methods you declare, or within JSP scriptlets and expressions.

Do note that you do not have direct access to any of the JSP implicit objects like request, response, session and so forth from within JSP methods. However, you should be able to pass any of the implicit JSP variables as parameters to the methods you declare.

Example:

```
<%!
    public String whereFrom(HttpServletRequest req) {
        HttpSession ses = req.getSession();
        ...
        return req.getRemoteHost();
    }
%>
<%
    out.print("Hi there, I see that you are coming in from ");
%>
<%= whereFrom(request) %>
```

Another Example:

file1.jsp

```
<%@page contentType="text/html"%>
<%!
    public void test(JspWriter writer) throws IOException {
        writer.println("Hello!");
    }
```

```
%>
```
file2.jsp
```
<%@include file="file1.jsp"%>
<html>
   <body>
      <%test(out);% >
   </body>
</html>
```

❖ **How can I enable session tracking for JSP pages if the browser has disabled cookies?**

We know that session tracking uses cookies by default to associate a session identifier with a unique user. If the browser does not support cookies, or if cookies are disabled, you can still enable session tracking using URL rewriting. URL rewriting essentially includes the session ID within the link itself as a name/value pair. However, for this to be effective, you need to append the session ID for each and every link that is part of your servlet response. Adding the session ID to a link is greatly simplified by means of of a couple of methods: response.encodeURL() associates a session ID with a given URL, and if you are using redirection, response.encodeRedirectURL() can be used by giving the redirected URL as input. Both encodeURL() and encodeRedirectedURL() first determine whether cookies are supported by the browser; if so, the input URL is returned unchanged since the session ID will be persisted as a cookie.

Consider the following example, in which two JSP files, say hello1.jsp and hello2.jsp, interact with each other. Basically, we create a new session within hello1.jsp and place an object within this session. The user can then traverse to hello2.jsp by clicking on the link present within the page.Within hello2.jsp, we simply extract the object that was earlier placed in the session and display its contents. Notice that we invoke the encodeURL() within hello1.jsp on the link used to invoke hello2.jsp; if cookies are disabled, the session ID is automatically appended to the URL, allowing hello2.jsp to still retrieve the session object. Try this example first with cookies enabled. Then disable cookie support, restart the brower, and try again. Each time you should see the maintenance of the session across pages. Do note that to get this example to work with cookies disabled at the browser, your JSP engine has to support **URL rewriting**.

hello1.jsp

```
<%@ page session="true" %>
<%
    Integer num = new Integer(100);
    session.putValue("num",num);
    String url =response.encodeURL("hello2.jsp");
%>
<a href='<%=url%>'>hello2.jsp</a>
```

hello2.jsp

```
<%@ page session="true" %>
<%
    Integer i= (Integer )session.getValue("num");
    out.println("Num value in session is "+i.intValue());
%>
```

❖ **How do I use a scriptlet to initialize a newly instantiated bean?**

A **jsp:useBean** action may optionally have a body. If the body is specified, its contents will be automatically invoked when the specified bean is instantiated. Typically, the body will contain scriptlets or jsp:setProperty tags to initialize the newly instantiated bean, although you are not restricted to using those alone. The following example shows the "today" property of the Foo bean initialized to the current date when it is instantiated. Note that here, we make use of a JSP expression within the jsp:setProperty action.

```
<jsp:useBean id="foo" class="com.Bar.Foo" >
<jsp:setProperty name="foo" property="today"
value="<%=java.text.DateFormat.getDateInstance().format(n
ew java.util.Date())
%>"/ >
<%-- scriptlets calling bean setter methods go here --%>
</jsp:useBean >
```

❖ **How does JSP handle run-time exceptions?**

You can use the errorPage attribute of the page directive to have uncaught runtime exceptions automatically forwarded to an error processing page.
For example:

```
<%@ page errorPage="error.jsp" %>
```

redirects the browser to the JSP page error.jsp if an uncaught exception is encountered during request processing. Within error.jsp, if you indicate that it is an error-processing page, via the directive:

```
<%@ page isErrorPage="true" %>
```

the Throwable object describing the exception may be accessed within the error page via the exception implicit object.
Note: You must always use a relative URL as the value for the errorPage attribute.

❖ **How do I prevent the output of my JSP or Servlet pages from being cached by the browser?**
You will need to set the appropriate HTTP header attributes to prevent the dynamic content output by the JSP page from being cached by the browser. Just execute the following scriptlet at the beginning of your JSP pages to prevent them from being cached at the browser. You need both the statements to take care of some of the older browser versions.

```
<%
   response.setHeader("Cache-Control","no-store");     //HTTP
   1.1
   response.setHeader("Pragma","no-cache"); //HTTP 1.0
   response.setDateHeader ("Expires", 0); //prevents caching
   at the proxy server
%>
```

❖ **How do I use comments within a JSP page?**
You can use "JSP-style" comments to selectively block out code while debugging or simply to comment your scriptlets. JSP comments are not visible at the client.
For example:

```
<%-- the scriptlet is now commented out
   <%
```

```
        out.println("Hello World");
    %>
--%>
```

You can also use HTML-style comments anywhere within your JSP page. These comments are visible at the client. For example:

```
<!-- (c) 2004 javagalaxy.com -->
```

Of course, you can also use comments supported by your JSP scripting language within your scriptlets. For example, assuming Java is the scripting language, you can have:

```
<%
    //some comment
    /**yet another comment **/
%>
```

❖ **Can I stop JSP execution while in the midst of processing a request?**

Yes. Preemptive termination of request processing on an error condition is a good way to maximize the throughput of a high-volume JSP engine. The trick (asuming Java is your scripting language) is to use the return statement when you want to terminate further processing. For example, consider:

```
<%
    if (request.getParameter("foo") != null) {
        //generate some html or update bean property
    } else {
        /*output some error message or provide redirection back
        to the input form after creating a memento bean updated
        with the 'valid' form elements that were input. This bean
        can now be used by the previous form to initialize the
        input elements that were valid then, return from the body
        of the _jspService() method to terminate further
        processing */
        return;
    }
%>
```

❖ **Is there a way to reference the "this" variable within a JSP page?**

Yes, there is. Under JSP 1.0, the page implicit object is equivalent to "this", and returns a reference to the servlet generated by the JSP page.

❖ **How do I perform browser redirection from a JSP page?**
You can use the response implicit object to redirect the browser to a different resource, as:
response.sendRedirect("http://www.exforsys.com/path/error.html");
You can also physically alter the Location HTTP header attribute, as shown below:
```
<%
   response.setStatus(HttpServletResponse.SC_MOVED_PERMANENTLY);
   String newLocn = "/newpath/index.html";
   response.setHeader("Location",newLocn);
%>
```

You can also use the: <**jsp:forward page="/newpage.jsp"** /> Also note that you can only use this before any output has been sent to the client. I beleve this is the case with the **response.sendRedirect()** method as well. If you want to pass any paramateres then you can pass using
```
<jsp:forward page="/servlet/login">
   <jsp:param name="username" value="Kishore" />
</jsp:forward>
```

❖ **How do I include static files within a JSP page?**
Answer Static resources should always be included using the JSP include directive. This way, the inclusion is performed just once during the translation phase. The following example shows the syntax:
<**%@ include file="copyright.html" %**>

Do note that you should always supply a relative URL for the file attribute. Although you can also include static resources using the action, this is not advisable as the inclusion is then performed for each and every request.

❖ **What JSP lifecycle methods can I override?**

You cannot override the _jspService() method within a JSP page. You can however, override the jspInit() and jspDestroy() methods within a JSP page. jspInit() can be useful for allocating resources like database connections, network connections, and so forth for the JSP page. It is good programming practice to free any allocated resources within jspDestroy().

The jspInit() and jspDestroy() methods are each executed just once during the lifecycle of a JSP page and are typically declared as JSP declarations:

```
<%!
    public void jspInit() {
        . . .
    }
%>
<%!
    public void jspDestroy() {
        . . .
    }
%>
```

❖ **Can a JSP page process HTML FORM data?**

Yes. However, unlike servlets, you are not required to implement HTTP-protocol specific methods like **doGet()** or **doPost()** within your JSP page. You can obtain the data for the FORM input elements via the request implicit object within a scriptlet or expression as:

```
<%
    String item = request.getParameter("item");
    int howMany = new
    Integer(request.getParameter("units")).intValue();
%>
```

or

```
<%= request.getParameter("item") %>
```

❖ **How do I mix JSP and SSI #include?**

If you're just including raw HTML, use the #include directive as usual inside your .jsp file.

```
<!--#include file="data.inc"-->
```

But it's a little trickier if you want the server to evaluate any JSP code that's inside the included file. If your data.inc file contains jsp code you will have to use <%@ vinclude="data.inc" %> The <!--#include file="data.inc"--> is used for including non-JSP files.

- **How can I implement a thread-safe JSP page?**
  You can make your JSPs thread-safe by having them implement the **SingleThreadModel** interface. This is done by adding the directive
  <%@ **page isThreadSafe="false"** % > within your JSP page.

- **How do I include static files within a JSP page?**
  Static resources should always be included using the JSP include directive. This way, the inclusion is performed just once during the translation phase. The following example shows the syntax: Do note that you should always supply a relative URL for the file attribute. Although you can also include static resources using the action, this is not advisable as the inclusion is then performed for each and every request.

- **How do you prevent the Creation of a Session in a JSP Page and why?**
  By default, a JSP page will automatically create a session for the request if one does not exist. However, sessions consume resources and if it is not necessary to maintain a session, one should not be created. For example, a marketing campaign may suggest the reader visit a web page for more information. If it is anticipated that a lot of traffic will hit that page, you may want to optimize the load on the machine by not creating useless sessions.

- **What is the page directive is used to prevent a JSP page from automatically creating a session:**
  <%@ page session="false">

- **Is it possible to share an HttpSession between a JSP and EJB? What happens when I change a value in the HttpSession from inside an EJB?**

You can pass the HttpSession as parameter to an EJB method, only if all objects in session are serializable.This has to be consider as "passed-by-value", that means that it's read-only in the EJB. If anything is altered from inside the EJB, it won't be reflected back to the HttpSession of the Servlet Container.The "pass-byreference" can be used between EJBs Remote Interfaces, as they are remote references. While it IS possible to pass an HttpSession as a parameter to an EJB object, it is considered to be "bad practice (1)" in terms of object oriented design. This is because you are creating an unnecessary coupling between back-end objects (ejbs) and front-end objects (HttpSession). Create a higher-level of abstraction for your ejb's api. Rather than passing the whole, fat, HttpSession (which carries with it a bunch of http semantics), create a class that acts as a value object (or structure) that holds all the data you need to pass back and forth between front-end/back-end. Consider the case where your ejb needs to support a non-http-based client. This higher level of abstraction will be flexible enough to support it. (1) Core J2EE design patterns (2001).

- **Can a JSP page instantiate a serialized bean?**
  No problem! The useBean action specifies the beanName attribute, which can be used for indicating a serialized bean. For example:

```
<jsp:useBean id="shop" type="shopping.CD" beanName="CD"
/>
<jsp:getProperty name="shop" property="album" />
```

  A couple of important points to note. Although you would have to name your serialized file "filename.ser", you only indicate "filename" as the value for the beanName attribute. Also, you will have to place your serialized file within the WEB-INFjspbeans directory for it to be located by the JSP engine.

- **Can you make use of a ServletOutputStream object from within a JSP page?**
  No. You are supposed to make use of only a JSPWriter object (given to you in the form of the implicit object out) for replying to clients. A JSPWriter can be viewed as a buffered version of the stream object returned by response.getWriter(), although

from an implementational perspective, it is not. A page author can always disable the default buffering for any page using a page directive as:
<%@ page buffer="none" %>

- **Can we implements interface or extends class in JSP?**
  No, we can't implements interface or extends class in JSP.

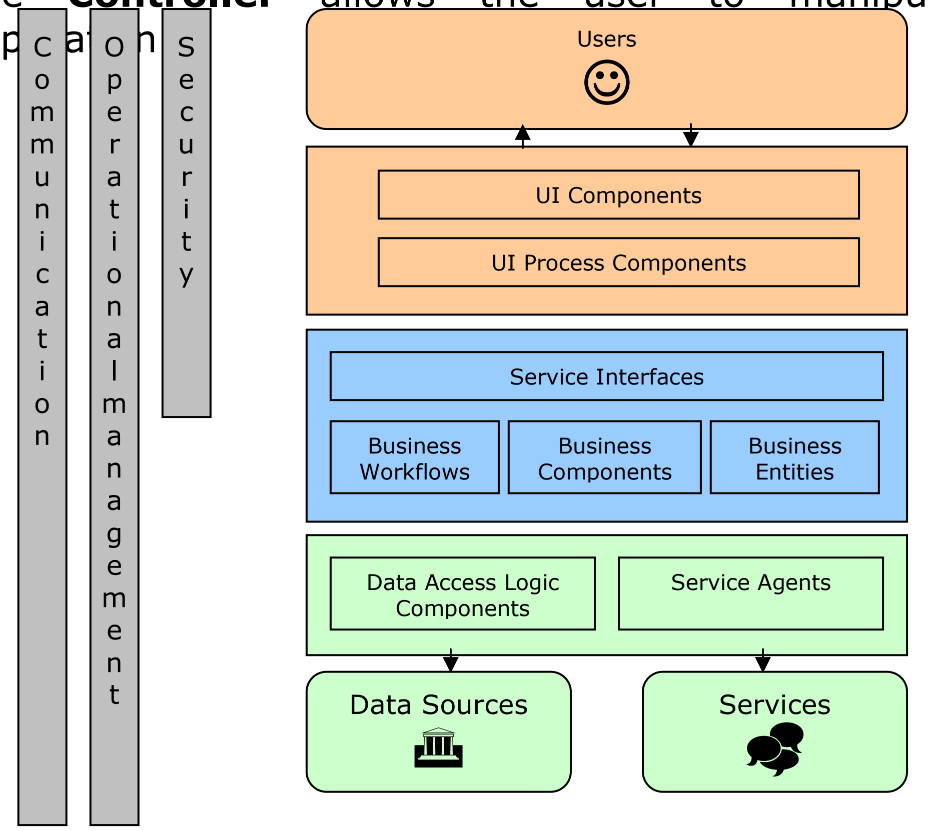
- **What are the steps required in adding a JSP Tag Libraries?**
  17. Create a TLD file and configure the required class Information.
  18. Create the Java Implementation Source extending the JSP Tag Lib Class (TagSupport).
  19. Compile and package it as loosed class file or as a jar under lib folder in Web Archive File for Class loading.
  20. Place the TLD file under the WEB-INF folder.
  21. Add reference to the tag library in the web.xml file.

Introduction to MVC (Model View Controler)

❖ Overview of MVC Architecture

The MVC design pattern divides applications into three components:

- The **Model** maintains the state and data that the application represents.
- The **View** allows the display of information about the model to the user.
- The **Controller** allows the user to manipulate the app[illegible]



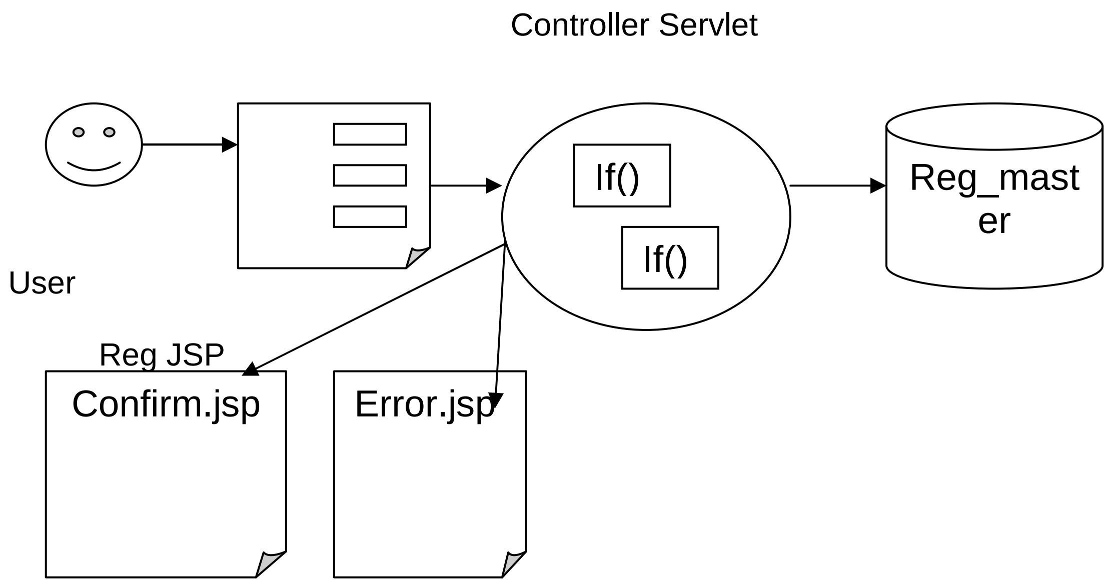
In Struts, the view is handled by JSPs and presentation components, the model is represented by Java Beans and the controller uses Servlets to perform its action.

By developing a familiar Web-based shopping cart, you'll learn how to utilize the Model-View-Controller (MVC) design pattern and truly separate presentation from content when using Java Server Pages.

❖ Applying MVC in Servlets and JSP

Many web applications are JSP-only or Servlets-only. With JSP, Java code is embedded in the HTML code; with Servlets the Java code calls println methods to generate the HTML code. Both approaches have their advantages and drawbacks; Struts gathers their strengths to get the best of their association.

Below you will find one example on registration form processing using MVC in Servlets and JSP:



1. In the above application Reg.jsp act as view accepts I/P from client and submits to Controller Servlet.
2. Controller Servlet validates the form data, if valid, stores the data into DB
3. Based on the validation and DB operations Controller Servlet decides to respond either Confirm.jsp or Error.jsp to client’s browser.

4. When the Error.jsp is responded, the page must include all the list of errors with detailed description.
5. The above shown application architecture is the model for MVC.
6. IF MVC Model 2 wants to be implemented in your application business logic and model operations must be separated from controller program.

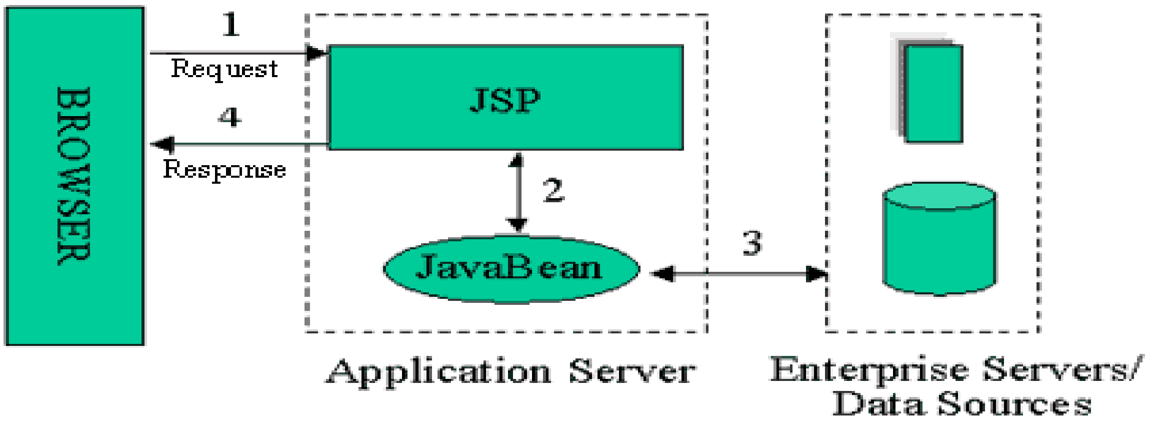
❖ **Struts**

Struts is an open source framework from Jakartha Project designed for developing the web applications with Java SERVLET API and Java Server Pages Technologies.Struts conforms the Model View Controller design pattern. Struts package provides unified reusable components (such as action servlet) to build the user interface that can be applied to any web connection. It encourages software development following the MVC design pattern.

❖ **View on JSP**

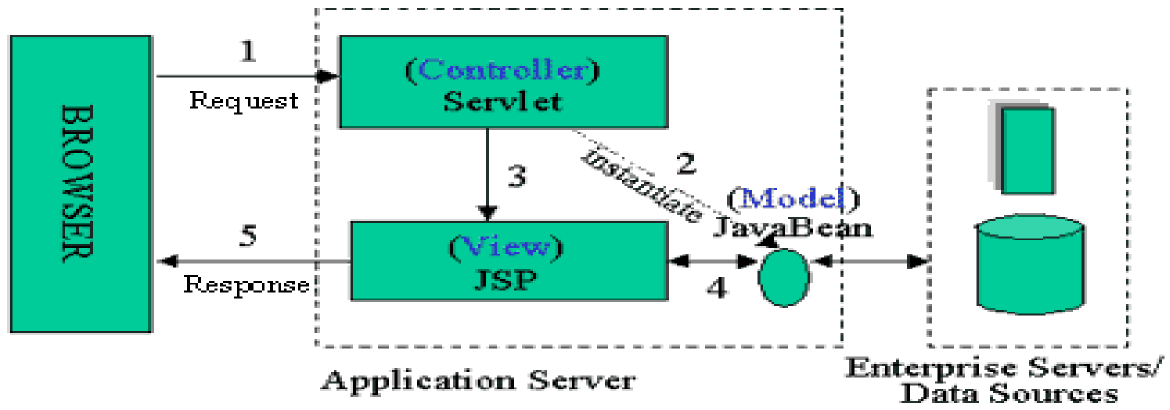
The early JSP specification follows two approaches for building applications using JSP technology. These two approaches are called as JSP Model 1 and JSP Model 2 architectures.

- **JSP Model 1 Architecture**



In **Model 1 architecture** the JSP page is alone responsible for processing the incoming request and replying back to the client. There is still separation of presentation from content, because all data access is performed using beans. Although the JSP Model 1 Architecture is more suitable for simple applications, it may not be desirable for complex implementations.

- **JSP Model 2 Architecture - MVC**

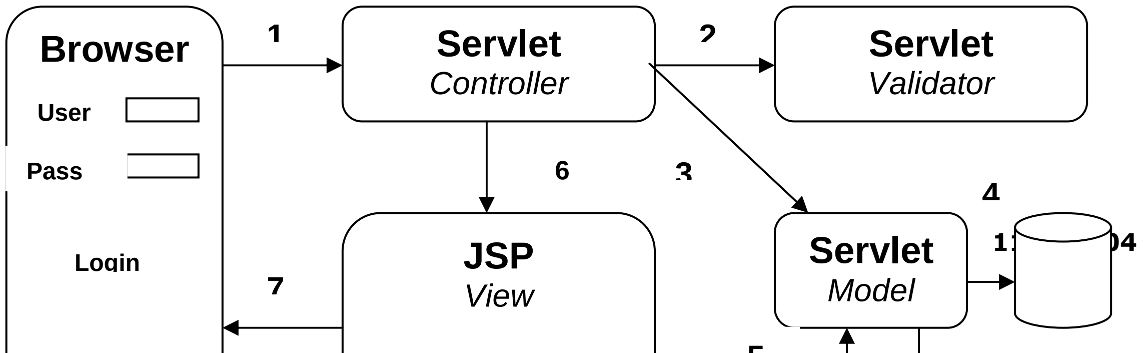


The **Model 2 Architecture** is an approach for serving dynamic content, since it combines the use of both Servlets and JSP. It takes advantages of the predominant strengths of both technologies, using JSP to generate the presentation layer and Servlets to perform process-intensive tasks. Here servlet acts as controller and is in charge of request processing and the creation of any beans or objects used by the JSP as well as deciding depending on the user's actions, which JSP page to forward the request to. Note that there is no processing logic within the JSP page itself; it is simply responsible for retrieving any objects or beans that may have been previously created by the servlet, and extracting the dynamic content from that servlet for insertion within static templates.

❖ **Limitation in traditional MVC approach**

The main limitation in the traditional MVC approach is, in that there is no separation of business logic (validation/ conditions/ anything related to business rules) from controller (is responsible for controlling of the application flow by using static/dynamic request dispatcher.

❖ **MVC Model 2 Architecture is Model View Controller**



1. Client submits login request to servlet application.
2. Servlet application acts as controller it first decides to request validator another servlet program which is responsible for not null checking (business rule).
3. control comes to controller back and based on the validation response, if the response is positive, servlet controller sends the request to model.
4. Model requests DB to verify whether the database is having the same user name and password, If found login operation is successful.
5. Beans are used to store if any data retrieved from the database and kept into HTTPSession.
6. Controller then gives response back to response JSP (view) which uses the bean objects stored in HTTPSession object.
7. and prepares presentation response on to the browser.

Overview of Struts Framework

❖ Introduction to Struts Framework

The goal of this project is to provide an open source framework for building Java web applications. The core of the Struts framework is a flexible control layer based on standard technologies like Java Servlets, JavaBeans, Resource Bundles, and XML, as well as various Jakarta Commons packages. Struts encourages application architectures based on the Model 2 approach, a variation of the classic Model-View-Controller (MVC) design paradigm.

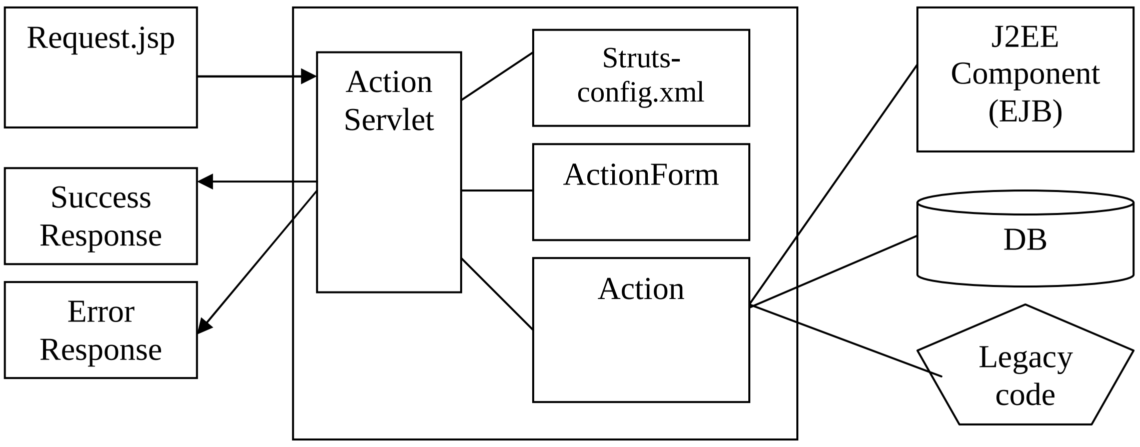
Struts provides its own Controller component and integrates with other technologies to provide the Model and the View.

For the Model, Struts can interact with standard data access technologies, like JDBC and EJB, as well as most any third-party packages, like Hibernate, iBATIS, or Object Relational Bridge.
For the View, Struts works well with Java Server Pages, including JSTL and JSF, as well as Velocity Templates, XSLT, and other presentation systems.
For Controller, ActionServlet and ActionMapping - The Controller portion of the application is focused on receiving requests from the client deciding what business logic function is to be performed, and then delegating responsibility for producing the next phase of the user interface to an appropriate View component. In Struts, the primary component of the Controller is a servlet of class ActionServlet. This servlet is configured by defining a set of ActionMappings. An ActionMapping defines a path that is matched against the request URI of the incoming request, and usually specifies the fully qualified class name of an Action class. Actions encapsulate the business logic, interpret the outcome, and ultimately dispatch control to the appropriate View component to create the response.

The Struts project was launched in May 2000 by Craig McClanahan to provide a standard MVC framework to the Java community. In July 2001.

In the MVC design pattern, application flow is mediated by a central Controller. The Controller delegates’ requests - in our case, HTTP requests - to an appropriate handler. The handlers are tied to a Model, and each handler acts as an adapter between the request and the Model. The Model represents, or encapsulates, an application's business logic or state. Control is usually then forwarded back through the Controller to the appropriate View. The forwarding can be determined by consulting a set of mappings, usually loaded from a database or configuration file. This provides a loose coupling between the View and Model, which can make applications significantly easier to create and maintain.

❖ Struts Architecture



❖ Front Controller

Context: The presentation-tier request handling mechanism must control and coordinate processing of each user across multiple requests. Such control mechanisms may be managed in either a centralized or decentralized manner.

Problem: The system requires a centralized access point for presentation-tier request handling to support the integration of system services, content retrieval, view management, and navigation. When the user accesses the view directly without going through a centralized mechanism,
Two problems may occur:

- Each view is required to provide its own system services, often resulting in duplicate code.
- View navigation is left to the views. This may result in commingled view content and view navigation.

Additionally, distributed control is more difficult to maintain, since changes will often need to be made in numerous places.

Solution: Use a controller as the initial point of contact for handling a request. The controller manages the handling of the request, including invoking security services such as authentication and authorization, delegating business processing, managing the choice of an appropriate view, handling errors, and managing the selection of content creation strategies.

The controller provides a centralized entry point that controls and manages Web request handling. By centralizing decision points and controls, the controller also helps reduce the amount of Java code, called scriptlets, embedded in the JavaServer Pages (JSP) page.

Centralizing control in the controller and reducing business logic in the view promotes code reuse across requests. It is a preferable approach to the alternative-embedding code in multiple views-because that approach may lead to a more error-prone, reuse-by-copy- and-paste environment.

Typically, a controller coordinates with a dispatcher component. Dispatchers are responsible for view management and navigation. Thus, a dispatcher chooses the next view for the user and vectors control to the resource. Dispatchers may be encapsulated within the controller directly or can be extracted into a separate component.

While the Front Controller pattern suggests centralizing the handling of all requests, it does not limit the number of handlers in the system, as does a Singleton. An application may use multiple controllers in a system, each mapping to a set of distinct services.

❖ **Structure**

Below figure represents the Front Controller class diagram pattern.

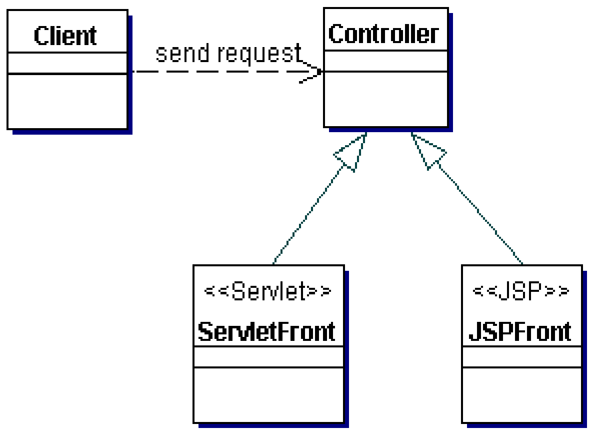


Figure: Front Controller class diagram

❖ **Participants and Responsibilities**

Below figure shows the sequence diagram representing the Front Controller pattern. It depicts how the controller handles a request.

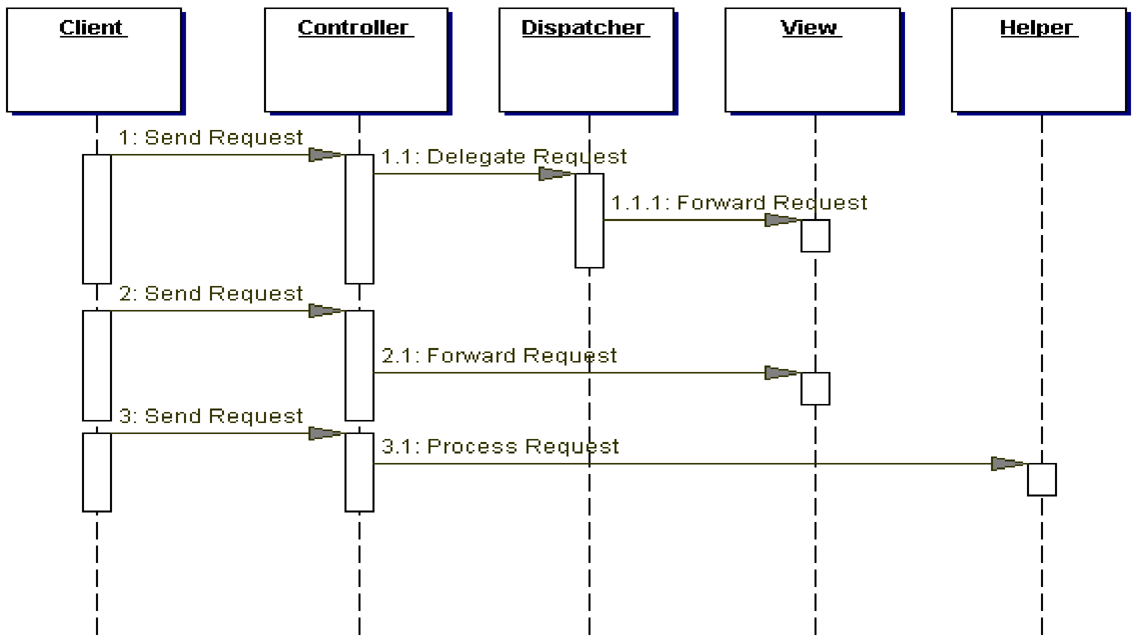


Figure: Front Controller sequence diagram

Controller: The controller is the initial contact point for handling all requests in the system. The controller may delegate to a helper to complete authentication and authorization of a user or to initiate contact retrieval.

Dispatcher: A dispatcher is responsible for view management and navigation, managing the choice of the next view to present to the user, and providing the mechanism for vectoring control to this resource.

A dispatcher can be encapsulated within a controller or can be a separate component working in coordination. The dispatcher provides either a static dispatching to the view or a more sophisticated dynamic dispatching mechanism.

The dispatcher uses the Request Dispatcher object (supported in the servlet specification) and encapsulates some additional processing.

Helper: A helper is responsible for helping a view or controller complete its processing. Thus, helpers have numerous responsibilities, including gathering data required by the view and storing this intermediate model, in which case the helper is sometimes referred to as a value bean. Additionally, helpers may adapt this data model for use by the view. Helpers can service requests for data from the view by simply providing access to the raw data or by formatting the data as Web content.

A view may work with any number of helpers, which are typically implemented as JavaBeans components (JSP 1.0+) and custom tags (JSP 1.1+). Additionally, a helper may represent a Command object, a delegate, or an XSL Transformer, which is used in combination with a stylesheet to adapt and convert the model into the appropriate form.

View: A view represents and displays information to the client. The view retrieves information from a model. Helpers support views by encapsulating and adapting the underlying data model for use in the display.

❖ **Controller Servlet – Action Servlet**

For those of you familiar with MVC architecture, the ActionServlet represents the C - the controller. The job of the controller is to:

- process user requests,

- determine what the user is trying to achieve according to the request,
- pull data from the model (if necessary) to be given to the appropriate view, and
- select the proper view to respond to the user.

The Struts controller delegates most of this grunt work to the Request Processor and Action classes.

In addition to being the front controller for your application, the ActionServlet instance also is responsible for initialization and clean-up of resources. When the controller initializes, it first loads the application config corresponding to the "config" init-param. It then goes through an enumeration of all init-param elements, looking for those elements who's name starts with config/. For each of these elements, Struts loads the configuration file specified by the value of that init-param, and assigns a "prefix" value to that module's ModuleConfig instance consisting of the piece of the init-param name following "config/". For example, the module prefix specified by the init-param config/foo would be "foo". This is important to know, since this is how the controller determines which module will be given control of processing the request. To access the module foo, you would use a URL like:
http://localhost:8080/myApp/foo/someAction.do
For each request made of the controller, the method process(HttpServletRequest, HttpServletResponse) will be called. This method simply determines which module should service the request and then invokes that module's RequestProcessor's process method, passing the same request and response.

❖ **Request Processor:**
The RequestProcessor is where the majority of the core processing occurs for each request. Let's take a look at the helper functions the process method invokes in-turn:

processPath	Determine the path that invoked us. This will be used later to retrieve an ActionMapping.
processLocale	Select a locale for this request, if one hasn't already been selected, and place it in the request.

processContent	Set the default content type (with optional character encoding) for all responses if requested.
processNoCache	If appropriate, set the following response headers: "Pragma", "Cache-Control", and "Expires".
processPreprocess	This is one of the "hooks" the RequestProcessor makes available for subclasses to override. The default implementation simply returns true. If you subclass RequestProcessor and override processPreprocess you should either return true (indicating process should continue processing the request) or false (indicating you have handled the request and the process should return)
processMapping	Determine the ActionMapping associated with this path.
processRoles	If the mapping has a role associated with it, ensure the requesting user is has the specified role. If they do not, raise an error and stop processing of the request.
processActionForm	Instantiate (if necessary) the ActionForm associated with this mapping (if any) and place it into the appropriate scope.
processPopulate	Populate the ActionForm associated with this request, if any.
processValidate	Perform validation (if requested) on the ActionForm associated with this request (if any).
processForward	If this mapping represents a forward, forward to the path specified by the mapping.
processInclude	If this mapping represents an include, include the result of invoking the path in this request.
processActionCreate	Instantiate an instance of the class specified by the current ActionMapping (if necessary).
processActionPerform	This is the point at which your action's perform or execute method will be called.
processFor	Finally, the process method of the

wardConfig	RequestProcessor takes the ActionForward returned by your Action class, and uses to select the next resource (if any). Most often the ActionForward leads to the presentation page that renders the response.

❖ Action class

The Action class defines two methods that could be executed depending on your servlet environment:

```
public ActionForward execute(ActionMapping mapping,
    ActionForm form,
    ServletRequest request,
    ServletResponse response)
    throws Exception;
public ActionForward execute(ActionMapping mapping,
    ActionForm form,
    HttpServletRequest request,
    HttpServletResponse response)
    throws Exception;
```

Since the majority of Struts projects are focused on building web applications, most projects will only use the "HttpServletRequest" version. A non-HTTP execute() method has been provided for applications that are not specifically geared towards the HTTP protocol.

The goal of an Action class is to process a request, via its execute method, and return an ActionForward object that identifies where control should be forwarded (e.g. a JSP, Tile definition, Velocity template, or another Action) to provide the appropriate response. In the MVC/Model 2 design pattern, a typical Action class will often implement logic like the following in its execute method:

- Validate the current state of the user's session (for example, checking that the user has successfully logged on). If the Action class finds that no logon exists, the request can be forwarded to the presentation page that displays the username and password prompts for logging on. This could occur because a user tried to enter an application "in the middle" (say, from a bookmark), or because the session has timed out, and the servlet container created a new one.

- If validation is not complete, validate the form bean properties as needed. If a problem is found, store the appropriate error message keys as a request attribute, and forward control back to the input form so that the errors can be corrected.
- Perform the processing required to deal with this request (such as saving a row into a database). This can be done by logic code embedded within the Action class itself, but should generally be performed by calling an appropriate method of a business logic bean.
- Update the server-side objects that will be used to create the next page of the user interface (typically request scope or session scope beans, depending on how long you need to keep these items available).
- Return an appropriate ActionForward object that identifies the presentation page to be used to generate this response, based on the newly updated beans. Typically, you will acquire a reference to such an object by calling findForward on either the ActionMapping object you received (if you are using a logical name local to this mapping), or on the controller servlet itself (if you are using a logical name global to the application).

In Struts 1.0, Actions called a perform method instead of the now-preferred execute method. These methods use the same parameters and differ only in which exceptions they throw. The elder perform method throws SerlvetException and IOException. The new execute method simply throws Exception. The change was to facilitate the Declarative Exception handling feature introduced in Struts 1.1.

The perform method may still be used in Struts 1.1 but is deprecated. The Struts 1.1 method simply calls the new execute method and wraps any Exception thrown as a ServletException.

❖ **Action Form class**

An ActionForm represents an HTML form that the user interacts with over one or more pages. You will provide properties to hold the state of the form with getters and setters to access them. ActionForms can be stored in either the session (default) or request scopes. If they're in the session it's important to implement the form's reset method to initialize the form before each use. Struts sets the ActionForm's properties from the request parameters and sends the validated form to the appropriate Action's execute method.

When you code your ActionForm beans, keep the following principles in mind:

- The ActionForm class itself requires no specific methods to be implemented. It is used to identify the role these particular beans play in the overall architecture. Typically, an ActionForm bean will have only property getter and property setter methods, with no business logic.
- The ActionForm object also offers a standard validation mechanism. If you override a "stub" method, and provide error messages in the standard application resource, Struts will automatically validate the input from the form (using your method). See "Automatic Form Validation" for details. Of course, you can also ignore the ActionForm validation and provide your own in the Action object.
- Define a property (with associated getXxx and setXxx methods) for each field that is present in the form. The field name and property name must match according to the usual JavaBeans conventions (see the Javadoc for the java.beans.Introspector class for a start on information about this). For example, an input field named username will cause the setUsername method to be called.
- Buttons and other controls on your form can also be defined as properties. This can help determine which button or control was selected when the form was submitted. Remember, the ActionForm is meant to represent your data-entry form, not just the data beans.
- Think of your ActionForm beans as a firewall between HTTP and the Action. Use the validate method to ensure all required properties are present, and that they contain reasonable values. An ActionForm that fails validation will not even be presented to the Action for handling.
- You may also place a bean instance on your form, and use nested property references. For example, you might have a "customer" bean on your ActionForm, and then refer to the property "customer.name" in your presentation page. This

would correspond to the methods customer.getName() and customer.setName(string Name) on your customer bean. See the Tag Library Developer Guides for more about using nested syntax with the Struts JSP tags.

- **Caution:** If you nest an existing bean instance on your form, think about the properties it exposes. Any public property on an ActionForm that accepts a single String value can be set with a query string. It may be useful to place beans that can affect the business state inside a thin "wrapper" that exposes only the properties required. This wrapper can also provide a filter to be sure runtime properties are not set to inappropriate values.

❖ Action class Design guidelines

Remember the following design guidelines when coding Action classes:

- Write code for a multi-threaded environment - The controller servlet creates only one instance of your Action class, and uses this one instance to service all requests. Thus, you need to write thread-safe Action classes. Follow the same guidelines you would use to write thread-safe Servlets. Here are two general guidelines that will help you write scalable, thread-safe Action classes:
  - Only Use Local Variables - The most important principle that aids in thread-safe coding is to use only local variables, not instance variables, in your Action class. Local variables are created on a stack that is assigned (by your JVM) to each request thread, so there is no need to worry about sharing them. An Action can be factored into several local methods, so long as all variables needed are passed as method parameters. This assures thread safety, as the JVM handles such variables internally using the call stack which is associated with a single Thread.
  - Conserve Resources - As a general rule, allocating scarce resources and keeping them across requests from the same user (in the user's session) can cause scalability problems. For example, if your application uses JDBC and you allocate a separate JDBC connection for every user, you are probably going to run in some scalability issues when your site suddenly shows up on Slashdot. You should strive to use pools and release resources (such as

database connections) prior to forwarding control to the appropriate View component -- even if a bean method you have called throws an exception.

- Don't throw it, catch it! - Ever used a commercial website only to have a stack trace or exception thrown in your face after you've already typed in your credit card number and clicked the purchase button? Let's just say it doesn't inspire confidence. Now is your chance to deal with these application errors - in the Action class. If your application specific code throws expections you should catch these exceptions in your Action class, log them in your application's log (servlet.log("Error message", exception)) and return the appropriate ActionForward.

  It is wise to avoid creating lengthy and complex Action classes. If you start to embed too much logic in the Action class itself, you will begin to find the Action class hard to understand, maintain, and impossible to reuse. Rather than creating overly complex Action classes, it is generally a good practice to move most of the persistence, and "business logic" to a separate application layer. When an Action class becomes lengthy and procedural, it may be a good time to refactor your application architecture and move some of this logic to another conceptual layer; otherwise, you may be left with an inflexible application which can only be accessed in a web-application environment. Struts should be viewed as simply the foundation for implementing MVC in your applications. Struts provides you with a useful control layer, but it is not a fully featured platform for building MVC applications, soup to nuts.

  The MailReader example application included with Struts stretches this design principle somewhat, because the business logic itself is embedded in the Action classes. This should be considered something of a bug in the design of the example, rather than an intrinsic feature of the Struts architecture, or an approach to be emulated. In order to demonstrate, in simple terms, the different ways Struts can be used, the MailReader application does not always follow best practices.

❖ **Action mapping implementation**

In order to operate successfully, the Struts controller servlet needs to know several things about how each request URI should be mapped to an appropriate Action class. The required knowledge has been encapsulated in a Java class named ActionMapping, the most important properties are as follows:

- **type** - Fully qualified Java class name of the Action implementation class used by this mapping.
- **name** - The name of the form bean defined in the config file that this action will use.
- **path** - The request URI path that is matched to select this mapping. See below for examples of how matching works and how to use wildcards to match multiple request URIs.
- **unknown** - Set to true if this action should be configured as the default for this application, to handle all requests not handled by another action. Only one action can be defined as a default within a single application.
- **validate** - Set to true if the validate method of the action associated with this mapping should be called.
- **forward** - The request URI path to which control is passed when this mapping is invoked. This is an alternative to declaring a type property.

❖ **Writing Action Mappings**

How does the controller servlet learn about the mappings you want? It would be possible (but tedious) to write a small Java class that simply instantiated new ActionMapping instances, and called all of the appropriate setter methods. To make this process easier, Struts uses the Jakarta Commons Digester component to parse an XML-based description of the desired mappings and create the appropriate objects initialized to the appropriate default values. See the Jakarta Commons website for more information about the Digester.

The developer's responsibility is to create an XML file named struts-config.xml and place it in the WEB-INF directory of your application. This format of this document is described by the Document Type Definition (DTD) maintained at http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd. This chapter covers the configuration elements that you will typically write as part of developing your application. There are several other elements that can be placed in the struts-config file to customize your application. See "Configuring Applications" for more about the other elements in the Struts configuration file.

The controller uses an internal copy of this document to parse the configuration; an Internet connection is not required for operation.

The outermost XML element must be <struts-config>. Inside of the <struts-config> element, there are three important elements that are used to describe your actions:

- <form-beans>
- <global-forwards>
- <action-mappings>

<**form-beans**>
This section contains your form bean definitions. Form beans are descriptors that are used to create ActionForm instances at runtime. You use a <form-bean> element for each form bean, which has the following important attributes:

- **name:** A unique identifier for this bean, which will be used to reference it in corresponding action mappings. Usually, this is also the name of the request or session attribute under which this form bean will be stored.
- **type:** The fully-qualified Java classname of the ActionForm subclass to use with this form bean.

<**global-forwards**>
This section contains your global forward definitions. Forwards are instances of the ActionForward class returned from an ActionForm's execute method. These map logical names to specific resources (typically JSPs), allowing you to change the resource without changing references to it throughout your application. You use a <forward> element for each forward definition, which has the following important attributes:

- **name:** The logical name for this forward. This is used in your ActionForm's execute method to forward to the next appropriate resource. Example: homepage

- **path:** The context relative path to the resource. Example: /index.jsp or /index.do
- **redirect:** True or false (default). Should the ActionServlet redirect to the resource instead of forward?

<**action-mappings**>
This section contains your action definitions. You use an <action> element for each of the mappings you would like to define. Most action elements will define at least the following attributes:

- **path:** The application context-relative path to the action.
- **type:** The fully qualified java classname of your Action class.
- **name:** The name of your <form-bean> element to use with this action

Other often-used attributes include:

- **parameter:** A general-purpose attribute often used by "standard" Actions to pass a required property.
- **roles:** A comma-delimited list of the user security roles that can access this mapping.

For a complete description of the elements that can be used with the action element, see the Struts Configuration DTD and the ActionMapping documentation.

❖ Action Mapping Example

Here's a mapping entry based on the MailReader example application. The MailReader application now uses DynaActionForms. But in this example, we'll show a conventinal ActionForm instead, to illustrate the usual workflow. Note that the entries for all the other actions are left out:

```
<struts-config>
   <form-beans>
      <form-bean
         name="logonForm"
         type="org.apache.struts.webapp.example.LogonForm"
         />
   </form-beans>
   <global-forwards
   type="org.apache.struts.action.ActionForward">
```

```
    <forward
      name="logon"
      path="/logon.jsp"
      redirect="false" />
  </global-forwards>
  <action-mappings>
    <action
      path   ="/logon"
      type
      ="org.apache.struts.webapp.example.LogonAction"
      name ="logonForm"
      scope ="request"
      input  ="/logon.jsp"
      unknown="false"
      validate="true" />
  </action-mappings>
</struts-config>
```

First the form bean is defined. A basic bean of class "org.apache.struts.webapp.example.LogonForm" is mapped to the logical name "logonForm". This name is used as a request attribute name for the form bean.

The "global-forwards" section is used to create logical name mappings for commonly used presentation pages. Each of these forwards is available through a call to your action mapping instance, i.e. mapping.findForward("logicalName").

As you can see, this mapping matches the path /logon (actually, because the MailReader example application uses extension mapping, the request URI you specify in a JSP page would end in /logon.do). When a request that matches this path is received, an instance of the LogonAction class will be created (the first time only) and used. The controller servlet will look for a bean in request scope under key logonForm, creating and saving a bean of the specified class if needed.

Optional but very useful are the local "forward" elements. In the MailReader example application, many actions include a local "success" and/or "failure" forward as part of an action mapping.

```
<!-- Edit mail subscription -->
<action
path="/editSubscription"
   type="org.apache.struts.webapp.example.EditSubscriptionA
   ction"
   name="subscriptionForm"
   scope="request"
   validate="false">
   <forward name="failure" path="/mainMenu.jsp"/>
   <forward name="success" path="/subscription.jsp"/>
</action>
```

Using just these two extra properties, the Action classes are almost totally independent of the actual names of the presentation pages. The pages can be renamed (for example) during a redesign, with negligible impact on the Action classes themselves. If the names of the "next" pages were hard coded into the Action classes, all of these classes would also need to be modified. Of course, you can define whatever local forward properties makes sense for your own application.

The Struts configuration file includes several other elements that you can use to customize your application. See "Configuring Applications" for details.

❖ **Using Action Mapping for pages**

Fronting your pages with ActionMappings is essential when using modules, since doing so is the only way you involve the controller in the request -- and you want to! The controller puts the application configuration in the request, which makes available all of your module-specific configuration data (including which message resources you are using, request-processor, datasources, and so forth).
The simplest way to do this is to use the forward property of the ActionMapping:

```
<action path="/view" forward="/view.jsp"/>
```

❖ **Configuring struts-config.xml file**

The Building Controller Components chapter covered writing the form-bean and action-mapping portions of the Struts configuration file. These elements usually play an important

role in the development of a Struts application. The other elements in Struts configuration file tend to be static: you set them once and leave them alone.
These "static" configuration elements are:

- controller
- message-resources
- plug-in
- data-sources

❖ **Controller configuration**

The <controller> element allows you to configure the ActionServlet. Many of the controller parameters were previously defined by servlet initialization parameters in your web.xml file but have been moved to this section of struts-config.xml in order to allow different modules in the same web application to be configured differently. For full details on available parameters see the struts-config_1_2.dtd or the list below.

- **bufferSize** - The size (in bytes) of the input buffer used when processing file uploads. [4096] (optional)
- **className** - Classname of configuration bean. [org.apache.struts.config.ControllerConfig] (optional)
- **contentType** - Default content type (and optional character encoding) to be set on each response. May be overridden by the Action, JSP, or other resource to which the request is forwarded. [text/html] (optional)
- **forwardPattern** - Replacement pattern defining how the "path" attribute of a <forward> element is mapped to a context-relative URL when it starts with a slash (and when the contextRelative property is false). This value may consist of any combination of the following:
  - $M - Replaced by the module prefix of this module.
  - $P - Replaced by the "path" attribute of the selected <forward> element.
  - $$ - Causes a literal dollar sign to be rendered.
  - $x - (Where "x" is any character not defined above) Silently swallowed, reserved for future use.

  If not specified, the default forwardPattern is consistent with the previous behavior of forwards. [MP] (optional)
- **inputForward** - Set to true if you want the input attribute of <action> elements to be the name of a local or global

ActionForward, which will then be used to calculate the ultimate URL. Set to false to treat the input parameter of <action> elements as a module-relative path to the resource to be used as the input form. [false] (optional)

- **locale** - Set to true if you want a Locale object stored in the user's session if not already present. [true] (optional)
- **maxFileSize** - The maximum size (in bytes) of a file to be accepted as a file upload. Can be expressed as a number followed by a "K", "M", or "G", which are interpreted to mean kilobytes, megabytes, or gigabytes, respectively. [250M] (optional)
- **multipartClass** - The fully qualified Java class name of the multipart request handler class to be used with this module. [org.apache.struts.upload.CommonsMultipartRequestHandler] (optional)
- **nocache** - Set to true if you want the controller to add HTTP headers for defeating caching to every response from this module. [false] (optional)
- **pagePattern** - Replacement pattern defining how the page attribute of custom tags using it is mapped to a context-relative URL of the corresponding resource. This value may consist of any combination of the following:
  - $M - Replaced by the module prefix of this module.
  - $P - Replaced by the "path" attribute of the selected <forward> element.
  - $$ - Causes a literal dollar sign to be rendered.
  - $x - (Where "x" is any character not defined above) Silently swallowed, reserved for future use.

  If not specified, the default pagePattern is consistent with the previous behavior of URL calculation. [MP] (optional)
- **processorClass** - The fully qualified Java class name of the RequestProcessor subclass to be used with this module. [org.apache.struts.action.RequestProcessor] (optional)
- **tempDir** - Temporary working directory to use when processing file uploads. [{the directory provided by the servlet container}]

This example uses the default values for several controller parameters. If you only want default behavior you can omit the controller section altogether.

```
<controller
   processorClass="org.apache.struts.action.RequestProcessor"
   debug="0"
contentType="text/html"/>;
```

❖ **Message Resource configuration**

Struts has built in support for internationalization (I18N). You can define one or more <message-resources> elements for your webapp; modules can define their own resource bundles. Different bundles can be used simultaneously in your application, the 'key' attribute is used to specify the desired bundle.

- **className** - Classname of configuration bean. [org.apache.struts.config.MessageResourcesConfig] (optional)
- **factory** - Classname of MessageResourcesFactory. [org.apache.struts.util.PropertyMessageResourcesFactory] (optional)
- **key** - ServletContext attribute key to store this bundle. [org.apache.struts.action.MESSAGE] (optional)
- **null** - Set to false to display missing resource keys in your application like '???keyname???' instead of null. [true] (optional)
- **parameter** - Name of the resource bundle. (required)

Example configuration:

```
<message-resources        parameter="MyWebAppResources"
null="false" />
```

This would set up a message resource bundle provided in the file MyWebAppResources.properties under the default key. Missing resource keys would be displayed as '???keyname???'.

❖ **PlugIn configuration**

Struts PlugIns are configured using the <plug-in> element within the Struts configuration file. This element has only one valid attribute, 'className', which is the fully qualified name of the Java class which implements the org.apache.struts.action.PlugIn interface.

For PlugIns that require configuration themselves, the nested <set-property> element is available.

This is an example using the Tiles plugin:

```
<plug-in className="org.apache.struts.tiles.TilesPlugin" >
   <set-property  property="definitions-config"  value="/WEB-
   INF/tiles-defs.xml"/>
</plug-in>
```

❖ **DataSource configuration**

Besides the objects related to defining ActionMappings, the Struts configuration may contain elements that create other useful objects.

The <data-sources> section can be used to specify a collection of DataSources [javax.sql.DataSource] for the use of your application. Typically, a DataSource represents a connection pool to a database or other persistent store. As a convenience, the Struts DataSource manager can be used to instantiate whatever standard pool your application may need. Of course, if your persistence layer provides for its own connections, then you do not need to specify a data-sources element.

Since DataSource implementations vary in what properties need to be set, unlike other Struts configuration elements, the data-source element does not pre-define a slate of properties. Instead, the generic set-property feature is used to set whatever properties your implementation may require. Typically, these settings would include:

- A driver class name
- A url to access the driver
- A description

And other sundry properties.

```
<data-source
type="org.apache.commons.dbcp.BasicDataSource">
<!-- ... set-property elements ... -->
</data-source>
```

In Struts 1.2.0, the GenericDataSource has been removed, and it is recommended that you use the Commons BasicDataSource or other DataSource implementation instead. In practice, if you need to use the DataSource manager, you should use whatever DataSource implementation works best with your container or database.

For examples of specifying a data-sources element and using the DataSource with an Action.

❖ The Struts configuration file

The Building Controller Components chapter covered writing the form-bean and action-mapping portions of the Struts configuration file. These elements usually play an important role in the development of a Struts application. The other elements in Struts configuration file tend to be static: you set them once and leave them alone.

These "static" configuration elements are:

- controller
- message-resources
- plug-in
- data-sources

❖ Controller Configuration

The <controller> element allows you to configure the ActionServlet. Many of the controller parameters were previously defined by servlet initialization parameters in your web.xml file but have been moved to this section of struts-config.xml in order to allow different modules in the same web application to be configured differently. For full details on available parameters see the struts-config_1_2.dtd or the list below.

- **bufferSize** - The size (in bytes) of the input buffer used when processing file uploads. [4096] (optional)
- **className** - Classname of configuration bean. [org.apache.struts.config.ControllerConfig] (optional)
- **contentType** - Default content type (and optional character encoding) to be set on each response. May be overridden by the Action, JSP, or other resource to which the request is forwarded. [text/html] (optional)
- **forwardPattern** - Replacement pattern defining how the "path" attribute of a <forward> element is mapped to a context-relative URL when it starts with a slash (and when the contextRelative property is false). This value may consist of any combination of the following:
  - $M - Replaced by the module prefix of this module.

  - o $P - Replaced by the "path" attribute of the selected <forward> element.
  - o $$ - Causes a literal dollar sign to be rendered.
  - o $x - (Where "x" is any character not defined above) Silently swallowed, reserved for future use.

  If not specified, the default forwardPattern is consistent with the previous behavior of forwards. [MP] (optional)
- **inputForward** - Set to true if you want the input attribute of <action> elements to be the name of a local or global ActionForward, which will then be used to calculate the ultimate URL. Set to false to treat the input parameter of <action> elements as a module-relative path to the resource to be used as the input form. [false] (optional)
- **locale** - Set to true if you want a Locale object stored in the user's session if not already present. [true] (optional)
- **maxFileSize** - The maximum size (in bytes) of a file to be accepted as a file upload. Can be expressed as a number followed by a "K", "M", or "G", which are interpreted to mean kilobytes, megabytes, or gigabytes, respectively. [250M] (optional)
- **multipartClass** - The fully qualified Java class name of the multipart request handler class to be used with this module. [org.apache.struts.upload.CommonsMultipartRequestHandler] (optional)
- **nocache** - Set to true if you want the controller to add HTTP headers for defeating caching to every response from this module. [false] (optional)
- **pagePattern** - Replacement pattern defining how the page attribute of custom tags using it is mapped to a context-relative URL of the corresponding resource. This value may consist of any combination of the following:
  - o $M - Replaced by the module prefix of this module.
  - o $P - Replaced by the "path" attribute of the selected <forward> element.
  - o $$ - Causes a literal dollar sign to be rendered.
  - o $x - (Where "x" is any character not defined above) Silently swallowed, reserved for future use.

  If not specified, the default pagePattern is consistent with the previous behavior of URL calculation. [MP] (optional)

- **processorClass** - The fully qualified Java class name of the RequestProcessor subclass to be used with this module. [org.apache.struts.action.RequestProcessor] (optional)
- **tempDir** - Temporary working directory to use when processing file uploads. [{the directory provided by the servlet container}]

This example uses the default values for several controller parameters. If you only want default behavior you can omit the controller section altogether.

```
<controller
    processorClass="org.apache.struts.action.RequestProcessor"
    debug="0"
contentType="text/html"/>;
```

❖ Message Resources Configuration

Struts has built in support for internationalization (I18N). You can define one or more <message-resources> elements for your webapp; modules can define their own resource bundles. Different bundles can be used simultaneously in your application, the 'key' attribute is used to specify the desired bundle.

- **className** - Classname of configuration bean. [org.apache.struts.config.MessageResourcesConfig] (optional)
- **factory** - Classname of MessageResourcesFactory. [org.apache.struts.util.PropertyMessageResourcesFactory] (optional)
- **key** - ServletContext attribute key to store this bundle. [org.apache.struts.action.MESSAGE] (optional)
- **null** - Set to false to display missing resource keys in your application like '???keyname???' instead of null. [true] (optional)
- **parameter** - Name of the resource bundle. (required)

Example configuration:

```
<message-resources        parameter="MyWebAppResources"
null="false" />
```

This would set up a message resource bundle provided in the file MyWebAppResources.properties under the default key. Missing resource keys would be displayed as '???keyname???'.

- **PlugIn Configuration**

  Struts PlugIns are configured using the <plug-in> element within the Struts configuration file. This element has only one valid attribute, 'className', which is the fully qualified name of the Java class which implements the org.apache.struts.action.PlugIn interface.

  For PlugIns that require configuration themselves, the nested <set-property> element is available.

  This is an example using the **Tiles** plugin:

  ```
  <plug-in
  className="org.apache.struts.tiles.TilesPlugin">
    <set-property property="definitions-config"
    value="/WEB-INF/tiles-defs.xml"/>
  </plug-in>
  ```

- **Data Source Configuration**

  Besides the objects related to defining ActionMappings, the Struts configuration may contain elements that create other useful objects.

  The <data-sources> section can be used to specify a collection of DataSources [javax.sql.DataSource] for the use of your application. Typically, a DataSource represents a connection pool to a database or other persistent store. As a convenience, the Struts DataSource manager can be used to instantiate whatever standard pool your application may need. Of course, if your persistence layer provides for its own connections, then you do not need to specify a data-sources element.

  Since DataSource implementations vary in what properties need to be set, unlike other Struts configuration elements, the data-source element does not pre-define a slate of properties. Instead, the generic set-property feature is used to set whatever properties your implementation may require. Typically, these settings would include:

  - A driver class name
  - A url to access the driver
  - A description

  And other sundry properties.

```
<data-source
type="org.apache.commons.dbcp.BasicDataSource">
<!-- ... set-property elements ... -->
</data-source>
```

In Struts 1.2.0, the GenericDataSource has been removed, and it is recommended that you use the Commons BasicDataSource or other DataSource implementation instead. In practice, if you need to use the DataSource manager, you should use whatever DataSource implementation works best with your container or database.

For examples of specifying a data-sources element and using the DataSource with an Action, see the Accessing a Database HowTo.

❖ **Configuring your application for modules**

Very little is required in order to start taking advantage of the Struts module feature. Just go through the following steps:

1. Prepare a config file for each module.
2. Inform the controller of your module.
3. Use actions to refer to your pages.

❖ **Module Configuration Files**

Back in Struts 1.0, a few "boot-strap" options were placed in the web.xml file, and the bulk of the configuration was done in a single struts-config.xml file. Obviously, this wasn't ideal for a team environment, since multiple users had to share the same configuration file.

In Struts 1.1, you have two options: you can list multiple struts-config files as a comma-delimited list, or you can subdivide a larger application into modules.

With the advent of modules, a given module has its own configuration file. This means each team (each module would presumably be developed by a single team) has their own configuration file, and there should be a lot less contention when trying to modify it.

❖ **Informing the Controller**

In struts 1.0, you listed your configuration file as an initialization parameter to the action servlet in web.xml. This is still done in 1.1, but it's augmented a little. In order to tell the Struts machinery about your different modules, you specify multiple config initialization parameters, with a slight twist. You'll still use "config" to tell the action servlet about your "default" module, however, for each additional module, you will list an initialization parameter named "config/module", where module is the name of your module (this gets used when determining which URIs fall under a given module, so choose something meaningful!). For example:

```
...
<init-param>
   <param-name>config</param-name>
   <param-value>/WEB-INF/conf/struts-default.xml</param-
   value>
</init-param>
<init-param>
   <param-name>config/module1</param-name>
   <param-value>/WEB-INF/conf/struts-
   module1.xml</param-value>
</init-param>
...
```

This says I have two modules. One happens to be the "default" module, which has no "/module" in it's name, and one named "module1" (config/module1). I've told the controller it can find their respective configurations under /WEB-INF/conf (which is where I put all my configuration files). Pretty simple!

(My struts-default.xml would be equivalent to what most folks call struts-config.xml. I just like the symmetry of having all my Struts module files being named struts-<module>.xml)

If you'd like to vary where the pages for each module is stored, see the forwardPattern setting for the Controller.

❖ Switching Modules

There are two basic methods to switching from one module to another. You can either use a forward (global or local) and specify the contextRelative attribute with a value of true, or

you can use the built-in org.apache.struts.actions.SwitchAction.
Here's an example of a global forward:

```
...
<struts-config>
    ...
    <global-forwards>
        <forward name="toModuleB"
        contextRelative="true"
        path="/moduleB/index.do"
        redirect="true"/>
        ...
    </global-forwards>
    ...
</struts-config>
```

You could do the same thing with a local forward declared in an ActionMapping:

```
...
<struts-config>
    ...
    <action-mappings>
        ...
        <action ... >
            <forward name="success"
            contextRelative="true"
            path="/moduleB/index.do"
            redirect="true"/>
        </action>
        ...
    </action-mappings>
    ...
</struts-config>
```

Finally, you could use org.apache.struts.actions.SwitchAction, like so:

```
...
<action-mappings>
    <action path="/toModule"
    type="org.apache.struts.actions.SwitchAction"/>
    ...
```

```
</action-mappings>
...
```

Now, to change to ModuleB, we would use a URI like this:
http://localhost:8080/toModule.do?prefix=/moduleB&page=/index.do

If you are using the "default" module as well as "named" modules (like "/moduleB"), you can switch back to the "default" module with a URI like this:
http://localhost:8080/toModule.do?prefix=&page=/index.do
That's all there is to it! Happy module-switching!

❖ **The Web Application Deployment Descriptor**

The final step in setting up the application is to configure the application deployment descriptor (stored in file WEB-INF/web.xml) to include all the Struts components that are required. Using the deployment descriptor for the example application as a guide, we see that the following entries need to be created or modified.

❖ **Configure the Action Servlet Instance**

Add an entry defining the action servlet itself, along with the appropriate initialization parameters. Such an entry might look like this:

```
<servlet>
    <servlet-name>action</servlet-name>
    <servlet-
    class>org.apache.struts.action.ActionServlet</servlet-
    class>
    <init-param>
        <param-name>config</param-name>
        <param-value>/WEB-INF/struts-config.xml</param-
        value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
```

The initialization parameters supported by the controller servlet are described below. (You can also find these details in the Javadocs for the ActionServlet class.) Square brackets describe the default values that are assumed if you do not provide a value for that initialization parameter.

- **config** - Context-relative path to the XML resource containing the configuration information for the default module. This may also be a comma-delimited list of configuration files. Each file is loaded in turn, and its objects are appended to the internal data structure. [/WEB-INF/struts-config.xml].

Warning - If you define an object of the same name in more than one configuration file, the last one loaded quietly wins.

- **config/${module}** - Context-relative path to the XML resource containing the configuration information for the application module that will use the specified prefix (/${module}). This can be repeated as many times as required for multiple application modules. (Since Struts 1.1)
- **convertNull** - Force simulation of the Struts 1.0 behavior when populating forms. If set to true, the numeric Java wrapper class types (like java.lang.Integer) will default to null (rather than 0). (Since Struts 1.1) [false]
- **rulesets** - Comma-delimited list of fully qualified classnames of additional org.apache.commons.digester.RuleSet instances that should be added to the Digester that will be processing struts-config.xml files. By default, only the RuleSet for the standard configuration elements is loaded. (Since Struts 1.1)
- **validating** - Should we use a validating XML parser to process the configuration file (strongly recommended)? [true]

Warning - Struts will not operate correctly if you define more than one <servlet> element for a controller servlet, or a subclass of the standard controller servlet class. The controller servlet MUST be a web application wide singleton.

❖ **Configure the Action Servlet Mapping**

Note: The material in this section is not specific to Struts. The configuration of servlet mappings is defined in the Java Servlet Specification. This section describes the most common means of configuring a Struts application.

There are two common approaches to defining the URLs that will be processed by the controller servlet - prefix matching and extension matching. An appropriate mapping entry for each approach will be described below.

Prefix matching means that you want all URLs that start (after the context path part) with a particular value to be passed to this servlet. Such an entry might look like this:

```
<servlet-mapping>
   <servlet-name>action</servlet-name>
   <url-pattern>/do/*</url-pattern>
</servlet-mapping>
```

which means that a request URI to match the /logon path described earlier might look like this:
http://www.mycompany.com/myapplication/do/logon
where /myapplication is the context path under which your application is deployed.

Extension mapping, on the other hand, matches request URIs to the action servlet based on the fact that the URI ends with a period followed by a defined set of characters. For example, the JSP processing servlet is mapped to the *.jsp pattern so that it is called to process every JSP page that is requested. To use the *.do extension (which implies "do something"), the mapping entry would look like this:

```
<servlet-mapping>
   <servlet-name>action</servlet-name>
   <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

and a request URI to match the /logon path described earlier might look like this:
http://www.mycompany.com/myapplication/logon.do
Warning - Struts will not operate correctly if you define more than one <servlet-mapping> element for the controller servlet.
Warning - If you are using the new module support in Struts 1.1, you should be aware that only extension mapping is supported.

❖ **Configure the Struts Tag Libraries**

Next, you must add an entry defining the Struts tag library.

The struts-bean taglib contains tags useful in accessing beans and their properties, as well as defining new beans (based on these accesses) that are accessible to the remainder of the page via scripting variables and page scope attributes. Convenient mechanisms to create new beans based on the value of request cookies, headers, and parameters are also provided.

The struts-html taglib contains tags used to create struts input forms, as well as other tags generally useful in the creation of HTML-based user interfaces.

The struts-logic taglib contains tags that are useful in managing conditional generation of output text, looping over object collections for repetitive generation of output text, and application flow management.

The struts-tiles taglib contains tags used for combining various view components, called "tiles", into a final composite view.

The struts-nested taglib is an extension of other struts taglibs that allows the use of nested beans.

Below is how you would define all taglibs for use within your application. In practice, you would only specify the taglibs that your application uses:

```
<taglib>
   <taglib-uri>/tags/struts-bean</taglib-uri>
   <taglib-location>/WEB-INF/struts-bean.tld</taglib-
   location>
</taglib>
<taglib>
   <taglib-uri>/tags/struts-html</taglib-uri>
   <taglib-location>/WEB-INF/struts-html.tld</taglib-
   location>
</taglib>
<taglib>
   <taglib-uri>/tags/struts-logic</taglib-uri>
```

```
    <taglib-location>/WEB-INF/struts-logic.tld</taglib-
    location>
</taglib>
<taglib>
    <taglib-uri>/tags/struts-tiles</taglib-uri>
    <taglib-location>/WEB-INF/struts-tiles.tld</taglib-
    location>
</taglib>
```
This tells the JSP system where to find the tag library descriptor for this library (in your application's WEB-INF directory, instead of out on the Internet somewhere).

- **Configure the Struts Tag Libraries (Servlet 2.3)**
  Servlet 2.3 Users only: The Servlet 2.3 specification simplifies the deployment and configuration of tag libraries. The instructions above will work on older containers as well as 2.3 containers (Struts only requires a servlet 2.2 container); however, if you're using a 2.3 container such as Tomcat 4.x, you can take advantage of a simplified deployment.

  All that's required to install the struts tag libraries is to copy struts.jar into your /WEB-INF/lib directory and reference the tags in your code like this:
  ```
  <%@ taglib uri=http://struts.apache.org/tags-html
  prefix="html" %>
  ```

  Note that you **must use the full uri** defined in the various struts tlds so that the container knows where to find the tag's class files. You don't have to alter your web.xml file or copy tlds into any application directories.

- **Add Struts Components To Your Application**
  To use Struts, you must copy the .tld files that you require into your WEB-INF directory, and copy struts.jar (and all of the commons-*.jar files) into your WEB-INF/lib directory.

- **Struts Bean Tags**
  This tag library contains tags useful in accessing beans and their properties, as well as defining new beans (based on these accesses) that are accessible to the remainder of the page via scripting variables and page scope attributes. Convenient

mechanisms to create new beans based on the value of request cookies, headers, and parameters are also provided.

Many of the tags in this tag library will throw a JspException at runtime when they are utilized incorrectly (such as when you specify an invalid combination of tag attributes). JSP allows you to declare an "error page" in the <%@ page %> directive. If you wish to process the actual exception that caused the problem, it is passed to the error page as a request attribute under key org.apache.struts.action.EXCEPTION.

If you are viewing this page from within the Struts Documentation Application (or online at http://struts.apache.org/), you can learn more about using these tags in the Bean Tags Developer's Guide.

Tag	**Description**
cookie	Define a scripting variable based on the value(s) of the specified request cookie.
define	Define a scripting variable based on the value(s) of the specified bean property.
header	Load the response from a dynamic application request and make it available as a bean
include	Render an internationalized message string to the response.
message	Expose a specified item from the page context as a bean.
page	Define a scripting variable based on the value(s) of the specified request parameter.
parameter	Load a web application resource and make it available as a bean.
resource	Define a bean containing the number of elements in a Collection or Map.
size	Expose a named Struts internal configuration object as a bean.
struts	Render the value of the specified bean property to the current JspWriter.

Struts

The core of the Struts framework is a flexible control layer based on standard technologies like Java Servlets, JavaBeans, ResourceBundles, and XML, as well as various Jakarta Commons packages. Struts encourages application architectures based on the Model 2 approach, a variation of the classic Model-View-Controller (MVC) design paradigm.

Struts provides its own Controller component and integrates with other technologies to provide the Model and the View. For the Model, Struts can interact with standard data access technologies, like JDBC and EJB, as well as most any third-party packages, like Hibernate, iBATIS, or Object Relational Bridge. For the View, Struts works well with JavaServer Pages, including JSTL and JSF, as well as Velocity Templates, XSLT, and other presentation systems.

The Struts framework provides the invisible underpinnings every professional web application needs to survive. Struts helps you create an extensible development environment for your application, based on published standards and proven design patterns.

❖ **What is the difference between Struts 1.0 and Struts 1.1**

The new features added to Struts 1.1 are

22. RequestProcessor class
23. Method **perform()** replaced by **execute()** in Struts base Action Class
24. Changes to web.xml and struts-config.xml
25. Declarative exception handling
26. Dynamic ActionForms
27. Plug-ins
28. Multiple Application Modules
29. Nested Tags
30. The Struts Validator
31. Change to the ORO package
32. Change to Commons logging
33. Removal of Admin actions
34. Deprecation of the GenericDataSource

- **Explain Struts navigation flow**

  A client requests a path that matches the Action URI pattern. The container passes the request to the ActionServlet. If this is a modular application, the ActionServlet selects the appropriate module. The ActionServlet looks up the mapping for the path. If the mapping specifies a form bean, the ActionServlet sees if there is one already or creates one. If a form bean is in play, the ActionServlet resets and populates it from the HTTP request. If the mapping has the validate property set to true, it calls validate on the form bean. If it fails, the servlet forwards to the path specified by the input property and this control flow ends. If the mapping specifies an Action type, it is reused if it already exists or instantiated.

  The Action's perform or execute method is called and passed the instantiated form bean (or null). The Action may populate the form bean, call business objects, and do whatever else is needed. The Action returns an ActionForward to the ActionServlet. If the ActionForward is to another Action URI, we begin again; otherwise, it's off to a display page or some other resource. Most often, it is a JSP, in which case Jasper, or the equivalent (not Struts), renders the page.

- **What is the difference between ActionForm and DynaActionForm**

  In struts 1.0, action form is used to populate the html tags in jsp using struts custom tag.when the java code changes, the change in action class is needed. To avoid the chages in struts 1.1 dyna action form is introduced.This can be used to develop using xml.The dyna action form bloats up with the struts-config.xml based definetion.

- **What is DispatchAction**

  The DispatchAction class is used to group related actions into one class. DispatchAction is an abstract class, so you must override it to use it. It extends the Action class.

  It should be noted that you dont have to use the DispatchAction to group multiple actions into one Action class.

You could just use a hidden field that you inspect to delegate to member() methods inside of your action.

- ❖ **How to call ejb from Struts**
  use the Service Locator patter to look up the ejbs
  Or You can use InitialContext and get the home interface.

- ❖ **What are the various Struts tag libraries**
  struts-html tag library - used for creating dynamic HTML user interfaces and forms. **struts-bean tag library** - provides substantial enhancements to the basic capability provided by.
  struts-logic tag library - can manage conditional generation of output text, looping over object collections for repetitive generation of output text, and application flow management.
  struts-template tag library - contains tags that are useful in creating dynamic JSP templates for pages which share a common format.

- ❖ **What is the difference between ActionErrors and ActionMessages**
  The difference between the classes is zero - all behavior in ActionErrors was pushed up into ActionMessages and all behavior in ActionError was pushed up into ActionMessage. This was done in the attempt to clearly signal that these classes can be used to pass any kind of messages from the controller to the view - errors being only one kind of message.

- ❖ **How you will handle errors and exceptions using Struts**
  There are various ways to handle exception:
  1. To handle errors server side validation can be used using ActionErrors classes can be used.
  2. The exceptions can be wrapped across different layers to show a user showable exception.
  3. using validators

- ❖ **How you will save the data across different pages for a particular client request using Struts**
  Several ways. The similar to the ways session tracking is enabled. Using cookies, URL-rewriting, SSLSession, and possibilty threw in the database.

- **What we will define in Struts-config.xml file. And explain their purpose**

  The main control file in the Struts framework is the struts-config.xml XML file, where action mappings are specified. This file's structure is described by the struts-config DTD file, which is defined at http://jakarta.apache.org/struts/. A copy of the DTD can be found on the /docs/dtds subdirectory of the framework's installation root directory. The top-level element is struts-config. Basically, it consists of the following elements:

  data-sources - A set of data-source elements, describing parameters needed to instantiate JDBC 2.0 Standard Extension DataSource objects

  form-beans - A set of form-bean elements that describe the form beans that this application uses

  global-forwards - A set of forward elements describing general available forward URIs

  action-mappings - A set of action elements describing a request-to-action mapping

- **What is the purpose of tiles-def.xml file, resourcebundle.properties file, validation.xml file?**

  The Tiles Framework is an advanced version of that comes bundled with the Struts Webapp framework. Its purpose is reduce the duplication between jsp pages as well as make layouts flexible and easy to maintain. It integrates with Struts using the concept of named views or definitions.

- **What is Action Class. What are the methods in Action class?**

  Action class is request handler in Struts. we will extend the Action class and over ride the execute() method in which we will specify the business logic to be performed.

- **Explain about token feature in Struts?**

  Tokens are used to check for invalid path for by the uer:

  1. if the user presses back button and submits the same page
  2. or if the user refreshes the page which will result to the resubmit of the previous action and might lead to unstabality..

  to solve the abv probs we use tokens

  1. in previous action type saveTokens(HttpServletreuest)

2. in current action check for duplication but if(!isValidToken())

❖ **What part of MVC does Struts represent?**
Bad question. Struts is a framework which supports the MVC pattern.

❖ **What are the core classes of struts?**
The core classes of struts are ActionForm, Action, ActionMapping, ActionForward etc.

❖ **What are the Important Components of Struts?**
1. Action Servlet
2. Action Classes
3. Action Form
4. Validator Framework
5. Message Resources
6. Struts Configuration XML Files
7. View components like JSP

❖ **What is Struts?**
Struts is a web page development framework and an open source software that helps developers build web applications quickly and easily. Struts combines Java Servlets, Java Server Pages, custom tags, and message resources into a unified framework. It is a cooperative, synergistic platform, suitable for development teams, independent developers, and everyone between.

❖ **How is the MVC design pattern used in Struts framework?**
In the MVC design pattern, application flow is mediated by a central Controller. The Controller delegates requests to an appropriate handler. The handlers are tied to a Model, and each handler acts as an adapter between the request and the Model. The Model represents, or encapsulates, an application's business logic or state. Control is usually then forwarded back through the Controller to the appropriate View. The forwarding can be determined by consulting a set of mappings, usually loaded from a database or configuration file. This provides a loose coupling between the View and Model, which can make an application significantly easier to create and maintain.

Controller - Servlet controller which supplied by Struts itself;
View - what you can see on the screen, a JSP page and presentation components;
Model - System state and a business logic JavaBeans.

❖ **Who makes the Struts?**
Struts is hosted by the Apache Software Foundation(ASF) as part of its Jakarta project, like Tomcat, Ant and Velocity.

❖ **Why it called Struts?**
Because the designers want to remind us of the invisible underpinnings that hold up our houses, buildings, bridges, and ourselves when we are on stilts. This excellent description of Struts reflect the role the Struts plays in developing web applications.

❖ **Do we need to pay the Struts if being used in commercial purpose?**
No. Struts is available for commercial use at no charge under the Apache Software License. You can also integrate the Struts components into your own framework just as if they were writtern in house without any red tape, fees, or other hassles.

❖ **What are the core classes of Struts?**
Action, ActionForm, ActionServlet, ActionMapping, ActionForward are basic classes of Structs.

❖ **What is the design role played by Struts?**
The role played by Structs is controller in Model/View/Controller(MVC) style. The View is played by JSP and Model is played by JDBC or generic data source classes. The Struts controller is a set of programmable components that allow developers to define exactly how the application interacts with the user.

❖ **How Struts control data flow?**
Struts implements the MVC/Layers pattern through the use of ActionForwards and ActionMappings to keep control-flow decisions out of presentation layer.

❖ **What configuration files are used in Struts?**

- ApplicationResourcesl.properties
- struts-config.xml

These two files are used to bridge the gap between the Controller and the Model.

❖ **What helpers in the form of JSP pages are provided in Struts framework?**

- struts-html.tld
- struts-bean.tld
- struts-logic.tld

❖ **Is Struts efficient?**

- The Struts is not only thread-safe but thread-dependent(instantiates each Action once and allows other requests to be threaded through the original object.
- ActionForm beans minimize subclass code and shorten subclass hierarchies
- The Struts tag libraries provide general-purpose functionality
- The Struts components are reusable by the application
- The Struts localization strategies reduce the need for redundant JSPs
- The Struts is designed with an open architecture
- subclass available
- The Struts is lightweight (5 core packages, 5 tag libraries)
- The Struts is open source and well documented (code to be examined easily)
- The Struts is model neutral

❖ **What is Jakarta Struts Framework?**

Jakarta Struts is open source implementation of MVC (Model-View-Controller) pattern for the development of web based applications. Jakarta Struts is robust architecture and can be used for the development of application of any size. Struts framework makes it much easier to design scalable, reliable Web applications with Java.

❖ **What is ActionServlet?**

The class org.apache.struts.action.ActionServlet is the called the ActionServlet. In the the Jakarta Struts Framework this class plays the role of controller. All the requests to the server goes through the controller. Controller is responsible for handling all the requests.

- **How you will make available any Message Resources Definitions file to the Struts Framework Environment?**
  Message Resources Definitions file are simple .properties files and these files contains the messages that can be used in the struts project. Message Resources Definitions files can be added to the struts-config.xml file through *<message-resources />* tag.
  Example:
  ```
  <message-resources parameter="MessageResources" />
  ```

- **What is Action Class?**
  The Action Class is part of the Model and is a wrapper around the business logic. The purpose of Action Class is to translate the HttpServletRequest to the business logic. To use the Action, we need to Subclass and overwrite the execute() method. In the Action Class all the database/business processing are done. It is advisable to perform all the database related stuffs in the Action Class. The ActionServlet (commad) passes the parameterized class to Action Form using the execute() method. The return type of the execute method is ActionForward which is used by the Struts Framework to forward the request to the file as per the value of the returned ActionForward object.

- **Write code of any Action Class?**
  Here is the code of Action Class that returns the ActionForward object.
  ```
  import javax.servlet.http.HttpServletRequest;
  import javax.servlet.http.HttpServletResponse;
  import org.apache.struts.action.Action;
  import org.apache.struts.action.ActionForm;
  import org.apache.struts.action.ActionForward;
  import org.apache.struts.action.ActionMapping;
  public class TestAction extends Action {
      public ActionForward execute (
          ActionMapping mapping,
          ActionForm form,
          HttpServletRequest request,
  ```

```
        HttpServletResponse response) throws Exception {
        return mapping.findForward(\"testAction\");
    }
}
```

❖ **What is ActionForm?**

An ActionForm is a JavaBean that extends org.apache.struts.action.ActionForm. ActionForm maintains the session state for web application and the ActionForm object is automatically populated on the server side with data entered from a form on the client side.

❖ **What is Struts Validator Framework?**

Struts Framework provides the functionality to validate the form data. It can be use to validate the data on the users browser as well as on the server side. Struts Framework emits the java scripts and it can be used validate the form data on the client browser. Server side validation of form can be accomplished by sub classing your From Bean with DynaValidatorForm class. The Validator framework was developed by David Winterfeldt as third-party add-on to Struts. Now the Validator framework is a part of Jakarta Commons project and it can be used with or without Struts. The Validator framework comes integrated with the Struts Framework and can be used without doing any extra settings.

❖ **Give the Details of XML files used in Validator Framework?**

The Validator Framework uses two XML configuration files validator-rules.xml and validation.xml. The validator-rules.xml defines the standard validation routines, these are reusable and used in validation.xml. to define the form specific validations. The validation.xml defines the validations applied to a form bean.

❖ **How you will display validation fail errors on jsp page?**

The following tag displays all the errors:

```
<html:errors/>
```

How you will enable front-end validation based on the xml in validation.xml?

The <html:javascript> tag to allow front-end validation based on the xml in validation.xml.

Example:

```
<html:javascript formName=”logonForm” dynamicJavascript=”true” staticJavascript=”true” />
```

generates the client side java script for the form “logonForm” as defined in the validation.xml file. The <html:javascript> when added in the jsp file generates the client site validation script.

EJB Enterprise Java Beans

❖ Agenda

- What is an EJB
- Bean Basics
- Component Contract
- Bean Varieties
  – Session Beans
  – Entity Beans
  – Message Driven Beans

❖ What is an EJB ?

Bean is a component

- A server-side component
- Contains business logic that operates on some temporary data or permanent database
- Is customizable to the target environment
- Is re-usable
- Is truly platform-independent

❖ So, what is an EJB?

- Ready-to-use Java component
  – Being Java implies portability, inter-operability
- Can be assembled into a distributed multi-tier application
- Handles threading, transactions
- Manages state and resources
- Simplifies the development of complex enterprise applications

❖ Benefits…

- Pure Java implies portability
  – exchange components without giving away the source.
- Provides interoperability
  – assemble components from anywhere, can all work together.

❖ Operational Benefits from EJB

- Transaction management service
- Distributed Transaction support
- Portability
- Scalability

- Integration with CORBA possible
- Support from multiple vendors

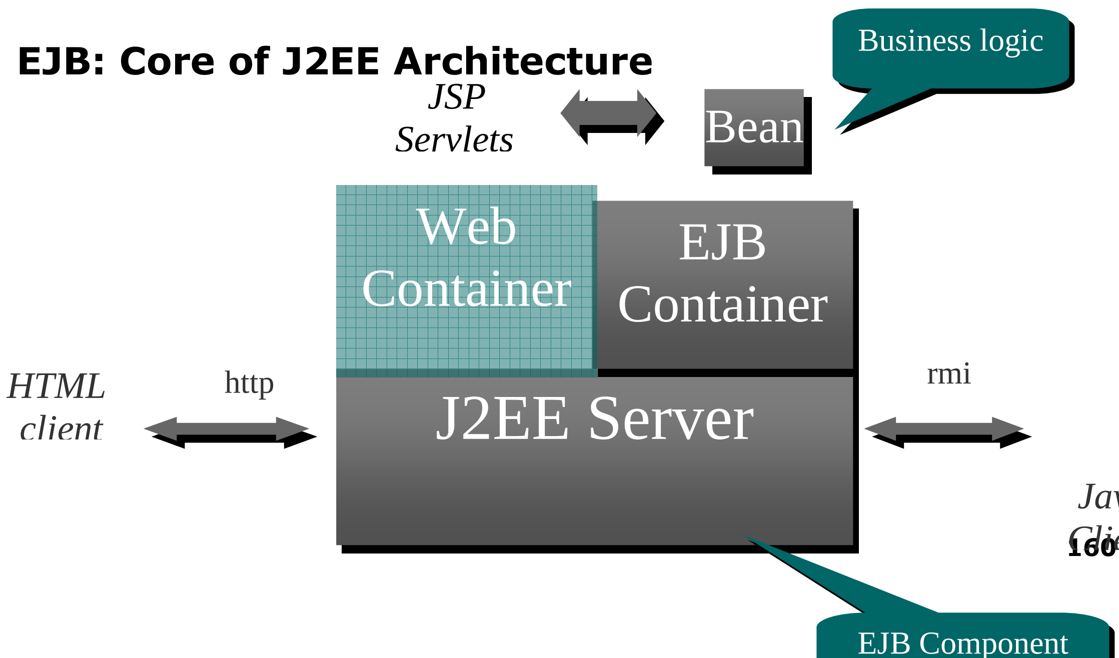
❖ **What Does EJB Really Define?**
- Component architecture
- Specification to write components using Java
- Specification to "component server developers"
- Contract between developer roles in a components-based application project

The basis of components used in distributed transaction-oriented enterprise applications.

❖ **The Componentized Application:**
- Application now consists of several re-usable components.
- **Instances** of components created at run-time for a client.
- Common object for all instances of the component, usually called the Factory Object
  – EJB calls it "Home Object"
- Common place where client can locate this **Home Object**
- Objects located from a remote client through **JNDI** (Java Naming and Directory Interface) service.

❖ **Application Server provides…**
- JNDI based naming service
- Implementation of Bean, Home and Remote
- Complete Life Cycle Management
- Resource pooling - beans, db connections, threads...
- Object persistence
- Transaction management
- Secured access to beans
- Scalability and availability

❖ **EJB: Core of J2EE Architecture**



❖ **The Architecture Scenario**

Application Responsibilities
- Create individual business and web components.
- Assemble these components into an application.
- Deploy application on an application server.
- Run application on target environment.

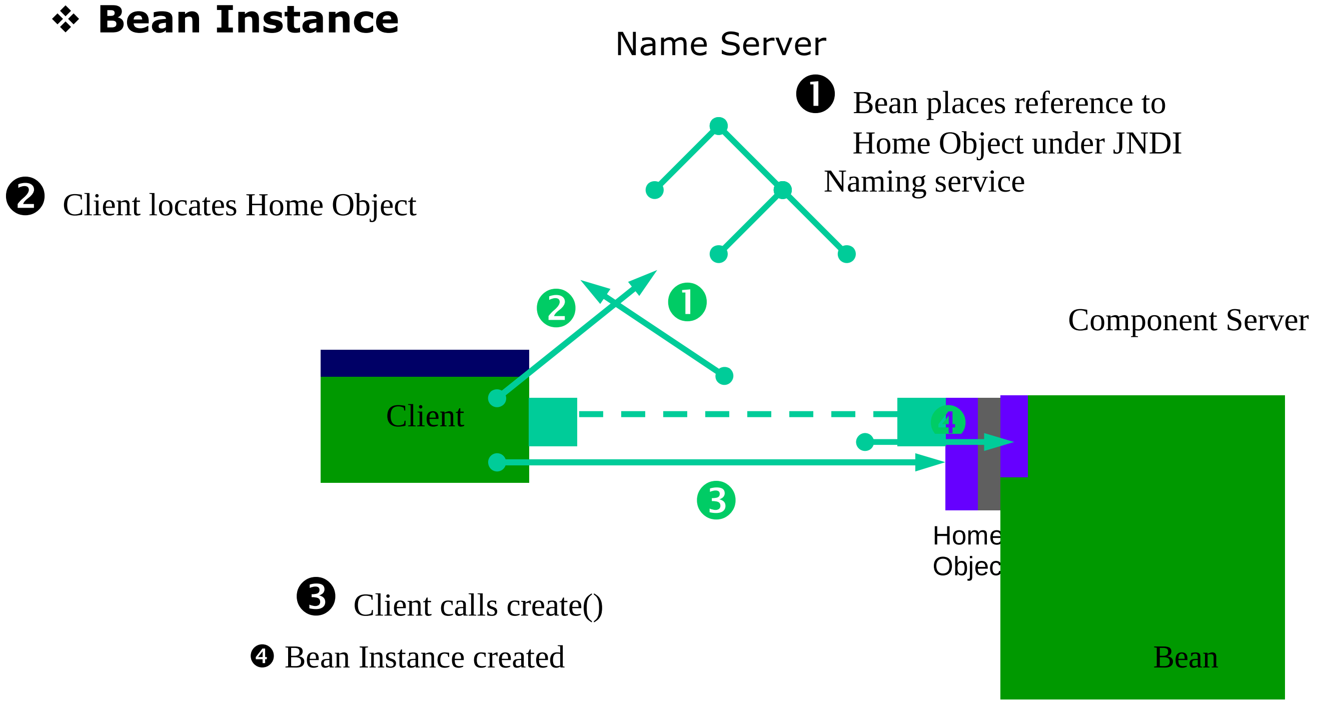
❖ **EJB Architecture Roles:** Appointed for Responsibilities

- Six roles in application development and deployment life cycle
  – Bean Provider
  – Application Assembler
  – Server Provider
  – Container Provider
  – Deployer
  – System Administrator
- Each role performed by a different party.
- Product of one role compatible with another.

Creating the Bean Instance

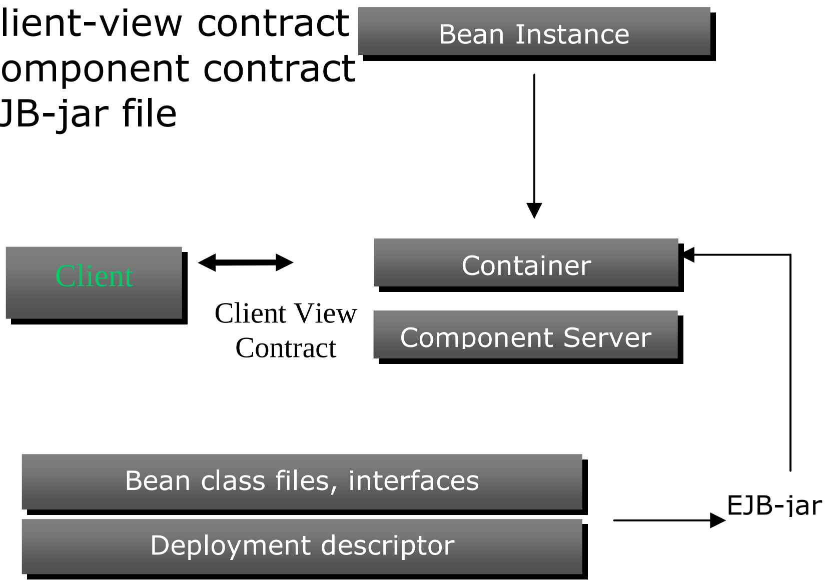
- Look up for the Home Object through JNDI
- Get the reference
- Call **create(…) method**
- The server generates the code for remote access using **RMI** (Remote Method Invocation).
- The RMI code in the form of stub and skeleton:
  – establishes connection,
  – marshals/unmarshals
  – places remote method calls

❖ Bean Instance



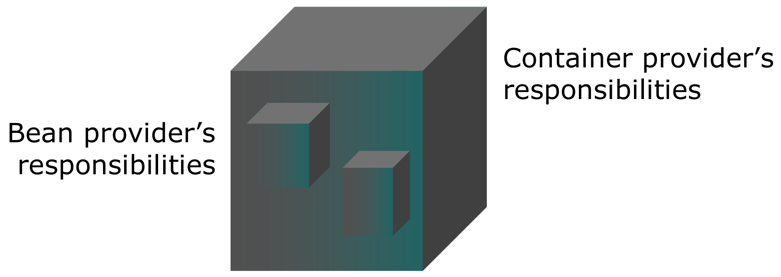
❖ Component Contract :

- Client-view contract
- Component contract
- EJB-jar file



❖ **Client-view contract:**
- Contract between client and container
- Uniform application development model for greater re-use of components
- View sharing by local and remote programs
- The Client can be:
  – another EJB deployed in same or another container
  – a Java program, an applet or a Servlet
  – mapped to non-Java clients like CORBA clients

❖ **Component contract:**
- Between an EJB and the container it is hosted by
- This contract needs responsibilities to be shared by:
  – the bean provider
  – the container provider



❖ **Bean provider's responsibility:**
- Implement business methods in the bean
- Implement *ejbCreate*, *ejbPostCreate* and *ejbRemove* methods, and *ejbFind* method (in the case of bean managed persistence)
- Define home and remote interfaces of the bean

- Implement container callbacks defined in the *javax.ejb.Session* bean interface
  – optionally the *javax.ejb.SessionSynchronization* interface
- Implement container callbacks defined in javax.ejb.EntityBean interfaces for entities
- Avoid programming practices that interfere with container's runtime management of bean instances

❖ **Container provider's responsibility:**

- Delegate client method invocations to the business methods
- Invoke appropriate methods during an EJB object creation, removal and lookup
- Provide classes that implement the home and remote interfaces of the bean
- Invoke *javax.ejb.SessionBean* interface and SessionSynchronization interface callbacks at appropriate times
- Invoke *javax.ejb.EntityBean* interface for entities and callbacks at appropriate times
- Implement persistence for entity beans with container managed persistence
- Provide *javax.ejb.SessionContext* and *javax.ejb.EntityContext* for session and entity bean instances, obtain the information from container
- Provide JNDI context with the bean's environment to the bean instances
- Manage transaction, security and exception for beans

❖ **Ejb-jar file**

- Standard format used by EJB tools for packaging (assembling) beans along with declarative information
- Contract between bean provider and application assembler, and between application assembler and application deployer
  The file includes:
  – Java class files of the beans
  – Finally, the Big Picture

❖ **Bean Varieties**

Three Types of Beans:
Session Beans: Short lived and last during a session.
Entity Beans: Long lived and persist throughout.

Message Driven Beans: Asynchronous Message Consumers Asynchronous.

❖ **Session Beans**
- A session object is a non-persistent object that implements some business logic running on the server.
- Executes on behalf of a single client.
- Can be transaction aware.
- Does not represent directly shared data in the database, although it may access and update such data.
- Is relatively short-lived.
- Is removed when the EJB container crashes. The client has to re-establish a new session object to continue computation

❖ **Types of Session Beans**
- There are two types of session beans:
  – Stateless
  – Stateful
  Message Consumers

❖ **Client's view of a Session Bean:**
- A client accesses a session object through the session bean's Remote Interface or Local Interface.
- Each session object has an identity which, in general, does not survive a crash

  Locating a session bean's home interface
  - Remote Home interface
    ```
    Context initialContext = new InitialContext();
    CartHome cartHome = (CartHome)
    javax.rmi.PortableRemoteObject.narrow(initialContext.loo
    kup("java:comp/env/ejb/cart"), CartHome.class);
    ```
  - Local Home Interface
    ```
    Context initialContext = new InitialContext();
    CartHome cartHome = (CartHome)
    initialContext.lookup("java:comp/env/ejb/cart");
    ```

❖ **JNDI:** used to locate Remote Objects created by bean.
portableRemoteObject Class: It uses an Object return by Lookup().
narrow() -> Call the create() of HomeInterface.
IntialContext Class:

Lookup() -> Searches and locate the distributed Objects.

❖ **Session Bean's Local Home Interface:**

- object that implements is called a session EJBLocalHome object.
- Create a new session object.
- Remove a session object.

❖ **Session Bean's Remote Home Interface**

- object that implements is called a session EJBHome object.
- Create a session object
- Remove a session object

❖ **Session Bean's Local Interface**

- Instances of a session bean's remote interface are called session EJBObjects
- business logic methods of the object.

❖ **Session Bean's Local Home Interface**

- instances of a session bean's local interface are called session EJBLocalObjects
- business logic methods of the object
- Creating an EJB Object
- Home Interface defines one or more create() methods
- Arguments of the create methods are typically used to initialize the state of the created session object

```
public interface CartHome extends javax.ejb.EJBHome {
    Cart create(String customerName, String account) throws
        RemoteException, BadAccountException,
        CreateException;
}
cartHome.create("John", "7506");
```

❖ **EJBObject or EJBLocalObject**

- Client never directly accesses instances of a Session Bean's class
- Client uses Session Bean's Remote Interface or Remote Home Interface to access its instance
- The class that implements the Session Bean's Remote Interface or Remote Home Interface is provided by the container.

❖ **Session Object Identity**

- Session Objects are meant to be private resources of the client that created them
- Session Objects, from the client's perspective, appear anonymous
- Session Bean's Home Interface must not define finder methods

Stateful Session Beans:

– A stateful session object has a unique identity that is assigned by the container at the time of creation.

– A client can determine if two object references refer to the same session object by invoking the isIdentical(EJBObject otherEJBObject) method on one of the references.

Stateless Session Beans:

– All session objects of the same stateless session bean, within the same home have the same object identity assigned by the container.

– isIdentical(EJBObject otherEJBObject) method always returns true.

❖ **Container Responsibilities:**

- Manages the lifecycle of session bean instances.
- Notifies instances when bean action may be necessary .
- Provides necessary services to ensure session bean implementation is scalable and can support several clients.

❖ **Activation and Passivation:**

- Session bean container may temporarily transfer state of an idle stateful session bean instance to some form of secondary storage.
- Transfer from working set to secondary storage is called **instance passivation**.
- Transfer back from the secondary storage to the instance variables is called **instance activation**.

❖ **Entity Beans**

Long Live Entity Beans!

- A component that represents an object-oriented view of some entities stored in a persistent storage like a database or an enterprise application.

- From its creation until its destruction, an entity object lives in a container.
- Transparent to the client, the Container provides security, concurrency, transactions, persistence, and other services to support the Entity Bean's functioning
  – Cainer Managed Persistence versus Bean Managed Persistence
- Multiple clients can access an entity object concurrently
- Container hosting the Entity Bean synchronizes access to the entity object's state using transactions
- Each entity object has an identity which usually survives a transaction crash
- Object identity is implemented by the container with help from the enterprise bean class
- Multiple enterprise beans can be deployed in a Container

❖ **Remote Clients:**
- Accesses an entity bean through the entity bean's remote and remote home interfaces
- Implements EJBObject and EJBHome Interfaces
- Location Independent
- Potentially Expensive, Network Latency
- Useful for coarse grained component access

❖ **Local Clients:**
- Local client is a client that is collocated with the entity bean and which may be tightly coupled to the bean.
- Implements EJBLocalObject and EJBLocalHome Interfaces
- Same JVM
- Enterprise bean can-not be deployed on a node different from that of its client – Restricts distribution of components.
- Better supports fine-grained component access

❖ **Locating the Entity Bean:**
- Location of EJB Container is usually transparent to Client
- Client locates Entity Bean's Home Interface using JNDI

Example

```
Context initialContext = new InitialContext();
AccountHome accountHome = (AccountHome)
initialContext.lookup("java:comp/env/ejb/accounts");
```

Entity Bean's Remote Home Interface

- Container provides the implementation of the Remote Home Interface for each Entity Bean deployed in the container
- Container makes the Remote Home Interface of all Entity Beans deployed in it accessible to Clients through JNDI
- The object that implements an Entity Bean’s Remote Home Interface is called an EJBHome object

Entity’ Bean’s Remote Home Interface

- Create new entity objects within the home
- Find existing entity objects within the home
- Remove an entity object from the home

❖ **Create Methods:**

- Entity Bean’s Remote Home Interface can define multiple create() methods, each defining a way of creating an entity object
- Arguments of create() initialize the state of the entity object
- Return type of a create() method is Entity Bean’s Remote Interface
- The throws clause of every create() method includes the java.rmi.RemoteException and javax.ejb.CreateException

❖ **finder Methods**

- Entity Bean’s Home Interface defines many finder methods
- Name of each finder method starts with the prefix “find”
- Arguments of a finder method are used by the Entity Bean implementation to locate requested entity objects
- Return type of a finder method must be the Entity Bean’s Remote Interface, or a collection of Remote Interfaces
- The throws clause of every finder method includes the java.rmi.RemoteException and javax.ejb.FinderException

❖ **Entity Bean’s Remote Interface**

- Client accesses an entity object through Entity Bean’s Remote Interface
- Entity bean’s Remote Interface must extend javax.ejb.EJBObject interface
- Remote Interface defines business methods which are callable by clients
- The container provides the implementation of the methods defined in the javax.ejb.EJBObject interface

- Only business methods are delegated to the instances of the enterprise bean class

❖ **Entity Bean's Local Home Interface**
- must extend the javax.ejb.EJBLocalHome interface
- Each method must be one of the:
  – Create methods
  – Find methods
  – Home methods

❖ **Entity Bean's Local Interface**
- Local client can access an entity object through the entity bean's local interface.
- must extend the javax.ejb.EJBLocalObject interface.
- defines the business methods callable by local clients.

❖ **Persistence Management**
- Data access protocol for transferring state of the entity between the Entity Bean instances and the database is referred to as object persistence
- There are two ways to manage this persistence during an application's lifetime
  – Bean-managed
  – Container-managed

❖ **Bean Managed Persistence:**
- Entity Bean provider writes database access calls directly into the enterprise bean class Container Managed Persistence
- Bean Provider need not write database calls in the bean
- Container provider's tools generate database access calls at deployment time

Advantage: Entity Bean can be mostly independent from the data source in which the entity is stored

Disadvantage: Sophisticated tools are needed at deployment time to map Entity Bean fields to data source

❖ **EJB QL**
- Need for standardizing queries
- Why not SQL?
- EJB QL: EJB Query Language

– Specification language
– Based on the CMP Data Model (Abstract Persistence Schema)
– Compiled to a target language: SQL

EJB QL Example

- SELECT OBJECT(l) From OrderBean o, in(o.lineItems) l
  SELECT l.LINEITEM_ID FROM LINEITEMEJBTABLE l, ORDEREJBTABLE o WHERE (l.ORDER_ID = o.ORDER_ID)
- SELECT OBJECT(o) FROM OrderBean o WHERE o.creditCard.expires = '03/05'"
  SELECT o.ORDER_ID FROM CREDITCARDEJBTABLE a_1, ORDEREJBTABLE o WHERE ((a_1.EXPIRES='03/05' AND o.CREDITCARD_ID = a_1.CREDITCARD_ID))

❖ **EJB QL Example**

```
SELECT c.address
FROM CustomerBeanSchema c
WHERE c.iD=?1 AND c.firstName=?2

SELECT  ADDRESS.ID
FROM  ADDRESS, CUSTOMER
WHERE  CUSTOMER.CUSTOMERID=?
    AND CUSTOMER.FIRSTNAME=?
    AND CUSTOMER.CUSTOMERID = ADDRESS.CUSTOMERID
```

❖ **EJB QL: Deployment Descriptor**

```
<query>
    <description>Method finds large orders</description>
    <query-method>
        <method-name>findAllCustomers</method-name>
        <method-params/>
    </query-method>
    <ejb-ql>SELECT OBJECT(c) FROM CustomerBeanSchema
    c</ejb-ql>
</query>
```

❖ **Home Business Methods**

- Methods in the Home Interface
- Implementation provided by Bean Provider with matching ejbHome<method> in the Bean
- Exposed to the Client View

- Not specific to any Bean instance

Select Methods

- Defined as abstract method in the Bean class
  – ejbSelect<method>
- Special type of a query method
- Specified using a EJB QL statement
- Not exposed to the Client View
- Usually called from a business method

❖ **Example of EJB 1.1 CMP Bean**

```
public class AccountBean implements EntityBean {
    // Bean Instance Variables
    public long account_number;
    public java.lang.String customer_name;
    public double balance;
    // Business Methods
    public void credit ( double amount ) {
        balance += amount;
    }
    public void debit ( double amount ) {
        balance -= amount;
    }
}
```

❖ **Example of EJB 2.0 CMP Bean**

```
public abstract class AccountBean implements EntityBean {
    // Virtual Fields
    public abstract long getAccount_number();
    public abstract void setAccount_number(long
    account_number);
    public abstract java.lang.String getCustomer_name();
    public abstract void setCustomer_name(String
    customer_name);
    public abstract double getBalance();
    public abstract void  setBalance(double  balance);
    // Business Method
    public void credit (double amount) {
        double balance = getBalance();
        balance += amount;
        setBalance(balance);
    }
}
```

❖ **Abstract Schema : Deployment Descriptor**

```
<abstract-schema-name>CustomerBeanSchema</abstract-schema-name>
<cmp-field>
    <description>id of the customer</description>
    <field-name>iD</field-name>
</cmp-field>
<cmp-field>
    <description>First name of the customer</description>
    <field-name>firstName</field-name>
</cmp-field>
<cmp-field>
    <description>Last name of the customer</description>
    <field-name>lastName</field-name>
</cmp-field>
<primkey-field>iD</primkey-field>
```

❖ **Container Managed Relationships:**

- Container Managed Relationships <cmr-field>
- Bean-Bean, Bean-Dependent, Dependent-Dependent
- Defined using Abstract Accessor Methods
- Unidirectional or Bi-directional
  – LineItem à Product
  – Student àß Course
- Cardinality
  – One to One
  – One to Many
  – Many to One
  – Many to Many

❖ **Example Entity Bean: Order**

```
public abstract OrderBean extends Entity Bean {
    // Virtual Fileds <cmp-fields>
    public abstract Long getOrderID();
    public abstract void setOrderID(Long orderID);
    // Virtual Fields <cmr-fields>
    public abstract Address getShipingAddress();
    public abstract void setShipingAddress (Address address);
    public abstract Collection getLineItems();
    public abstract void setLineItems (Collection lineItems);
}
```

❖ **Example Entity Bean: Product**

```
public abstract OrderBean extends Entity Bean {
    // Virtual Fields <cmp-field>
    public abstract Long getProductID();
    public abstract void setProductID(Long orderID);
    // Virtual Fields <cmp-field>
    public abstract String getProductCategory();
    public abstract void setProductCategory (String  category);
    // NO – Relationship Fields
}
```

❖ **Relationships: Deployment Descriptor**

```
<ejb-relation>
    <description>ONE-TO-ONE:            Customer            and
    Address</description>
    <ejb-relation-name>Customer-Address</ejb-relation-
    name>
    <ejb-relationship-role>
        <ejb-relationship-role-name>customer       has       one
        addresss</ejb-relationship-role-name>
        <multiplicity>one</multiplicity>
        <relationship-role-source>
            <ejb-name>CustomerBean</ejb-name>
        </relationship-role-source>
        <cmr-field>
            <cmr-field-name>address</cmr-field-name>
        </cmr-field>
    </ejb-relationship-role>
    <ejb-relationship-role>
        <ejb-relationship-role-name>Address   belong   to   the
        Customer</ejb-relationship-role-name>
        <multiplicity>one</multiplicity>
        <cascade-delete/>
        <relationship-role-source>
            <ejb-name>AddressBean</ejb-name>
        </relationship-role-source>
    </ejb-relationship-role>
</ejb-relation>
```

EJB

❖ **What is the difference between normal Java object and EJB?**
Java Object: It's a reusable componet
EJB: is reusable and deployable component which can be deployed in any container. EJB is a distributed component used to develop business applications. Container provides runtime environment for EJBs. EJB is an Java object implemented according EJB specification. Deployability is a feature.

❖ **What is the difference between JavaBean and EJB?**
Java Beans: is intra-process component, JavaBeans is particularly well-suited for asynchronous, intra-application communications among software.
EJB: is an Inter-Process component.

❖ **What is EJB?**
Enterprise Java Bean is a specification for server-side scalable,transactional and multi-user secure enterprise-level applications. It provides a consistant component architecture for creating distributed n-tier middleware.

Enterprise JavaBeans (EJB) is a technology that based on J2EE platform. EJBs are server-side components. EJB are used to develop the distributed, transactional and secure applications based on Java technology.

❖ **What is Session Bean. What are the various types of Session Bean?**
SessionBeans: They are usually associated with one client. Each session bean is created and destroyed by the particular EJB client that is associated with it. These beans do not survive after system shutdown.
These Session Beans are of two types:
a) **Stateful Session Beans:** They maintain conversational state between subsequest calls by a client
b) **Stateful Session Beans:** These beans have internal states. They can be stored (getHandle()) and restored (getEJBObject()) across client sessions.Since they can be

persistence, they are also called as Persistence Session Beans.

a) **Stateless Session Bean:** Consider this as a servlet equivalent in EJB. It is just used to service clients regardless of state and does not maintain any state.
b) **Stateless Session Beans:** These beans do not have internal States. They need not be passivated. They can be pooled into service multiple clients.

❖ **What is the difference between Stateful session bean and Stateless session bean?**

Stateful:

- Stateful s.Beans have the passivated and Active state which the Stateless bean does not have.
- Stateful beans are also Persistent session beans. They are designed to service business processes that span multiple method requests or transactions.
- Stateful session beans remembers the previous requests and reponses.
- Stateful session beans does not have pooling concept.
- Stateful Session Beans can retain their state on behave of an individual client.
- Stateful Session Beans can be passivated and reuses them for many clients.
- Stateful Session Bean has higher performance over stateless sessiob bean as they are pooled by the application server.

Stateless:

- Stateless Session Beans are designed to service business process that last only for a single method call or request.
- Stateless session beans do not remember the previous request and responses.
- Stattless session bean instances are pooled.
- Stateless Session Beans donot maintain states.
- Stateless Session Beans, client specific data has to be pushed to the bean for each method invocation which result in increase of network traffic.

❖ **What is the life cycle of Stateful session bean?**

Stateful Session Bean has three states. Does not exists, Method Ready and Passivated states.

Like Stateless beans, when the Stateful Session Bean hasnt been instantiated yet (so it is not an instance in memory) is it in the Does not exists state.

Once a container creates one or more instances of a Stateful Session Bean it sets them in a Method Ready state. In this state it can serve requests from its clients. Like Stateless Session Beans, a new instance is created (Class.newInstance()), the context is passed (setSessionContext()) and finally the bean is created with ejbCreate().

During the life of a Stateful Session Bean, there are periods of inactivity. In these periods, the container can set the bean to the Passivate state. This happens through the ejbPassivate() method. From the Passivate state the bean can be moved back to the Method Ready state, via ejbActivate() method, or can go directly to the Does Not Exists state with ejbRemove().

❖ **What is the life cycle of Stateless session bean?**
Stateless session bean has only two states: Does Not Exists and Method Ready Pool.
A bean has not yet instantiated (so it is not an instance in memory) when it is in the Does Not Exists state.

When the EJB container needs one or more beans, it creates and set them in the Method Ready Pool state. This happens through the creation of a new instance (Class.newInstance()), then it is set its context (setSessionContext()) and finally calls the ejbCreate() method.

The ejbRemove() method is called to move a bean from the Method Ready Pool back to Does Not Exists state.

❖ **What are the call back methods in Session bean?**
Session bean callback methods differ whether it is Stateless or stateful Session bean. Here they are.
Stateless Session Bean:
1. setSessionContext()
2. ejbCreate()
3. ejbRemove()

Stateful Session Bean:

1. setSessionContext()
2. ejbCreate()
3. ejbPassivate()
4. ejbActivate()
5. ejbRemove()

- **When you will chose Stateful session bean and Stateless session bean?**

  Stateful session bean is used when we need to maintain the client state. Example of statefull session is Shoping cart site where we need to maintain the client state.

  Stateless session bean will not have a client state it will be in pool.

  To maintain the state of the bean we prefer stateful session bean and example is to get mini statement in

  ATM we need sessions to be maintained.

- **What is Entity Bean. What are the various types of Entity Bean?**

  Entity bean represents the real data which is stored in the persistent storage like Database or file system. For example, There is a table in Database called Credit_card. This table contains credit_card_no,first_name, last_name, ssn as colums and there are 100 rows in the table. Here each row is represented by one instance of the entity bean and it is found by an unique key (primary key) credit_card_no.

  There are two types of entity beans.

  1. Container Managed Persistence(CMP)
  2. Bean Managed Presistence(BMP)-

- **What is the difference between CMP and BMP?**

  CMP means Container Managed Persistence. When we write CMP bean , we dont need to write any JDBC code to connect to Database. The container will take care of connection our enitty beans fields with database. The Container manages the persistence of the bean. Absolutely no database access code is written inside the bean class.

BMP means Bean Managed Persistence. When we write BMP bean, it is programmer responsiblity to write JDBC code to connect to Database.

❖ **What is the lifecycle of Entity Bean?**

The following steps describe the life cycle of an entity bean instance

An entity bean instances life starts when the container creates the instance using newInstance and then initialises it using setEntityContext.

The instance enters the pool of available instances. Each entity bean has its own pool. While the instance is in the available pool, the instance is not associated with any particular entity object identity. Any of these pooled instances may be used to execute finder (ejbFind) or home (ejbHome) methods.

An instance transitions from the pooled state to the ready state when the container selects that instance to service a client call to an entity object. There are two possible transitions from the pooled to the ready state: through the creation of an entity (ejbCreate and ejbPostCreate) or through the activation of an entity (ejbActivate).

When an entity bean instance is in the ready state, the instance is associated with a specific entity object identity. While the instance is in the ready state, the container can synchronize the instance with its representation in the underlying data source whenever it determines the need to using ejbLoad and ejbStore methods. Business methods can also be invoked zero or more times on an instance. An ejbSelect method can be called by a business method, ejbLoad or ejbStore method.

The container can choose to passivate an entity bean instance within a transaction. To passivate an instance, the container first invokes the ejbStore method to allow the instance to prepare itself for the synchronization of the database state with the instance’s state, and then the container invokes the ejbPassivate method to return the instance to the pooled state.

There are three possible transitions from the ready to the pooled state: through the ejbPassivate method, through the ejbRemove method (when the entity is removed), and because of a transaction rollback for ejbCreate, ejbPostCreate,or ejbRemove.

The container can remove an instance in the pool by calling the unsetEntityContext() method on the instance.

❖ **What are the call back methods in Entity bean?**
Entity Bean:
1. setEntityContext()
2. ejbCreate()
3. ejbPostCreate()
4. ejbActivate()
5. ejbPassivate()
6. ejbRemove()
7. unsetEntityContext()

❖ **When you will choose CMP and BMP?**
BMP
- Bean managed persistence
- Developer has to write persistence code for ejbLoad(),ejbStore() for entity beans
- Should follow this approach only when its bare necessity to write your own persistence logic.Usually container managed persistence is quite sufficient and less error prone.

CMP
- Container managed persistence
- Developer maps the bean fields with the database fields in the deployment descriptors.
- Developer need not provide persistence logic (JDBC) within the bean class.
- Containiner manages the bean field to DB field synchronization.

The point is only ENTITY beans can have theier pesristence mechanism as CMP or BMP. Session beans, which usually contains workflow or business logic should never have persistence code.Incase you choose to write persistence within your session bean, its usefull to note that the persistence is managed by the container BMP.Session beans cannot be CMP and its not possibel to provide field mapping for session bean.

BMPs are much harder to develop and maintain than CMPs.All other things are being equal,choose CMPs over BMPs for pure maintainability.

There are limitations in the types of the data sources that may be supported for CMPs by a container provide.Support for non JDBC type data sources,such as CICS,are not supported by the current CMP mapping and invocation schema.Therefore accessing these would require a BMP.

Complex queries might not be possible with the basic EJBQL for CMPs.So prefer BMPs for complex queries.

If relations between entity beans are established then CMPs may be necessary.CMR has ability to define manage relationships between entity beans.
Container will tyr to optimize the SQL code for the CMPs,so they may be scalable entity beans than the BMPs.

BMPs may be inappropriate for larger and more performance sesitive applications.

❖ **What are advantages and disadvantages of CMP and BMP?**
CMP: Container managed persistence
Advantages:
1. Easy to develop and maintain.
2. Relationships can be maintained between different entities.
3. Optimization of SQL code will be done.
4. Larger and more performance applications can be done.
Disadvantages:
1. Will not support for some nonJDBC data sources,i.e,CICS.
2. Complex queries cannot be developed with EJBQL.

BMP: Bean managed persistence
Advantages:
1. Support for nonJDBC data sources.

2. Complex queries can be build.

Disadvantages:

1. Hard to develop and maintain.
2. We cannot maintain the relationships between different entities.
3. Optimization of SQL code cannot be done by the container,because bean it self contains the code.
4. Not appropriate for larger and complex applications.

❖ **What is difference between EJB 1.1 and EJB 2.0?**

EJB 2.0 adds the local beans, which are accessible only from within the JVM where beans are running in.

In **EJB 1.1**, we had to implement remote client views for all these beans, even if we had no remote clients.

❖ **What is Message Driven Bean?**

Message Driven Bean (MDB) is an enterprise bean which runs inside the EJB container and it acts as Listener for the JMS asynchronous message . It does not have Home and Remote interface as Session or Entity bean. It is called by container when container receives JMS asynchronous message. MDB has to implement MessageListener which has a method onMessage(Message msg). When the container calls the MDB it passes the message to onMesage() method and then MDB process that message.

❖ **What is the life cycle of MDB?**

The lifetime of an MDB instance is controlled by the container. Only two states exist: Does not exist and Ready , as illustrated in the following figure:

The life of an MDB instance starts when the container invokes newInstance() on the MDB class to create a new instance. Next, the container calls setMessageDrivenContext() followed by ejbCreate() on the instance. The bean then enters the Ready state and is ready to consume messages.

When a message arrives for the bean, the container invokes the onMessage() method of one of the available instances, passing a Message object in argument. Message s can be consumed and processed concurrently by using multiple instances of the same type.

The container invokes ejbRemove() on the bean instance when it no longer needs the instance. The bean instance can perform clean up operations here.

- **What is local interface. How values will be passed?**
  If Client and EJB classes are in the same machine (Same JVM) then we can use Local linterface instead of Remote interface. Since Client and EJB are in same JVM, values are passed by referance.

- **What is the difference between local interface and remote interface?**
  We can describe the following common rules for choosing whether to use remote client view or local client view:
  When you will potentially use a distributed environment (if your enterprise bean should be independent of its deployment place), you should obviously choose remote client view.

  Use remote client view when you need to be sure that parameters passed between your EJB and the client (and/or other enterprise beans) should be passed "by value" instead of "by reference." With pass-by-value, the bean will have its own copy of the data, completely separated from the copy of the data at the client. With local client view, you can do pass-by-reference, which means your bean, as well as the client, will work directly with one copy of the data. Any changes made by the bean will be seen by the client and vice versa. Pass-by-reference eliminates time/system expenses for copying data variables, which provides a performance advantage.

  If you create an entity bean, you need to remember that it is usually used with a local client view. If your entity bean needs to provide access to a client outside of the existing JVM (i.e., a remote client), you typically use a session bean with a remote client view. This is the so-called Session Facade pattern, the goal of which is that the session bean provides the remote client access to the entity bean.

  If you want to use container-managed relationship (CMR) in your enterprise bean, you must expose local interfaces, and thus use local client view. This is mentioned in the EJB specification.

Enterprise beans that are tightly coupled logically are good candidates for using local client view. In other words, if one enterprise bean is always associated with another, it is perfectly appropriate to co-locate them (i.e., deploy them both in one JVM) and organize them through a local interface.

- **What is EJB Query Language?**

  EJB QL is somewat similar to SQL. But ejb ql is used to retrieve data from bean objects where as sql is used to retrieve data from tables.

- **What is ACID?**

  ACID is releated to transactions. It is an acronyam of Atomic, Consistent, Isolation and Durable. Transaction must following the above four properties to be a better one

  Atomic: It means a transaction must execute all or nothing at all.

  Consistent: Consistency is a transactional characteristic that must be enforced by both the transactional system and the application developer

  Isolation: Transaation must be allowed to run itselft without the interference of the other process or transactions.

  Durable: Durablity means that all the data changes that made by the transaction must be written in some type of physical storage before the transaction is successfully completed. This ensures that transacitons are not lost even if the system crashes.

- **What are the various isolation levels in a transaction and differences between them?**

  There are three isolation levels in Transaction. They are

  1. **Dirrty Reads:** If transaction A updates a record in database followed by the transaction B reading the record then the transaction A performs a rollback on its update operation, the result that transaction B had read is invalid as it has been rolled back by transaction A.
  2. **NonRepeatable Reads:** If transaction A reads a record, followed by transaction B updating the same record, then transaction A reads the same record a second time,

transaction A has read two different values for the same record.

3. **Phantom Reads:** If transaction A performs a query on the database with a particular search criteria (WHERE clause), followed by transaction B creating new records that satisfy the search criteria, followed by transaction A repeating its query, transaction A sees new, phantom records in the results of the second query.

- **What are the various transaction attributes and differences between them?**

There are six transaction attributes that are supported in EJB.

```
1. quired          -     T1---T1
                      0---T1
2. RequiresNew     -     T1---T2
                      0---T1
3. Mandatory       -     T1---T1
                      0---Error
4. Supports  -     T1---T1
                      0---0
5. NotSupported    -     T1---0
                      0---0
6. Never           -     T1---Error
                      0---0
```

- **What is the difference between activation and passivation?**

Activation and Passivation is appilicable for only Stateful session bean and Entity bean.

When Bean instance is not used for a while by client then EJB Container removes it from memory and puts it in secondary storage (often disk) so that the memory can be reused. This is called Passivation.

When Client calls the bean instance again then Container takes the passivated bean from secondary storage and puts it in memory to serve the client request. This is called Activation.

- **What is Instance pooling?**

pooling of instances n stateless session beans and Entity Beans server maintains a pool of instances.whenever server got a request from client, it takes one instance from the pool and serves the client request.

- ❖ **What is the difference between HTTPSession and Stateful Session Bean?**

From a logical point of view, a Servlet/JSP session is similar to an EJB session. Using a session, in fact, a client can connect to a server and maintain his state.

But, is important to understand, that the session is maintained in different ways and, in theory, for different scopes.

A session in a Servlet, is maintained by the Servlet Container through the HttpSession object, that is acquired through the request object. You cannot really instantiate a new HttpSession object, and it does not contains any business logic, but is more of a place where to store objects.

A session in EJB is maintained using the SessionBeans. You design beans that can contain business logic, and that can be used by the clients. You have two different session beans: Stateful and Stateless. The first one is somehow connected with a single client. It maintains the state for that client, can be used only by that client and when the client "dies" then the session bean is "lost".

A Stateless Session Bean does not maintain any state and there is no guarantee that the same client will use the same stateless bean, even for two calls one after the other. The lifecycle of a Stateless Session EJB is slightly different from the one of a Stateful Session EJB. Is EJB Containers responsability to take care of knowing exactly how to track each session and redirect the request from a client to the correct instance of a Session Bean. The way this is done is vendor dependant, and is part of the contract.

- ❖ **What is the difference between find and select methods in EJB?**

select method is not there in EJBs

A select method is similar to a finder method for Entity Beans, they both use EJB-QL to define the semantics of the method.

They differ in that an ejbSelect method(s) are not exposed to the client and the ejbSelect method(s) can return values that are defined as cmp-types or cmr-types.

❖ **What are the optional clauses in EJB QL?**
Three optional clauses are available in EJB Ql.
1. SELECT
2. FROM
3. WHERE
The EJB QL must always contain SELECT and FROM clauses. The WHERE clause is optional.

The FROM clause provides declarations for the identification variables based on abstract schema name, for navigating through the schema. The SELECT clause uses these identification variables to define the return type of the query, and the WHERE clause defines the conditional query.

❖ **What is handle in EJB?**
To get hold the session state of the Stateful Session bean.

A handle is an abstraction of a network reference to an EJB object. A handle is intended to be used as a "robust" persistent reference to an EJB object.

❖ **What is the difference between JNDI context, Initial context, session context and ejb context?**
JNDI Context Provides a mechanism to lookup resources on the network
Initial Context constructor provides the initial context.
Session Context has all the information a session bean would require from the container
Entity Context has all the information that an Entity bean would need from a container
Ejb Context contains the information that is common to both the session and entity bean.

- **What is the difference between sessioncontext and entitycontext?**
  Session Context Contains information that a Session Bean would require from the container
  Entity Context contains the information that an Entity Bean would require from a container

- **What is the difference between EAR, JAR and WAR file**
  In J2EE application modules are packaged as EAR, JAR and WAR based on their functionality
  JAR: Java Archive File
  EJB modules which contains enterprise java beans class files and EJB deployment descriptor are packed as JAR files with .jar extenstion
  WAR: Web Archive File
  Web modules which contains Servlet class files,JSP FIles,supporting files, GIF and HTML files are packaged as JAR file with .war(web achive) extension
  EAR: Enterprise File
  All above files(.jar and .war) are packaged as JAR file with .ear (enterprise archive) extension and deployed into Application Server.

- **What is deployment descriptor?**
  Deployment Descriptor is a XML document with .xml extenion. It basically descripes the deployment settings of an application or module or the component. At runtime J2EE server reads the deployment descriptor and understands it and then acts upon the component or module based the information mentioned in descriptor.

  For example EJB module has a deployment descriptor ejb-jar.xml where we mention whether it is session or entity or mesage driven bean and where is the home, remore and Bean classes are located and what type of transaction etc. In a simple word, without deployment descritor the Container (EJB/Servlet/JSP container) will not know what to do with that module.

  Deployment Descriptor is a file located in the WEB-INF directory that controls the behaviour of Servlets and JSP.

The file is called Web.xml and contains

```
                xmlHeader
Web.xml             DOCTYPE               Sevlet name
                Web-appelements    ------→   Servlet Class
                                   Init-parm
```

Servlet Configuration:

```
<web-app>
   <Servlet>
      <Servlet-name>Admin</Servlet-name>
      <Servlet-Class>com.ds.AdminServlet</Servlet-class>
   </Servlet>
   <init-param>
      <param-value></param-value>
      <param-name>admin.com</param-name>
   </init-param>
   <Servlet-mapping>
      <Servlet-name>Admin</Servlet-name>
      <url-pattern>/Admin</url-pattern>
   </Servlet-mapping>
</web-app>
```

EJB Deployment descriptor:

```
                Ejb-jar.xml
META-INF
                Weblogic-ejb-jar.xml
```

```
<ejb-jar>
   <enterprise-bean>
      </Session>
         <ejb-name>Statefulfinacialcalcu</ejb-name>
         <home>fincal.stateful.fincalc</home>
         <remote>fincal.stateful.fincalc</remote>
         <ejb-Class>fincal.stateful.fincalcEJB<ejb-Class>
         <session-type>Stateful</session-type>
         <transaction-type>Container</transaction-type>
      </Session>
   </enterprise-bean>
   <assembly-descriptor>
      <container-transaction>
         <method>
            <ejb-name>Statefulfinacialcalcu</ejb-name>
            <method-name> * </method-name>
         </method>
```

```
            <transaction-attribute>supports</transaction-
            attribute>
        </container-transaction>
    <assembly-descriptor>
<ejb-jar>
```

weblogic-ejb-jar.xml:

```
<weblogic-ejb-jar>
    <weblogic-enterprise-bean>
        <ejb-name>Statefulfinacialcalcu</ejb-name>
        <jndi-name>statefulfinacalc</jndi-name>
    </weblogic-enterprise-bean>
</weblogic-ejb-jar>
```

❖ **What is CMR?**

CMR - Container Managed Relationships allows the developer to declare various types of relationships between the entity beans.

❖ **What is the difference between CMP 1.1 and CMP 2.0?**

CMR and sub classing of the CMP bean by the container.

❖ **What is the difference between optimistic locking and pessimistic locking?**

Optimistic locking assumes that no one would read or change the data while changes are being by a bean.
Pessimistic locking would rather lock down the data so that no one can access it.

❖ **What is lazy loading?**

Lazy loading is a characteristic of an application when the actual loading and instantiation of a class is delayed until the point just before the instance is actually used. The goal is to only dedicate memory resources when necessary by only loading and instantiating an object at the point when it is absolutely needed.

Tools such as Eclipse have popularized the lazy-loading approach as they use the facility to control the load and initialization of heavyweight plug-ins. This gives the double bonus of speeding up the initial load time for the application, as not all plug-ins are loaded straightaway; ensuring efficiency as only the plug-ins that are used are loaded at all.

- **Is Decorator an EJB design pattern?**
  No, If i throw a custom ApplicationException from a business method in Entity bean which is participating in a transaction, would the transaction be rolled back by container. Does container rolls back transaction only in case of SystemExceptions.

  Yes, the rollback will occur For declarative transactions, container will rollback on systemException. Container has no way to know whether a speicific application exception is serious enough to roll back the participated transaction. Use setRollbackOnly() to doom the transaction.

- **What are simple rules that a Primary key class has to follow?**
  Implement the equals and hashcode methods.

- **What is abstract schema?**
  CMP uses abstract schema to map to the physical database.

- **What is re-entrant. Is session beans reentrant. Is entity beans reentrant?**
  Re-entrant means where Bean A calls methods of Bean B and then Bean B turns around and calls methods of Bean A. The above all within a single thread of control. This is also called as loopback.

  Entity beans are the only one bean that is reentrant. Neither Session bean nor Message Driven Bean are reentrant. When Entity bean, we have to declare in the deployment descriptor whether it is reentrant (true or false).

  Why an onMessage call in Message-driven bean is always a seperate transaction.

  The MDB is stateless and inherently each message is unique with respect to the MDB.

Each message needs to be processed independently. Hence the need for separate transactions

- **Does Stateful Session bean support instance pooling?**
  All Beans support Instance Pooling
  statefull session bean does not maintain instance pooling,
  stateless session beans and entity beans can maintain instance pooling

- **Why does EJB needs two interfaces(Home and Remote Interface)?**
  Home is to provide Lookup from JNDI while Remote is to provide Object Instantiated

- **Can I invoke Runtime.gc() in an EJB?**
  No

- **Can a Session Bean be defined without ejbCreate() method?**
  No

- **Why are ejbActivate() and ejb Passivate() included for stateless session bean even though they are never required as it is nonconversational bean?**
  To have a consistent interface, so that there is no different interface that you need to implement for Stateful Session Bean and Stateless Session Bean. Both Stateless and Stateful Session Bean implement javax.ejb.SessionBean and this would not be possible if stateless session bean is to remove ejbActivate and ejbPassivate from the interface. You could argue that the two (stateful and stateless) are so different that they should have their own interface but Sun did not think so. They made both session beans implement the same interface and provided deployment descriptor to denote which one is it that you are deploying.

  With EJB 1.1 specs, why is unsetSessionContext() not provided in Session Beans, like unsetEntityContext() in Entity Beans.

This was the answer Provided by some one... According to Mohan this one is not correct. Please see Mohan's reply below and more in the comments section.

ejbRemove() is called for session beans every time the container destroyes the bean. So you can use this method to do the stuff you typically would do in unsetEntityContext(). For entity beans ejbRemove() is only called if the user explicitly deletes the bean. I think that is the reason why the engineers at SUN invented the unsetEntityContext() for this kind of bean.

❖ **What is the difference between ejbStore() and ejbLoad()?**
When the EJB container needs to synchronize the instance variables of an entity bean with the corresponding values stored in a database, it invokes the ejbLoad and ejbStore methods. The ejbLoad method refreshes the instance variables from the database, and the ejbStore method writes the variables to the database. The client cannot call ejbLoad and ejbStore.

❖ **What is the difference between ejbCreate() and ejbPostCreate()?**
Session and Message Driven Bean will have only ejbCreate() method and no ejbPostCreate() method. Entity bean will have both ejbCreate() and ejbPostCreate() methods.

The ejbPostCreate method returns void, and it has the same input parameters as the ejbCreate method. If we want to set a relationship field to initialize the bean instance, we should do so in the ejbPostCreate method. we cannot set a relationship field in the ejbCreate method.

The ejbPostCreate() allows the bean to do any post-create processing before it begins serving client requests. For every ejbCreate() there must be a matching (matching arguments) ejbPostCreate() method.

❖ **Is stateless Sessiob bean create() method contains any parameters?**

No. This method must not contain any input parameters and cannot be overloaded as well.

- **How can i retrieve from inside my Bean(Stateless session and Entity CMP) the user name which i am serving (the user name of user just logged in my web application)?**
  Inside an EJB you may retrieve the "Caller" name, that is the login id by invoking: sessionContext.getCallerIdentity().getName() where sessionContext is the instance of "SessionContext" (setSessionContext) passed to the Session Bean, or the instance of "EntityContext" (setEntityContext) passed to the Entity Bean.

- **What is EJB architecture(components)**
  EJB Architecture consists of:
  a) EJB Server
  b) EJB containers that run on these servers,
  c) Home Objects, Remote EJB Objects and Enterprise Beans that run within these containers,
  d) EJB Clients and
  e) Auxillary systems like JNDI (Java Naming and Directory Interface), JTS(Java Transaction Service) and security services.

If my session bean with single method insert record into 2 entity beans, how can I know that the process is done in same transaction (the attributes for these beans are Required)
If your method in the session bean is already running under a transaction the calls to any other bean which have been deployed with trans-attribute 'Required' will be executed within the same transaction context.

So if your session bean is using container-managed transactions and your method is deployed with 'Required', 'RequiresNew' or 'Mandatory', you can safely assume that the calls to your entity beans are handled under same transaction. If you're not running in a transaction, a separate transaction will be set up for each call to your entity beans.

If your session bean is using bean-managed transactions, you can ensure that the calls are handled in the same transaction by:

```
javax.transaction.UserTransaction tran= null;
try {
   tran=ctx.getUserTransaction();
   tran.begin();
   myBeanHome1.create(....);
   myBeanHome2.create(...);
   tran.commit();
} catch(...) {
}
```

You may want to check if you're already running in a transaction by calling tran.getStatus().

- **Is there a way to get the original exception object from inside a nested or wrapped Exception (for example an EJBException or RemoteException)?**

  Absolutely yes, but the way to do that depends on the Exception, since there are no standards for that.
  Some examples: When you have an javax.ejb.EJBException, you can use the getCausedByException() that returns a java.lang.Exception. ·A java.rmi.RemoteException there is a public field called detail of type java.lang.Throwable ·With a java.sql.SQLException you need to use the method getNextException() to get the chained java.sql.SQLException. ·When you have an java.lang.reflect.InvocationtargetException, you can get the thrown target java.lang.Throwable using the getTargetException() method.

- **Can undefined primary keys are possible with Entity beans?If so, what type is defined?**

  Yes,undefined primary keys are possible with Entity Beans.The type is defined as java.lang.Object.

- **When two entity beans are said to be identical?Which method is used to compare identical or not?**

  Two Entity Beans are said to be Identical,if they have the same home inteface and their primary keys are the same.To test for this ,you must use the component inteface's isIdentical() method.

- **Why CMP beans are abstract classes?**
  We have to provide abstract data to object mapping that maps the fields in our bean to a batabase, and abstract methods methods that corelate these fields.

- **Is instance pooling necessary for entity beans?**
  One of the fundamental concepts of Entity Beans is that they are the pooled objects.Instance pooling is the service of the container that allows the container to reuse bean instances,as opposed to creating new ones every time a request for a bean is made.This is a perfomance optimizatio done by the container.

- **What is the difference b/w sendRedirect() and <jsp: forward>?**
  sendredirect will happen on clint side & request , rsponse will be newly created, for forward action it is server side action & request, response is passed & not modified or destroyed.

- **How the abstract classes in CMP are converted into concrete classes?**
  EJB2.0 allows developer to create only abstract classes and at the time of deployement the container creates concrete classes of the abstract. It is easy for container to read abstract classes and appropriately generate concrete classes.

Questions

1. **A developer successfully creating and tests a stateful bean following deployment, intermittent "NullpointerException" begin to occur, particularly when the server is hardly loaded. What most likely to related problem.?**
   a) setSessionContext
   b) ejbCreate
   c) ejbPassivate
   d) beforeCompletion
   e) ejbLoad

2. **2 example implementations os Proxy are RMI & Ejb?**

3. **If an RMI parameter implements java.rmi.Remote, how is it passed "on-the-wire?"**
   Choice 1: It can never be passed.
   Choice 2: It is passed by value.
   Choice 3: It cannot be passed because it implements java.rmi.Remote.
   Choice 4: It cannot be passed unless it ALSO implements java.io.Serializable.
   Choice 5: It is passed by reference.
   Ans: 2

4. **public synchronized void txTest(int i) {**
   System.out.println("Integer is: " + i);
   }
   What is the outcome of attempting to compile and execute the method above, assuming it is implemented in a stateful session bean?
   Choice 1: Run-time error when bean is created.
   Choice 2: The method will run, violating the EJB specification.
   Choice 3: Compile-time error for bean implementation class.
   Choice 4: Compile-time error for remote interface.
   Choice 5: Run-time error when the method is executed.
   Ans: 2

5. **What is the CORBA naming service equivalent of JNDI?**
   Choice 1: Interface Definition Language.

Choice 2: COS Naming.
Choice 3: Lightweight Directory Access Protocol.
Choice 4: Interoperable Inter-Orb Protocol.
Choice 5: Computer Naming Service
Ans: 2

6. **InitialContext ic = new InitialContext();
TestHome th = (TestHome)
ic.lookup("testBean/TestBean");
TestRemote beanA = th.create();
TestRemote beanB = th.create();
TestRemote beanC = th.create();
beanC.remove();
TestRemote beanD = th.create();
TestRemote beanE = th.create();
beanC = th.create();
Given the above code, container passivates which bean instance first if the container limited the bean pool size to four beans and used a "least-recently-used" algorithm to passivate?**
Choice 1 : Bean A
Choice 2 : Bean B
Choice 3 : Bean C
Choice 4 : Bean D
Choice 5 : Bean E

7. **Which one of the following phenomena is NOT addressed by read-consistency?**
a) Phantom read
b) Cached read
c) Dirty read
d) Non-repeatable read
e) Fuzzy read
Ans: b,e

8. **Which one of the following methods is generally called in both ejbLoad() and ejbStore()?**
a) getEJBObject()
b) getHandle()
c) remove()
d) getEJBHome()

e) getPrimaryKey()
Ans: e

9. **public void ejbCreate(int i) {**
```
   System.out.println("ejbCreate(i)");
}
```
Given a currently working stateless session bean, what will be the outcome upon deploying and executing the bean if you added the above unique method to the implementation class of a stateless session bean (and made no other changes)?
a) Compile time error during stub/skeleton generation.
b) Compile time error for home interface.
c) Code will compile without errors.
d) Compile time error for remote interface.
e) Compile time error for bean implementation.
Ans: a

10. **Given the above code in your stateless session bean business method implementation, and the transaction is container-managed with a Transaction Attribute of TX_SUPPORTS, which one of the following is the first error generated?**
a) Error when compiling home interface
b) Error while generating stubs and skeletons
c) NullPointerException during deployment
d) Runtime error
e) Compile-time error for the bean implementation
Ans: b

11. **Which one of the following is the result of attempting to deploy a stateless session bean and execute one of the method M when the bean implementation contains the method M NOT defined in the remote interface?**
a) Compile time error for remote interface.
b) Compile time error for bean implementation.
c) Compile time error during stub/skeleton generation.
d) Code will compile without errors.
e) Compile time error for home interface.
Ans: d

12. **Which one of the following characteristics is NOT true of RMI and Enterprise Java Beans?**
   a) They must execute within the confines of a Java virtual machine (JVM).
   b) They serialize objects for distribution.
   c) They require .class files to generate stubs and skeletons.
   d) They do not require IDL.
   e) They specify the use of the IIOP wire protocol for distribution.
   Ans: a

13. **Which one of the following is the result of attempting to deploy a stateless session bean and execute one of the method M when the bean implementation contains the method M NOT defined in the remote interface?**
   a) Compile time error for remote interface.
   b) Compile time error for bean implementation.
   c) Compile time error during stub/skeleton generation.
   d) Code will compile without errors.
   e) Compile time error for home interface.

14. **If a unique constraint for primary keys is not enabled in a database, multiple rows of data with the same primary key could exist in a table. Entity beans that represent the data from the table described above are likely to throw which exception?**
   a) NoSuchEntityException
   b) FinderException
   c) ObjectNotFoundException
   d) RemoveException
   e) NullPointerException

15. **A developer needs to deploy an Enterprise Java Bean, specifically an entity bean, but is unsure if the bean container is able to create and provide a transaction context. Which attribute below will allow successful deployment of the bean?**
   a) BeanManaged
   b) RequiresNew
   c) Mandatory
   d) Required

e) Supports

16. **What is the outcome of attempting to compile and execute the method above, assuming it is implemented in a stateful session bean?**
    a) Compile-time error for remote interface
    b) Run-time error when bean is created
    c) Compile-time error for bean implementation class
    d) The method will run, violating the EJB specification.
    e) Run-time error when the method is executed

17. **Which one of the following is the result of attempting to deploy a stateless session bean and execute one of the method M when the bean implementation contains the method M NOT defined in the remote interface?**
    a) Compile time error for remote interface.
    b) Compile time error for bean implementation.
    c) Compile time error during stub/skeleton generation.
    d) Code will compile without errors.
    e) Compile time error for home interface.

18. **If a unique constraint for primary keys is not enabled in a database, multiple rows of data with the same primary key could exist in a table. Entity beans that represent the data from the table described above are likely to throw which exception?**
    a) NoSuchEntityException
    b) FinderException
    c) ObjectNotFoundException
    d) RemoveException
    e) NullPointerException

There are two Enterprise Java Beans, A and B. A method in "A" named "Am" begins execution, reads a value v from the database and sets a variable "X" to value v, which is one hundred. "Am" adds fifty to the variable X and updates the database with the new value of X. "Am" calls "Bm", which is a method in B. "Bm" begins executing. "Bm" reads an additional value from the database. Based on the value, "Bm" determines that a business rule has been violated and aborts the transaction. Control is returned to "Am".Requirement: If "Bm" aborts the transaction, it is imperative that the original value be read from the database and stored in variable X.

19. Given the scenario above, which Transaction Attributes will most likely meet the requirements stated?

a) A-RequiresNew, B-Mandatory
b) A-Mandatory, B-RequiresNew
c) A-RequiresNew, B-Supports
d) A-NotSupported, B-RequiresNew
e) A-RequiresNew, B-RequiresNew

20. If an RMI parameter implements java.rmi.Remote, how is it passed "on-the-wire?"

Choice 1: It can never be passed.
Choice 2: It is passed by value.
Choice 3: It cannot be passed because it implements java.rmi.Remote.
Choice 4: (Correct) It cannot be passed unless it ALSO implements java.io.Serializable.
Choice 5: It is passed by reference.

21.
```
public synchronized void  txTest(int i)  {
    System.out.println("Integer is: " + i);
}
```
What is the outcome of attempting to compile and execute the method above, assuming it is implemented in a stateful session bean?

Choice 1: Run-time error when bean is created.
Choice 2: The method will run, violating the EJB specification.
Choice 3: (Correct) Compile-time error for bean implementation class.
Choice 4: Compile-time error for remote interface.
Choice 5: Run-time error when the method is executed.

22. What is the CORBA naming service equivalent of JNDI?

Choice 1: Interface Definition Language
Choice 2: (Correct) COS Naming
Choice 3: Lightweight Directory Access Protocol
Choice 4: Interoperable Inter-Orb Protocol

Choice 5: Computer Naming Service

**23. InitialContext ic = new InitialContext();
TestHome th = (TestHome)
ic.lookup("testBean/TestBean");
TestRemote beanA = th.create();
TestRemote beanB = th.create();
TestRemote beanC = th.create();
beanC.remove();
TestRemote beanD = th.create();
TestRemote beanE = th.create();
beanC = th.create();
Given the above code, container passivates which bean instance first if the container limited the bean pool size to four beans and used a "least-recently-used" algorithm to passivate?**

Choice 1: Bean A
Choice 2: Bean B
Choice 3: Bean C
Choice 4: Bean D
Choice 5: Bean E

Ans: 4 (Correct, Since only Statefull session bean and Entity Bean can be passivated, and Entitybean can not call as th.create() normally, I take it as statefull session bean)

24. Which one of the following phenomena is NOT addressedby read-consistency?

a) Phantom read (Correct)
b) Cached read
c) Dirty read
d) Non-repeatable read
e) Fuzzy read

25. Which one of the following methods is generally called in both ejbLoad() and ejbStore()?

a) getEJBObject()
b) getHandle()
c) remove()
d) getEJBHome()
e) getPrimaryKey() (Correct)